UPPSALA
UNIVERSITET

UPTEC X 15 017
Examensarbete 30 hp
Mars 2016

# Towards automatic smartphone analysis for point-of-care microarray assays

Julia Erkers

UPPSALA UNIVERSITET

# Degree Project in Bioinformatics

Masters Programme in Molecular Biotechnology Engineering,
Uppsala University School of Engineering

| **UPTEC X 15 017** | **Date of issue 2016-03** |
|---|---|


Author

**Julia Erkers**


Title

## Towards automatic smartphone analysis for point-of-care microarray assays


Abstract

Poverty and long distances are two reasons why some people in the third world countries has difficulties seeking medical help. A solution to the long distances could be if the medical care was more mobile and diagnostically tests could be performed on site in villages. A new point-of-care test based on a small blood shows promising results both in run time and mobility. However, the method still needs more advanced equipment for analysis of the resulting microarray. This study has investigated the potential to perform the analysis within a smartphone application, performing all steps from image capturing to a diagnostic result. The project was approach in two steps, starting with implementation and selection of image analysis methods and finishing with implementing those results into an Android application. A final application was not developed, but the results gained from this project indicates that a smartphone processing power is enough to perform heavy image analysis within a sufficient amount of time. It also imply that the resolution in the evaluated images taken with a Nexus 6 together with an external macro lens most likely is enough for the whole analysis, but further work must be done to ensure it.



Keywords

Point-of-care test, image analysis, diagnostic tool, smartphone application



Supervisors

**Jesper Gantelius**
**Science for Life Laboratory**

Scientific reviewer

**Ida-Maria Sintorn**
**Uppsala University**


| Project name | Sponsors |
|---|---|
| Language **English** | Security |
| **ISSN 1401-2138** | Classification |
| Supplementary bibliographical information | Pages **60** |


**Biology Education Centre** Biomedical Center Husargatan 3, Uppsala
Box 592, S-751 24 Uppsala Tel +46 (0)18 4710000 Fax +46 (0)18 471 4687

# Towards automatic smartphone analysis for point-of-care microarray assays

*Julia Erkers*

## Populärvetenskaplig sammanfattning

Inom sjukvården används idag avancerad utrustning för att ställa diagnoser på patienter och analysverktygen är ofta stora laboratoriemaskiner som kräver utbildad personal och tar tid, det kan ta uppemot dagar innan en patient får sitt provsvar. Detta är ett problem i alla samhällen men kanske främst i fattiga- och krigshärjadeländer där den vårdsökande kan ha problem både att ta sig till ett sjukhus samt inte har möjlighet att återkomma för att få provsvaret. En forskargrupp vid Kungliga tekniska högskolan i Stockholm har tagit fram en patientnärametod som är baserad på en liten mängd blodprov vilket körs på en mikropappersmatris där blodprovet flödar vertikalt genom pappersmatrisen. Metoden är snabb, analystiden ligger på ungefär 10 minuter, och billig men dess främsta kvalitet är mobiliteten. Hela metoden körs i en liten cylindrisk behållare där provet appliceras med hjälp av en spruta. Det patientnäratestet har hittills testats i form av ett allergitest där det visat lovande resultat och hoppet är att med den ökade kunskapen om biomarkörer kunna använda metoden för diagnostisering av sjukdomar. Problemet med den vertikala flödesmetoden är att analysen av resultatet kräver mer avancerad utrustning och därmed är en begränsande faktor i mobiliteten. Detta projekt har undersökt möjligheten att köra analysen av papperamatrisen i en mobilapplikation, om det är möjligt skulle metoden bli fullständigt mobil och patienten skulle kunna få provsvaren inom en timme. Resultatet från det patientnäratestet är en liten pappersmatris med prickar på, vilka har en diameter på 120 µm. Desto tydligare en prick framträder i matrisen desto högre är koncentrationen i blodet finns det av den substans som pricken motsvarar.

Projektet delades in i två faser, den första i vilken bildanalysmetoder studerades och valdes ut och en andra fas där dessa metoder implementerades i en mobilapplikation. Bildanalysen involverade förbehandling av bilden för att få maximal information om prickarna på pappersmatrisen men även rotation, beskärning av bilden och en mappning av mönstret i bilden men prickarnas kända koordinater, de kända koordinaterna innehöll information om vilken substans varje prick motsvarade. Resultatet av studien har indikerat att dagen telefoner har tillräcklig processorkraft för att kunna utföra tunga bildanalytiska beräkningar inom en önskvärd tid. Studien har inte utvecklat en fullständig mobilapplikation men resultaten av förbildbehandlingen har gett väldigt lovande resultat. Bildkvaliteten, i bilder som tagits med en telefonkamera med ett externt makroobjektiv, har i analyserna visat på hög kvalitet och varje prick har representerats av en stor mängd pixlar vilket är en viktig del för att i senare steg kunna analysera prickarnas intensitet, vilket sedan översätts till koncentration. För att säkert kunna avgöra om en det går att utföra analysen av en pappersmatris med hjälp av en mobilapplikation krävs att applikationen färdigställs och jämförs med tidigare resultat. Men denna studie pekar i riktning på att smarttelefoner kommer spela en allt viktigare roll inom sjukvården i framtiden.

**Examensarbete 30 hp**
**Civilingenjörsprogrammet i Molekylär bioteknik,**
**inriktning Bioinformatik**

**Uppsala universitet, mars 2016**

# Table of Content

# Abbreviations

| | |
|---|---|
| ADT | Android Developer Tool |
| GUI | Graphical User Interface |
| IDE | Integrated Development Environment |
| NDK | Native Developer Kit |
| OpenCV | Open Computer Vision |
| PCA | Principal Component Analysis |
| POC | Point-of-care |
| QR code | Quick Response Code |
| SE | Structural Element |
| SRG | Seed Region Growing |
| TADP | Tegra Android Developer Pack |
| UI | User Interface |

# 1. Introduction

## 1.1 Background

Immunoassay-based affinity proteomic analysis can be used for a wide variety of diagnostics such as autoimmunity, clinical microbiology and oncology (Chinnasamy et al., 2014). With new knowledge originating from the development of protein libraries, for example The Human Protein Atlas, it has become possible to discover novel biomarkers which can be used to distinguish between certain diseases with high accuracy. However, current multiplexed affinity proteomic tools have suffered from long assay times and the need of highly sophisticated laboratory equipment as well as trained operators. Chinnasamy et al states that these limitations has gained an increased interest for paper-based arrays among researchers. There are also many potential benefits with paper-based arrays which is another urge in the research, benefits could for example be: a lower cost per assay, shorter run time, an ease of use and robustness regardless of environment.

Chinnasamy et al. (2014) published a paper showing proof of concepts regarding their own development of a point-of-care (POC) vertical flow allergen microarray assay. The progress was mainly driven by the ambition to lower analysis cost, speed up the assay time, and make a method that is more mobile, not dependent on sophisticated laboratory equipment. The vertical flow microarray assay developed at the KTH group is a paper based method, using a nitrocellulose membrane on which a high-density microarray pattern is printed. The pattern consists of microspots with different human allergens, with a spot size of about 120 µm in diameter. The analysis is performed using a sandwich-based immunoassay in a vertical flow approach. To keep a constant flow through the array, a syringe pump with a controlled flow is used. The paper array is printed with different allergens and when serum is flowed through, IgE antibodies will recognise and bind to their specific allergen. In order to retrieve a colorimetric representation of the antibodies, the microarray is then exposed by anti-hIgE-conjugated gold nanoparticles. The method has an assay time less than 10 minutes and has shown a sensitivity of 1 ng/mL. However, the membrane imaging and the data analysis is still dependent on scanner and computer respectively.

## 1.2 Project scope

This study is a partial study of an ongoing research project between the division of Nanobiotechnology, KTH, Scilifelab, the department of Global Health, Karolinska Institutet in Sweden and the MSF Epicenter / Mbarara University Hospital in Uganda. The project has identified a need of new diagnostic tools which must be portable for bringing out to small villages in the country, this because of big distances and poverty. This partial project has investigated the potential of using a smartphone application for both imaging paper array result and perform a data analysis which could result in a more portable and easy to use diagnostic tool. The data analysis consists of an image processing part in which the array spots must be recognised and their respective values extracted and paired with the allergen which was printed in the spot. The project included an exhaustive literature study which was performed to identify all the needed steps in the whole smartphone application. It also included parts that could cause problems even though it showed no signs of it during this study. Because of the short time span of the project everything in the literature study was not implemented in the smartphone application. Instead, it was limited to only develop a pipeline

for some parts of the image analysis, which were information about the array spots and include image enhancement, rotation and cropping. During the project, image analysis was performed in both Matlab and Android. Matlab was chosen since it was familiar to me and provides a great amount of documentation. Android was then chosen for the smartphone application development as a request from the client. This project is highly motivated because if it is possible, then the analysis can be carried out anywhere and most importantly on site with patients. It means that patients who are not able to reach a doctor because of long distances, disabilities, poverty etc. could receive a housecall from a doctor, who then could perform a POC test to search for infections and diseases.

## 1.3 Aim

The aim of this thesis was to investigate the possibility to develop a smartphone application that could manage the processing requirements to carry out image analysis within a sufficient amount of time. Requirements stated in the project concerned the total assay time, the application should not take more than 20-30 minutes to run, and the developing was not allowed any great investments, i.e., the developing process and license was asked for to be kept as cheap as possible. A reason to keep down the cost and for example reject expensive licences was because it would result in a cheaper application which can be used worldwide. The users, doctors and organisations, would not have to pay for using the licence. It would probably result in a greater dispersion of the application and thereby help more patients than otherwise.

# 2. Image analysis

Image analysis is a major field within computer science which has the purpose to extract valuable information from images. It is comparatively simple to read a barcode or apply a filter for noise reduction but significantly harder to extract or recognize objects. Images consists of pixels that are the smallest component in an image, having a square shape. Each pixel is assigned an intensity value, i.e., a value describing how the colour in that particular pixel should be expressed. Images are made up by many pixels and when working with 2-dimensional image, the image is described as a matrix $f$ of size M×N, where M and N corresponds to the number of rows and columns respectively. When referring to a pixel intensity in an image the convention $f(x, y)$ is used to describe the intensity in the image at position (x, y).

The process of acquiring an image is an imperfect process that results in a degraded version of the original object or scene. Errors in the acquisition can occur from technical aspects such as lens aberration or wrong focus, as well as other factors such as the surrounding light which can affect the interpretation of the image. In image analysis these factors are adjusted in the different steps shown in Figure 1. Errors that might occur due to lens aberration needs to be taken care of before any analysis can take part. Image distortion is when an image appear to be curved, which is important to correct for if any feature extraction will be performed. A pre-processing step is implemented both to adjust the image for uneven illumination as well as for reducing noise. Image rotation and cropping is implemented here to make it possible to perform a mapping between the array spots in the image with a coordinate file, which contains information about what substance that is found in each spot. The matching is based on centroids in the image and a coordinate file, when the matching is done a segmentation is carried out which assigns pixels to either be a part of a spot or a part of the backrest. The substance concentration is measured by extracting the spot intensity value from the segmented spot, a value that is normalised to be comparable with other results and then presented in an easily interpretable way for the user.

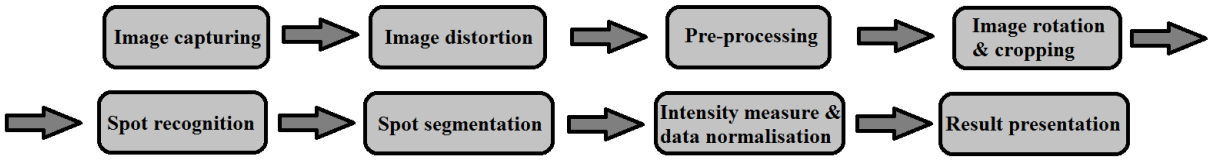


*Figure 1: The considered work flow for the image analysis. From start when the image is captured to the final result with the concentration within each array spot.*

## 2.1 Image distortion

If the ambition is to extract or measure an image's geometrical features, image distortion might be of concern. Geometrical distortion is when the image is curved or bent at the edges. Fish-eye lenses are examples of distortion devices employed to create an extra curved effect to the image. Geometrical distortion refers to point position displacement and results from the camera lens design. The lenses have been imperfectly assembled to each other, causing a non-perfect symmetry which might be due to precision problems. There are two types of geometrical distortions, negative- and positive displacement. Negative displacement, also called barrel distortion, narrows the points closer together and closer to the optical centre of the image, see Figure 2a. The result of barrel distortion is not simply straight lines that appear

curved, but it also causes a decrease in scale further away from the optical centre. Positive displacement or pincushion distortion results in placing the image points further apart from each other and away from the optical centre, see Figure 2b. This will create the illusion of the object's scale to increase at the edges of the image (Weng et al., 1992).

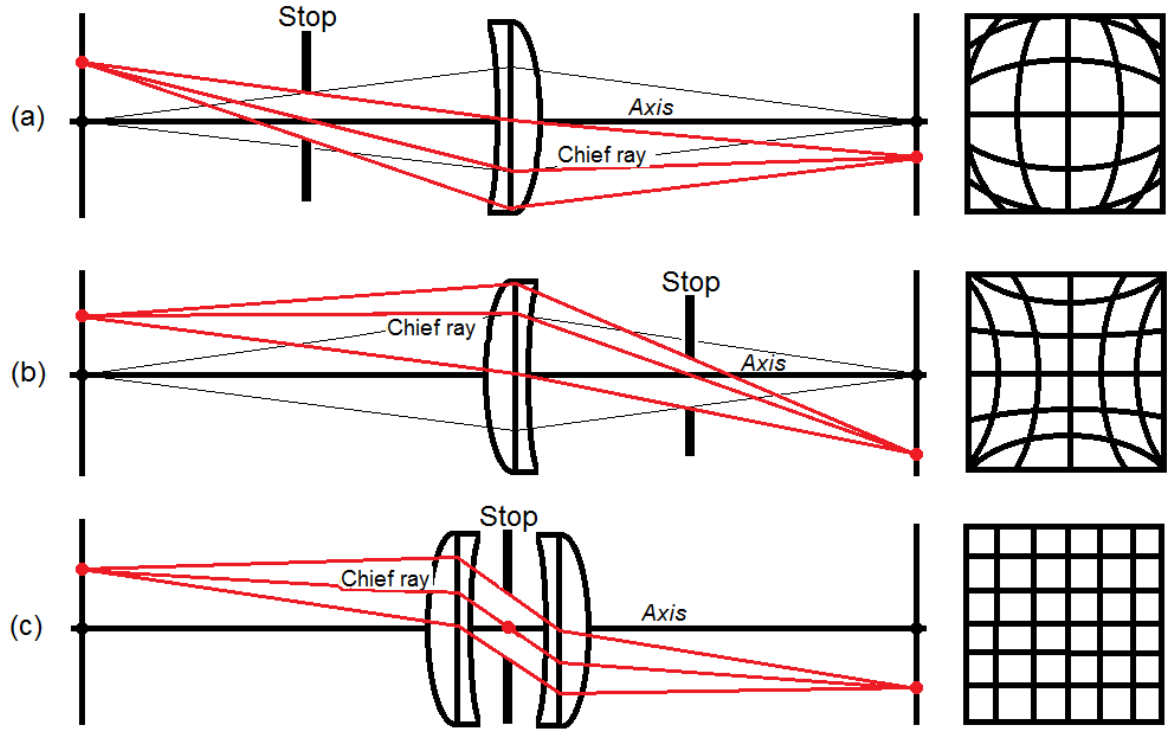


*Figure 2: Different types of radial distortion and how they arise. The figure shows how distortion arises when introducing a stop before and after the lens (a-b) and how radial distortion can be avoided by introducing the stop between two mirrored lenses (c) (Jenkins and White, 1957). The different radial distortion is barrel distortion (a), pincushion distortion (b) and orthoscopic doublet, i.e. correction for distortion (c).*

Distortion arises when a stop is placed before or after a thin lens, see Figure 2a-b, left. If a stop is placed in front of the lens, rays from an object close to the axis will also pass the lens close to the axis. Because of the better symmetry in the middle of the lens, rays passing there will result in the image point placed where it is expected to. Chief rays, which are rays coming from objects placed off-axis and which passes through the centre of a stop (seen as the middle red line in Figure 2a), will cause the rays to refract only in the lower part of the lens. The stop causes a change in the ratio between the object and image, which in turn lowers the lateral magnification of the rays compared to the magnification of rays belonging to objects close to the axis, which results in positioning the image point closer to the optical centre. In pincushion distortion for which the stop is placed after the lens which will lower the distance to the image i.e., increase the ray magnification and therefore spread the image points further away from the optical centre (see Figure 2b, left). To overcome the problem two identical thin lenses are placed as mirrors of each other and a stop is placed between them as seen in Figure 2c. In the optimal case, where the symmetry between them is perfect and the magnification is constant, the resulting images will be free from distortion (Jenkins and White, 1957).

Barrel distortion is more common in cheap, wide-angle camera lenses (Alvarez et al., 2009; Park et al., 2009; Xiaochuan et al., 2009) and signs of barrel distortion is often seen in

smartphones and external camera lenses for smartphones. Research has significantly increased in this area as machine vision and precision matching has become more important (Xiaochuan et al., 2009). But there was no need to look further into image distortion in this project since no signs of distortion was found in the images. It was however important to keep image distortion in mind throughout the project since the images that were analysed were captured by a smartphone with an external camera lens, which should be considered a high risk for developing image distortion.

## 2.2 Pre-processing

The paper-microarrays is scanned into a computer and their fluorescence intensity level is analysed against a dark background, which may results in images with low noise levels and high contrast. Here the images are captured by a smartphone camera which not only capture the desired array but also part of the area beyond the array, such as for example a table. The light in the image may vary and can even result in a light gradient. Another problem that might arise is if the paper array is damage, which might be result from scratches and holes. Such image artefacts can have impact on the overall result in the image analysis and must therefore be dealt with, see Figure 3 which exemplifies the problem with damaged paper arrays. When taking all these issues into account, some extra steps are required before the image analysis can be performed.

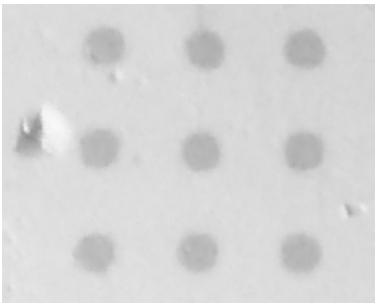

*Figure 3: A part of a paper array used in the project. To the left of the array is a hole which can cause problem with the image analysis.*

Filtering the image is a common way to reduce noise and correct for uneven illumination. A filter is defined as a matrix of size N×N which is applied on all positions in the image. For example, an average filter of size 3×3 will average the nine pixels within the filter and use the result as the new pixel value, see Figure 4. The filter is then moved one step at the time - resulting in new pixel values. An average filter is used to reduce the amount of noise and the result is a more or less blurred version of the original image.

To correct an image which contains uneven illumination another type of filter can be applied, namely a morphological Top-Hat operation or a morphological Bot-Hat. These operations are defined as subtraction of an image with an applied opening or closing operation from the original image. The choice of Top-Hat or Bot-Hat for an illumination correction depends on whether the object in the image are light (1) or dark (2). The operations are defined as:

$$f_{TOP}(x,y) = f - (f \circ s) \quad (1)$$

$$f_{BOT}(x,y) = f - (f \bullet s) \quad (2)$$

where s is a defined structuring element (SE), i.e., a defined shape of the filter, the white circle is the symbol for Opening and the black circle means Closing. Opening and Closing are two morphological operations, other than Top-Hat and Bot-Hat, they both tend to smooth contours in the image to some extent. The general difference between Opening and Closing is that Opening breaks or eliminates small objects and stripes while Closing fuses narrow objects and fill holes in objects (Gonzalez and Woods, 2007). What a morphological operation (Top-Hat, Bot-Hat, Opening, Closing etc.) will result in, is dependent on the size

and shape of the SE. In this project the SE was in most cases chosen in order to eliminate small, unwanted objects.

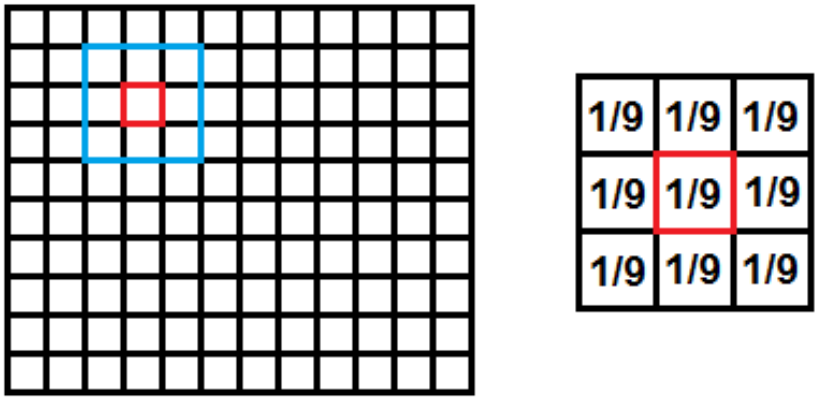


*Figure 4: Average filter. An image with an applied average filter, blue lines indicate pixels used to calculate the red pixel's new value (left). The structure of the filter (right).*

## 2.3 Image rotation

The array orientation is important for matching the frame spots of the array to a coordinate file, which contain information about how all spots lie in relation to each other. The matching is a step which should result in identification of spots with little or no spot intensity i.e., spots that cannot be detected with the eye. When the image of the paper array is captured it could be convenient to assume that the orientation of the array is not optimal and that it is necessary to rotate the image somewhere between $-180° \leq \theta \leq 180°$.

Two different methods for identifying the angle to rotate the image has been investigated: principal component analysis (PCA) and Hu's moment invariants. Hu's moment invariants was chosen for this task, since the moment invariants was found to be more robust and less sensitive to noise than the PCA.

### 2.3.1 Principal component analysis

PCA is considered to be one of the most used and also one of the oldest methods and has been traced back to the 19th century (Abdi and Williams, 2010). It was given its name and the modern formulation, which is still used today, by the mathematical statistician Hotelling. PCA is used to extract important underlying information from observed data and presents this in a new set of orthogonal variables which is called principal components.

In image analysis PCA provides a useful way to e.g., normalize object boundaries and regions for variation in translation, size and orientation. The generated segmented objects (i.e., pixels in the binary image with a value of 1) are treated as the observations or as data points. Each pixel is described as a 2-D vector $\mathbf{x} = [x_1, x_2]^T$ in which $x_1$ and $x_2$ represent the pixel coordinates and the T indicates transpose (Gonzalez and Woods, 2007). The collection of all object points are then used to calculate the mean vector $\mathbf{m_x}$ and the covariance matrix $\mathbf{C_x}$. For a sample of N x-vectors the $\mathbf{m_x}$ vector is calculated as an ordinary averaging:

$$m_x = \left(\frac{1}{N}\right) \sum_{n=1}^{N} (x_n) \qquad (3)$$

$\mathbf{C_x}$ is then approximated from the pixels and the mean vector as follows:

$$C_x = \frac{1}{N}\sum_{n=1}^{N}(x_n x_n^T - m_n m_n^T) \quad (4)$$

The generated eigenvectors from the covariance matrix display the direction of data distribution, where the first eigenvector is associated with the greatest eigenvalue, i.e., the greatest data spread. The largest eigenvector can therefore be used along with a vector which points out the desired direction. Linear algebra is used for calculating the angle needed for rotating the image.

### 2.3.2 Moment invariants

Another way to calculate the image rotation is to use moment invariants, a method based on the objects physical properties. The rotation method used in the Android smartphone application is based on Hu's seven moment invariants. It is now over 50 years since Hu published his paper in which he stated the mathematical foundation of 2D moment invariants for translation, scaling and rotation. He was also the first one to highlight their utility in image analysis. Since then, his invariants has been further developed and more invariants have been generated and used by other researchers (Flusser et al., 2009). Hu's seven moments are based on a continuous function, for which the moments are strictly invariant (Hu, 1962). The 2D moment is defined as:

$$m_{pq} = \iint x^p y^q f(x,y)dxdy \quad (5)$$

where the moment's order $m_{pq}$ is defined as (p + q); p and q are greater or equal to zero. The corresponding central moment's $\mu_{pq}$ is as follows:

$$\mu_{pq} = \iint (x - x_c)^p (y - y_c)^q f(x,y)dxdy \quad (6)$$

where $x_c = m_{10}/m_{00}$ and $y_c = m_{01}/m_{00}$ which are the centroid coordinates. For digital images, which are used here, the continuous function can be rewritten as a discrete expression:

$$\mu_{pq} = \sum\sum (x - x_c)^p (y - y_c)^q f(x,y) \quad (7)$$

The central moments provide information about how the moments deviate from the mean of the standard deviation in the image's different directions. The central moments can be normalized as following:

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\frac{(p+q+1)}{2}}} \quad (8)$$

Hu's seven moment invariants can be written based on the normalized central moments as stated:

$$\phi_1 = \eta_{20} + \eta_{02} \quad (9)$$

$$\phi_2 = (\eta_{20} + \eta_{02})^2 + 4\eta_{11}^2 \quad (10)$$

$$\phi_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \mu_{03})^2 \quad (11)$$

$$\phi_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} - \mu_{03})^2 \quad (12)$$

$$\phi_5 = (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] \quad (13)$$

$$\phi_6 = (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03}) \quad (14)$$

$$\phi_7 = (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] - (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] \quad (15)$$

It has been proven that Hu's seven moments are invariant and generate proper results for continuous functions which are free from noise. However, for non-continuous functions the moment invariants might change during geometrical transformations (Huang and Leng, 2010). This project deals with digital images that are neither continuous nor free from noise and pixels are defined to have finite values. It has been shown that the capability to reduce noise is in conflict with how well the invariants reconstruct an image. Higher order moments are more sensitive to noise than lower order moments (Teh and Chin, 1988). But, only high moment orders are able to reconstruct fine details in an image. Reconstruction is required when scaling and rotating an image. In this project the moment invariants are applied to a binary image which means that no fine grayscale details will be present and noise has been removed in a previous step. This means that by using a binary image the conflict between fine detail reconstruction and noise can be omitted. Using Hu's moment invariants on a binary image also ease the basis for changing moment method in the future.

## 2.4 Spot recognition

There are several known methods which deal with spot recognition, also called gridding, and when dealing with microarray the gridding is often performed automatically (Guo-Chuan et al., 2011; Leung and Cavalieri, 2003). The automatic gridding method works best for images without noise, concerning both scanning/image capturing noise but also biological noise, which can alter the shape of the spots. But noise free images are not common and the resulting grid from an automatic gridding might need to be manually adjusted (Hess et al., 2001; Jain et al., 2002; Leung and Cavalieri, 2003). Gridding can be based on summing up all intensities at the x-axis of the image and the y-axis respectively across the whole image. The resulting values can then be plotted which will result in a pattern of peaks and valleys, see Figure 5, which gives an indication of how the grid pattern is located. Based on the resulted grid pattern it is possible to make an assumption of the position for each spot's centroid (Daskalakis et al., 2007; Jain et al., 2002).

It is known that the printing precision is not perfect and that the printing robot movement can result in a difference between how the coordinates were formulated and how they were printed (Hess et al., 2001). The printer is supposed to print the centroids in straight lines but it is shown in reality that the result is to some extent fluctuating, as for example in Figure 5. However, within this project it is possible to overcome this issue by using the exact positions of each spot. The information is provided by GenePix, a software which can analysis a scanned array and through a pipeline of image analysis, automatically find the spots positions in the array, i.e., the spot's centroids. Using the coordinates provided by GenePix for the centroid positions it is possible to generate more accurate results, which can influence the segmentation of spots in later implementation, i.e., more accurate distinguish between spots

and background.

One way to use the known true spot centroid's positions is to map them to the microarray image. In order to receive a proper mapping it is desired to normalize the positions with the microarray image, i.e., giving them the same scale, centre of mass and the same rotation. As described in the image rotation section Hu derived seven invariant moments under translation, rotation and scaling. The central moments (6) have the property of translation invariance. Invariance for scaling can be achieved by applying the defined normalization (equation 8) to the seven moments (equation 9-15). Another factor first introduced to the image normalization is contrast, i.e., how the contrast between two (non-binary) images is different (Maitra, 1979). Comparing rotation, scaling, translation and contrast between two images $f_1(x,y)$ and $f_2(x,y)$ one expect the images to be unchanged during these two conditions:

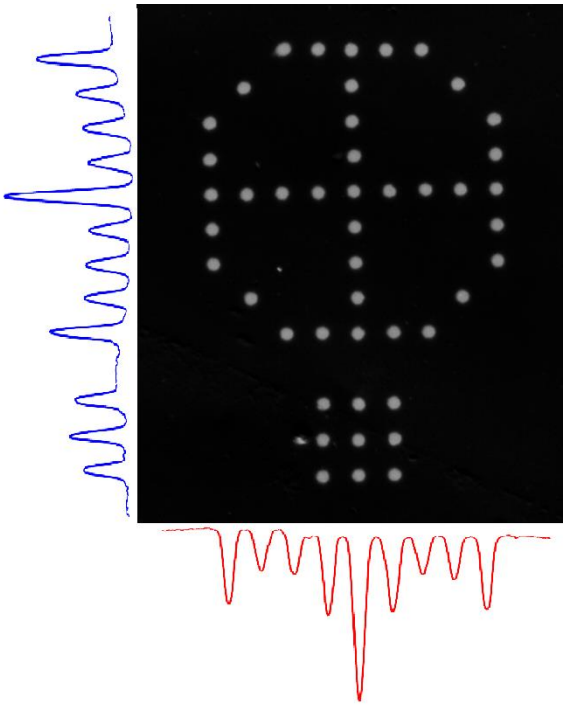

*Figure 5: Gridding. Determination of spot centroid positions by intensity summation.*

$$f_1(x,y) = k f_2(x,y) \tag{16}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = c \begin{bmatrix} cos\theta & sin\theta \\ -sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} a \\ b \end{bmatrix} \tag{17}$$

where k is the change in contrast, θ the difference in rotation, (a, b) is the translation distinctness between the images and c the difference in scale. For binary images, which has been used here, equation 16 is omitted since binary images do not have contrast. For the purpose in this project image normalization and coordinate matching should be robust enough. The expected robustness increase is due to the fact that the true coordinates should map to the image centroids and overlap spots with no or little expression. Using a gridding technique you will end up with coordinate positions for all spots, but the resulting positions of them will be more general and unsure.

## 2.5 Spot segmentation

Spot segmentation is meant to classify the spot as foreground and the backrest as background, a crucial step needed for later intensity calculation. The segmentation of spots can be classified into four major groups namely: Fixed circle segmentation, Adaptive circle segmentation, Adaptive shape segmentation and Histogram segmentation (Yang et al., 2001). Adaptive shape segmentation neglects the shape of a spot, a feature which is beneficial in a sample where noise is present, i.e., biological noise in the sample can cause irregular shapes and sizes of the spots (Daskalakis et al., 2007).

One well-studied adaptive shape segmentation method is the Seeded Region Growing (SRG) method. SRG only requires seeding points, i.e., starting points, and will then expand the foreground area and the backrest area until all pixels are assigned to be a part of a spot or the

backrest (Yang et al., 2001). The allocation of pixels are performed in a nearest-neighbour sense where all pixels close to an already assigned pixel are considered to be the next to be grouped. The selection of the pixel to group is based on the pixel's values, where the pixel with the closest pixel value is grouped. However, two major drawbacks with SRG for this project are: (1) some spots will not be expressed or expressed at such a low level that they will not be detected, which will cause SRG to classify these spots as the background. (2) The whole segmentation will be dependent on which pixels are chosen as seeds, it is important that a representative seed is selected for each spot and most importantly that the seed is a part of the spot if it is considered to be a spot-seed.

Another promising method requiring less computational power is the Histogram segmentation. It determines the classification based on a local histogram of the pixel values within an area containing both the spot and some background. Based on the histogram, using low value percentiles for background and high value percentiles for foreground, the classification is performed resulting in representative pixel values for the foreground and background respectively. A limitation for the histogram segmentation is the unstable result when the array contains spots of various sizes (Yang et al., 2001).

The segmentation method chosen has to be able to identify spots with high expression levels as well as those spots with low- or no expressions level, i.e., no- or small intensity difference from the background. Fixed circle segmentation is a method that may be easily implemented. It's only required parameter is the radius, within which pixels are classified as foreground. The major limitation of the fixed circular segmentation is the assumption of circularity as well as a uniform spot size. To overcome that problem it is possible to choose a smaller diameter which works for all the array spots. However, if one choses to use a local background for normalization in a later step, it is important to not include the area closest outside the defined spot area. The reason for this is because the spots might have an irregular shape to some extent, i.e., some spots are larger and will therefor include some of their spot pixels to the background. The problem can be solved by defining a grey zone area corresponding to the area with a diameter greater than the smallest spot and equal or greater than the greatest spot diameter, see Figure 6. The area outside of the grey zone can then be used to determine the local background, as described in chapter 2.6.

## 2.6 Intensity measure and data normalization

*Figure 6: Fixed spot segmentation. The red circle corresponds to the smallest spot size, which will be classified as foreground for all the array spots. The white area can be used to determine the local background and the light grey is a grey zone.*

In order to generate a reliable test answer some control spots will be printed in the array. These control spots will be generated from a dilution series, the concentration in these spots will therefore be known and can serve as a concentration measure for the allergen spots, i.e., the test spots with unknown concentration. Images may contain contamination, for example serum that does not flow through properly in non-spot areas or light gradients from the scene. One way to solve these problems is to print each control spot in at least two copies and spread them all over the microarray. This will enable controls for even flow through over the whole array as well as control for potential differences in the background.

Not only is the spot's intensity required in order to extract

information about the concentration in a spot. To generate a value which can be compared with the control spots, all spots need to be normalised but also the background needs to be considered as it might vary over the image. In this project both the foreground pixels and the backrest pixels have been considered when the data normalisation expression was selected.

For the foreground pixels, the red circle in Figure 6, the spot's mean pixel value should be used to represent the spot intensity. The expression level of each spot may be measured as the sum of pixel intensities within a spot (Yang et al., 2001). When presenting the data and in order to compare different results, one ought to use the spot's ratio of intensity. It is computed as the mean of the spot pixel intensities, which is the same result as the ratio of the sums of all the spot's pixel values. The result can therefore be interpreted as a ratio of expression and is given a biological meaning, the same is not true for a median pixel intensity.

One method to derive the impact of the background is to implement a morphological opening as Yang et al. (2001) describes. It should have a window size of at least twice the spot diameter, to reduce the impact of the spots and reduce variability. A morphological opening is a filtering method which consists of first an erosion, which erodes objects smaller than the window. It is followed by a dilation in order to dilate objects that have been shrunk to generate their original size. Here the SE was chosen to generate a result where noise and the spots had been removed, i.e., the resulting image contained estimated values of the background. To set background values for each individual spot one can take the mean value of the same area used for the foreground estimation. One way then to normalize and phrase the spot expression level is:

$$S_{norm} = \frac{1}{(mean(F) - mean(B))} \quad (18)$$

where $S_{norm}$ is a normalised spot expression value, mean(F) is the mean foreground intensity of the spot and mean(B) is the corresponding median background intensity of the same spot, after applying a morphological opening. Negative values should be omitted or interpreted as having no expression level.

To derive a background intensity several methods can be used. A morphological opening has proven to give a more robust result than other more simple methods like local background estimation. Yang et al. (2001) argue that a morphological opening should be less variable than other investigated background methods. Yang et al. (2001) used calculations from a large window that operated over the image, which for them resulted in a less bias result for pixels with extreme values. However, how well morphological operations perform depend on the choice of SE, both considered the size and shape. Some other, simpler techniques, only use a small area outside the foreground area to calculate the background intensity, and using these methods each individual pixel has a larger impact on the calculation of the background (Yang et al., 2001). A reason to use morphological opening for background estimation is because it results in fewer negative values, i.e., fewer values that cannot generate a proper result. The software SPOT, which uses morphological opening, has shown to generate much few negative values and can be compared with GenePix, which uses a median background estimation, which often generates up to 30 % negative values (Smyth and Speed, 2003).

# 3. Implementation

The image analysis was performed on images captured by Motorola's Nexus 6 with an additional macro lens from Photojojo. During the project several images of different paper arrays were analysed. The final result which included fine tuning of parameters and the overall performance evaluation was based on four images taken with a fixed distance from the camera, these images can be seen in Figure 11.

## 3.1 Image processing in Matlab

Matlab R2015a was used along with its Image Processing toolbox in order to determine what methods to use in everything from noise reduction to image rotation and cropping. The reason for using Matlab was its scripting language that does not require the programmer to set the data structures and it provides an easy and flexible program implementation in my opinion. Other benefits were the available documentation and that I as developer already was familiar with the Image Processing toolbox when the project was initiated.

### 3.1.1 Pre-processing

This step aimed for reduction of noise, enhancement in contrast between the spots and the background which would result in a better segmentation of the spot. The images had a gradient of light which needed to be corrected, having a more even spot intensity over the whole image would reduce the risk of classifying foreground spots as background and vice versa during the segmentation. The illumination correction was performed by a small average filter, only corresponding to about 4 % of the spot size, and a Bot-Hat operation. The Bot-Hat SE had the shape of a square and the size was determined based on the image size, which for the four tested images corresponded to a size of approximately 40-45 pixels, details can be found in Appendix A.

Threshold segmentation was implemented to enable calculations of the array's orientation and to crop the image safely around the array. Matlab's *graythresh* method were chosen which implements a global search for an optimal threshold value were the intraclass variance between black and white pixels are minimised. When converting the image from a grayscale image to a binary image the threshold value determined the pixel's new value, i.e., pixels with an intensity value above the threshold, holds the new binary value 1 (object) and pixels with a value below are allocated the value 0 (background). The segmentation method was chosen with respect to the low computational power required; which is necessary in order to have the same method run in a smartphone with even less power. A morphological erosion was performed on the resulting binary image to reduce very small objects. It was followed by labelling the objects and extracting the area's features. Objects that were considered to be too small to be an array spot were removed with Matlab's function *bwareaopen*, the determination of sizes considered to be too small was based on an area feature histogram. A morphological dilation was applied to merge the spots into becoming one big object which then could be recognized as the largest object in the image, which was extracted and used in further analysing steps.

### 3.1.2 Image rotation and cropping

To find the image array's orientation and the angle φ it needs to be rotated, a function based on Hu's moment invariants which are described in Gonzalez and Woods's (2009) was implemented as seen in Algorithm 1.

Algorithm 1 uses the central moments to determine the angle φ, the image is then rotated by Matlab's function *imrotate*, details is found in the supplementary. After the rotation an image cropping were performed on the binary image with dilated objects. The cropping used a bounding-box which was set to have a minimal size but still contain all pixels belonging to the segmented objects. The angle φ and the bounding-box's coordinates were saved and later used on the original grayscale image in order to remove noise and object not needed in the analysis, i.e., it would generate a new image in the right orientation and only containing the desired array pattern. The image can later be used for the spot intensity measure.

---

***Algorithm 1:*** *Image rotation*

---

1. Let x and y be two vectors with the segmented array object's coordinates.
2. x_bar = mean(x)
3. y_bar = mean(y)
4. $my_{11}$ = SUM((x - x_bar)*(y - y_bar))
5. $my_{02}$ = SUM((y - y_bar)$^2$)
6. $my_{20}$ = SUM((x - x_bar)$^2$)
7. phi = 0.5*$\tan^{-1}$((2$my_{11}$)/($my_{20}$-$my_{02}$))

---

## 3.2 Android

The Matlab code was translated into Java, which is the language used for Android application developing, and the image analysis methods were implemented through the open source library OpenCV. Android was chosen because there are a greater amount of different smartphones using Android's operational system and Android smartphones has a greater cost range than iOS smartphones. Another benefit was that the developing in Android is based on Java which is known to be a good choice when working with visualization and Java was a familiar coding language.

### 3.2.1 Implementation concept

In this study the aim has been to develop a smartphone application containing all parts from Figure 7. Only the image processing has been implemented and the other parts are described below as well as in chapter 5.1, future work. The image analysis carried out in this smartphone application will be dependent on external information, for example a coordinate file generated from scanning the microarray in order to get the array spot's exact position and also information that tells the application what is expressed in each spot. That information is thought to be imported by a QR-scan. The idea is that each paper microarray will have its own associated QR-code. The code will download a file when it's scanned and the file should contain the required information stated above. Alternatively, the QR-scan will point to a storing position within the smartphone containing the file needed, see step 1 in Figure 7. The next step is to capture an image of the array using the smartphone's camera. There are a lot of

camera applications available to download for free and the image capturing part will therefore call an external camera application to perform this part. The image processing, step 3 in Figure 7, will be the heavy computational part of the application and will therefore be run at multi threads to keep the user interface (UI) responsive. All the image processing is carried out using the open source library, OpenCV.

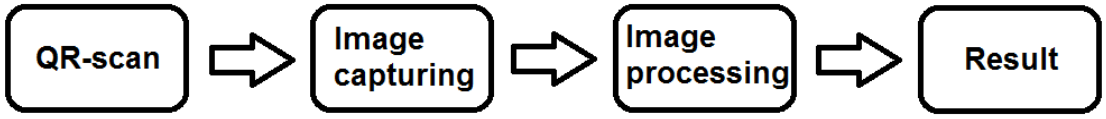


*Figure 7: Smartphone application workflow overview. The first step is a QR-scan to import important information. The second step is image capturing of the array followed by the image analysis of the image. The result is then graphically presented in the last step of the application.*

In order to make the smartphone application project easy to understand, some considerations concerning how to ease the interpretation of the code and the way the program communicate with the methods, were taken. Android uses something called *Activities*, which are linked together as a strand, see Figure 8b. How the linking between activities works is, in my opinion, not always obvious and the troubleshooting can be harder compared to Figure 8a. However, building an Android application requires the use of activities. But whenever it is possible the implementation idea is to use a more independent way of handling methods, see Figure 8a. For example, all image analysis is run in a Java class implementing the independent method structure.

The application is implemented to run on two threads, an UI thread and a background thread. It is essential to divide the application into two threads because of the heavy image processing calculation which will otherwise block the UI. According to Android's developer the UI will stop and tell the user it is not responding if the UI is blocked for more than five seconds.

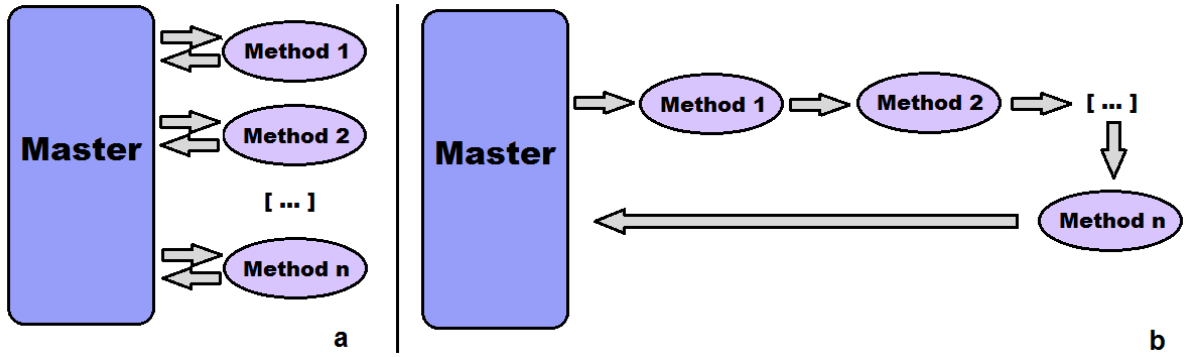


*Figure 8: Different way of setting up methods. (Left) Methods can be ran from the master and required to deliver their result independently. (Right) A method is initialized by the master and then the next method is called from within the previous.*

The work in Android was performed in a most-important-first sense, which meant that parts which were crucial, like the image analysis, were performed first. Other parts like image capturing and QR-scan were ranked lower since there are ready code packages available on the web to use and these parts were not required in order to obtain a result.

### 3.2.2 Development environment

For development environment, two alternatives were considered: Android Studio and Eclipse with Android developer tool (ADT). Android Studio was, when this project was carried out, new on the market and the initial release was in December, 2014 (only one month before the project started). In my opinion, Android Studio's emulator suffered from being considerably slower than Eclipse's emulator and because of its recent entrance on the market, little documentation was available other than Android's own documentation. Therefore, Eclipse and the ADT were chosen as the development environment.

There were some problems with setting up the developing environment which needed to compile both Java and C++ code in the same program. Downloading all the needed packages and installing them turned out to be difficult, most because of the large amount of packages. Therefor the free package: Tegra Android Developer Pack (TADP) was used, it is a package provided by NVIDIA which automatically acquires all the necessary packages required to develop OpenCV-applications in Android.

### 3.2.3 Open source

The original idea of Open Source was to provide unlimited access to program source code. Anyone would be able to use, modify and share the code without having to pay anything for it. Within this project all programs and libraries, with the exception of Matlab, is open source. Both Android and Eclipse are open source to some extend and also the libraries: OpenCV and ZXing.

It might be tempting to use a lot of different libraries, each specified for the particular task it is considered to carry out. However, if one wishes to build a project which can easily be handed over to someone else to continue the development, it is necessary to keep down the amount of libraries. Having a small amount of libraries will reduce the time needed to get acquainted with the project, which includes learning all the libraries used in the project and how they are structured. In the beginning of this project, the open source library D3 was a candidate for data visualization of the results, presented in the final step of the Android application. However, if OpenCV or Android has tools for visualization of data I recommend to use them instead of introducing another library to the Android project.

OpenCV, *open source computer vision library*, is an open source library with over 2500 algorithms for computer vision and machine learning. Here, OpenCV is used to implement the image processing to the Android environment. Because of its great amount of algorithms and wide documentation it was to me an obvious decision of library to work with.

ZXing is an open source library for decoding and encoding 1D- and 2D-barcodes. It supports a variety of format and among them QR-scan which is the barcode type thought to be implemented in this project. Documentation, available tutorials and whole code packages for creating a barcode scanner is reason to consider ZXing as a library to use for the implementation.

### 3.2.4 Translation from Matlab to Java and OpenCV

Translating the Matlab code into Java does not only means translating the Matlab methods to the corresponding OpenCV methods, it also requires to assure that the parameters such as filter sizes, SE shape etc. will generate a result corresponding to the Matlab result. Since the image analysis will be run on a smartphone this part also involve setting up an Android

activity for testing and presenting the result as well as setting up multiple threads and coordinate the computations on each individual thread.

The application is built within an Android project containing a main activity, which hold all the Android classes and functionality, and a Java class, implementing all the image analysis through the open source library OpenCV. The Android part, seen in green in Figure 9, can be seen as a shell only containing the visual parts of the application such as the graphical user interface (GUI). It is the Android activity that initialises the image processing in the Java class. The Java class function that is called starts up by initializing a new thread, called a background thread, in order to not block the UI thread and more importantly not cause the application to crash. When the background thread is setup all image analysis methods are run on the thread and all the partial results are reported back to the UI thread (indicated with purple arrows in Figure 9) in order to indicate the progress to the user, the image view of the activity is updated, see Figure 10.

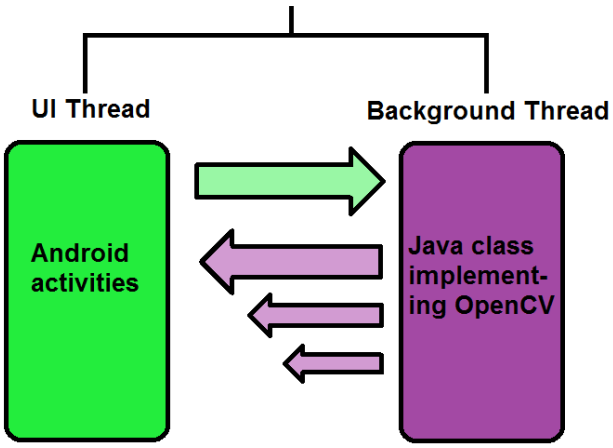


*Figure 9: Representation of the application setup. The application makes up of an Android part (green) and a Java class (purple). These two parts are running on two different threads a UI thread and a background thread. The Android part call the Java class when the image analysis is to be performed and the Java class report its progress by sending the partial results back to the Android activity.*

The initial plan was to translate the Matlab code into Java with the aid of OpenCV, without concerning any Android programming. When that had been done it was supposed to be imported to an Android project and the framework of the smartphone application appearance could take form. The idea did not succeed, even though OpenCV supports Java and Window's operation system it was not manageable to set up the development environment required for OpenCV. Putting together different libraries and programs can sometimes be a hard job and as it turned out, so was the case here. The final solution was to download everything needed for a totally clean environment install from NVIDIA, i.e., the package TADP. The installation was performed by carefully following the step-by-step guide at OpenCV's website for android developing.

Since both Matlab and OpenCV consists of a great variety of image processing methods and tools the translation went easy, all the methods in Matlab had some corresponding method in OpenCV. But as in the Matlab implementation a lot of time was needed for setting the parameters once again, using the values from Matlab did not work which was expected since it is different implementations. This resulted in one more exhaustive search for the optimal parameter values a time consuming task.

### 3.2.5 Graphical user interface

In a project dealing with image analysis the aim is to keep the result objective. By implementing a variety of tuneable parameters by for example allowing the user to select filter shapes and sizes, will generate a more subjective result where the user can rerun the application and receive different results every time. Therefore, the aim has been to provide the user with information by continuously update in the progress and erasing as much selection options as possible in order to keep the result objective, i.e., the interaction should be kept low in future development. Due to the lack of time only one activity was implemented, that activity however did not contain any buttons or other interactions with the user, see Figure 10.

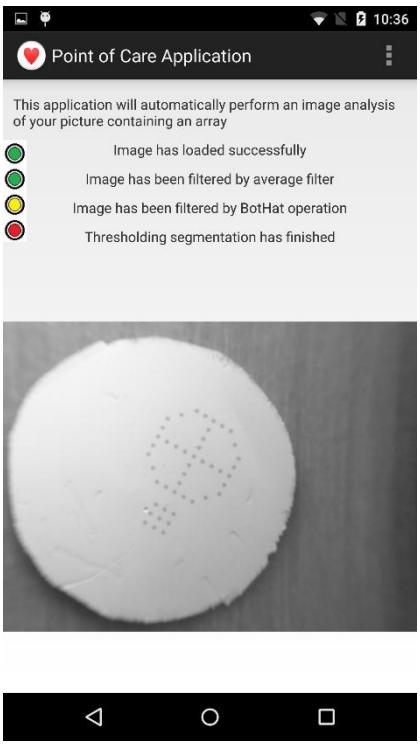


*Figure 10: GUI of the smartphone application.*

# 4. Result

The purpose was to investigate the possibility to carry out a diagnostic image analysis through a smartphone application. The project has examined image processing tools, both in Matlab and the open source library OpenCV, as well as the computational requirements of the resulting program with respect to the lower processing power in smartphones.

During the development phase when selecting the image analysis methods fifteen different images capturing four different paper-arrays were used to test the methods on. When setting the final parameters only four images of the same paper-array captured with the same distance from the camera lens were used. The reason was that only one paper-array with the final array structure were available and there was a shortness of time. The four images were captured in different angles and was slightly different positioned in the images, see Figure 11.

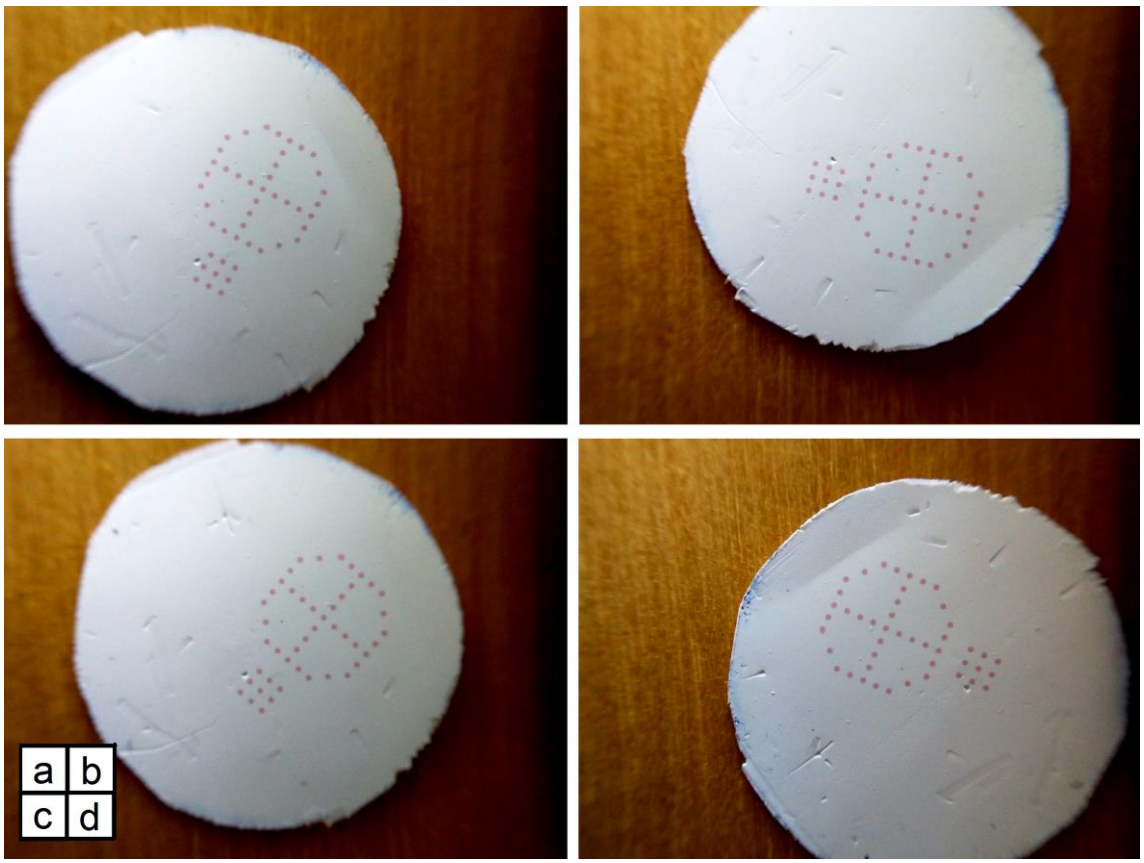


*Figure 11: The paper array used for the final image analysis testing, it has been captured in different orientations. The images are referred to as: DSC_0034 (a), DSC_0035 (b), DSC_0036 (c), and DSC_0037 (d).*

## 4.1 Matlab - noise reduction

Determining how to convert the captured colour image to a greyscale image both Matlab's built in function *rgb2gray* and the three different colour channels (red, green and blue) were considered. The result from each of them is presented in Figure 12. The red colour channel (Figure 12b) shows the lowest contrast between the array spots and the paper array itself. The three other methods are more similar in contrast even though the blue colour channel (Figure 12d) has an overall darker image. The choice of grey scale conversion method was based on

the contrast between the spots and their closest environment, the overall background, i.e., the area outside the paper array was not concerned. The green colour channel was selected based on a little higher spot contrast compared to the blue channel and Matlab's resulting image from *rgb2gray* method, this is also supported by the histogram analysis in Figure 13.

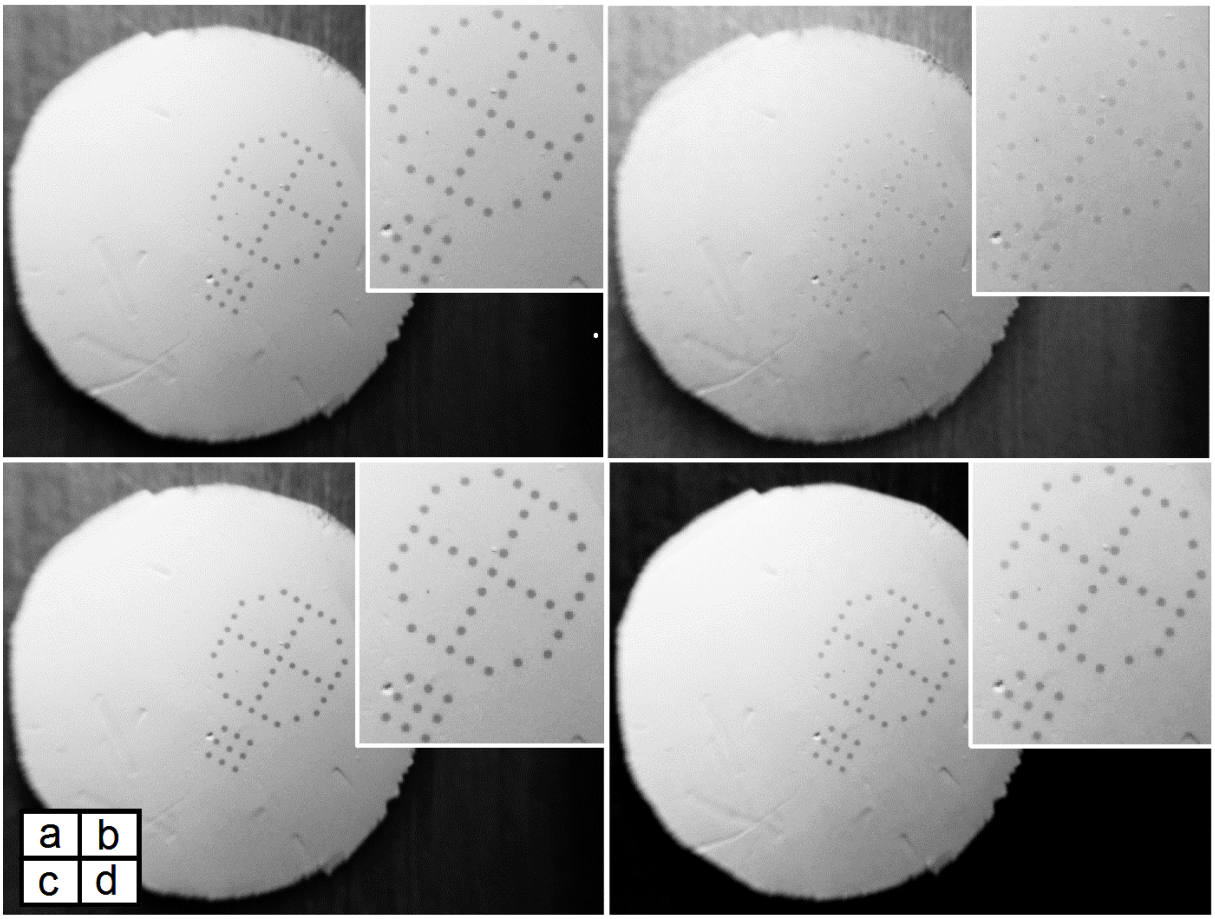


*Figure 12: Graphical presentation different colour conversion methods. Matlab's function rgb2gray (a), the red colour channel (b), the green colour channel (c) and the blue colour channel (d).*

The motivation to perform a histogram analysis was to get an objective result rather than a subjective result based on one person's interpretation. The histogram is a representation that shows the number of pixels that has a particular pixel intensity value. In Figure 13 it is seen that Matlab's *rgb2gray* function has a histogram close to the blue colour channel. The green colour channel had a more dominant tilt at the red marker than the other two's histograms, which indicates the boundary between foreground pixels and background pixels. The more distinct change of foreground and background the greater the contrast between them is considered.

When the grey colour conversion method had been chosen, the work of noise reduction in the image was introduced. It involved a substantial amount of time spent on setting the parameters for the different methods. It was easy to choose the methods since there were a lot of demands to fulfil, but optimizing the filter sizes, thresholding values etc. was the time consuming part. Most of the parameters have been set due to a trial-and-error approach, which can be considered to some extent exhaustive.

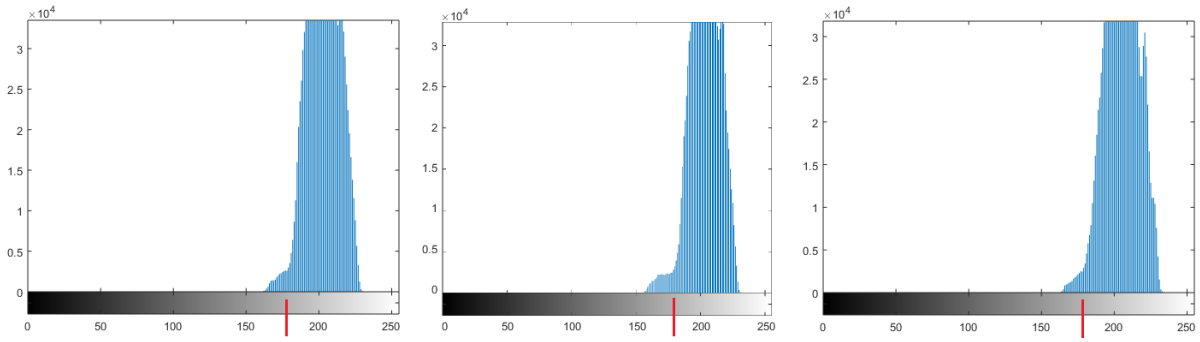

*Figure 13: Histogram analysis of image contrast. The leftmost histogram corresponds to grey image generated by Matlab's function rgb2gray, the centre histogram shows the green colour channel and the right histogram is the blue colour channel. The red lines indicate the shift between spot intensities and background.*

The pre-processing in Matlab (described in chapter 3) starts with a slight blurring of the image, the averaging filter size is approximately 4 % of the size of an array spot. In three out of four tested images the averaging filter does not affect the overall result of the rotation. But for some images in the pre-testing phase and one in the last testing phase showed a better segmentation result after applying an average filter, therefor it is suggested to keep the average filter. For example, on image DSC_0037 (the improved result is shown in Figure 14, left) has shown a decrease of noise and a slightly better result, without applying the average filter the resulting orientation was wrong with approximately 5-10°.

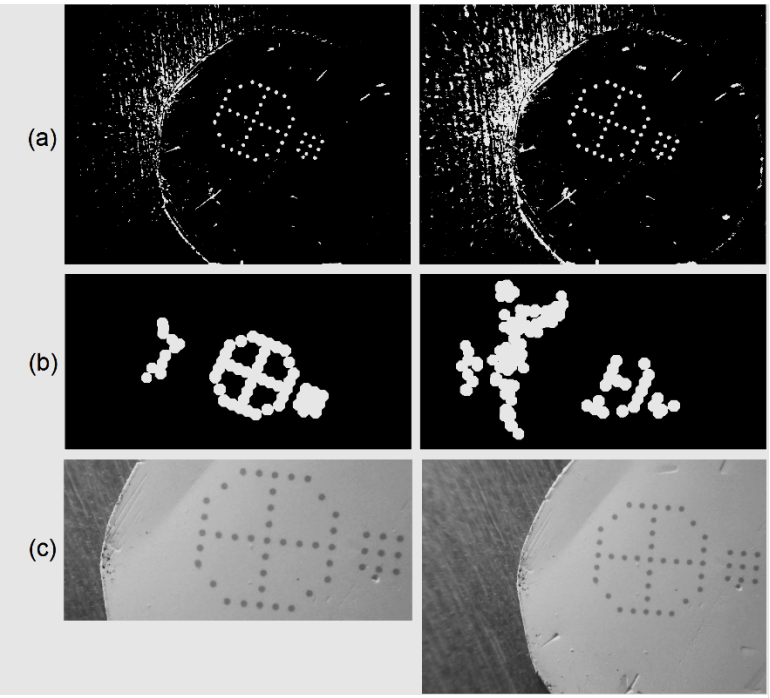


*Figure 14: Comparing a square shaped SE with a disk shaped SE. The left column shows results from using a square shaped SE and the right column show the results of a disk shaped. After applying a Bot-Hat filter and a thresholding segmentation (a). Result from removing objects considered to be too small to be a part of the array objects (b). The final rotated and cropped image (c).*

After the Bot-Hat has been applied a segmentation function is called which first performs a threshold segmentation. It is followed by erosion and dilation, removing small objects respectively merging the array spots to become one large object. The last step in the pre-processing is to remove everything except the two largest objects that should be the array objects, which has been shown for three out of four tested images, image DSC_0037 which failed is shown in Figure 14c right.

Due to the limited time in the project and the urgency to proceed and work in Android the development in Matlab was not finished. A result where each spot intensity is linked to an expression id and presented in some easy interpretable diagram is left for future development. Moving over to Android instead of finishing was important, this in order to investigate the hypothesis, whether or not it

is possible to develop a smartphone application to analyse diagnostic results with assistance of image analysis.

## 4.2 Matlab - array rotation

The rotation is based on Algorithm 1, chapter 3. The reason why PCA, described in chapter 2.3.1, was not chosen is shown in Table 1. It displays that the PCA does not provide a reliable answer. The reason could be that it has been incorrectly implemented, most likely then would be an error in the vector analysis for more information see Appendix B, or that PCA is not robust enough for this application. In Table 1 it is noticeable that the PCA sometimes generates two angles were one of these is an imaginary angle. When the angles were examined it appeared that the angles were often not even close to the real rotation angle, which was the reason moment invariant were implemented. Since Matlab did not had its own function for Hu's seven moments it was implemented from scratch and can be seen in Appendix A.

Table 1: *Two methods were analysed to calculate the angle of image rotation. The first method was Principal component analysis but it resulted in some imaginary angles as well as an overall bad approximation of the rotation. Hu's moment invariants on the other hand showed very promising results even though some images needed an additional check for whether the orientation was the expected portrait or the wrong landscape orientation, i.e., if they needed an additional rotation of 90 or 270 degrees.*

| **Image [.jpg]** | **Principal component analysis [°]** | **Moment invariants [°]** | **Correction for landscape orientation [°]** |
|---|---|---|---|
| DSC_0034 | 90.0000 – 50.4990i | 28.5262 | + 0 |
| DSC_0035 | 38.7834 | 14.1174 | + 90 |
| DSC_0036 | 42.7980 | 40.8497 | + 0 |
| DSC_0037 | 90.0 – 50.4990i | 22.9038 | + 270 |

The implemented rotation method worked correctly for two of the images but for DSC_0035 and DSC_0037 the final orientation was wrong with ±90°. These two images should have been rotated more than 90° but the method only rotated them as stated in Table 1 which was 14.1258° and 17.4610° respectively. From that result the following assumption could be made, that the moment invariant method only rotates an image to be parallel to the nearest vertical/horizontal line. It was not the desired outcome of the method but could be adjusted by adding a check to see if the image orientation after the rotation was landscape or portrait, if the orientation was landscape the image was rotated again and this time 90°. Only image DSC_0037, which earlier had shown the worst result in every partial result, had a final orientation which was 180° inaccurate. Time was not further spent on searching for a better rotation method. Instead time could be spent on the smartphone application, introducing a view showing the user how to place the paper array in order to receive a result. It is also possible to implement a warning if the angle is found to be close to 90° or small, such as for example below 10°.

## 4.3 Result from the smartphone application

Much time in the project was spent on finding proper methods to use for the image analysis and trying the candidate methods in Matlab. An unexpected time consuming part was the environmental setup for the smartphone programming, it was planned to take one week and

within that week also include programming repetition. In the reality it took two weeks due to the complexity of assemble the IDE Eclipse with the OpenCV library and Android's developer tool kit, which also required the native developer kit (NDK). The OpenCV library is originally coded in C++ and making it possible to run on a Java machine the NDK was needed because it enables the code to compile in a C++ compiler and run on the java IDE. Another week was spent on implementing the threading, described in chapter 3, which is a requirement to be able to run any image processing without causing the application to crash. Despite these shortcomings and the lack of time a smartphone application was developed, the code is found in Appendix D. It does not have all the implementation as the developed Matlab function, see section 4.1 and 4.2. The parts that were implemented, are filtering and morphological methods, which are considered to be some of the most computationally intensive methods the application will run. With the implementation in this project it has an analysing time of 13 seconds. In earlier implementations using a circular Bot-Hat filter it had an analysing time of 151 seconds, which is more than 10 times slower, see Table 2.

Table 2: *The different running time depending on implemented filter shape.*

| **Morphological filter shape:** | **Total application run time:** |
|---|---|
| Elliptical shape<br>(i.e., implemented as circular) | 151 seconds |
| Square shape | 13 seconds |

Interesting results were obtained comparing the major implemented parts: the morphological Bot-Hat operation and the thresholding segmentation. The resulting images both have noise of such low contrast against the background, that the images needed a contrast enhancement to show any differences. In Figure 15 both the contrast and light in the image has been increased to better show the differences in the two images. The image resulted from Matlab's Bot-Hat operation has an evenly spread powder-like noise, Figure 15 (left), while the OpenCV has most of its noise outside of the array and the noise is of an abundant pattern, Figure 15 (right).

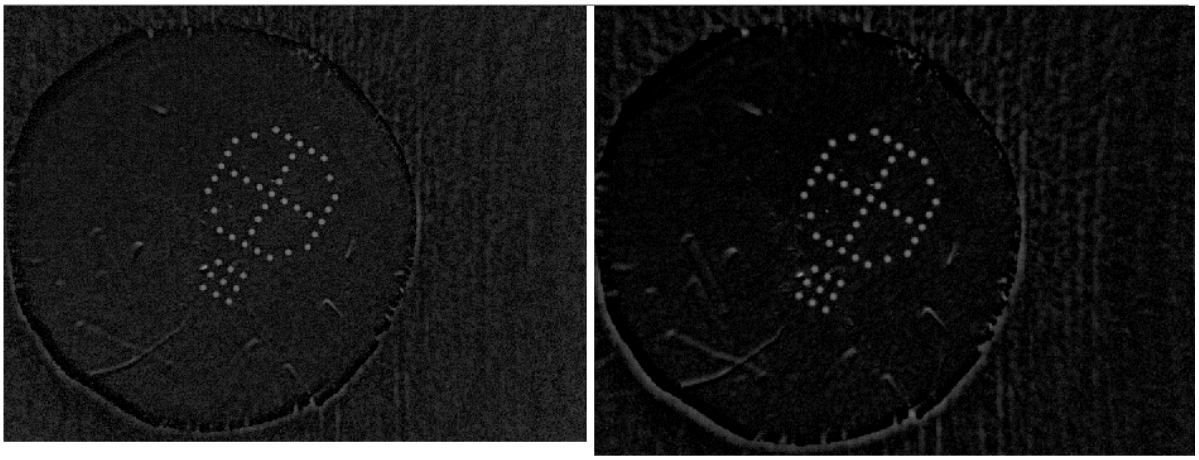

*Figure 15: Difference of the Bot-Hat operation results. The contrast has been enhanced for both images by multiplying the images with a factor three. (Left) Matlab and (right) OpenCV.*

The thresholding results has a major difference in the outcome of the segmented array spots. In Matlab they are segmented as solid circles (Figure 16, top) while OpenCV only have segmented the outer edge of the spots (Figure 16, bottom). The total amount of pixels segmented as object pixels, i.e., appear white in Figure 16, are larger in the Matlab method, both considering true-positive segmented object pixels and false-positive pixels.

The differences found in Figure 15 and 16 can be explained from the fact that the methods originate from the same method, but has been implemented independently in different software. One should expect having some differences when comparing methods from different software and therefor these results are not a surprise. Investigating of which methods to use for the image processing in the smartphone application has provided a pipeline of methods. But additionally methods can be needed as for example in Figure 16, a morphological closing might be a solution for the OpenCV to generate solid spots.

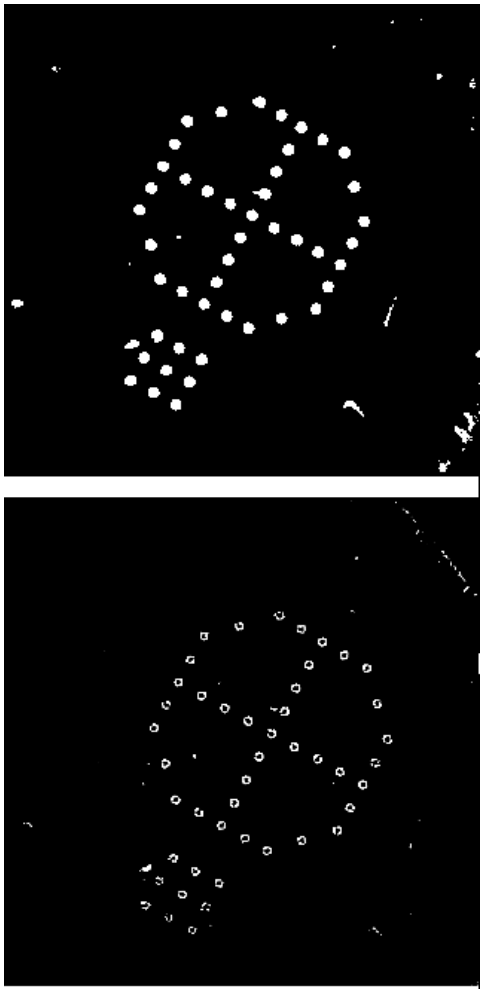

*Figure 16: Analysing the distinctness between Matlab's and OpenCV's thresholding result. Matlab (upper), OpenCV (lower).*

# 5. Discussion

Images were captured by Nexus 6 and an additional macro lens objective which together showed great quality and resolution, Figure 11. Since the images showed great sharpness and contrast no further inspection has been made and it was assumed that even a cheap macro lens could be enough for such an analysis as performed here.

The results presented from the pre-processing have a considerable amount of noise for some images, see Figure 14a-b. Evaluating the images it should be taken into consideration that they are a part of a bigger whole and the images should be viewed as a series in which every image should provide progress in the overall analysis. The goal for the pre-processing was to generate a binary image only containing the array as an object. It was not possible for all four images but the major shortage was probably not the image processing but the damaged paper array. Holes and scratches on the paper array have proven to appear as noise in the image, as in Figure 3, were the array has a big hole beside the spots which has impacted both segmentation and rotation.

The morphological Bot-Hat filter mentioned in chapter 2, has a filter size which depends on the image size. It is defined to be the image maximum length divided by 95, giving a filter size of about 40 pixels, which has shown to correspond well to the size of the spots. The meaning of not setting a fixed filter size value is the fact that different cameras generate different image sizes with different pixel density, i.e., a measure of the image resolution. However, it has not been tested if the filter size expression is valid for other cameras, something that ought to be considered for future work. The Bot-Hat filter has the shape of a square since it had a more positive impact on the followed segmentation compared to the disk shaped SE, see Figure 14. Meaning, the choice of SE for the Bot-Hat operation have a great impact on both the runtime and the amount of noise in the resulting threshold segmentation. When a squared SE is used (Figure 14a, left) the resulting image have considerably less noise compared to a disk shaped SE (Figure 14a, right). Following the thresholding, objects considered to be too small to be a part of the array are removed (Figure 14b) and here as well as the final rotated and cropped image (Figure 14c), it is possible to visually see the different result originate from different SEs.

Rotating the image into an optimal position for mapping them with the true coordinates, which contains information about the substance in respectively spot, might not result in an optimal orientation at all. As mentioned above, noise due to a rough treatment of the paper array can impact the segmentation and therefor also the resulting rotation. The desired array orientation is as in Figure 5, where the array has the smaller rectangle bellow the greater octagon. However, saying if the orientation is optimal is not possible, partly because of the hole so close to the spots (see Figure 3) and partly because of the printing pattern, which is fluctuating to some extent. The fluctuation in the array spots appearance makes it harder to determine the optimal orientation with respect to the true coordinates compared to a case were all spots are structured in straight lines. Efforts have been taken to rotate those images which requires more than 90° rotation and another additional solution could be to require the user to position the array in the range: $-89° \leq \theta \leq 89°$.

During the project there has been suggestions on building a housing for the image capturing. If it becomes reality, the image capturing will have a fixed distance between the camera and the paper array and most likely the illumination of all images would be similar. It could come to ease an optimisation of the image processing and especially the image rotation.

For future improvements, this study has chosen methods with consideration to be both easy to implement as well as computational light. This means that the result presented here does not have to be the optimal solution. For instance, there exists methods which for example actively search for circles, none of them were considered here because of their need in more processing power. Another reason was that the thresholding segmentation, which fulfils both demands, performed a sufficient segmentation. The result could be improved by some post-processing, as described in chapter 3.1.1, which also was the case here.

## 5.1 Future development

This project has shown that it is possible to develop a smartphone application which can perform image analysis in a sufficient amount of time. The aim was to investigate the possibility to develop a smartphone application that would provide the user with reliable test answers, it has not been shown in this study and further developing is required.

This project has come half way with the image analysis, see Figure 1, all parts involving the spots are left to implement as well as the result calculation. Looking at the whole application work still remains on the image capturing and a QR-scan functionality. The image capturing is implemented in a separate project but not merged to the POC application Android project, code can be found in Appendix C. Both the image capturing and the QR-scan was ranked low in the implementation list, mostly because code for these kinds of implementation is available on the internet together with a lot of documentation.

A suggestion is to examine how the already implemented image processing would perform with compressed images. The application analysis 16-bit grayscale images which are captured with a Nexus 6, which are considered to be high-resolution. By compressing the image, it should run faster and if the performance is equivalent, information about the spot's centroid position could still be extracted and used later on the high-resolution image. For the result calculation, i.e., the spot intensity measuring, the original high-resolution image must be used for achieving the most accurate result.

# 6. Conclusion

This thesis has investigated the possibility of using a smartphone application to carry out diagnostic computationally intensive image processing. The purpose has been to examine if it is feasible to incorporate the diagnostic tools from the hospital's large analysing machines into a smartphone, which would result in a mobility where the tests could be analysed on site. The project has examined a POC allergy test where the sample is flown through a paper array which is analysed by image analysis. To study the practicable in this matter, intensive image processing methods was implemented in an Android application to examine both the computational time and the image processing quality.

The results from this project has shown proof of the possibility to develop a smartphone application that can carry out the desired diagnostic analysis. The heavy computations already implemented in the application indicates that even low power machines, such as a smartphone, are able to perform heavy computations. As seen in table 2 the choice of method, or in this case the shape of the filter, can have a great impact on the run time.

The image analysis in this project has not been finished but; the first partial result indicates that it is operationally possible to carry out the image analysis on images captured by a smartphone camera. It is worth noticing that no conclusion considering quality of the final spot intensity values and the interpretation of these can be made in this thesis. Which means that in the worst case it might not be feasible to perform the POC tests in a smartphone even though all evidence in this thesis indicates that it is. Therefore, to receive a final result it is necessary to complete all the image processing and data extraction. It should be followed by an exhaustive comparison of results derived from the smartphone application and from laboratory machines, which are used today.

# 7. Acknowledgment

I would like to thank my supervisor Jesper Gantelius and the research group at Science for Life Laboratory for the opportunity to do this Master thesis. It has been very instructive to work and structure such a big project and it has been interesting to learn about your method and the work done in Uganda. Two very important persons in this project has been my scientific reviewer Ida-Maria Sintorn and my programming supervisor Johan Öfverstedt, thank you for all your time and effort. I would not have come as far as I did in the project if it was not for you two, your ideas and suggestions as well as your small lectures has been invaluable! Another group of important people I like to give gratitude to, who has supported me throughout not only this project but the five years of study, is my family and friends. Finally, the one person who literally has carried me through this: Jonas Fors – Thank you for always being there for me.

# 8. Reference list


Abdi, H., Williams, L.J., 2010. Principal component analysis. Wiley Interdiscip. Rev. Comput. Stat. 2, 433–459. doi:10.1002/wics.101

Alvarez, L., Gómez, L., Sendra, J.R., 2009. An Algebraic Approach to Lens Distortion by Line Rectification. J. Math. Imaging Vis. 35, 36–50. doi:10.1007/s10851-009-0153-2

Chinnasamy, T., Segerink, L.I., Nystrand, M., Gantelius, J., Svahn, H.A., 2014. Point-of-Care Vertical Flow Allergen Microarray Assay: Proof of Concept. Clin. Chem. 60, 1209–1216. doi:10.1373/clinchem.2014.223230

Daskalakis, A., Cavouras, D., Bougioukos, P., Kostopoulos, S., Glotsos, D., Kalatzis, I., Kagadis, G.C., Argyropoulos, C., Nikiforidis, G., 2007. Improving gene quantification by adjustable spot-image restoration. Bioinformatics 23, 2265–2272. doi:10.1093/bioinformatics/btm337

Flusser, J., Zitova, B., Suk, T., 2009. Moments and Moment Invariants in Pattern Recognition. John Wiley & Sons, Hoboken, NJ, USA.

Gonzalez, R.C., Woods, R.E., 2007. Digital Image Processing, Third Edition, 3 ed. ed. Pearson education (US).

Guo-Chuan, L., Lin, L., Fujun, S., 2011. The preliminary study on microarray automatic spot identification method, in: 2011 International Symposium on Bioelectronics and Bioinformatics (ISBB). Presented at the 2011 International Symposium on Bioelectronics and Bioinformatics (ISBB), pp. 186–190. doi:10.1109/ISBB.2011.6107677

Hess, K.R., Zhang, W., Baggerly, K.A., Stivers, D.N., Coombes, K.R., Zhang, W., 2001. Microarrays: handling the deluge of data and extracting reliable information. Trends Biotechnol. 19, 463–468. doi:10.1016/S0167-7799(01)01792-9

Huang, Z., Leng, J., 2010. Analysis of Hu's moment invariants on image scaling and rotation, in: 2010 2nd International Conference on Computer Engineering and Technology (ICCET). Presented at the 2010 2nd International Conference on Computer Engineering and Technology (ICCET), pp. V7–476–V7–480. doi:10.1109/ICCET.2010.5485542

Hu, M.-K., 1962. Visual pattern recognition by moment invariants. IRE Trans. Inf. Theory 8, 179–187. doi:10.1109/TIT.1962.1057692

Jain, A.N., Tokuyasu, T.A., Snijders, A.M., Segraves, R., Albertson, D.G., Pinkel, D., 2002. Fully Automatic Quantification of Microarray Image Data. Genome Res. 12, 325–332. doi:10.1101/gr.210902

Leung, Y.F., Cavalieri, D., 2003. Fundamentals of cDNA microarray data analysis. Trends Genet. 19, 649–659. doi:10.1016/j.tig.2003.09.015

Maitra, S., 1979. Moment invariants. Proc. IEEE 67, 697–699. doi:10.1109/PROC.1979.11309

Park, J., Byun, S.-C., Lee, B.-U., 2009. Lens distortion correction using ideal image coordinates. IEEE Trans. Consum. Electron. 55, 987–991. doi:10.1109/TCE.2009.5278053

Smyth, G.K., Speed, T., 2003. Normalization of cDNA microarray data. Methods, Candidate Genes from DNA Array Screens: application to neuroscience 31, 265–273. doi:10.1016/S1046-2023(03)00155-5

Teh, C.-H., Chin, R.T., 1988. On image analysis by the methods of moments. IEEE Trans. Pattern Anal. Mach. Intell. 10, 496–513. doi:10.1109/34.3913

Weng, J., Cohen, P., Herniou, M., 1992. Camera Calibration with Distortion Models and Accuracy Evaluation. IEEE Trans. Pattern Anal. Mach. Intell. 14, 965–980. doi:10.1109/34.159901

Xiaochuan, Z., Qingsheng, L., Baoling, H., Xiyu, L., 2009. An image distortion correction algorithm based on quadrilateral fractal approach controlling points, in: 4th IEEE Conference on Industrial Electronics and Applications, 2009. ICIEA 2009. Presented at the 4th IEEE Conference on Industrial Electronics and Applications, 2009. ICIEA 2009, pp. 2676–2681. doi:10.1109/ICIEA.2009.5138693

Yang, Y.H., Buckley, M.J., Speed, T.P., 2001. Analysis of cDNA microarray images. Brief. Bioinform. 2, 341–349. doi:10.1093/bib/2.4.341

# 9. Appendices

**Appendix A:** The generated Matlab code containing all the functions used from the first colour image until the final rotation and cropping of the image array.

**Appendix B:** Principal Component Analysis and the calculation of angle to rotate the image. The reason for choosing Hu's moment invariants over PCA is based on the results generated from this code and the function moment_invariant in appendix A.

**Appendix C:** Code from a Java project for image capturing, it calls for an already existing camera application which will take the photo and then send the photo's storing position back to this application.

**Appendix D:** Code for the final smartphone application.

***MainActivity:*** The GUI part of the application which contains all the part the user can see in the smartphone application.

***ImageProcessing:*** This class is run on a background thread and is performing all the image analysis.

***activity_main:*** Is an XML-file containing information about the layout of the smartphone application, information that is used in the MainActivity.

# Appendix A.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Name:               POC_analysis
% Author:             Julia Erkers
%
% Description:        Takes an image as input. Performs: colour conversion,
%                     reduce noise and image enhancement as a preprocessing
%                     step along with a thresholding. Then the binary image
%                     is rotated and cropped to only contain the image array
%
% Input:              Image captured by a smartphone
% Output:             The original image after rotation and cropping
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function imAnalysis = POC_analysis(image)

% Read in the image as a matrix
im = imread(image);

% Transform from rgb to grayscale, using the green color chanel
imG = im(:,:,2);

% Call a preprocessing function
s = size(imG);
rad = round(max(s)/95);
imPreproc = preprocess(imG, rad);

% Call function to segment the image and remove everything except for the
% array objects.
imSeg = IM_segmentation(imPreproc);

% Call a function to calculate rotation angle using Hu's moments
phi = rotMoments(double(imSeg));
imRot = imrotate(imSeg, phi);

% To make sure the whole array is included in the bounding box, defined as
% the area to crop, a convexhull function is applied on the image to define
% where to crop.
CH = bwconvhull(imRot);
BB = regionprops(CH, 'BoundingBox');
rect2 = BB.BoundingBox;

% Image rotation and cropping is applied to the original image and if the
% resulting image orientation is portrait an additional rotation of 90
% degrees are applied.
imAnalysis = imrotate(imG, phi);
imAnalysis = imcrop(imAnalysis, rect2);
[row, col] = size(imAnalysis);
if (row < col)
    imAnalysis = imrotate(imAnalysis, 90);
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Name:               preprocess
% Author:             Julia Erkers
%
% Description:        Performs a preprocessing to reduce noise and increase
%                     contrast between spots and the backrest
%
% Input:              Greyscale image
% Output:             Greyscale image
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function im_bot = preprocess(image, bot_size)

% Apply a small average filter to image
AvFilter = fspecial('average', 3);
imFiltered = imfilter(image, AvFilter);

% Bothat filter to enhance the contrast between spots and background
im_bot = imbothat(imFiltered,strel('square',bot_size));
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Name:               IM_segmentation
% Author:             Julia Erkers
%
% Description:        Takes an image and integer as input and performs
%                     segmentation on the image along with morphological
%                     operations in order to retreive an image containing
%                     only the array objects.
%
% Input:              Greyscale image,
%
% Output:             Binary image
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function imOut = IM_segmentation(image)

% Thresholding segmentation the image resulting in a binary image.
Tvalue = graythresh(image);
imBinary = im2bw(image, Tvalue);

% By eroding the image and dilate the image small objects are removed and
% the array object are dilated into one/two objects
imErode = imerode(imBinary, strel('disk', 15));
imDilate = imdilate(imErode, strel('disk', 75));

% The value used in bwareaopen are estimated based on a area feature
% histogram. With images used in this project we found that objects smaller
% than 60.000 pixels where too small to be the array object. Therefore
% everything smaller than 60.000 pixels is removed.
imOut = bwareaopen(imDilate, 60000);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Name:               rotMoments
% Author:             Julia Erkers
%
% Description:        Computes the angle (phi), which is how much the image
%                     has to be rotated to get the right orientation.
%
%                     The function is based on: Gnozales, R.C., Woods, R.E.,
%                     2007. Digital Image Processing, Third Edition,
%                     Chapter 11.3.4
%
% Input:              Binary image
% Output:             Angle PHI
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function PHI = rotMoments(I)


if (~ismatrix(I) || issparse(I) || ~isreal(I) || ~isnumeric(I) || islogi-
cal(I))
    error('Image must be 2-dimensional, real, nonsparse, numerical or logi-
cal matrix');
end

I = double(I);

[y, x] = find(I);

% Computing the moments
x_bar = mean(x);
y_bar = mean(y);
my11 = sum( (x - x_bar) .* (y - y_bar) );
my02 = sum( (y - y_bar).^2 );
my20 = sum( (x - x_bar).^2 );

% Calculating the angle to rotate the image
phiR = 0.5*atan( (2*my11) / (my20 - my02) );
PHI = radtodeg(phiR);
```

# Appendix B.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Name:               pca_analysis
% Author:             Julia Erkers
%
% Description:        This function calculates the angle to rotate an image
%                     based on Principal Component Analysis (PCA). It takes
%                     a binary image as input and the angle to rotate is
%                     calculated based on how the image objects are spread
%                     in the image.
%
% Input:              Binary image
% Output:             Vector of integers
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function angle = pca_analysis(image)

[row, col] = find(image);
matrix = [row, col];
covariance = pca(matrix);
[eigvect, eigval] = eig(covariance);

% Chose the eigenvector with the highest associated eigenvalue
if (max(eigval(:,1)) > max(eigval(:,2)))
    v = eigvect(:,1);
else
    v = eigvect(:,2);
end

v1 = v./norm(v,1);
% v2 is a vector with the desired rotation
v2 = [0, 1];

%Calculate angle to rotate image by using the cross product
angle = acosd( dot(v1,v2) / (v1 * norm(v2)) );
```

# Appendix C.

```
/*************************************************************************
    Project name:     useexistingcameraapp
    File name:        MainActivity.java
    Author:           Julia Erkers

    Description:      Open an existing camera application on the smartphone
                      and capture an image by it, save it to the device and
                      should later open it in this application (the last
                      part not implemented).
*************************************************************************/

package com.example.julia.useexistingcameraapp;

import android.app.Activity;
import android.content.Intent;
import android.graphics.Bitmap;
import android.net.Uri;
import android.os.Bundle;
import android.os.Environment;
import android.provider.MediaStore;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.ImageView;
import android.widget.Toast;

import java.io.File;
import java.text.SimpleDateFormat;
import java.util.Date;

import static android.widget.Toast.LENGTH_LONG;


public class MainActivity extends Activity {

    public static final int MEDIA_TYPE_IMAGE = 1;
    public static final int CAPTURE_IMAGE_ACTIVITY_REQUEST_CODE = 100;
    ImageView imgView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        imgView = (ImageView) findViewById(R.id.imageView1);
        imgView.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new
Intent(MediaStore.ACTION_IMAGE_CAPTURE);

                Uri fileUri =
Uri.fromFile(getOutputMediaFileUri(MEDIA_TYPE_IMAGE));
                intent.putExtra(MediaStore.EXTRA_OUTPUT, fileUri);

                startActivityForResult(intent, 0);
```

```
            }
        });
    }


    private static File getOutputMediaFileUri(int type) {
        File mediaStorageDir = new
File(Environment.getExternalStoragePublicDirectory(
                Environment.DIRECTORY_PICTURES), "ExternalCameraApp");

        // Create the storage directory if it does not exist
        if (! mediaStorageDir.exists()) {
            if (! mediaStorageDir.mkdirs()) {
                Log.d("ExternalCameraApp", "failed to create directory");
                return null;
            }
        }

        // Create a media file name
        String timeStamp = new
SimpleDateFormat("yyyyMMdd_HHmmss").format(new Date());
        File mediaFile;
        if (type == MEDIA_TYPE_IMAGE) {
            mediaFile = new File(mediaStorageDir.getPath() + File.separator
+
                    "IMG_" + timeStamp + ".jpg");
        } else {
            return null;
        }
        return mediaFile;
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent
data) {
        super.onActivityResult(requestCode, resultCode, data);

        if (requestCode == CAPTURE_IMAGE_ACTIVITY_REQUEST_CODE) {
            if (resultCode == RESULT_OK) {
                Bitmap bm = (Bitmap) data.getExtras().get("data");
                imgView.setImageBitmap(bm);
                Toast.makeText(this, "Image saved to:\n" + data.getData(),
                        LENGTH_LONG).show();
            } else if (resultCode == RESULT_CANCELED) {
                // User cancelled the image capture
                Toast.makeText(this, "Application were stopped by user",
LENGTH_LONG).show();
            } else {
                // Image capture failed
                Toast.makeText(this, "Image capture failed, something is
wrong",
                        LENGTH_LONG).show();
            }
        }
    }


    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
```

```
        // Inflate the menu; this adds items to the action bar if it is
present.
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();

        //noinspection SimplifiableIfStatement
        if (id == R.id.action_settings) {
            return true;
        }

        return super.onOptionsItemSelected(item);
    }
}
```

```
/*********************************************************************
    Project name:     useexistingcameraapp
    File name:        activity_main.xml
    Author:           Julia Erkers

    Description:      Describing and setting the layout for the
                      application: useexistingcameraapp
*********************************************************************/

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    tools:context=".MainActivity">

    <ImageView
        android:id="@+id/imageView1"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:contentDescription="@string/hello_world"
        android:src="@drawable/ic_launcher"
        android:layout_marginLeft="34dp"
        android:layout_marginRight="36dp" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/tap"
        android:id="@+id/textView"
        android:layout_alignBottom="@+id/imageView1"
        android:layout_centerHorizontal="true"
        android:textSize="20sp"/>

    <RadioButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Yes"
        android:id="@+id/radioButton"
        android:layout_alignParentTop="true"
        android:layout_alignParentStart="true"
        android:checked="false" />

    <RadioButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="No"
        android:id="@+id/radioButton2"
        android:layout_alignParentTop="true"
        android:layout_toEndOf="@+id/radioButton"
        android:layout_marginStart="28dp"
        android:checked="true" />


</RelativeLayout>
```

```
/*******************************************************************************
    Project name:     useexistingcameraapp
    File name:        AndroidManifest.xml
    Author:           Julia Erkers

    Description:      The Android Manifest is a file containing essential
                      information about the application and permission to
                      interact with other components in the smartphone etc.
*******************************************************************************/

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.julia.useexistingcameraapp" >

    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission an-
droid:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-feature android:name="android.hardware.camera2" an-
droid:required="true" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER"
/>
            </intent-filter>
        </activity>
    </application>

</manifest>
```

# Appendix D.

```
/*************************************************************************
    Project name:     pointofcareapplication
    File name:        MainActivity.java
    Author:           Julia Erkers

    Description:      The MainActivity initialise the image processing
                      which is performed in another thread (a background
                      thread). The MainActivity also retrieves partial
                      results which are displayed for the user and
                      eventually also the final result. The MainActivity
                      does not perform any image analysis but it holds all
                      the graphics and interaction with the user.
*************************************************************************/

package com.example.pointofcareapplication;

import java.io.File;

import ImageAnalysis.ImageProcessing;

import org.opencv.android.BaseLoaderCallback;
import org.opencv.android.LoaderCallbackInterface;
import org.opencv.android.OpenCVLoader;
import org.opencv.android.Utils;
import org.opencv.core.Mat;

import android.app.Activity;
import android.graphics.Bitmap;
import android.os.Bundle;
import android.os.Environment;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.ImageView;

public class MainActivity extends Activity {

            // Initiate a TAG for messages.
            private static final String TAG = "POC:Application";
            Activity act;
            Mat m;

            static {
                        if (!OpenCVLoader.initDebug()) {
                                    // Handle initialisation error
                        }
            }

            // The OpenCV loader callback
            private BaseLoaderCallback mLoaderCallback = new
BaseLoaderCallback(this) {
                        @Override
                        public void onManagerConnected(int status) {
                                    switch (status) {
                                    case LoaderCallbackInterface.SUCCESS:
                                    {
```

```
                                        Log.i(TAG, "OpenCV loaded
successfully");
                                        // now we can call opencv code !
                                        String mPath =
Environment.getExternalStorageDirectory()+
            "/DCIM/100_CFV5/DSC_0034.JPG";
                                        File imgFile = new File(mPath);
                                        if (imgFile.exists()) {

            ImageProcessing.startImageProcessing(imgFile, act);
                                        }
                                } break;
                                default:
                                {
                                        Log.i(TAG, "BaseLoaderCallback
did not load successfully");

            super.onManagerConnected(status);
                                } break;
                                }
                        }
            };

            public MainActivity() {
                        act = this;
                        Log.i(TAG, "Instantiated new " + this.getClass());
            }

            @Override
            protected void onCreate(Bundle savedInstanceState) {
                        super.onCreate(savedInstanceState);
                        setContentView(R.layout.activity_main);
            }


            @Override
            protected void onResume() {
                        try {
                                super.onResume();
                                mLoaderCallback.onManagerConnected
(LoaderCallbackInterface.SUCCESS);
                        } catch (final Exception e) {
                                Log.e(TAG, "Exception");
                        }

                        if (!OpenCVLoader.initDebug()) {
                                if (!OpenCVLoader.initAsync(

            OpenCVLoader.OPENCV_VERSION_2_4_2, this, mLoaderCallback)) {
                                        Log.e(TAG, "Cannot connect to
OpenCV Manger");
                                }
                        } else {
                                mLoaderCallback.onManagerConnected
(LoaderCallbackInterface.SUCCESS);
                        }
            }

            public void printImage(Mat m) {
                        // Convert to bitmap
```

```
                        Bitmap bm = Bitmap.createBitmap(m.cols(), m.rows(),
Bitmap.Config.ARGB_8888);
                        Utils.matToBitmap(m, bm);

                        // Find the imageview and draw it.
                        ImageView iv = (ImageView)
findViewById(R.id.array_img);
                        iv.setImageBitmap(bm);
            }


            @Override
            public boolean onCreateOptionsMenu(Menu menu) {
                        /* Inflate the menu; this adds items to the action
                        bar if it is present.*/
                        getMenuInflater().inflate(R.menu.main, menu);
                        return true;
            }

            @Override
            public boolean onOptionsItemSelected(MenuItem item) {
                        /* Handle action bar item clicks here. The action bar
                        will automatically handle clicks on the Home/Up
                        button, so long as you specify a parent activity in
                        AndroidManifest.xml. */
                        int id = item.getItemId();
                        if (id == R.id.action_settings) {
                                    return true;
                        }
                        return super.onOptionsItemSelected(item);
            }
}
```

```
/*************************************************************************
    Project name:     pointofcareapplication
    File name:        ImageProcessing.java
    Author:           Julia Erkers

    Description:      This class implements all the image analysis used in
                      the application from the library: OpenCV. When it is
                      called by the MainActivity it immediately implements
                      threading to prevent the application from crashing.
                      Then it read in the image and perform image analysis
                      as stated in the thesis.
*************************************************************************/

package ImageAnalysis;

import java.io.File;
import java.util.ArrayList;

import org.opencv.android.Utils;
import org.opencv.core.Core;
import org.opencv.core.CvType;
import org.opencv.core.Mat;
import org.opencv.core.Size;
import org.opencv.highgui.Highgui;
import org.opencv.imgproc.Imgproc;

import com.example.pointofcareapplication.R;

import android.app.Activity;
import android.graphics.Bitmap;
import android.util.Log;
import android.widget.ImageView;

public class ImageProcessing {

            // Variables and constants
            private static int mHeight;
            private static int mWidth;
            private Mat mImage = new Mat(new Size(mWidth, mHeight),
CvType.CV_8U);
            private Mat mGrayImage = new Mat(new Size(mWidth, mHeight),
CvType.CV_8U);
            private Mat mBinary = new Mat(new Size(mWidth, mHeight),
CvType.CV_8U);
            private static Bitmap mBitmap;
            private static Bitmap mBitmap2;
            private static Bitmap mBitmap3;

            private ArrayList<Mat> mColourChannels = new ArrayList<Mat>(4);
            private static final String TAG = "ImageProc";
            private static ImageProcessing imgProc;
            private static Mat mLoadImage;


            /*
             * Constructor
             * Read the image from a file direction and set the height and
             * width variables. The constructor also send a message
             * depending on whether or not the load was successful.
             */
```

```
            public ImageProcessing(File mImageName) {
                        mImage =
Highgui.imread(mImageName.getAbsolutePath());
                        if (!mImage.empty()) {
                                    Log.i(TAG, "Image loaded successfully.
Height: " + mImage.height() + " width: " + mImage.width());
                        } else {
                                    Log.e(TAG, "Unable to load image from
path: " + mImageName.getAbsolutePath());
                        }

                        /*
                         * Convert the image to a gray scale image by using
                         * the green channel of the image. Observe: OpenCV
                         * does not use RGB but BGR
                         */
                        Core.split(mImage, mColourChannels);
                        mGrayImage = mColourChannels.get(1);
                        if (mGrayImage.empty()) {
                                    Log.e(TAG, "Convert image to grayscale
failed");
                                    return;
                        }
                        mHeight = mImage.height();
                        mWidth = mImage.width();
                        mLoadImage = mImage;
            }


            /*
             * The method called from the MainActivity which starts off by
             * initialising a new thread on which the method will run all
             * the image processing on.
             */
            public static void startImageProcessing(final File imagePath,
final Activity act) {
                        new Thread(){
                                    @Override
                                    public void run() {
                                                final ImageView iv =
(ImageView)act.findViewById(R.id.array_img);
                                                final ImageView load_btn =
(ImageView)act.findViewById(R.id.loadImg);
                                                final ImageView filter_btn =
(ImageView)act.findViewById(R.id.filterImg);
                                                final ImageView bothat_btn =
(ImageView)act.findViewById(R.id.bothatImg);
                                                final ImageView segment_btn =
(ImageView)act.findViewById(R.id.segmentImg);

                                                imgProc = new
ImageProcessing(imagePath);

                                                mBitmap =
createBitmap(mLoadImage);
                                                act.runOnUiThread(new Runnable()
{
                                                            @Override
                                                            public void run() {
```

```
            iv.setImageBitmap(mBitmap);
            load_btn.setImageResource(R.drawable.greenbtn);
            filter_btn.setImageResource(R.drawable.yellowbtn);
                                                    }
                                                });
                                                try {
                                                        Thread.sleep(2000);
// Set the thread to sleep for 2 seconds so the user are able
                                                } catch (Exception e) { // to
view image before it's replaced

            e.getLocalizedMessage();
                                                }

                                                // Run a blur-method and convert
resulting Mat to a bitmap image.
                                                Mat mPreBImage =
imgProc.preBlur();
                                                mBitmap =
createBitmap(mPreBImage);
                                                act.runOnUiThread(new Runnable()
{
                                                        @Override
                                                        public void run() {

            iv.setImageBitmap(mBitmap);
            filter_btn.setImageResource(R.drawable.greenbtn);
            bothat_btn.setImageResource(R.drawable.yellowbtn);
                                                        }
                                                });

                                                // Run a Bot-hat operation and
convert resulting Mat to a bitmap image.
                                                Mat mPreImage =
imgProc.preBotFilter();
                                                mBitmap2 =
createBitmap(mPreImage);
                                                act.runOnUiThread(new Runnable()
{
                                                        @Override
                                                        public void run() {

            iv.setImageBitmap(mBitmap2);
            bothat_btn.setImageResource(R.drawable.greenbtn);
            segment_btn.setImageResource(R.drawable.yellowbtn);
                                                        }
                                                });
                                                try {
                                                        Thread.sleep(2000);
                                                } catch (Exception e) {

            e.getLocalizedMessage();
                                                }

                                                // Segmenting the image and set
ImageView of UI to the segmented image.
                                                Mat mSegmentImage =
imgProc.segmentation(mPreBImage);
```

```
                                                mBitmap3 =
createBitmap(mSegmentImage);
                                                act.runOnUiThread(new Runnable()
{
                                                          @Override
                                                          public void run() {

            iv.setImageBitmap(mBitmap3);
            segment_btn.setImageResource(R.drawable.greenbtn);
                                                          }
            });
                                    }
                        }.start();
            }


            /*
             * Method that takes an image Mat and creates a Bitmap.
             */
            private static Bitmap createBitmap(Mat image) {
                        Bitmap mBitmap = Bitmap.createBitmap(image.cols(),
image.rows(), Bitmap.Config.ARGB_8888);
                        Utils.matToBitmap(image, mBitmap);
                        return mBitmap;
            }

            /*
             * Method for splitting up the colour layer and convert from
             * RGB to Gray-scale by choosing the green layer.
             */
            private Mat preBlur() {
                        // Blurs the image using an average filter
                        Imgproc.blur(mGrayImage, mGrayImage, new Size(7, 7));
                        return mGrayImage;
            }

            /*
             * The filter size radius for the blackhat transformation is
             * determined by dividing the greater value of the image height
             * and width divided by 95. However, OpenCV does not use radius
             * as a measure for the filter size so we have to multiply our
             * radius by 2 to get the right measure.
             */
            private Mat preBotFilter() {
                        Size radius = new Size(0, 0); // Initialised to speed
up the process
                        Mat mBotImage = new Mat(new Size(mWidth, mHeight),
CvType.CV_8U);

                        if (mHeight > 1000 || mWidth > 1000) {
                                    radius = new Size(63, 63);

                        } else { // If the image is to small set the size to
3x3
                                    radius = new Size(5, 5);
                        }

                        /*
                         * Sets the structural element to have elliptic shape
                         * and the size calculated based on the image sizes.
```

```
                        * The mOffset is set to the centre of the kernel.
                        * Then the morphological operation Blackhat is used
                        * on the image. The destination image replaces the
                        * source image.
                        */
                       Mat mStructuralElement =
Imgproc.getStructuringElement(
                                                   Imgproc.MORPH_RECT, radius);
                       Imgproc.morphologyEx(
                                                   mGrayImage, mBotImage,
Imgproc.MORPH_BLACKHAT, mStructuralElement);
                       return mBotImage;
           }

           private Mat segmentation(Mat image) {
                       /*
                        * Apply an adaptive thresholding segmentation to the
                        * image.
                        */
                       Imgproc.adaptiveThreshold(image, mBinary, 255,
Imgproc.ADAPTIVE_THRESH_MEAN_C , Imgproc.THRESH_BINARY, 17, 3);

                       /*
                        * Remove small objects by the morphological
                        * operation: erode.
                        */
                       return mBinary;
           }
}
```

```
/*********************************************************************
    Project name:     pointofcareapplication
    File name:        activity_main.xml
    Author:           Julia Erkers

    Description:      Describing and setting the layout for the main
                      activity in the application: pointofcareapplication.
*********************************************************************/

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.pointofcareapplication.MainActivity" >

    <TextView
        android:id="@+id/intro_text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:paddingBottom="10dp"
        android:paddingLeft="10dp"
        android:paddingRight="10dp"
        android:paddingTop="15dp"
        android:text="@string/introText" />

    <ImageView
        android:id="@+id/loadImg"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/intro_text"
        android:layout_alignParentLeft="true"
        android:src="@drawable/redbtn" />
    <TextView
        android:id="@+id/text_load"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/intro_text"
        android:layout_centerHorizontal="true"
        android:text="@string/image_load" />

    <ImageView
        android:id="@+id/filterImg"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/loadImg"
        android:layout_alignParentLeft="true"
        android:src="@drawable/redbtn" />
    <TextView
        android:id="@+id/text_filter"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/text_load"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="10dp"
        android:text="@string/image_filtered" />

    <ImageView
        android:id="@+id/bothatImg"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
        android:layout_below="@+id/filterImg"
        android:layout_alignParentLeft="true"
        android:src="@drawable/redbtn" />
    <TextView
        android:id="@+id/text_bothat"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/text_filter"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="10dp"
        android:text="@string/image_bothat" />

    <ImageView
        android:id="@+id/segmentImg"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/bothatImg"
        android:layout_alignParentLeft="true"
        android:src="@drawable/redbtn" />
    <TextView
        android:id="@+id/text_segment"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/text_bothat"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="10dp"
        android:text="@string/image_segmented" />

    <ImageView
        android:id="@+id/array_img"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_below="@+id/text_segment"
        android:layout_marginTop="15dp"
        android:src="@drawable/ic_launcher" />

</RelativeLayout>
```