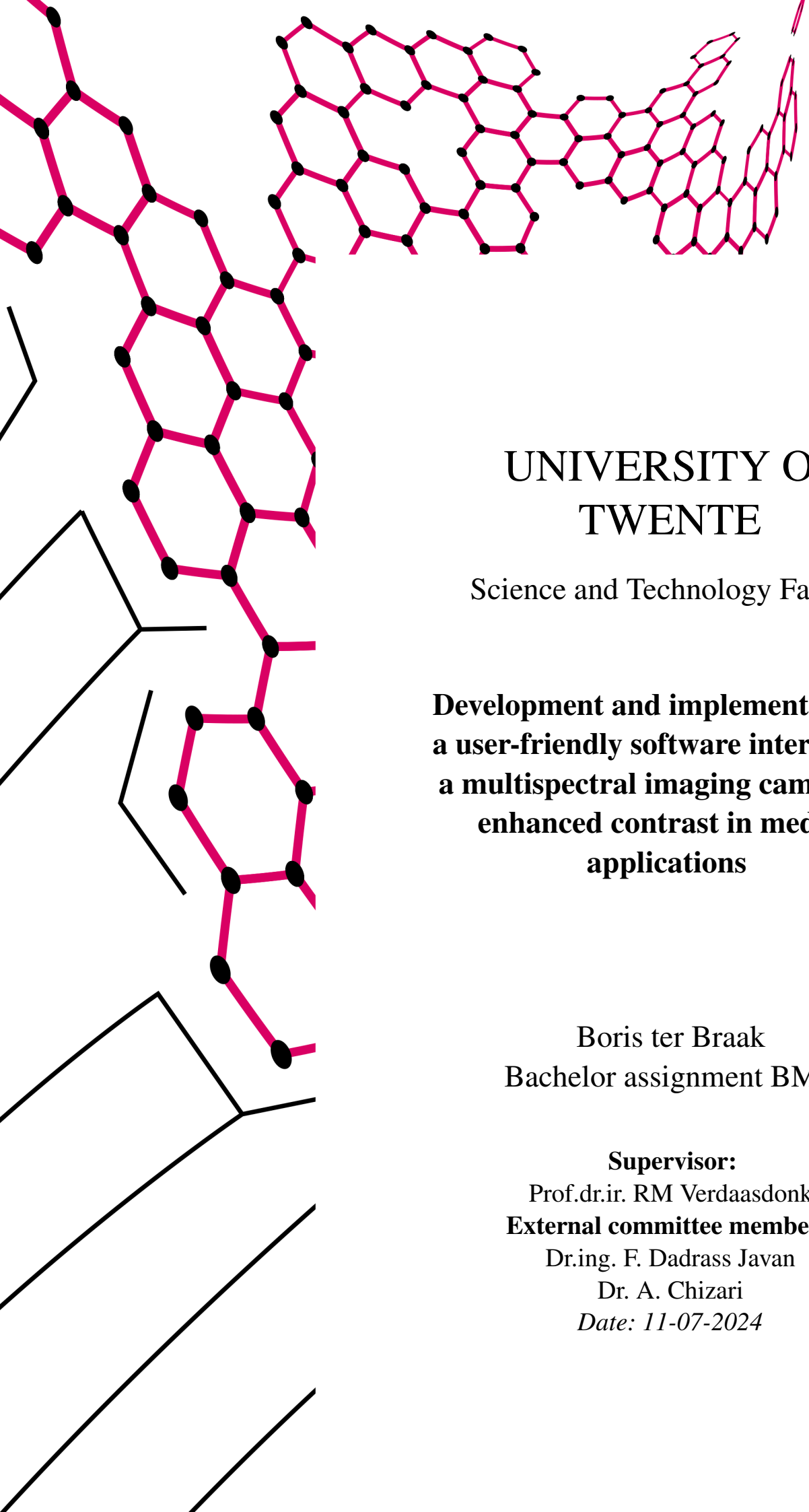


UNIVERSITY OF TWENTE

Science and Technology Faculty

Development and implementation of a user-friendly software interface for a multispectral imaging camera for enhanced contrast in medical applications

Boris ter Braak
Bachelor assignment BMT

Supervisor:
Prof.dr.ir. RM Verdaasdonk
External committee members:
Dr.ing. F. Dadrass Javan
Dr. A. Chizari
Date: 11-07-2024

Contents

1 Abstract **2**

2 Introduction **3**

3 Theoretical background **4**
3.1 The principles of multispectral imaging . . . 4
3.2 Spectral filter array cameras . . . 4
3.2.1 Array . . . 4
3.2.2 Image imperfections . . . 5
3.3 Skin optics . . . 6
3.3.1 Perfusion . . . 6
3.3.2 Oxygenation . . . 7
3.3.3 Pigment . . . 7
3.3.4 Abnormal tissue . . . 7
3.4 Image enhancement . . . 7

4 Methods **8**
4.1 Requirement elicitation . . . 8
4.2 Tools and specifications . . . 8
4.3 Initial design . . . 9
4.3.1 Software design . . . 9
4.3.2 Pre-processing . . . 10

5 Results **12**
5.1 First version . . . 12
5.1.1 Overview . . . 12
5.1.2 Strengths . . . 13
5.1.3 Shortcomings and solutions . . . 13
5.2 Final design . . . 18
5.2.1 Overview . . . 18
5.2.2 Resulting images . . . 22

6 Discussion **23**
6.1 Requirements . . . 23
6.2 Images . . . 23
6.3 Shortcomings and possibilities for future improvements . . . 23

7 Conclusion **25**

References **26**

8 Appendix **28**
8.1 Code . . . 28

1 Abstract

Multispectral cameras have great applications in the medical industry due to their inherent ability to take the same image using multiple wavelengths. This makes these cameras useful for visualizing chromophores and processes residing in/beneath the outer layer of skin. Image (pre-)processing is required to obtain high-quality data, which include spatial rearrangement and corrections. The purpose of this study is to create a user-friendly application for medical professionals to create contrasting images with enhanced features. The development process is described and feasability of the program is tested. The results show significant image enhancement using presets created with the program and the program meets all requirements set. It is user-friendly due to the simple design and limited functions and shows to be useful for imaging with medical applications. Improvements could be made in image quality and expanding the support base for the program.

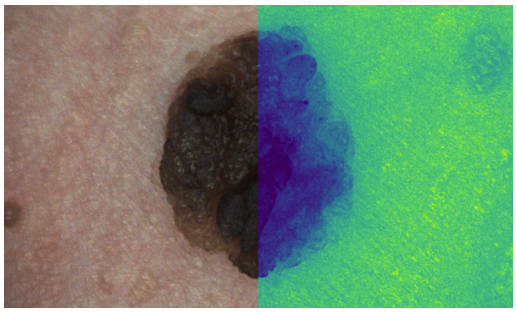
2 Introduction

The skin is the largest organ of the human body and investigating it can provide crucial information about someones well-being. Not only does our skin protect us from the outside world, it also contains various structures and chromophores giving physicians inspecting it an idea about the health condition.

Visual inspection is the best method possible to do so as it allows inspection without the need to perform invasive actions and having the added risk of infection with unwanted side-effects as a result. Visual inspection is based on how light interacts with different tissues. The amount of absorption and reflection of the skin at certain wavelengths contains information the structures underneath the skin, invisible to the naked eye.

Regular cameras film in RGB, thus having only three broad colorbans (red, green and blue) with which to inspect the skin. This does not give sufficient discrimination to look at structures beneath the outer layer of skin. Multispectral cameras on the other hand can have between 3 to 25 bands or even more giving a significant increase in wavelengths to choose from. Next to giving a wider spectrum of wavelengths (sometimes including even infrared and/or ultraviolet), it also allows for a far more specific selection of wavelengths and thus a higher spectral resolution. Altogether, multispectral cameras give the ability to enhance certain features with applications such as pre-operation assessments.

(a) RGB image of a hand (left) compared to a multispectral image of the same hand (right) with enhanced visualization of blood vessels [1]



(b) RGB image of melanoma (left) compared to a multispectral image of the same melanoma (right) [2]

Figure 1: Examples of medical applications of multispectral imaging. (a) shows enhanced perfusion and (b) shows the possibility of detecting malignant melanomas

Some multispectral cameras (like the one used for this research) are spectral filter array (SFA) cameras. A single snapshot image contains information of 16 or 25 wavelengths in the filter. Processing of this image is needed to extract the desired information and to be able to visualize chromophores or physiological processes.

Most companies deliver specialized applications along with their cameras in order to get and manipulate multispectral image data for scientific applications, but not for user-friendly medical applications.

The goal of this research is to create such a program for a multispectral camera to combine any of the available wavelength images with a simple analytical formula to enhance specific features in an image like anatomical structures and/or physiologic processes. A researcher should be able to test several combinations by inserting images in the formula and saving them as a preset. Then, from within another simple and user-friendly interface meant for medical professionals, these presets can be selected for specific diagnostics.

3 Theoretical background

3.1 The principles of multispectral imaging

There are some key aspects to consider when working with a multispectral camera. They are displayed in figure 2.

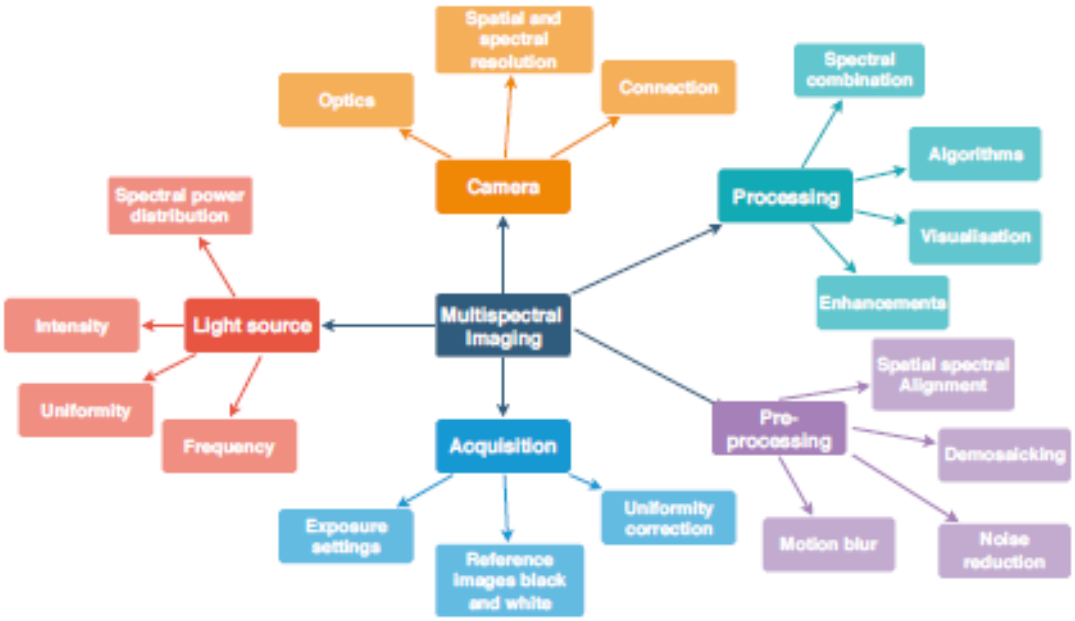


Figure 2: Key aspects of multispectral imaging [3]

For uniform and reliable results, one should consider the light source in the optical setup, the specifications of the camera, the parameters for acquisition and the processing steps. While this report mostly focuses on the pre-processing- and processing steps, the other aspects are still taken into consideration.

3.2 Spectral filter array cameras

3.2.1 Array

SFA cameras, like the one used in this research, use an array covering a megapixel sensor. For standard RGB cameras this array is a 2x2 grid, mostly organized like in figure 3. So each megapixel gets multiple signals representing the colors of the array. From this, a full image can be reconstructed through image processing.

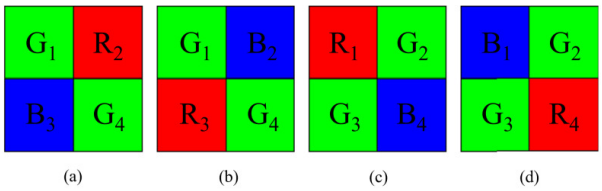
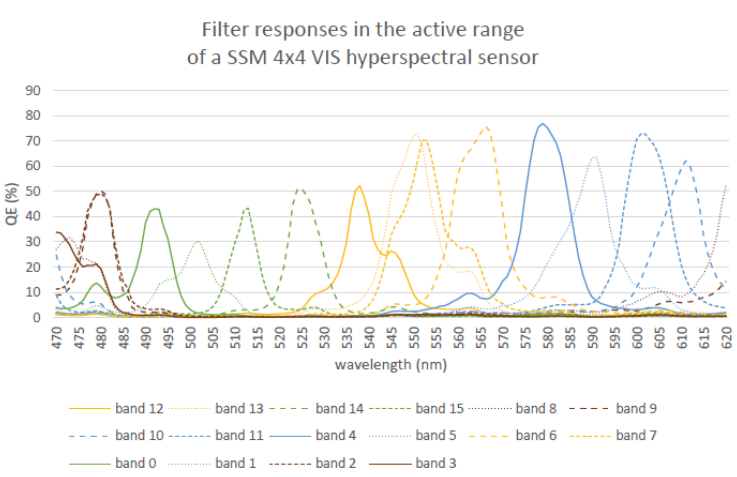


Figure 3: 2x2 bayer patterns [4]

This 2x2 array is scaled up for SFA cameras. The camera used for this research uses a 4x4 grid as seen in figure 4a. This array has 16 colors (wavelengths between 490nm and 618nm) and with a higher spectral resolution, because the bands (figure 4b) are smaller than those of the standard bayer pattern RGB cameras use. This allows for a precise selection of wavelengths without as much signal from unwanted wavelengths around the peak wavelength.

0 490	1 500	2 477	3 478
4 577	5 591	6 563	7 553
8 618	9 616 478	10 613	11 600
12 538	13 549	14 523	15 511

(a) Spectral filter array of the camera with peak wavelengths in [nm]



(b) Spectral response for each band in the array

Figure 4: Layout of the spectral filter array (a) and their respective responses (b). Extracted from a from a file containing specifications, given along with the camera

3.2.2 Image imperfections

Spectral imperfections

Images received from hyperspectral cameras are not entirely uniform in the spectral dimension. For example, the SFA does not have the same sensitivity for each color. Some colors are received with higher intensity than others [5].

Besides that, pixels can influence the signal of neighbouring pixels. This is called 'cross-talk' [3]. Instead of only receiving a signal from the intended wavelength as defined by the filters spectral sensitivity for the given pixel, that pixel also receives some signal from another wavelength. They show a second order peak next to the intended peak wavelength.

For figure 4b, a uniform lightsource is used. Every lightsource has its own emission spectrum [6]. Using a lightsource without a uniform spectrum (the same emission for each wavelength) could cause imperfections where some wavelengths would get significantly more (or less) signal than others, if not accounted for.

Spatial imperfections

The illumination over the image is not constant. This phenomenon, called 'vignetting' causes a fall-off of the signal towards the edges in comparison to the center of the image [7]. The illumination follows a Gaussian curve over the width and length of the image.

Noise

The sensor creates a certain amount of noise in the image, even if no signal is received, for example when the light source is turned off [5]. This 'dark noise' creates inconsistencies in the illumination of the final image. This issue is amplified when images have a low signal-to-noise ratio (SNR). If there is a relatively high amount of noise compared to a low amount of signal, the influence of noise in the final image will become more apparent. Two ways to go about this are to decrease the noise or increase the signal, such that the SNR becomes higher and the relative influence of noise lower.

3.3 Skin optics

Chromophores are the particles within the skin that interact with incoming light. The most important chromophores can be seen in figure 5.

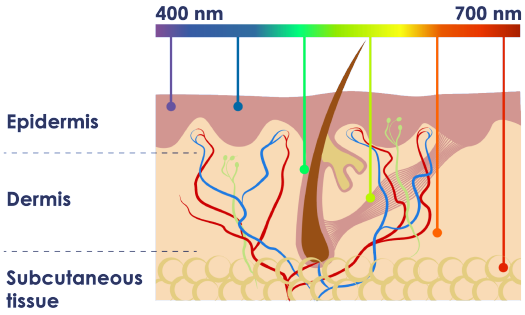


Figure 5: Layers of the skin and a visualization of how far different wavelengths penetrate the skin [8]

The epidermis is the outer layer of skin and is also what gives the skin its color due to the presence of melanin. The layer below is where blood vessels and hair follicles are found. The difference in reflection and absorbance curves can be used to distinguish between these chromophores. However, there are some large absorbers in the skin that result in some spectral regions being useless for visualization. Proteins absorb the short wavelength ultraviolet light and most of the longer wavelengths from infrared onwards are absorbed by water and tissue within the skin. This leaves the spectral region of visible wavelengths and Near Infrared (NIR) wavelengths as the useful so-called optical window. Within this window, multiple chromophores can be found such as melanin and (oxy)hemoglobin (figure 6), but also some physiological processes like perfusion and oxygenation. Some of these chromophores and processes are explained below.

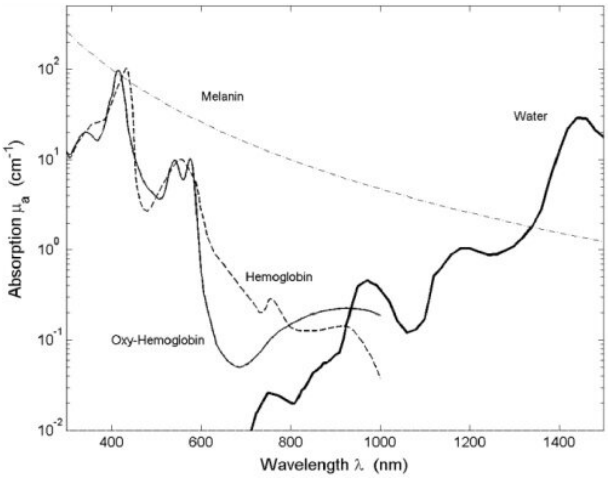


Figure 6: Absorption spectra of chromophores present in the skin [9]

3.3.1 Perfusion

Perfusion is the flow of blood through tissue. It can be visualized by using the absorption spectra for (oxy-)hemoglobin in figure 6 as these chromophores are only present in blood.

3.3.2 Oxygenation

Oxygenation is the amount of oxygen present within the blood. The way to visualize this is to look at the differences between the spectra of oxygenated- and deoxygenated hemoglobin (figure 6). Hemoglobin is a molecule in blood to bind oxygen and transport it. Binding oxygen changes the absorption spectrum of this molecule. Because of this, the absorption spectrum of oxygenated hemoglobin differs from that of deoxygenated hemoglobin [10].

3.3.3 Pigment

Pigment is mainly caused by the presence of melanin in the skin. This polymer has a characteristically wide absorption spectrum resulting in darkness of the skin when present in high concentrations. It just absorbs that much of the visible spectrum [11]. The absorption spectrum can be seen in figure 6.

3.3.4 Abnormal tissue

Abnormal tissue is tissue with an abnormal growth rate, otherwise known as tumor tissue. There is no singular absorption spectrum for cancerous tissue, but there are indicators based on abnormal concentrations of certain molecules. It has been found that abnormal tissue shows an increased concentration of hemoglobin due to angiogenesis [12], which is the process of new capillaries being formed, excessively present in tumor tissue as rapidly growing populations of cells need oxygen to survive [13]. So abnormal tissue could also be detected by looking for abnormally high levels of blood in similar fashion to 3.3.1. Furthermore, melanoma can be detected by looking at irregular deposition of melanin in the skin. Melanoma originate from melanocytes which are responsible for the production of melanin [14]. Malignant growth of these melanocytes causes dark pigment spots on the skin by which cancerous cells could be detected.

3.4 Image enhancement

Image enhancement is the process of making images more useful. This could mean making images more visually appealing, but also (and especially in this research's context) bringing out specific features of an image. Image enhancement to bring out the features above from 3.3 is done by looking at differences in absorption/reflection at different wavelengths. A method to visualize perfusion is performed by He and Wang [15]. The formula used is a sum of weighted averages between wavelengths shown below in formula 1.

$$Perfusion = (556nm - k * 625nm) + (529nm - l * 603nm) + (556nm - m * 615nm) \tag{1}$$

With k, l and m weight factors (mostly between 0.5 and 2) of the subtraction. They use the same principle for pigmentation according to formula 2.

$$Pigmentation = (482nm - k * 543nm) + (482nm - l * 529nm) + (482nm - m * 544nm) \tag{2}$$

It can be seen from these formulas that the principle behind image enhancement is choosing wavelengths in the weighted subtraction such that there is a difference in absorption of the desired chromophore between those wavelengths. Summing multiple weighted subtractions together only amplifies the contrast of the result. The same principle can be applied to any other chromophore within the active spectral range of the sensor.

4 Methods

4.1 Requirement elicitation

The most important part of designing a program is defining what functionalities it needs to hold. There should be a clear understanding of the requirements before taking any steps in implementation. For this application, the most important requirements are:

- The application must be able to show a real-time capture of a combination of images from the camera.
- The researcher must be able to combine images of wavelengths by simple analytic manipulations (+,-,/,x).
- The researcher must be able to save the formula as a preset which can later be accessed again.
- The doctor must be able to access and select any of the presets
- The application must be able to combine the images in such a way that enables the researcher to create presets that show certain structures in the skin with optimal contrast.
- The application should have an additional page where RGB images can be made by selecting wavelengths for the individual 'R'-, 'G'- and 'B'-channels.
- The user should be able to save images
- The user should be able to display saved images and edit them

4.2 Tools and specifications

Image acquisition was done with the XIMEA MQ022HG-IM hyperspectral camera with the SM4X4-VIS3 filter [16]. It has a spectral range of 460-600nm and consists of 16 spectral bands. The exposure was set to 100ms and was constant throughout the whole process.
Programming was done with python v3.11. Functions to access XIMEAs camera were provided in their xiapi python module. Other imported modules included Pillow and OpenCV (for computer vision and real-time image processing) and Numpy (for calculations with matrices). Lastly, the script is turned into an executable using PyInstaller. It also analyzes the script to discover every necessary library and module to run the code.

4.3 Initial design

4.3.1 Software design

Based on the requirements from 4.1, a flowchart was made with all necessary functionality (figure 7).

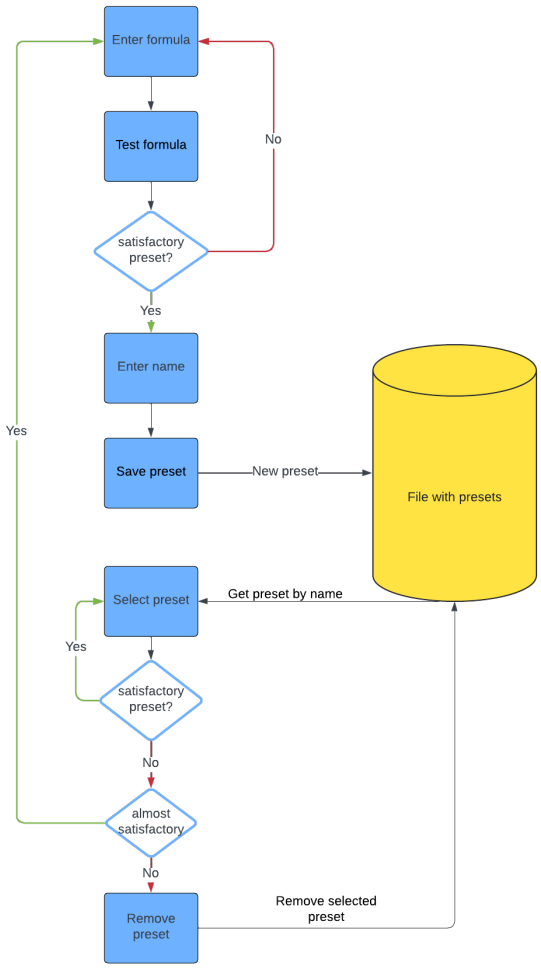


Figure 7: Flowchart of how presets are added and selected in the initial design

A researcher can enter a formula containing analytical operators and individual images as a string in a textbox and then test the formula, after which the result of that formula is displayed on screen. If they decide they want to keep the preset, they can enter a name for the preset and press a 'save' button which saves the preset in a file, recognizable by its name. The same preset can then be selected from the file, giving them the ability to edit or remove the preset. A doctor will have access to the same file with presets from a different interface where all they can do is select the desired preset from a list with all presets.

4.3.2 Pre-processing

A schematic overview of the necessary pre-proccesing steps is given below in figure 8.

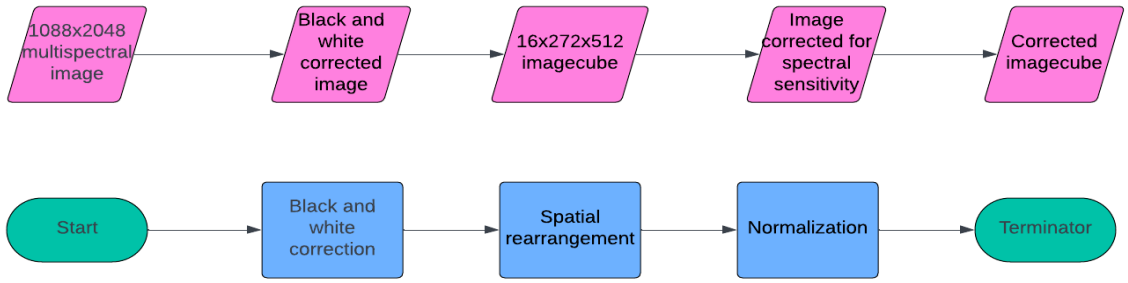


Figure 8: Flowchart of the pre-processing steps

First of all, a black- and white correction should be applied to account for the spectral imperfections named in 3.2.2. A process for this is given by [17] and [5], which has been shown to account for inhomogeneous illuminations. A black- and white reference image should be taken with the same setup as with which the camera is about to be used. The black image can be acquired by fully blocking the camera and then taking an image. This will account for any imperfections that are present, even when no signal is obtained. As for the white image, a white sheet of paper should be held at some distance from the camera to correct for uneven signal strength along the spatial dimensions with the current optical setup (in combination with surrounding light sources). From then on, using these images, a formula (3) should be applied to the desired image to get the corrected image.

$$I_{corrected} = \frac{I - I_D}{I_W - I_D} \tag{3}$$

With I the acquired image, I_D the dark reference image and I_W the white reference image. All are arrays with numbers representing the received signal strengths per pixel.

The resulting image has a size of 1088x2048 pixels. This image can be separated into 16 272x512 images. The full image contained repeated grids of 4x4 pixels of different wavelengths in accordance with image 4a. So by reducing the spatial resolution to 1/4 in both x and y direction, images were acquired of each individual wavelength, thereby creating a so-called 'imagecube' [18]. An image with, not only, an x- and y-axis as the spatial dimensions, but also an extra spectral dimension representing all wavelengths (as shown in figure 9).

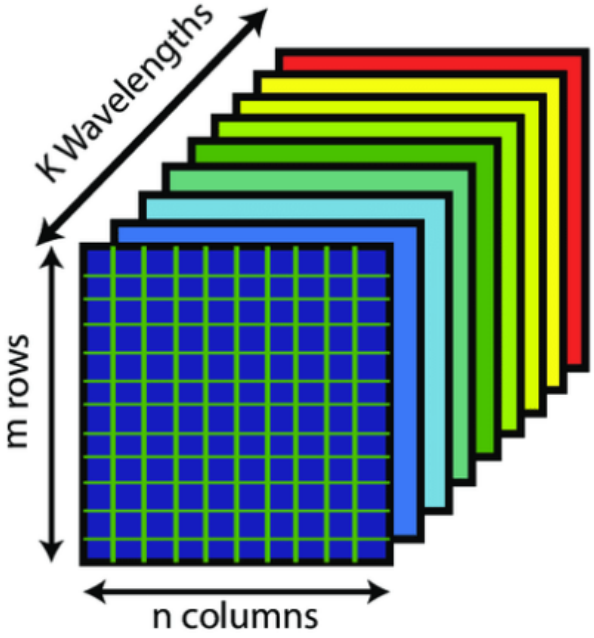


Figure 9: Image cube with two spatial- and one spectral dimension [19]

Furthermore, a normalization is applied, because of the difference in sensitivities between the different wavelengths (section 3.2.2). Intensities are read as a value between 0 and 255 with, 0 being no received signal and 255 being full received signal strength. To make sure that every wavelength returns the same signal strength, the image is multiplied by a gain to stretch the intensity such that the maximum intensity is 255 and therefore equal for all wavelengths. This is done by:

$$I_{Normalized}(\lambda) = I(\lambda) * Gain(\lambda) \tag{4}$$

$$Gain(\lambda) = \frac{255}{I_{max}(\lambda)} \tag{5}$$

With I the array of the corresponding image. This formula makes sure that even wavelengths that receive less signal can be used in calculations without losing their information.

5 Results

5.1 First version

5.1.1 Overview

A sequence diagram is made as a summary of the program (shown in figure 10). Some details are missing regarding for example checking if selected and created presets are valid and how the images are created, but it is a good, simplified visualization for what happens behind the scenes for the most important functionalities.

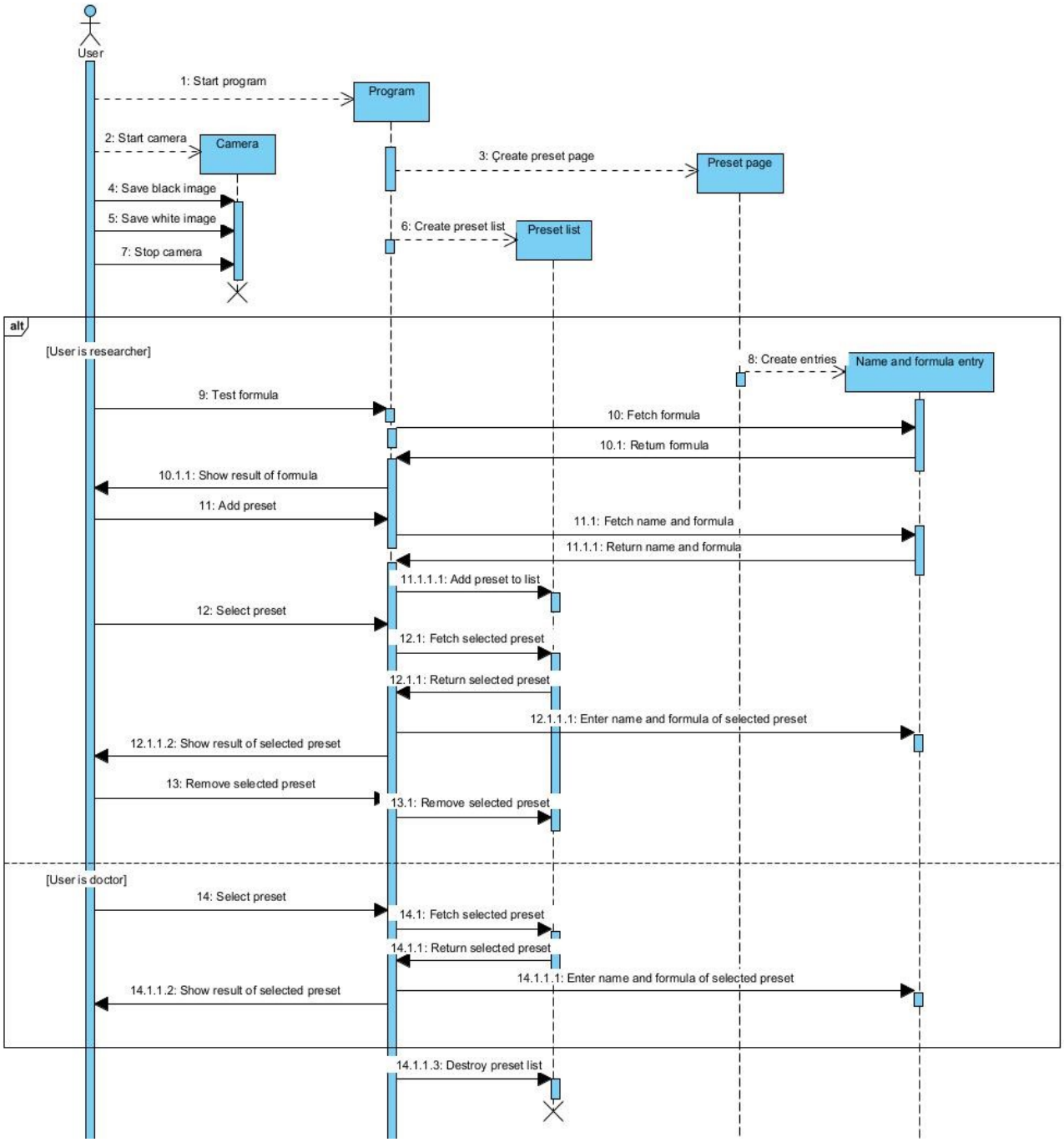


Figure 10: Sequence diagram of the first version of the final design

5.1.2 Strengths

The design has all necessary functionality for adding and removing presets. Presets can be edited and overwritten. Upon selecting a preset, the name and formula of that preset would be visible in the textboxes, such that the name and formula can be changed.

5.1.3 Shortcomings and solutions

Unexpected behaviour

First of all, errors weren't caught by the system which resulted in problematic behaviour as certain unwanted inputs could crash the system. For example, the buttons for starting and stopping the camera and the ones for saving a black and white reference image could be pressed without a camera being connected. An error would then be displayed in the terminal that no camera was found and the program would freeze. In addition to that, a user could enter a formula that is not of the expected form. This would give the same result, just with a different error. There are more cases for which this kind of behaviour would occur. These sort of unexpected actions should be accounted for such that the program continues working after these inputs.
To account for unexpected inputs, so-called 'try-except'-blocks were added where needed for every possible wrong action such that the system would never crash anymore, but it would display pop-ups with the error message instead informing the user of what went wrong and then resuming the program as it was before. These blocks execute the code written within the try-statement, unless an error occurs. Then it executes the code within the except-statement. This allows the program to catch errors and handle them

Black and white correction

Implementing the black and white correction as described in section 4.3.2 resulted in issues, mostly having to do with division by zero. Any pixel which was divided by zero was automatically set to positive infinity by the program. This resulted in noise over the image, reducing the quality. A work-around was to set all pixels with value positive infinity to zero. This ended up working, but another problem surfaced, previously undiscovered. Doing the correction this way severely reduced the intensity of the final image, as the denominator was bigger than the numerator by some magnitude. As a result, all pixels got a value between 0 and 1 making the image invisible. Stretching out the values back to a maximum of 255 was tried, but the resulting image had a low resolution and lots of noise.
Multiple attempts were made to fix the problems described above, but none sufficed. The array representing the white image always contains quite a lot of zeros. This means that element-wise division creates division-by-zero problems. Removing all the zeros from the array and resizing back to the original size altered the image too much for it to be useful. After careful consideration, the decision was made to not implement a black and white correction. It was found that the black image (with the camera covered correctly, for example with the lens cover) was fully black and did not show any inherent noise. As for the white image, the problem would be vignetting as described in section 3.2.2. However, the intensity from the furthest edge to the center of the image only did not differ significantly (a difference of at most 20 on a scale of 0-255, this is barely even visible for the naked eye). While not perfect, it was deemed sufficient for the intended purpose of the application.

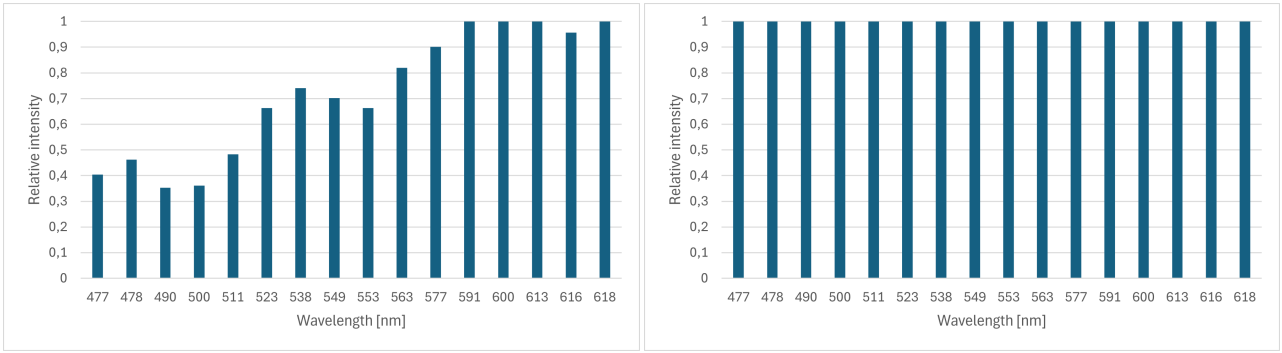
Normalization

Normalization as described in section 4.3.2 didn't have the desired effect. Although each wavelength displayed the same intensity, problems arose because of the inconsistency of the maximum value for each wavelength. During the initial design phase, it was assumed that the maximum value would be relatively constant and the on-screen intensity would gradually be adjusted. Instead, the maximum value changed each frame resulting in flickering images for frames without movement, let alone when movement was introduced. Moreover, because some wavelengths received less signal than others, they had the tendency to be underexposed causing noise to appear in the image. Amplifying the image by a constant resolves the intensity issue, but also amplifies noise. At first, a solution to this problem was to not apply normalization at all. This ensured that flickering was no longer an issue, but it meant that intensities per wavelength would differ immensely due to the sensitivity of the sensor. When combining multiple images into a formula, it is preferred for them to have the same intensity such that each wavelength used has a similar influence on the final result. After consideration, it was decided

that the best method would be to upscale lower intensity wavelengths as a ratio compared to the one with the highest intensity, to keep the gain constant. A record was made of the maximum intensity of each wavelength with constant settings (see table 1, figure 14). Then, for each iteration, multiple images are taken and added together per wavelength however many times necessary to reach the desired intensity (displayed in the final column of table 1), solving the intensity issue and simulating a variable exposure time within the snapshot images. This also decreases noise, because the amount of signal used for each wavelength becomes higher, thus increasing the SNR as described in section 3.2.2. This yielded the results shown in figure 14 and figure 12b.

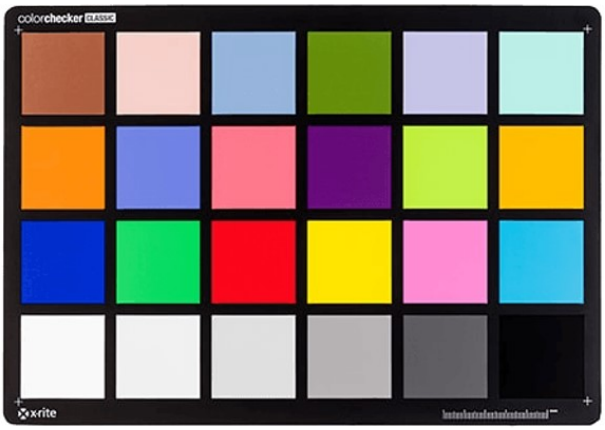
Table 1: Table of wavelengths and their intensities and respective gain used for the program

Wavelength [nm]	Intensity	Gain (255/intensity)
477	103	2.48
478	118	2.16
490	90	2.83
500	92	2.77
511	123	2.07
523	169	1.51
538	189	1.35
549	179	1.42
553	169	1.51
563	209	1.22
577	230	1.11
591	255	1.00
600	255	1.00
613	255	1.00
616	244	1.05
618	255	1.00

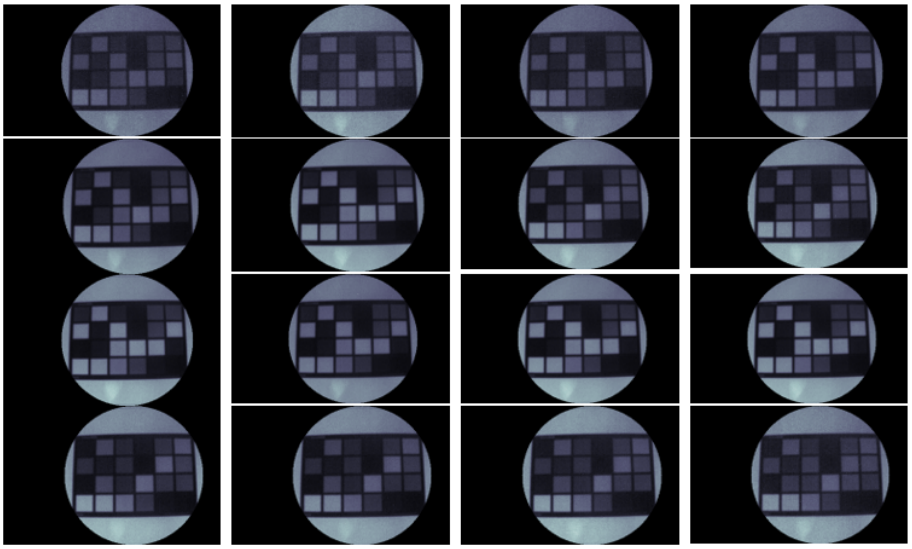


(a) (b)

Figure 11: Graph of the sensitivity per wavelength compared to the wavelength for which the sensor has the highest sensitivity before (a) and after (b) normalization



(a) Color checker used for 12b



(b) Image for each wavelength organised the same as the spectral filter array (section 4a) after normalization is applied

Figure 12: Reference color checker (a) and an image of the color checker taken for each wavelength after normalization (b)

Preset storage

Furthermore, presets were only saved temporarily while running the program. Closing the program caused all created presets to be forgotten (as can be seen by the destroyed preset list at the bottom of figure 10). It is required that saved presets could still be accessed after closing and reopening the program. Otherwise doctors would not be able to access created presets.
In the final design, storing presets is done via a json-file accessed by the script. The script checks whether such a file already exists. If not, it creates an empty one within the same folder and writes an empty dictionary to the file. If a file does exist, the dictionary contained within the file is accessed by the program. This file is saved automatically with each update. This means that any added preset will still be available the next time the interface is used after closing it. It has the added advantage of being able to send created presets to others using the program elsewhere. A copy of the file can be exported and then placed by someone else into the dedicated folder where their program can access it in the way described above.

Selecting presets

In the initial design, a doctor using the system can select presets from a list of all presets created by a researcher. Evaluation resulted in a slight change of the requirements. Where before the doctor was able to select any of the presets (also seen in figure 10, where the doctor has access to the entire preset list), it now became clear that the doctor only needs to be able to access the most important presets and they should be able to do so quickly and easily whereas in the initial design it is unnecessarily cumbersome.

To make the process of selecting a preset quicker and easier for a doctor using the system, buttons were created which could have presets assigned to them by a researcher. Fixing a single preset to a button was not a viable solution, as presets could still be changed by a researcher afterwards. This is why the decision was made to assign presets by name. If the researcher creates with the same name as displayed on the button, that preset will be assigned to the respective button. Clicking on the button then displays the result of that preset on-screen. Since no 2 presets can have the same name, no situation will occur where more presets are added to a single button. Furthermore, clicking on a button when no preset is assigned returns no error, but just shows a black screen instead. Therefore, this method is foolproof and meets the requirement set.

RGB image

After evaluation of the design, it became clear that the requirement to be able to show an RGB image in fashion of figure 13 was not yet met. Previously, images were only displayed as a greyscale image with a representation of the intensity per pixel. This gave useful results, but an additional page to create an RGB-image was desired.

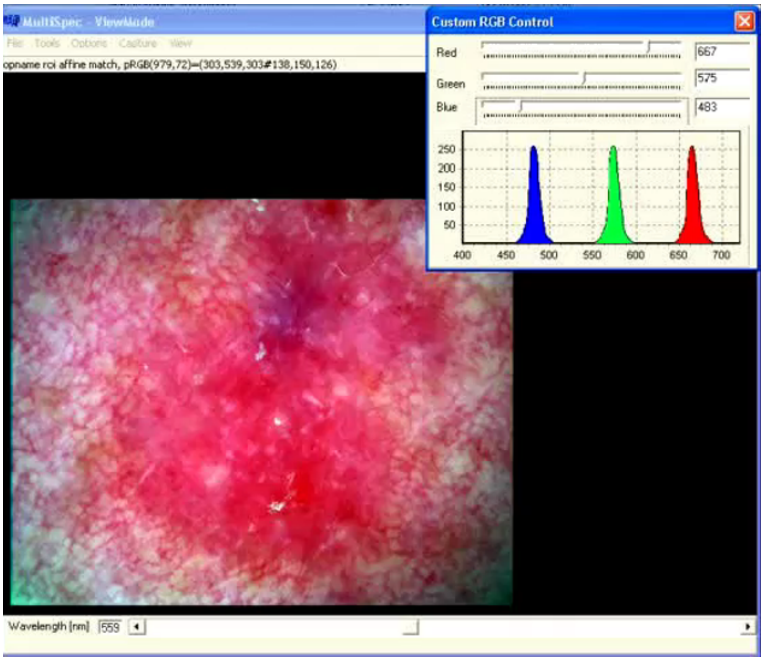
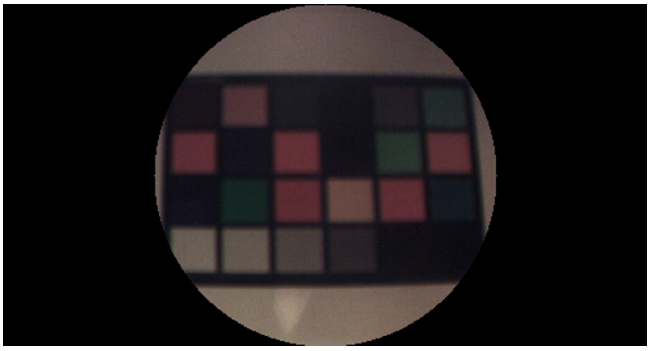
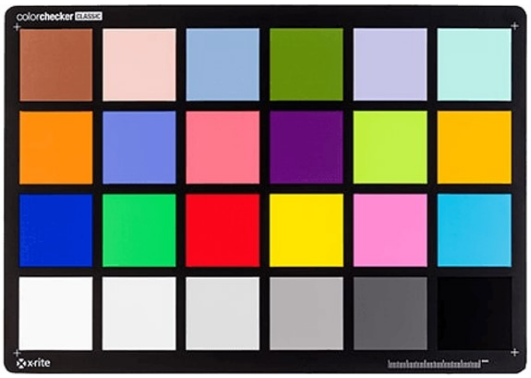


Figure 13: Desired way of creating an RGB-image. Program has three sliders, one for each of the channels to assign any wavelength to the respective channel

To enable the user to create an RGB image within the same interface. A new page was added to the program. When the researcher interacts with this page, they can select a wavelength for each of the channels and increase or decrease their respective intensities. This gives the researcher a lot of possibilities in order to create their optimal image. Doctors interacting with this page again see buttons with names which have presets assigned to them as desribed above.
Saving presets for this page was a problem to overcome. Instead of saving only one formula, presets of this form need to hold 6 independent pieces of information. One for each of the channels and one for each channel intensity. After careful consideration and multiple failed tests, the final solution was to save presets as a dictionary entry, similar to how grayscale images are saved from the other page. For this preset however, all necessary bits of information are added as a tuple in the dictionary. The program distinguishes between the elements of the tuple by assigning all elements to their dedicated place on the page.
This created a different issue of being able to select any type of preset in any page. Since the form of these presets is different, presets from one page are not compatible with the other, resulting in errors being given by the system. That is why presets in the RGB form get a signature (_RGB) added to their name and presets in the grayscale form can not contain that signature anywhere in their name or the system notifies the user that the entered name is invalid. Based on this method, only presets containing the signature are available on the RGB page and vice versa.



(a) Image created with the RGB-page of the color checker in (b) with wavelengths: 618nm (R), 563nm (G) and 500nm (B)



(b) Color checker used for (a)

Figure 14: Image created with the RGB page (a) next to reference image (b)

Saving and uploading images

Lastly, the first version of the design did not yet have the possibility to save images and upload/edit previously created images, but this was one of the requirements. For the final design, these functionalities have been implemented.

To save images, firstly, directories are made specifically to store these images if they don't yet exist. The general directory is called 'Saved images' in which are 2 more directories called 'Preset images' and 'RGB images' for images created with the preset pages from figure 16 and RGB pages from figure 17 respectively. A saved image is put into either of these directories depending on the type of image, along with its preset and image cube (if they exist). The preset is saved as a json-file containing a single dictionary entry and the image cube is saved as another directory with a png-images for each wavelength.

Uploading images is done by asking the user to select a file from the 'Saved images' directory. Depending on the page from which this is done, a different folder is opened. The user can only select the '.png' file found in the initial folder, selecting any other file than a '.png' returns in a black screen (but no error of some sorts) and the image cube '.png' files are just meant as building blocks for the image displayed on-screen.

5.2 Final design

5.2.1 Overview

An updated version of the sequence diagram shown in figure 10 is shown below in figure 15. The diagram misses some functionalities like switching to the RGB page and saving/uploading images, but is again a good, very simplified visualization of the most important functions. A full description of all functions of the program can be found in section 8.1.

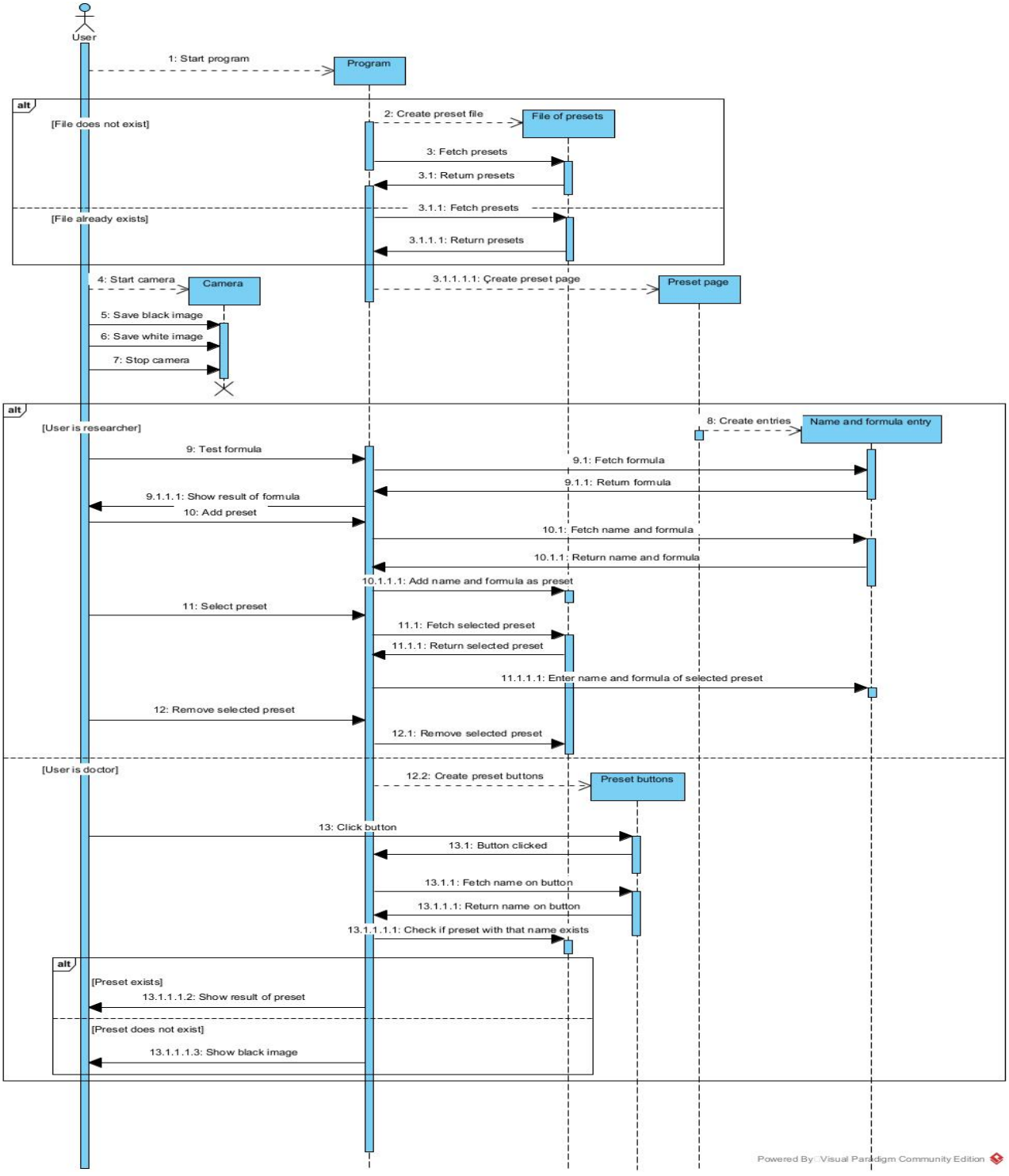
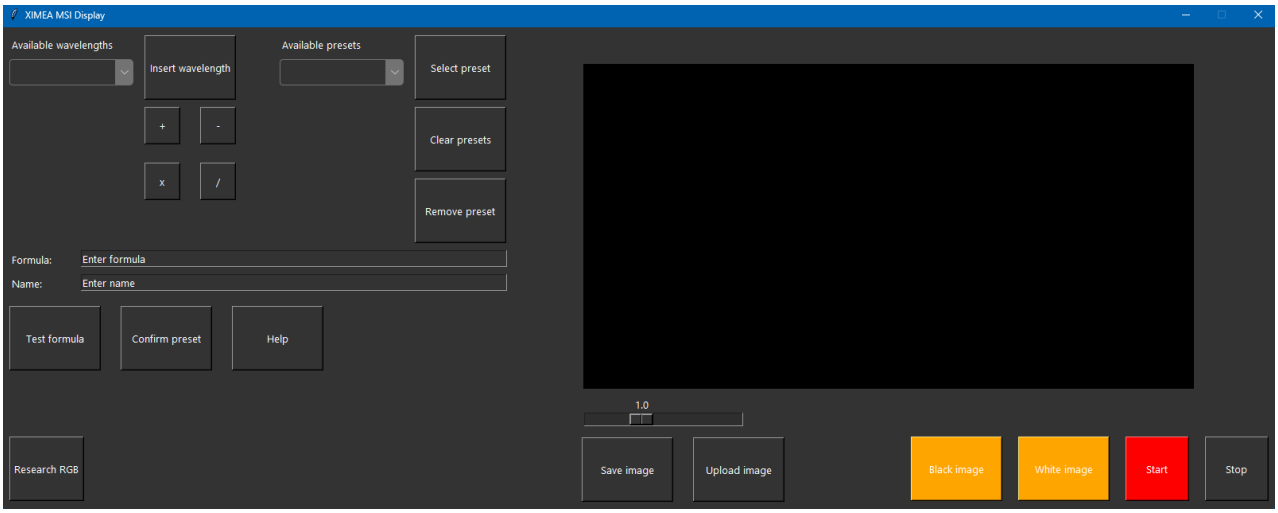


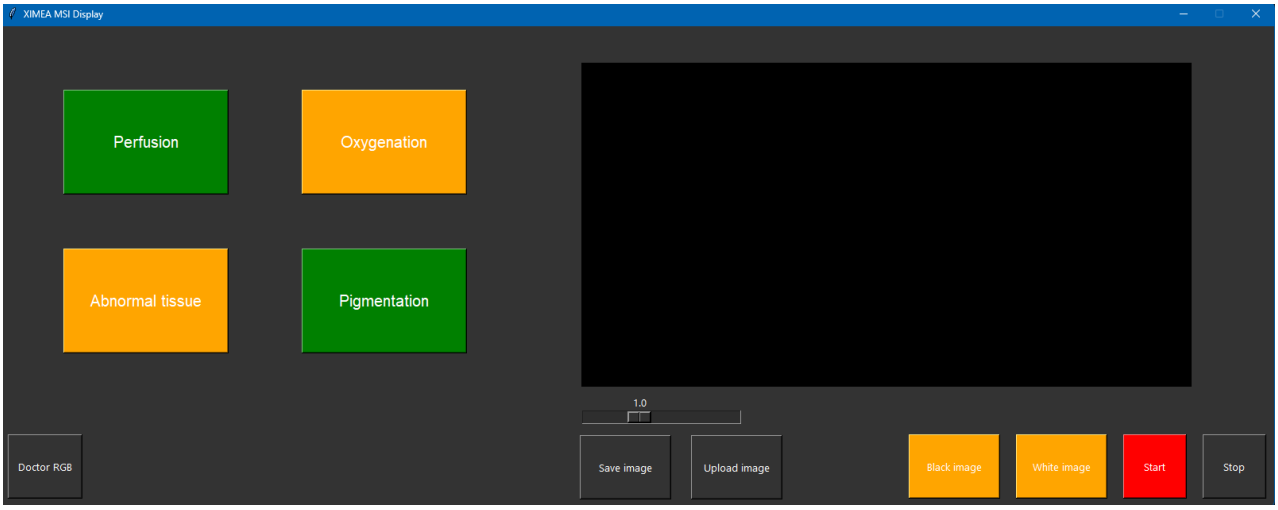
Figure 15: Sequence diagram of the most important functions available on the preset page of the final version of the design. For the RGB page, the sequence diagram is almost the same, except that presets are a combinations of channels and not just a formula.

It can be seen that some of the issues from section 5.1.3 are now solved. The preset list is now a file of presets, which does not get removed when the program is closed. When opening the program again, it is first checked whether the file already exists and if so, it gets all presets saved within that file. Furthermore, doctors no longer have access to the entire list of presets. They just have buttons which may or may not have presets assigned to them based on their name. Clicking the button will either show the result of the assigned, or a black screen if no preset is assigned.
A further description of the final product is given below.

Preset pages



(a)



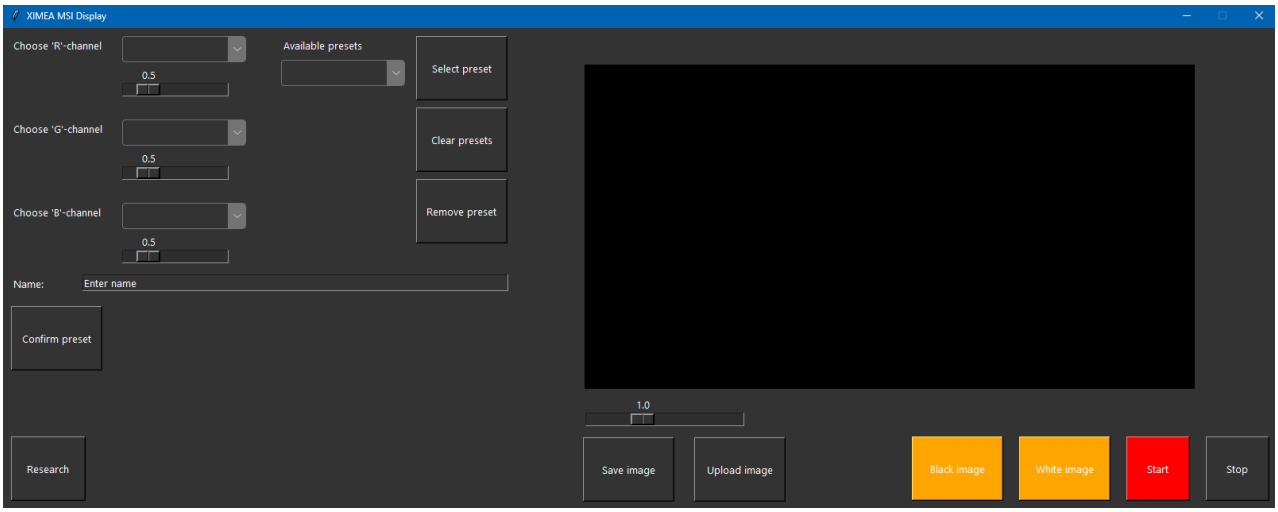
(b)

Figure 16: Preset pages for a researcher (a) and a doctor (b)

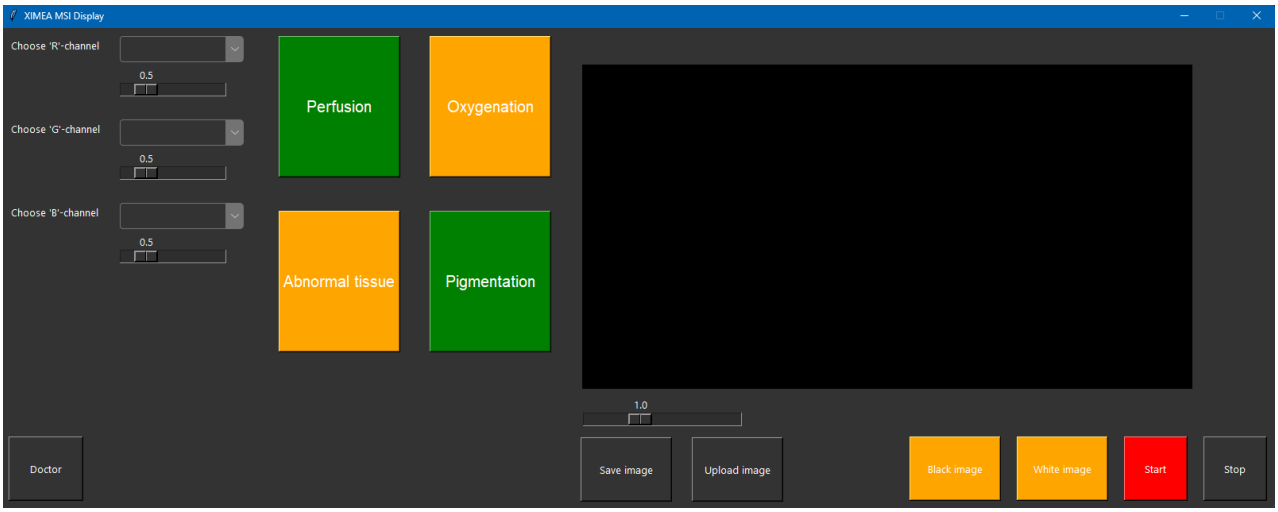
Figure 16 shows the interfaces a user gets to see depending on their role. A researcher (figure 16a) has all necessary functionality to create and remove presets. They can create presets by adding wavelengths and operations into the formula entry. Wavelengths can be selected from the list of available wavelengths in the top left of the screen and then clicking 'Insert wavelength'. This automatically puts the representation for that wavelength into the formula such that the program can access it. Clicking any of the operators underneath the 'Insert wavelength' button automatically inserts the Numpy (4.2) representation of that operator. When a formula is inserted in the entry, the researcher can test the formula by clicking 'Test formula'. This shows the result of the currently entered formula on screen, but does not save the preset yet. If the result is good, the researcher can choose to save the preset by entering a name for the preset and then clicking 'Confirm

preset'. The preset then gets added to the list of available presets. From that list, the researcher can also select a preset that is already created by choosing that preset from the list and clicking 'Select preset'. This shows the result of that preset on screen and inserts the formula and name of that presets into the formula- and name entry respectively. The currently selected preset can also be removed by clicking 'Remove preset. And if the researcher wants to reset all presets they can press 'Clear presets', which will remove any created preset (a second confirmation has to be given in order to proceed with the action, such that this will not happen accidentally).
The preset page for the doctor (figure 16b) is a lot simpler than the one for the researcher. A doctor can only choose between four presets assigned to the four buttons displayed on screen. When a button is green, this means that it has a preset assigned to it (as described in section 5.1.3).
Both pages have a button to go to the RGB-page of their respective role (shown in figure 17).

RGB pages



(a)



(b)

Figure 17: RGB pages for a researcher (a) and a doctor (b)

The RGB pages have a lot of the same functionalities as the standard preset pages. Except presets are now more complicated. Instead of just adding a formula to an entry, the researcher can select any of the wavelengths (or already created presets) for any of the R-, G- and B-channels including a relative intensity determined by the scale underneath each channel. When a valid value is entered for each channel, the result of that combination is automatically displayed on screen, so there is no more need for a 'Test formula'-button. The RGB page for the doctor again has four buttons which can have presets assigned to them by name.

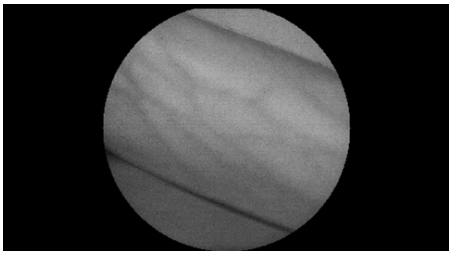
General functionality

It can be seen in figures 16 and 17 that all pages have a scale underneath the image display section to up- or downscale the intensity of the image to finetune the result. Saving and uploading images should be possible for all users, so every page has these functionalities as well. They also all have buttons to start and stop the camera and to save a black and white reference image such that the camera can be started and calibrated on any page. These buttons will also turn green once that action is performed. The start-button starts off red to indicate that starting the camera is necessary to run the program, while the program runs just fine without the black- and white images (also when a black- and white correction is implemented). Using these colors as indicators to the state of the buttons makes the application more user-friendly.

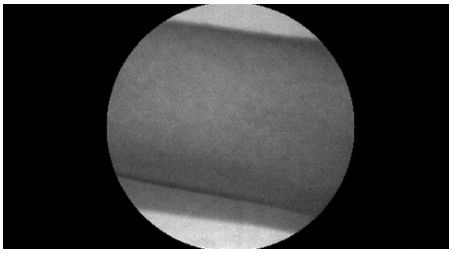
5.2.2 Resulting images

To show the results of image processing and demonstrate the possibilities of the program, images (seen in figure 18) were made to compare different presets with each other.

(a) Regular RGB-image of an arm taken by a mobile phone

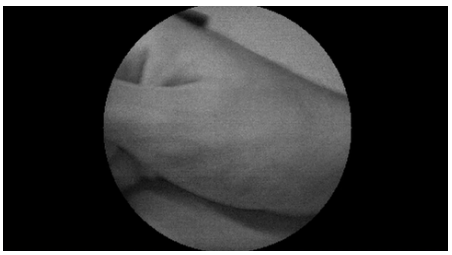


(b) Perfusion image of the same arm as 'a'

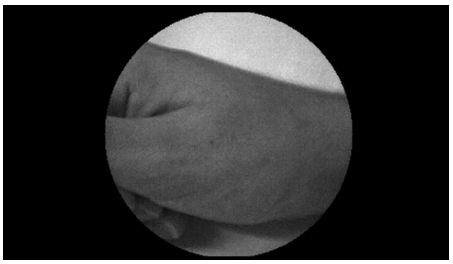


(c) Pigmentation image of the arm

(d) Regular RGB-image of the hand (with a birthmark) taken by a mobile phone



(e) Perfusion image of the hand in 'd'

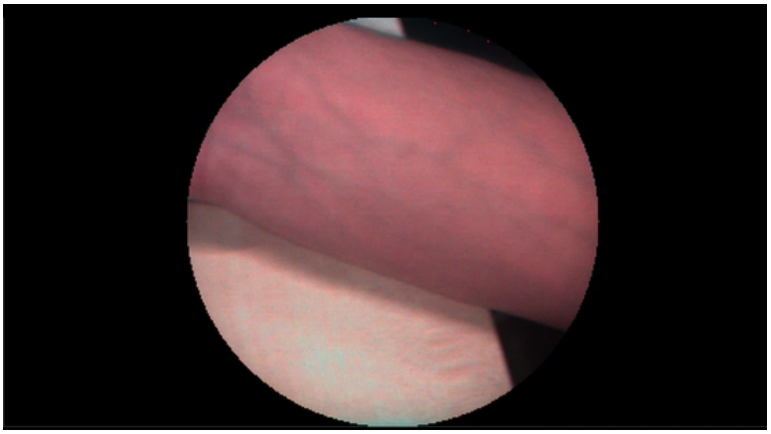


(f) Pigmentation image of the hand in 'd'

Figure 18: Images made with the program with a perfusion preset (b and e) and a pigmentation preset (c and f) compared to RGB-images of the same part of the body (a and d).
The perfusion preset is of the form of equation 1, with weighted subtractions between wavelengths 618nm and 490nm, 616nm and 500nm, 613nm and 477nm.
The pigmentation preset has the same form but with weighted subtractions between wavelengths 553nm and 477nm, 563nm and 490nm.

The images above (figures 18b, 18c, 18e, 18f) are made in the page shown in figure 16a. The images below (figure 19) showcase the possibilities of presets made with the page from figure 17.

(a) Regular 'RGB'-image of an arm taken by a mobile phone



(b) Image of the arm in 'a' with a perfusion preset created with the RGB page of figure 17a

Figure 19: Comparison between a regular RGB-image (a) and an RGB image with enhanced visualization of perfusion (b).
For (b), the perfusion preset, used for figure 18b, is set as the 'R'-channel, 563nm is set as the 'G'-channel and 549nm is set as the 'B'-channel.

6 Discussion

6.1 Requirements

Part of the aim of this study was to create a user-friendly application for medical purposes. The final product meets all of the requirements set in 4.1regarding the software design. Using the program, a researcher has the ability to create a formula, test it and then possibly save the formula along with a name as a preset. The researcher can select presets, remove presets and clear all presets. They can do this for both regular presets and presets in the form of RGB-images. The doctor has buttons to select presets that were created by a researcher (if they are assigned to the button as explained in section 5.2). Both users have the ability to start and stop the camera, to save a black- and white image and to save and upload images. The image is shown on the right-hand side of the screen whereas the buttons users can interact with are on the left-hand side.
The program also has some additional features to make the application more user-friendly. For example, buttons get a color depending on the state of their functions and the list of presets only shows presets of the current page.
Furthermore, the program behaves as intended and unexpected behaviour is handled appropriately by the system catching any possible error.

6.2 Images

Another requirement was to be able to create contrasting images using the program. The results from presets created with the application in the form of equations 1 and 2 for perfusion and pigmentation respectively, show the desired chromophores as intended. Figures 18a - 18c show that the perfusion presets significantly enhances the blood vessels (already slightly visible in figure 18a) in the arm, while they are not at all visible in figure 18c as this preset shows pigmentation. This is confirmed by images 18d - 18f. A small birthmark (representing a local high amount of melanin) is visible in figure 18d and more so in figure 18f (the tiny black dot at the base of the thumb), while barely visible in figure 18e.
Image enhancement is achieved as contrast is improved in the created images compared to the standard RGB-image. The same can be seen in figure 19 where perfusion in the arm is significantly enhanced compared to the reference image made with a mobile phone.

6.3 Shortcomings and possibilities for future improvements

First of all, like said in section 4.3, no black and white correction is applied at the moment. Although the results (section 5.2.2) are of sufficient quality for the intended purpose of this study, a black and white correction could improve the quality, removing any dark noise and vignetting from the image.
Secondly, it was mentioned in section 3.2.2 that not using a uniform lightsource could cause imperfections. The method of normalization used in 5.2 does account for the non-uniform lightsource, but it does not account for the fact that other lightsources will have different emission spectra. The emission spectrum of the lightsource used is not determined and therefore nothing can be said about correcting the normalization for other surrounding light. A possible solution could be to recalculate the intensities per wavelength using a spectrally uniform LED-ring. This will give absolute intensities per wavelength that are independent of the emission spectrum of the surrounding light. This way, you could divide the resulting gain by the relative emission per wavelength of the used lightsource and apply normalization this way. Or you could simply perform any measurement under the illumination of a uniform lightsource. Thirdly, the colors of presets created with the RGB-page seem to be off when comparing figure 14a to figure 12a. This could partly be because of a sub-optimal choice of wavelength. But more likely is that the color balance is wrong. For this, a color correction matrix could be calculated, a method for which is proposed by [20]. A future version of this program could include such a color correction matrix to make the results created with the RGB-page more accurate.
Lastly, the program only shows the correct wavelengths for the camera used in this research. Using this program for any other camera with a different array than shown in figure 4a will require accurate inspection to find out what wavelength corresponds to what index in the array, because the names of the wavelengths in the list of available wavelengths on page 16a won't change depending on the connected camera. A way to read

out the peak wavelengths of the filter used is desired in order to make the application less type-specific. This would also enable usage of the program for cameras with an array with a layout of a size other than 4x4, which currently would cause the program to crash.

7 Conclusion

A program was made for a multispectral camera for user-friendly medical applications. The study shows the steps of the process regarding both software design and image processing. Feasability of the program was analyzed. The program meets all requirements and has a user-friendly interface depending on the expertise of the user. Some additional features are added to expand functionality and convenience. Images made using a preset created with the program show enhanced contrast compared to reference images, demonstrating its possibilities. Aspects of image processing need to be improved upon, including the black and white correction and color correction. The program should also undergo changes to allow usage of the program with other multispectral cameras and increase the support base.

References

[1] Goel M, Whitmire E, Mariakakis A, Saponas TS, Joshi N, Morris D, et al. HyperCam: hyperspectral imaging for ubiquitous computing applications. In: Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing. UbiComp '15. New York, NY, USA: Association for Computing Machinery; 2015. p. 145–156. Available from: https://doi.org/10.1145/2750858.2804282. doi:10.1145/2750858.2804282.

[2] Authors W. Detecting skin cancer using hyperspectral images - Advanced Science News — advancedsciencenews.com;. [Accessed 08-07-2024]. https://www.advancedsciencenews.com/detecting-skin-cancer-using-hyperspectral-images/.

[3] Bauer JR, Bruins AA, Hardeberg JY, Verdaasdonk RM. A Spectral Filter Array Camera for Clinical Monitoring and Diagnosis: Proof of Concept for Skin Oxygenation Imaging. J Imaging. 2019 Jul;5(8).

[4] Chung KL, Chen HY, Hsieh TL, Chen YB. Compression for Bayer CFA images: Review and performance comparison. Sensors (Basel). 2022 Oct;22(21):8362.

[5] Ji Y, Kwak Y, Park SM, Kim YL. Compressive recovery of smartphone RGB spectral sensitivity functions. Opt Express. 2021 Apr;29(8):11947-61.

[6] Abdel-Rahman F, Okeremgbo B, Alhamadah F, Jamadar S, Anthony K, Saleh M. Caenorhabditis elegans as a model to study the impact of exposure to light emitting diode (LED) domestic lighting. Journal of environmental science and health Part A, Toxic/hazardous substances environmental engineering. 2017 01;52:1-7. doi:10.1080/10934529.2016.1270676.

[7] Jiang J, Zheng H, Ji X, Cheng T, Tian Y, Zhu Y, et al. Analysis and Evaluation of the Image Preprocessing Process of a Six-Band Multispectral Camera Mounted on an Unmanned Aerial Vehicle for Winter Wheat Monitoring. Sensors. 2019;19(3). Available from: https://www.mdpi.com/1424-8220/19/3/747. doi:10.3390/s19030747.

[8] How our system works | Phovia — phovia.vetoquinol.ca;. [Accessed 07-07-2024]. https://phovia.vetoquinol.ca/en/how-our-system-works.

[9] Setchfield K, Gorman A, Simpson AHRW, Somekh MG, Wright AJ. Relevance and utility of the in-vivo and ex-vivo optical properties of the skin reported in the literature: a review [Invited]. Biomedical Optics Express. 2023 Jun;14(7):3555. Available from: http://dx.doi.org/10.1364/BOE.493588. doi:10.1364/boe.493588.

[10] Prahl S; 1999. Available from: https://omlc.org/spectra/hemoglobin/.

[11] Riley PA. Melanin. The International Journal of Biochemistry Cell Biology. 1997;29(11):1235-9. Available from: https://www.sciencedirect.com/science/article/pii/S1357272597000137. doi:https://doi.org/10.1016/S1357-2725(97)00013-7.

[12] Anderson PG, Kainerstorfer JM, Sassaroli A, Krishnamurthy N, Homer MJ, Graham RA, et al. Broadband optical mammography: chromophore concentration and hemoglobin saturation contrast in breast cancer. PLoS One. 2015 Mar;10(3):e0117322.

[13] Lugano R, Ramachandran M, Dimberg A. Tumor angiogenesis: causes, consequences, challenges and opportunities. Cell Mol Life Sci. 2020 May;77(9):1745-70.

[14] Heistein JB, Acharya U, Mukkamalla SKR. Malignant Melanoma. In: StatPearls. Treasure Island (FL): StatPearls Publishing; 2024. .

[15] He Q, Wang RK. Analysis of skin morphological features and real-time monitoring using snapshot hyperspectral imaging. Biomed Opt Express. 2019 Oct;10(11):5625-38.

[16] ;. Available from: https://www.ximea.com/en/products/hyperspectral-cameras-based-on-usb3-xispec/mq022hg-im-sm4x4-vis.

[17] McCann J. In: Luo MR, editor. Retinex Theory. New York, NY: Springer New York; 2016. p. 1118-25. Available from: https://doi.org/10.1007/978-1-4419-8071-$7_2$60.*doi* : 10.1007/978 − 1 − 4419 − 8071 − $7_2$60.

[18] Cucci C, Delaney JK, Picollo M. Reflectance Hyperspectral Imaging for Investigation of Works of Art: Old Master Paintings and Illuminated Manuscripts. Acc Chem Res. 2016 Sep;49(10):2070-9.

[19] Dissanayake D, Herath V, Godaliyadda GMR, Ekanayake MP, Bandara C, Prabhath GW. Design of a Multispectral Imaging System for Industrial Applications; 2019. doi:10.13140/RG.2.2.36834.68800.

[20] Chen Z, Wang X, Liang R. RGB-NIR multispectral camera. Opt Express. 2014 Mar;22(5):4985-94. Available from: https://opg.optica.org/oe/abstract.cfm?URI=oe-22-5-4985. doi:10.1364/OE.22.004985.

8 Appendix

8.1 Code

__init()__

This block of code is meant for initializing variables to be used later in the script. It creates the window for the program and creates the first page upon opening (for either researcher or doctor depending on the user). The function also checks whether a preset file already exists and creates one if it doesn't (as can be seen at the top of figure 15). Explanations for the variable names and their functionality within the program are given as comments in the code.

```
# Ximea module
from ximea import xiapi

# Tkinter imports
import tkinter as tk
from tkinter import ttk
from tkinter import messagebox as mb
from tkinter import simpledialog
from tkinter.filedialog import askopenfilename

# Arrays and calculation
import numpy as np

# Computer vision and image processing
import cv2
from PIL import Image, ImageTk

# Json file
import json

# Pop-up messages
import warnings

# Command prompt arguments
import sys

# Functions for files and directories
import os

# Class containing all the functions for the app
class XIMEA_DisplayApp():
    # Initializes all necessary values
    def __init__(self, window, user):
        # Create a window for the app
        self.user = user
        self.window = window
        self.window.title("XIMEA MSI Display")
        # Create the theme
        root.tk.call('source', 'Azure/azure.tcl')
        root.tk.call('set_theme', 'dark')
        # Create the canvas
        self.width = 1400
        self.height = 500
        self.canvas = tk.Canvas(self.window, width=self.width,height=self.height)
        self.canvas.pack(expand=True, padx=100)

        # Keeps track of what page is currently in use
        # Page 1: Preset page researcher
        # Page 2: Preset page doctor
        # Page 3: RGB page researcher
        # Page 4: RGB page doctor
```

```
52          self.current_page = None
53
54          # Checks if black and white images have been saved
55          self.saved_black = False
56          self.saved_white = False
57
58          # Sets the initial mode for the type of image on screen (LIVE: image from ...
                camera or IM: uploaded image)
59          self.mode = "LIVE"
60
61          # Initializes the color of certain buttons
62          self.white_img_color = "ORANGE"
63          self.black_img_color = "ORANGE"
64          self.start_button_color = "RED"
65
66          # Standard formula for the image
67          self.black = np.zeros((272,512))
68
69          # List of all wavelengths
70          self.wavelengths = {'490nm':"self.n490", '500nm':"self.n500", ...
                '477nm':"self.n477", '478nm':"self.n478",
71                              '577nm':"self.n577", '591nm':"self.n591", ...
                                    '563nm':"self.n563", '553nm':"self.n553",
72                              '618nm':"self.n618", '616nm':"self.n616", ...
                                    '613nm':"self.n613", '600nm':"self.n600",
73                              '538nm':"self.n538", '549nm':"self.n549", ...
                                    '523nm':"self.n523", '511nm':"self.n511"}
74
75          # Opens the json file and puts all presets in a dictionary
76          JFile = 'presets.json'
77          try:
78              with open(JFile, 'r') as fp:
79                  self.presets = dict(json.load(fp))
80                  fp.close()
81          except:
82              with open(JFile, 'w') as fp:
83                  json.dump({}, fp)
84                  fp.close()
85              with open(JFile, 'r') as fp:
86                  self.presets = dict(json.load(fp))
87                  fp.close()
88
89          # Starts the program with page according to user
90          if self.user == "Researcher" or self.user == "Tester":
91              self.page1()
92          elif self.user == "Doctor":
93              self.page2()
94
95          # Create instance for camera
96          self.cam = xiapi.Camera()
97
98          # Create instances for images
99          self.img = xiapi.Image()
100
101         self.img1 = xiapi.Image()
102         self.img2 = xiapi.Image()
103         self.img3 = xiapi.Image()
104
105         self.black_img = xiapi.Image()
106         self.white_img = xiapi.Image()
107
108         # Update the image
109         self.update()
```

Pages

The following functions are meant for creating the pages and all of their widgets. Calling any of the functions destroys all widgets currently on-screen and creates all widgets belonging to that page (buttons, text entries, lists, etc.). The names of the widgets describe their corresponding function. All widgets are made using the tkinter module mentioned in section 4.2.

```
    # Research page
    def page1(self) -> None:
        self.current_page = 'Page 1'
        self.set_current_formula("self.black")
        # Destroy all widgets from the previous page
        for widget in self.window.winfo_children():
            if widget is not self.canvas:
                widget.destroy()

        # Miscellaneous buttons
        self.stop_button = tk.Button(self.window, text="Stop", ...
            command=self.stop_camera, height=4, width=10)
        self.stop_button.pack(side = 'right', padx=10, pady=10)
        self.start_button = tk.Button(self.window, text="Start", ...
            command=self.setup, height=4, width=10, bg=self.start_button_color)
        self.start_button.pack(side = 'right', padx=10, pady=10)
        self.save_white_image_button = tk.Button(self.window, text="White image", ...
            command=self.save_white_image, height=4, width=15, ...
            bg=self.white_img_color)
        self.save_white_image_button.pack(side="right",padx=10,pady=10)
        self.save_black_image_button = tk.Button(self.window, text="Black image", ...
            command=self.save_black_image, height=4, width=15, ...
            bg=self.white_img_color)
        self.save_black_image_button.pack(side="right",padx=10,pady=10)
        if self.user == "Tester":
            self.ROI_button = tk.Button(self.window, text="Select ROI", ...
                command=self.get_ROI, height = 4, width=15)
            self.ROI_button.place(x=1010,y=515)
        self.save_image_button = tk.Button(self.window, text="Save image", ...
            command= self.save_image, height=4, width=15)
        self.save_image_button.place(x=730,y=515)
        self.upload_image_button = tk.Button(self.window, text="Upload image", ...
            command= self.upload_image, height=4, width=15)
        self.upload_image_button.place(x=870,y=515)
        if self.user == "Tester":
            self.page2_button = tk.Button(self.window, text='Doctor', ...
                command=self.page2, height=4, width=12)
            self.page2_button.pack(side="left", padx=10,pady=10)
        if self.user == "Tester" or self.user == "Researcher":
            self.page3_button = tk.Button(self.window, text="Research RGB", ...
                command=self.page3, height=4,width=12)
            self.page3_button.pack(side="left", padx=10,pady=10)
        if self.user == "Tester":
            self.page4_button = tk.Button(self.window, text="Doctor RGB", ...
                command=self.page4, height=4, width=12)
            self.page4_button.pack(side="left", padx=10,pady=10)

        # Current formula buttons
        self.test_formula_button = tk.Button(self.window, text='Test formula', ...
            command=self.test_formula, height=4, width=15)
        self.test_formula_button.place(x=10,y=350)
        self.confirm_preset_button = tk.Button(self.window, text="Confirm ...
            preset", command=self.add_preset_page1, height=4, width=15)
        self.confirm_preset_button.place(x=150,y=350)

        # Help button
```

```
43          self.help_button = tk.Button(self.window, text="Help", command= ...
                self.show_help, height = 4, width = 15)
44          self.help_button.place(x=290, y=350)
45
46          # Preset buttons
47          self.remove_preset_button = tk.Button(self.window, text="Remove preset", ...
                command=self.remove_preset, height=4, width=15)
48          self.remove_preset_button.place(x=520, y=190)
49          self.clear_preset_button = tk.Button(self.window, text="Clear presets", ...
                command=self.clear_presets, height=4, width=15)
50          self.clear_preset_button.place(x=520, y=100)
51          self.set_current_preset_button = tk.Button(self.window, text="Select ...
                preset", command=self.set_as_current_preset, height=4, width=15)
52          self.set_current_preset_button.place(x=520,y=10)
53
54          # Create formula buttons
55          self.insert_wavelength_button = tk.Button(self.window, text='Insert ...
                wavelength', ...
                command=lambda:self.insert_wavelength(self.wavelength_box.get()), ...
                height=4, width=15)
56          self.insert_wavelength_button.place(x=180, y=10)
57          self.insert_minus_button = tk.Button(self.window, text="-", ...
                command=lambda:self.insert_sign("-"), height=2, width=5)
58          self.insert_minus_button.place(x=250,y=100)
59          self.insert_plus_button = tk.Button(self.window, text="+", ...
                command=lambda:self.insert_sign("+"), height=2, width=5)
60          self.insert_plus_button.place(x=180,y=100)
61          self.insert_divide_button = tk.Button(self.window, text="/", ...
                command=lambda:self.insert_sign("/"), height=2, width=5)
62          self.insert_divide_button.place(x=250,y=170)
63          self.insert_multiply_button = tk.Button(self.window, text="x", ...
                command=lambda:self.insert_sign("x"), height=2, width=5)
64          self.insert_multiply_button.place(x=180,y=170)
65
66          # Text entries
67          formula_label = tk.Label(self.window, text="Formula:")
68          formula_label.place(x=10,y=280)
69          self.formula_entry = tk.Entry(self.window, width=76)
70          self.formula_entry.insert(0, "Enter formula")
71          self.formula_entry.place(x=100, y=280)
72          name_label = tk.Label(self.window, text="Name:")
73          name_label.place(x=10,y=310)
74          self.name_entry = tk.Entry(self.window, width=76)
75          self.name_entry.insert(0, "Enter name")
76          self.name_entry.bind("<FocusIn>", self.delete_name)
77          self.name_entry.place(x=100, y=310)
78
79          # Comboboxes for wavelengths and presets
80          wavelength_label = tk.Label(self.window, text="Available wavelengths")
81          wavelength_label.place(x=10, y=10)
82          self.wavelength_box = ttk.Combobox(self.window, ...
                values=list(self.wavelengths.keys()))
83          self.wavelength_box.place(x=10, y=40)
84          preset_label = tk.Label(self.window, text="Available presets")
85          preset_label.place(x=350, y=10)
86          self.preset_box = ttk.Combobox(self.window, ...
                values=list(self.get_preset_without_rgb().keys()))
87          self.preset_box.place(x=350, y=40)
88
89          # Intensity scale
90          self.intensity_scale = tk.Scale(self.window, from_=0, to=3, ...
                resolution=0.1, orient="horizontal", length=200)
91          self.intensity_scale.place(x=730, y=460)
92          self.intensity_scale.set(1)
93
```

```
94      # Doctor page
95      def page2(self) -> None:
96          self.current_page = 'Page 2'
97          self.set_current_formula("self.black")
98          # Destroy all widgets from previous page
99          for widget in self.window.winfo_children():
100             if widget is not self.canvas:
101                 widget.destroy()
102
103         # Miscellaneous buttons
104         self.stop_button = tk.Button(self.window, text="Stop", ...
                command=self.stop_camera, height=4, width=10)
105         self.stop_button.pack(side = 'right', padx=10, pady=10)
106         self.start_button = tk.Button(self.window, text="Start", ...
                command=self.setup, height=4, width=10, bg=self.start_button_color)
107         self.start_button.pack(side = 'right', padx=10, pady=10)
108         self.save_white_image_button = tk.Button(self.window, text="White image", ...
                command=self.save_white_image, height=4, width=15, ...
                bg=self.white_img_color)
109         self.save_white_image_button.pack(side="right",padx=10,pady=10)
110         self.save_black_image_button = tk.Button(self.window, text="Black image", ...
                command=self.save_black_image, height=4, width=15, ...
                bg=self.black_img_color)
111         self.save_black_image_button.pack(side="right",padx=10,pady=10)
112         self.save_image_button = tk.Button(self.window, text="Save image", ...
                command= self.save_image, height=4, width=15)
113         self.save_image_button.place(x=730,y=515)
114         self.upload_image_button = tk.Button(self.window, text="Upload image", ...
                command= self.upload_image, height=4, width=15)
115         self.upload_image_button.place(x=870,y=515)
116         if self.user == "Tester":
117             self.page1_button = tk.Button(self.window, text="Research", ...
                    command=self.page1, height=4, width=12)
118             self.page1_button.pack(side="left", padx=10,pady=10)
119             self.page3_button = tk.Button(self.window, text="Research RGB", ...
                    command=self.page3, height=4,width=12)
120             self.page3_button.pack(side="left", padx=10,pady=10)
121         if self.user == "Tester" or self.user == "Doctor":
122             self.page4_button = tk.Button(self.window, text="Doctor RGB", ...
                    command=self.page4, height=4, width=12)
123             self.page4_button.pack(side="left", padx=10,pady=10)
124
125         # Preset buttons
126         self.perfusion_button = tk.Button(self.window, text="Perfusion", command= ...
                lambda:self.set_button_preset("Perfusion"), height=5,width=18, ...
                font=('Arial', 15), bg=self.set_button_color("Perfusion"))
127         self.perfusion_button.place(x=80,y=80)
128         self.oxygenation_button = tk.Button(self.window, text="Oxygenation", ...
                command= lambda:self.set_button_preset("Oxygenation"), ...
                height=5,width=18, font=('Arial', 15), ...
                bg=self.set_button_color("Oxygenation"))
129         self.oxygenation_button.place(x=380,y=80)
130         self.carcinoma_button = tk.Button(self.window, text="Abnormal tissue", ...
                command= lambda:self.set_button_preset("Abnormal tissue"), ...
                height=5,width=18, font=('Arial', 15), ...
                bg=self.set_button_color("Abnormal tissue"))
131         self.carcinoma_button.place(x=80,y=280)
132         self.pigment_button = tk.Button(self.window, text="Pigmentation", ...
                command= lambda:self.set_button_preset("Pigmentation"), ...
                height=5,width=18, font=('Arial', 15), ...
                bg=self.set_button_color("Pigmentation"))
133         self.pigment_button.place(x=380,y=280)
134
135         # Intensity scale
```

```
136         self.intensity_scale = tk.Scale(self.window, from_=0, to=3, ...
                resolution=0.1, orient="horizontal", length=200)
137         self.intensity_scale.place(x=730, y=460)
138         self.intensity_scale.set(1)
139 
140     # Research RGB page
141     def page3(self) -> None:
142         self.current_page = "Page 3"
143         self.set_current_formula("self.black")
144         for widget in self.window.winfo_children():
145             if widget != self.canvas:
146                 widget.destroy()
147         self.rgb_presets = self.wavelengths.copy()
148         self.copied_values = self.presets.copy()
149         self.copied_values.pop("White")
150         self.copied_values.pop("Black")
151         for key in self.copied_values.keys():
152             self.rgb_presets.update({key:self.presets.get(key)})
153 
154         # Comboboxes and scales for r, g and b value
155         r_label = tk.Label(self.window, text="Choose 'R'-channel")
156         r_label.place(x=10,y=10)
157         self.r_value = ttk.Combobox(self.window, values= ...
                list(self.get_preset_without_rgb()) + list(self.wavelengths))
158         self.r_value.place(x=150,y=10)
159         self.r_scale = tk.Scale(self.window, from_=0, to=3, resolution=0.1, ...
                orient="horizontal", length=134)
160         self.r_scale.place(x=147, y=45)
161         self.r_scale.set(0.5)
162         g_label = tk.Label(self.window, text="Choose 'G'-channel")
163         g_label.place(x=10,y=115)
164         self.g_value = ttk.Combobox(self.window, values= ...
                list(self.get_preset_without_rgb()) + list(self.wavelengths))
165         self.g_value.place(x=150,y=115)
166         self.g_scale = tk.Scale(self.window, from_=0, to=3, resolution=0.1, ...
                orient="horizontal", length=134)
167         self.g_scale.place(x=147, y=150)
168         self.g_scale.set(0.5)
169         B_label = tk.Label(self.window, text="Choose 'B'-channel")
170         B_label.place(x=10,y=220)
171         self.b_value = ttk.Combobox(self.window, values= ...
                list(self.get_preset_without_rgb()) + list(self.wavelengths))
172         self.b_value.place(x=150,y=220)
173         self.b_scale = tk.Scale(self.window, from_=0, to=3, resolution=0.1, ...
                orient="horizontal", length=134)
174         self.b_scale.place(x=147, y=255)
175         self.b_scale.set(0.5)
176 
177         # Preset widgets
178         self.confirm_preset_button = tk.Button(self.window, text="Confirm ...
                preset", command=self.add_preset_page3, height=4, width=15)
179         self.confirm_preset_button.place(x=10,y=350)
180         name_label = tk.Label(self.window, text="Name:")
181         name_label.place(x=10,y=310)
182         self.name_entry = tk.Entry(self.window, width=76)
183         self.name_entry.insert(0, "Enter name")
184         self.name_entry.bind("<FocusIn>", self.delete_name)
185         self.name_entry.place(x=100, y=310)
186         preset_label = tk.Label(self.window, text="Available presets")
187         preset_label.place(x=350, y=10)
188         self.preset_box = ttk.Combobox(self.window, ...
                values=list(self.get_preset_rgb()))
189         self.preset_box.place(x=350, y=40)
190         self.set_current_preset_button = tk.Button(self.window, text="Select ...
                preset", command=self.set_as_current_preset, height=4, width=15)
```

```
191         self.set_current_preset_button.place(x=520,y=10)
192         self.remove_preset_button = tk.Button(self.window, text="Remove preset", ...
                command=self.remove_preset, height=4, width=15)
193         self.remove_preset_button.place(x=520, y=190)
194         self.clear_preset_button = tk.Button(self.window, text="Clear presets", ...
                command=self.clear_presets, height=4, width=15)
195         self.clear_preset_button.place(x=520, y=100)
196
197         # Miscellaneous buttons
198         self.stop_button = tk.Button(self.window, text="Stop", ...
                command=self.stop_camera, height=4, width=10)
199         self.stop_button.pack(side = 'right', padx=10, pady=10)
200         self.start_button = tk.Button(self.window, text="Start", ...
                command=self.setup, height=4, width=10, bg=self.start_button_color)
201         self.start_button.pack(side = 'right', padx=10, pady=10)
202         self.save_white_image_button = tk.Button(self.window, text="White image", ...
                command=self.save_white_image, height=4, width=15, ...
                bg=self.white_img_color)
203         self.save_white_image_button.pack(side="right",padx=10,pady=10)
204         self.save_black_image_button = tk.Button(self.window, text="Black image", ...
                command=self.save_black_image, height=4, width=15, ...
                bg=self.white_img_color)
205         self.save_black_image_button.pack(side="right",padx=10,pady=10)
206         if self.user == "Tester":
207             self.ROI_button = tk.Button(self.window, text="Select ROI", ...
                    command=self.get_ROI, height = 4, width=15)
208             self.ROI_button.place(x=1010,y=515)
209         self.save_image_button = tk.Button(self.window, text="Save image", ...
                command= self.save_image, height=4, width=15)
210         self.save_image_button.place(x=730,y=515)
211         self.upload_image_button = tk.Button(self.window, text="Upload image", ...
                command= self.upload_image, height=4, width=15)
212         self.upload_image_button.place(x=870,y=515)
213         if self.user == "Tester" or self.user == "Researcher":
214             self.page1_button = tk.Button(self.window, text="Research", ...
                    command=self.page1, height=4, width=12)
215             self.page1_button.pack(side="left", padx=10,pady=10)
216         if self.user == "Tester":
217             self.page2_button = tk.Button(self.window, text='Doctor', ...
                    command=self.page2, height=4, width=12)
218             self.page2_button.pack(side="left", padx=10,pady=10)
219             self.page4_button = tk.Button(self.window, text="Doctor RGB", ...
                    command=self.page4, height=4, width=12)
220             self.page4_button.pack(side="left", padx=10,pady=10)
221
222         # Intensity scale
223         self.intensity_scale = tk.Scale(self.window, from_=0, to=3, ...
                resolution=0.1, orient="horizontal", length=200)
224         self.intensity_scale.place(x=730, y=460)
225         self.intensity_scale.set(1)
226
227     # Doctor RGB page
228     def page4(self) -> None:
229         self.current_page = "Page 4"
230         self.set_current_formula("self.black")
231         for widget in self.window.winfo_children():
232             if widget != self.canvas:
233                 widget.destroy()
234         self.rgb_presets = self.wavelengths.copy()
235         self.copied_values = self.presets.copy()
236         self.copied_values.pop("White")
237         self.copied_values.pop("Black")
238         for key in self.copied_values.keys():
239             self.rgb_presets.update({key:self.presets.get(key)})
240
```

```
241         # Comboboxes and scales for r, g and b value
242         r_label = tk.Label(self.window, text="Choose 'R'-channel")
243         r_label.place(x=10,y=10)
244         self.r_value = ttk.Combobox(self.window, values= ...
                list(self.get_preset_without_rgb()) + list(self.wavelengths))
245         self.r_value.place(x=150,y=10)
246         self.r_scale = tk.Scale(self.window, from_=0, to=3, resolution=0.1, ...
                orient="horizontal", length=134)
247         self.r_scale.place(x=147, y=45)
248         self.r_scale.set(0.5)
249         g_label = tk.Label(self.window, text="Choose 'G'-channel")
250         g_label.place(x=10,y=115)
251         self.g_value = ttk.Combobox(self.window, values= ...
                list(self.get_preset_without_rgb()) + list(self.wavelengths))
252         self.g_value.place(x=150,y=115)
253         self.g_scale = tk.Scale(self.window, from_=0, to=3, resolution=0.1, ...
                orient="horizontal", length=134)
254         self.g_scale.place(x=147, y=150)
255         self.g_scale.set(0.5)
256         B_label = tk.Label(self.window, text="Choose 'B'-channel")
257         B_label.place(x=10,y=220)
258         self.b_value = ttk.Combobox(self.window, values= ...
                list(self.get_preset_without_rgb()) + list(self.wavelengths))
259         self.b_value.place(x=150,y=220)
260         self.b_scale = tk.Scale(self.window, from_=0, to=3, resolution=0.1, ...
                orient="horizontal", length=134)
261         self.b_scale.place(x=147, y=255)
262         self.b_scale.set(0.5)
263
264         # Preset buttons
265         self.perfusion_button = tk.Button(self.window, text="Perfusion", command= ...
                lambda:self.set_button_preset("Perfusion_RGB"), height=7,width=13, ...
                font=('Arial', 15), bg=self.set_button_color("Perfusion_RGB"))
266         self.perfusion_button.place(x=350,y=10)
267         self.oxygenation_button = tk.Button(self.window, text="Oxygenation", ...
                command= lambda:self.set_button_preset("Oxygenation_RGB"), ...
                height=7,width=13, font=('Arial', 15), ...
                bg=self.set_button_color("Oxygenation_RGB"))
268         self.oxygenation_button.place(x=540,y=10)
269         self.carcinoma_button = tk.Button(self.window, text="Abnormal tissue", ...
                command= lambda:self.set_button_preset("Abnormal tissue_RGB"), ...
                height=7,width=13, font=('Arial', 15), ...
                bg=self.set_button_color("Abnormal tissue_RGB"))
270         self.carcinoma_button.place(x=350,y=230)
271         self.pigment_button = tk.Button(self.window, text="Pigmentation", ...
                command= lambda:self.set_button_preset("Pigmentation_RGB"), ...
                height=7,width=13, font=('Arial', 15), ...
                bg=self.set_button_color("Pigmentation_RGB"))
272         self.pigment_button.place(x=540,y=230)
273
274         # Miscellaneous buttons
275         self.stop_button = tk.Button(self.window, text="Stop", ...
                command=self.stop_camera, height=4, width=10)
276         self.stop_button.pack(side = 'right', padx=10, pady=10)
277         self.start_button = tk.Button(self.window, text="Start", ...
                command=self.setup, height=4, width=10, bg=self.start_button_color)
278         self.start_button.pack(side = 'right', padx=10, pady=10)
279         self.save_white_image_button = tk.Button(self.window, text="White image", ...
                command=self.save_white_image, height=4, width=15, ...
                bg=self.white_img_color)
280         self.save_white_image_button.pack(side="right",padx=10,pady=10)
281         self.save_black_image_button = tk.Button(self.window, text="Black image", ...
                command=self.save_black_image, height=4, width=15, ...
                bg=self.white_img_color)
282         self.save_black_image_button.pack(side="right",padx=10,pady=10)
```

```
283         self.save_image_button = tk.Button(self.window, text="Save image", ...
                command= self.save_image, height=4, width=15)
284         self.save_image_button.place(x=730,y=515)
285         self.upload_image_button = tk.Button(self.window, text="Upload image", ...
                command= self.upload_image, height=4, width=15)
286         self.upload_image_button.place(x=870,y=515)
287         if self.user == "Tester":
288             self.page1_button = tk.Button(self.window, text="Research", ...
                    command=self.page1, height=4, width=12)
289             self.page1_button.pack(side="left", padx=10,pady=10)
290         if self.user == "Tester" or self.user == "Doctor":
291             self.page2_button = tk.Button(self.window, text='Doctor', ...
                    command=self.page2, height=4, width=12)
292             self.page2_button.pack(side="left", padx=10,pady=10)
293         if self.user == "Tester":
294             self.page3_button = tk.Button(self.window, text="Research RGB", ...
                    command=self.page3, height=4,width=12)
295             self.page3_button.pack(side="left", padx=10,pady=10)
296
297         # Intensity scale
298         self.intensity_scale = tk.Scale(self.window, from_=0, to=3, ...
                resolution=0.1, orient="horizontal", length=200)
299         self.intensity_scale.place(x=730, y=460)
300         self.intensity_scale.set(1)
```

Main loop

The main loop is what executes the program. This is the function that is called every frame. It obtains images and calls the functions for image processing, then it creates the image and displays it on-screen.

```
    # Function that is looped over. This is the fundament of the script
    def update(self) -> None:
        if self.cam.CAM_OPEN and self.mode == "LIVE":
            # Get images from camera as a numpy array
            self.cam.get_image(self.img1)
            self.cam.get_image(self.img2)
            self.cam.get_image(self.img3)
            self.data = self.img1.get_image_data_numpy().astype(np.uint8), ...
                self.img2.get_image_data_numpy().astype(np.uint8), ...
                self.img3.get_image_data_numpy().astype(np.uint8)

            # Black and white correction (not yet implemented)
            self.black_and_white_correction()

            # Extract individual images and put a mask over them
            self.create_image_cube()
            self.mask_images()

        # Update displayed image
        self.set_image(self.create_image())

        # Call function again to maintain loop
        self.window.after(15, self.update)
```

Setup

The setup function is called when the users presses 'Start camera'. It then starts the camera and updates the button color (to do so, it first removes all widgets, then changes the variable of the button color and then recreates all removed buttons, but with updated colors). The same principle is used for the functions of saving the black and white image. These functions are meant for the black-and-white correction. Upon pressing the buttons, they give instructions to the user indicating the steps they need to take to correctly save a black and white reference image.

```
    # Start the camera
    def setup(self) -> None:
        self.mode = "LIVE"
        try:
            # Starting camera and initializing settings
            self.cam.open_device()
            self.cam.set_exposure(50000)
            self.cam.start_acquisition()
            # Change button colors
            self.start_button.destroy()
            self.stop_button.destroy()
            self.save_white_image_button.destroy()
            self.save_black_image_button.destroy()
            self.start_button_color = "GREEN"
            self.stop_button = tk.Button(self.window, text="Stop", ...
                command=self.stop_camera, height=4, width=10)
            self.stop_button.pack(side = 'right', padx=10, pady=10)
            self.start_button = tk.Button(self.window, text="Start", ...
                command=self.setup, height=4, width=10, bg=self.start_button_color)
            self.start_button.pack(side = 'right', padx=10, pady=10)
            self.save_white_image_button = tk.Button(self.window, text="White ...
                image", command=self.save_white_image, height=4, width=15, ...
                bg=self.white_img_color)
            self.save_white_image_button.pack(side="right",padx=10,pady=10)
            self.save_black_image_button = tk.Button(self.window, text="Black ...
                image", command=self.save_black_image, height=4, width=15, ...
                bg=self.white_img_color)
            self.save_black_image_button.pack(side="right",padx=10,pady=10)
        except:
            mb.showerror("Error", "No device found")
            self.start_button_color = "RED"

    # Save the black image for correction
    def save_black_image(self) -> None:
        try:
            if self.cam.CAM_OPEN:
                mb.showinfo("Instruction", "Block the camera lens, then press ok")
            self.cam.get_image(self.black_img)
            self.black_img_data = self.black_img.get_image_data_numpy()
            self.black_img_data[self.black_img_data<0] = 0
            # Change button colors
            self.start_button.destroy()
            self.stop_button.destroy()
            self.save_white_image_button.destroy()
            self.save_black_image_button.destroy()
            self.black_img_color = "GREEN"
            self.stop_button = tk.Button(self.window, text="Stop", ...
                command=self.stop_camera, height=4, width=10)
            self.stop_button.pack(side = 'right', padx=10, pady=10)
            self.start_button = tk.Button(self.window, text="Start", ...
                command=self.setup, height=4, width=10, bg=self.start_button_color)
            self.start_button.pack(side = 'right', padx=10, pady=10)
```

```
45              self.save_white_image_button = tk.Button(self.window, text="White ...
                    image", command=self.save_white_image, height=4, width=15, ...
                    bg=self.white_img_color)
46              self.save_white_image_button.pack(side="right",padx=10,pady=10)
47              self.save_black_image_button = tk.Button(self.window, text="Black ...
                    image", command=self.save_black_image, height=4, width=15, ...
                    bg=self.black_img_color)
48              self.save_black_image_button.pack(side="right",padx=10,pady=10)
49              self.saved_black = True
50          except:
51              mb.showerror("Error", "Device not open")
52
53      # Save the white image for correction
54      def save_white_image(self) -> None:
55          try:
56              if self.cam.CAM_OPEN:
57                  mb.showinfo("Instruction", "Hold a white sheet of paper in front ...
                        of the camera, then press ok")
58              self.cam.get_image(self.white_img)
59              self.white_img_data = self.white_img.get_image_data_numpy()
60              self.white_img_data = self.white_img_data[self.white_img_data != 0]
61              self.white_img_data = cv2.resize(self.white_img_data, (2048,1088), ...
                    interpolation=cv2.INTER_CUBIC)
62              self.white_img_data = self.white_img_data/np.max(self.white_img_data)
63              # Change button colors
64              self.start_button.destroy()
65              self.stop_button.destroy()
66              self.save_white_image_button.destroy()
67              self.save_black_image_button.destroy()
68              self.white_img_color = "GREEN"
69              self.stop_button = tk.Button(self.window, text="Stop", ...
                    command=self.stop_camera, height=4, width=10)
70              self.stop_button.pack(side = 'right', padx=10, pady=10)
71              self.start_button = tk.Button(self.window, text="Start", ...
                    command=self.setup, height=4, width=10, bg=self.start_button_color)
72              self.start_button.pack(side = 'right', padx=10, pady=10)
73              self.save_white_image_button = tk.Button(self.window, text="White ...
                    image", command=self.save_white_image, height=4, width=15, ...
                    bg=self.white_img_color)
74              self.save_white_image_button.pack(side="right",padx=10,pady=10)
75              self.save_black_image_button = tk.Button(self.window, text="Black ...
                    image", command=self.save_black_image, height=4, width=15, ...
                    bg=self.black_img_color)
76              self.save_black_image_button.pack(side="right",padx=10,pady=10)
77              self.saved_white = True
78          except:
79              mb.showerror("Error", "Device not open")
80
81      # Turn the camera off and close it
82      def stop_camera(self) -> None:
83          try:
84              self.cam.stop_acquisition()
85              self.cam.close_device()
86              # Change button colors
87              self.start_button.destroy()
88              self.stop_button.destroy()
89              self.save_white_image_button.destroy()
90              self.save_black_image_button.destroy()
91              self.black_img_color = "ORANGE"
92              self.white_img_color = "ORANGE"
93              self.start_button_color = "RED"
94              self.stop_button = tk.Button(self.window, text="Stop", ...
                    command=self.stop_camera, height=4, width=10)
95              self.stop_button.pack(side = 'right', padx=10, pady=10)
```

```
96              self.start_button = tk.Button(self.window, text="Start", ...
                    command=self.setup, height=4, width=10, bg=self.start_button_color)
97              self.start_button.pack(side = 'right', padx=10, pady=10)
98              self.save_white_image_button = tk.Button(self.window, text="White ...
                    image", command=self.save_white_image, height=4, width=15, ...
                    bg=self.white_img_color)
99              self.save_white_image_button.pack(side="right",padx=10,pady=10)
100             self.save_black_image_button = tk.Button(self.window, text="Black ...
                    image", command=self.save_black_image, height=4, width=15, ...
                    bg=self.white_img_color)
101             self.save_black_image_button.pack(side="right",padx=10,pady=10)
102         except:
103             mb.showerror("Error", "No device open")
104
105     # Help pop-up
106     def show_help(self) -> None:
107         mb.showinfo("Info", "np.subtract(b1, b2) = b1 - b2\n" +
108                             "np.add(b1, b2) = b1 + b2\n" +
109                             "np.multiply(b1, b2) = b1 * b2\n" +
110                             "np.divide(b1, b2) = b1 / b2")
```

Processing

Processing the image is done according to figure 8. With the addition of a circular image mask (as described in section 5.1.3 under '*Black and white correction*') and the exception that normalization is done simultaneously with creating the image cube. It can be seen that immediately when seperating the wavelengths, for some wavelengths multiple images are added together (like mentioned in section 5.1.3 under '*Normalization*').

```
    #TODO To be implemented (within 'try'-block, 'except'-block gets executed as long as no white or black image is saved)
    # If there is a black and white image, implements the formula for black and white correction
    def black_and_white_correction(self) -> None:
        # Correction formula for flat-field correction
        try:
            # Formula for black and white correction should go here in place of the code below
            # Write the formula for the black and white correction for each of the arrays (1, 2, 3)
            # Last line is correct
            self.corrected_data_1 = self.data[0]
            self.corrected_data_2 = self.data[1]
            self.corrected_data_3 = self.data[2]
            self.corrected_data = [self.corrected_data_1, self.corrected_data_2, self.corrected_data_3]
        except:
            self.corrected_data_1 = self.data[0]
            self.corrected_data_2 = self.data[1]
            self.corrected_data_3 = self.data[2]
            self.corrected_data = [self.corrected_data_1, self.corrected_data_2, self.corrected_data_3]

    # Separate wavelengths from full signal
    def create_image_cube(self) -> None:
        # Image for every wavelength (1088x2048 -> 272x512)
        # The corrected_data numpy array has all wavelength pixels in a 4x4 grid, so by choosing what pixels to select per wavelength you get individual images for each
        # Multiple images are added together depending on the relative signal for each wavelength
        self.n490 = self.corrected_data[0][np.s_[0::4],np.s_[0::4]] + self.corrected_data[1][np.s_[0::4],np.s_[0::4]] + self.corrected_data[2][np.s_[0::4],np.s_[0::4]]*0.833
        self.n500 = self.corrected_data[0][np.s_[0::4],np.s_[1::4]] + self.corrected_data[1][np.s_[0::4],np.s_[1::4]] + self.corrected_data[2][np.s_[0::4],np.s_[1::4]]*0.772
        self.n477 = self.corrected_data[0][np.s_[0::4],np.s_[2::4]] + self.corrected_data[1][np.s_[0::4],np.s_[2::4]] + self.corrected_data[2][np.s_[0::4],np.s_[2::4]]*0.476
        self.n478 = self.corrected_data[0][np.s_[0::4],np.s_[3::4]] + self.corrected_data[1][np.s_[0::4],np.s_[3::4]] + self.corrected_data[2][np.s_[0::4],np.s_[3::4]]*0.161

        self.n577 = self.corrected_data[0][np.s_[1::4],np.s_[0::4]] + self.corrected_data[1][np.s_[1::4],np.s_[0::4]]*0.109
        self.n591 = self.corrected_data[0][np.s_[1::4],np.s_[1::4]]
        self.n563 = self.corrected_data[0][np.s_[1::4],np.s_[2::4]] + self.corrected_data[1][np.s_[1::4],np.s_[2::4]]*0.220
        self.n553 = self.corrected_data[0][np.s_[1::4],np.s_[3::4]] + self.corrected_data[1][np.s_[1::4],np.s_[3::4]]*0.509

        self.n618 = self.corrected_data[0][np.s_[2::4],np.s_[0::4]]
        self.n616 = self.corrected_data[0][np.s_[2::4],np.s_[1::4]] + self.corrected_data[1][np.s_[2::4],np.s_[1::4]]*0.045
```

```
36          self.n613 = self.corrected_data[0][np.s_[2::4],np.s_[2::4]]
37          self.n600 = self.corrected_data[0][np.s_[2::4],np.s_[3::4]]
38
39          self.n538 = self.corrected_data[0][np.s_[3::4],np.s_[0::4]] + ...
                self.corrected_data[1][np.s_[3::4],np.s_[0::4]]*0.349
40          self.n549 = self.corrected_data[0][np.s_[3::4],np.s_[1::4]] + ...
                self.corrected_data[1][np.s_[3::4],np.s_[1::4]]*0.425
41          self.n523 = self.corrected_data[0][np.s_[3::4],np.s_[2::4]] + ...
                self.corrected_data[1][np.s_[3::4],np.s_[2::4]]*0.509
42          self.n511 = self.corrected_data[0][np.s_[3::4],np.s_[3::4]] + ...
                self.corrected_data[1][np.s_[3::4],np.s_[3::4]] + ...
                self.corrected_data[2][np.s_[3::4],np.s_[3::4]]*0.073
43
44      # Creates a circular mask from the center to put over the image
45      def create_circular_mask(self, h, w, center = None, radius = None) -> bool:
46          # Set the center and radius
47          if center is None:
48              center = (w/2, h/2)
49          if radius is None:
50              radius = min(center[0], center[1], w-center[0], h-center[1])
51
52          Y,X = np.ogrid[:h, :w]
53          # Formula to calculate how far everything is from the center set previously
54          dist_from_center = np.sqrt((X-center[0])**2 + (Y-center[1])**2)
55          # Create a tuple containing boolean values indicating whether each pixel ...
                is inside or outside the circular mask
56          mask = dist_from_center ≤ radius
57          return mask
58
59      # Values outside of the max are made black such that only the signal at the ...
            center remains
60      def mask_images(self) -> None:
61          self.mask = self.create_circular_mask(272,512)
62
63          self.n490[self.mask==False] = 0
64          self.n500[self.mask==False] = 0
65          self.n477[self.mask==False] = 0
66          self.n478[self.mask==False] = 0
67          self.n577[self.mask==False] = 0
68          self.n591[self.mask==False] = 0
69          self.n563[self.mask==False] = 0
70          self.n553[self.mask==False] = 0
71          self.n618[self.mask==False] = 0
72          self.n616[self.mask==False] = 0
73          self.n613[self.mask==False] = 0
74          self.n600[self.mask==False] = 0
75          self.n538[self.mask==False] = 0
76          self.n549[self.mask==False] = 0
77          self.n523[self.mask==False] = 0
78          self.n511[self.mask==False] = 0
```

Create image

This block of code contains all necessary functions for displaying in image on-screen. The 'set_image' functions is called from the main loop (8.1) and sets the image of the canvas to the given input image, which is obtained via 'create_image'. For the preset pages, this function evaluates the formula in the formula entry and turns it into a greyscale image (if the formula is valid according to section 8.1). It does so except for when an image is uploaded ('self.mode = "IM"', this variable is also explained in section 8.1). Then that uploaded image becomes the displayed image. For the RGB page, this functions does the same, but the image is no longer a greyscale image from the evaluation of the formula entry, but rather a merged image of the wavelengths used for each of the three RGB-channels. The 'correct_result' function removes any unwanted value from the final image (which could arise because of division by zero or subtraction resulting in negative values).

```
    # Removes values such as negative and infinite numbers from the result which ...
        would otherwise cause discrepancies in the final image
    def correct_result(self, result) -> None:
        warnings.simplefilter('ignore', RuntimeWarning)
        result = np.nan_to_num(result, copy=True, nan=255, posinf=255, neginf=0)
        result[result<0] = 0
        return result

    # Creates the image to be displayed on screen based on the page and the ...
        current formula
    def create_image(self) -> Image:
        if self.current_page == "Page 1" or self.current_page == "Page 2":
            if self.mode == "LIVE":
                self.image = ...
                    cv2.cvtColor(self.correct_result(self.eval_formula(self.get_current_formula()
                    * self.intensity_scale.get()).astype(np.uint8), ...
                    cv2.COLOR_BGR2RGB)
            elif self.mode == "IM":
                try:
                    self.image = ...
                        self.correct_result(eval(self.get_current_formula()) * ...
                        self.intensity_scale.get()).astype(np.uint8)
                except:
                    self.image = np.asarray(Image.open(self.current_file))

        if self.current_page == "Page 3" or self.current_page == "Page 4":
            if self.mode == "LIVE":
                # Channels are not empty
                if self.r_value.get() != "" and self.g_value.get() != "" and ...
                    self.b_value.get() != "":
                    # Formulas in the channels are valid formulas
                    if ...
                        self.valid_formula_no_error(self.rgb_presets.get(self.r_value.get())) ...
                        and ...
                        self.valid_formula_no_error(self.rgb_presets.get(self.g_value.get())) ...
                        and ...
                        self.valid_formula_no_error(self.rgb_presets.get(self.b_value.get())):
                        self.merge_RGB_image()
                        self.image = self.correct_result(self.image.astype(np.uint8))
                    else:
                        # Display black image
                        self.image = np.zeros((272,512))
                else:
                    # Display black image
                    self.image = np.zeros((272,512))
            elif self.mode == "IM":
                if self.r_value.get() != "" and self.g_value.get() != "" and ...
                    self.b_value.get() != "":
                    if ...
                        self.valid_formula_no_error(self.rgb_presets.get(self.r_value.get())) ...
```

```
                            and ...
                            self.valid_formula_no_error(self.rgb_presets.get(self.g_value.get())) ...
                            and ...
                            self.valid_formula_no_error(self.rgb_presets.get(self.b_value.get())):
36                          self.merge_RGB_image()
37                          self. image = ...
                                self.correct_result(self.image.astype(np.uint8))
38                      else:
39                          # Display uploaded image
40                          self.image = np.asarray(Image.open(self.current_file))
41                  else:
42                      # Display uploaded image
43                      self.image = np.asarray(Image.open(self.current_file))
44          return Image.fromarray(self.image.astype(np.uint8)).resize((768,408))
45
46      # Merges the results from each channel together into one RGB image
47      def merge_RGB_image(self):
48          self.image = cv2.merge([self.r_scale.get() * self.intensity_scale.get() * ...
                self.eval_formula(self.rgb_presets.get(self.r_value.get())),
49                                 self.g_scale.get() * self.intensity_scale.get() * ...
                                       self.eval_formula(self.rgb_presets.get(self.g_value.get())),
50                                 self.b_scale.get() * self.intensity_scale.get() * ...
                                       self.eval_formula(self.rgb_presets.get(self.b_value.get()))])
51
52      # Set the input image as the image being displayed on the canvas
53      def set_image(self, image):
54          self.photo = ImageTk.PhotoImage(image=image)
55          self.canvas.create_image(self.width, 0.5*self.height, ...
                image=self.photo,anchor=tk.E)
```

Formula

The following block contains all functions for the formula. The formula is defined as the text in the formula entry that is to be evaluated and turned into an expression. Not all entries can be turned into expression. That's why the function 'valid_formula' exists. It is used throughout the script wherever it is necessary to check if a formula can be evaluated. Using an invalid formula as an image causes problems in the script so this functions prevents that. It throws an error when the formula is invalid, indicating to the user that they should change it. The script also has this functions without throwing the error, which is necessary in certain situations. The 'self.current_formula is the formula that is used for creating the image (as can be seen in section 8.1). This variable has a getter and setter to make these functionalities easier in other parts of the script. Setting the current formula to the given formula is only done when the given formula is valid. The function 'test_formula' is called upon pressing that respective button. It just sets the formula entry to be the current formula thus displaying the result, but does not yet save it as a preset.

```
    # Set current formula to what is given as the parameter
    # Only if it is a valid formula
    def set_current_formula(self, formula) -> None:
        if self.valid_formula_no_error(formula):
            self.current_formula = formula
        else:
            self.current_formula = "self.black"

    # Returns the current formula
    def get_current_formula(self) -> str:
        return self.current_formula

    # Returns the evaluation of the current formula if the formula is valid
    def eval_formula(self, formula) -> np.array:
        if self.valid_formula(formula):
            return eval(formula)

    # Tests the formula currently entered in the formula entry
    def test_formula(self) -> None:
        if self.valid_formula(self.formula_entry.get()):
            self.set_current_formula(self.formula_entry.get())

        # Checks whether it's possible to evaluate the formula
    # Returns True or False depending on it
    # Gives an error for invalid formula if it cannot evaluate the formula
    def valid_formula(self, formula) -> None:
        try:
            eval(formula)
            return True
        except:
            mb.showerror("Error", "Invalid formula")
            return False

    # Same function as valid_formula, but doesn't give error message
    # Necessary in some situations
    def valid_formula_no_error(self, formula) -> None:
        try:
            eval(formula)
            return True
        except:
            return False
```

Preset

The functions in the following block all have to do with presets. Firstly there are the functions for adding presets. The functions for the preset- and RGB-page are seperate, as these functions are quite complicated already. There are a lot of cases to be identified when adding a preset, which are described as comments in the code (figure 15 shows a simplified version without all these cases described). Adding a preset results in removing the list of presets, then updating the list and lastly recreating the list (otherwise the updated preset list would not show in the program). The 'set_button_preset' function sets the current formula to the formula associated with the button that is pressed (only if a preset is associated to that button, described in section 5.1.3 under '*Selecting presets*'). The function called 'set_as_current_preset' is called when the button 'Select preset' is clicked. This function makes sure that the result of that preset is shown on-screen by setting the current formula to the formula associated with the preset. The functions for removing and clearing presets first ask for a confirmation and then determine what presets to remove. Then updating the list goes the same as with adding presets. First, the list is removed, then the presets are updated and lastly the list is added again. Nothing special happens in the final functions for inserting wavelengths and signs and also deleting the name and formula from their respective entries. Explanations for these bits of code are given as comments above the functions.

```
    # Check if the name already exists in presets
    # Check if the formula is valid
    # Add the preset to the dictionary of presets
    def add_preset_page1(self) -> None:
        name = self.name_entry.get()
        formula = self.formula_entry.get()
        if name.__contains__("_RGB"):
            mb.showerror("Error", "Invalid name")
        else:
            # Case 1: name does not yet exist
            if name not in self.presets.keys():
                # Case 1.1: Name entry is empty
                if name == "":
                    mb.showerror("Error", "Please fill in a name for the preset")
                # Case 1.2: Formula entry is empty
                elif formula == "":
                    mb.showerror("Error", "Please fill in a formula")
                # Case 1.3: Neither are empty
                else:
                    # Case 1.3.1: Formula is valid
                    if self.valid_formula_no_error(formula):
                        self.presets.update({name:formula})
                        self.preset_box.destroy()
                        self.preset_box = ttk.Combobox(self.window, ...
                            values=list(self.get_preset_without_rgb().keys()))
                        self.preset_box.place(x=350, y=40)
                        with open('presets.json', 'w') as fp:
                            json.dump(self.presets, fp)
                            fp.close()
                        self.delete_name(e='')
                        self.delete_formula(e='')
                    # Case 1.3.2: Formula is invalid
                    else:
                        if mb.askokcancel("Warning", "This formula is invalid, ...
                            are you sure you want to add it?"):
                            self.presets.update({name:formula})
                            self.preset_box.destroy()
                            self.preset_box = ttk.Combobox(self.window, ...
                                values=list(self.get_preset_without_rgb().keys()))
                            self.preset_box.place(x=350, y=40)
                            with open('presets.json', 'w') as fp:
                                json.dump(self.presets, fp)
                                fp.close()
                            self.delete_name(e='')
```

```
                            self.delete_formula(e='')
            # Case 2: Name already exist in presets
            else:
                # Case 2.1: Formula is valid
                if self.valid_formula_no_error(formula):
                    if mb.askokcancel("Warning", "Are you sure you want to ...
                        overwrite the preset?"):
                        self.presets[name] = formula
                        self.preset_box.destroy()
                        self.preset_box = ttk.Combobox(self.window, ...
                            values=list(self.get_preset_without_rgb().keys()))
                        self.preset_box.place(x=350, y=40)
                        with open('presets.json', 'w') as fp:
                            json.dump(self.presets, fp)
                            fp.close()
                        self.delete_name(e='')
                        self.delete_formula(e='')
                # Case 2.2: Formula is invalid
                else:
                    if mb.askokcancel("Warning", "This formula is invalid, are ...
                        you sure you want to add it?"):
                        if mb.askokcancel("Warning", "Are you sure you want to ...
                            overwrite the preset?"):
                            self.presets[name] = formula
                            self.preset_box.destroy()
                            self.preset_box = ttk.Combobox(self.window, ...
                                values=list(self.get_preset_without_rgb().keys()))
                            self.preset_box.place(x=350, y=40)
                            with open('presets.json', 'w') as fp:
                                json.dump(self.presets, fp)
                                fp.close()
                            self.delete_name(e='')
                            self.delete_formula(e='')
    def add_preset_page3(self) -> None:
        name = self.name_entry.get() + "_RGB"
        # Case 1: name does not yet exist
        if name not in self.presets.keys():
            # Case 1.1: Name entry is empty
            if name == "":
                mb.showerror("Error", "Please fill in a name for the preset")
            # Case 1.2: Name entry is not empty
            else:
                # Case 1.3.1: Formula is valid
                if ...
                    self.valid_formula_no_error(self.rgb_presets.get(self.r_value.get())) ...
                    and ...
                    self.valid_formula_no_error(self.rgb_presets.get(self.g_value.get())) ...
                    and ...
                    self.valid_formula_no_error(self.rgb_presets.get(self.b_value.get())):
                    self.presets.update({name:(self.r_value.get(), ...
                        self.g_value.get(), self.b_value.get(), ...
                        self.r_scale.get(), self.g_scale.get(), self.b_scale.get())})
                    self.preset_box.destroy()
                    self.preset_box = ttk.Combobox(self.window, ...
                        values=list(self.get_preset_rgb().keys()))
                    self.preset_box.place(x=350, y=40)
                    with open('presets.json', 'w') as fp:
                        json.dump(self.presets, fp)
                        fp.close()
                    self.delete_name(e='')
                # Case 1.3.2: Formula is invalid
                else:
                    if mb.askokcancel("Warning", "This formula is invalid, are ...
                        you sure you want to add it?"):
```

```
92                          self.presets.update({name:(self.r_value.get(), ...
                                self.g_value.get(), self.b_value.get(), ...
                                self.r_scale.get(), self.g_scale.get(), ...
                                self.b_scale.get())})
93                          self.preset_box.destroy()
94                          self.preset_box = ttk.Combobox(self.window, ...
                                values=list(self.get_preset_rgb().keys()))
95                          self.preset_box.place(x=350, y=40)
96                          with open('presets.json', 'w') as fp:
97                              json.dump(self.presets, fp)
98                              fp.close()
99                          self.delete_name(e='')
100         # Case 2: Name already exist in presets
101         else:
102             # Case 2.1: Formula is valid
103             if ...
                    self.valid_formula_no_error(self.rgb_presets.get(self.r_value.get())) ...
                    and ...
                    self.valid_formula_no_error(self.rgb_presets.get(self.g_value.get())) ...
                    and ...
                    self.valid_formula_no_error(self.rgb_presets.get(self.b_value.get())):
104                 if mb.askokcancel("Warning", "Are you sure you want to overwrite ...
                        the preset?"):
105                     self.presets[name] = (self.r_value.get(), self.g_value.get(), ...
                            self.b_value.get(), self.r_scale.get(), ...
                            self.g_scale.get(), self.b_scale.get())
106                     self.preset_box.destroy()
107                     self.preset_box = ttk.Combobox(self.window, ...
                            values=list(self.get_preset_rgb().keys()))
108                     self.preset_box.place(x=350, y=40)
109                     with open('presets.json', 'w') as fp:
110                         json.dump(self.presets, fp)
111                         fp.close()
112                     self.delete_name(e='')
113             # Case 2.2: Formula is invalid
114             else:
115                 if mb.askokcancel("Warning", "This formula is invalid, are you ...
                        sure you want to add it?"):
116                     if mb.askokcancel("Warning", "Are you sure you want to ...
                            overwrite the preset?"):
117                         self.presets[name] = (self.r_value.get(), ...
                                self.g_value.get(), self.b_value.get(), ...
                                self.r_scale.get(), self.g_scale.get(), ...
                                self.b_scale.get())
118                         self.preset_box.destroy()
119                         self.preset_box = ttk.Combobox(self.window, ...
                                values=list(self.get_preset_rgb().keys()))
120                         self.preset_box.place(x=350, y=40)
121                         with open('presets.json', 'w') as fp:
122                             json.dump(self.presets, fp)
123                             fp.close()
124                         self.delete_name(e='')
125
126     # Sets the presets of the buttons according to their names
127     def set_button_preset(self, preset) -> None:
128         # Checks if a preset exists with the name of the button
129         # If so, displays the result of that preset
130         # If not, displays black image
131         if preset == "Perfusion":
132             if self.presets.__contains__(preset):
133                 self.set_current_formula(self.presets.get(preset))
134             else:
135                 self.set_current_formula("self.black")
136         elif preset == "Oxygenation":
137             if self.presets.__contains__(preset):
```

```
138                 self.set_current_formula(self.presets.get(preset))
139             else:
140                 self.set_current_formula("self.black")
141         elif preset == "Abnormal tissue":
142             if self.presets.__contains__(preset):
143                 self.set_current_formula(self.presets.get(preset))
144             else:
145                 self.set_current_formula("self.black")
146         elif preset == "Pigmentation":
147             if self.presets.__contains__(preset):
148                 self.set_current_formula(self.presets.get(preset))
149             else:
150                 self.set_current_formula("self.black")
151         elif preset == "Perfusion_RGB":
152             if self.presets.__contains__(preset):
153                 self.r_value.set(self.presets.get(preset)[0])
154                 self.g_value.set(self.presets.get(preset)[1])
155                 self.b_value.set(self.presets.get(preset)[2])
156                 self.r_scale.set(self.presets.get(preset)[3])
157                 self.g_scale.set(self.presets.get(preset)[4])
158                 self.b_scale.set(self.presets.get(preset)[5])
159             else:
160                 self.r_value.set("")
161                 self.g_value.set("")
162                 self.b_value.set("")
163                 self.r_scale.set(0.5)
164                 self.g_scale.set(0.5)
165                 self.b_scale.set(0.5)
166         elif preset == "Oxygenation_RGB":
167             if self.presets.__contains__(preset):
168                 self.r_value.set(self.presets.get(preset)[0])
169                 self.g_value.set(self.presets.get(preset)[1])
170                 self.b_value.set(self.presets.get(preset)[2])
171                 self.r_scale.set(self.presets.get(preset)[3])
172                 self.g_scale.set(self.presets.get(preset)[4])
173                 self.b_scale.set(self.presets.get(preset)[5])
174             else:
175                 self.r_value.set("")
176                 self.g_value.set("")
177                 self.b_value.set("")
178                 self.r_scale.set(0.5)
179                 self.g_scale.set(0.5)
180                 self.b_scale.set(0.5)
181         elif preset == "Abnormal tissue_RGB":
182             if self.presets.__contains__(preset):
183                 self.r_value.set(self.presets.get(preset)[0])
184                 self.g_value.set(self.presets.get(preset)[1])
185                 self.b_value.set(self.presets.get(preset)[2])
186                 self.r_scale.set(self.presets.get(preset)[3])
187                 self.g_scale.set(self.presets.get(preset)[4])
188                 self.b_scale.set(self.presets.get(preset)[5])
189             else:
190                 self.r_value.set("")
191                 self.g_value.set("")
192                 self.b_value.set("")
193                 self.r_scale.set(0.5)
194                 self.g_scale.set(0.5)
195                 self.b_scale.set(0.5)
196         elif preset == "Pigmentation_RGB":
197             if self.presets.__contains__(preset):
198                 self.r_value.set(self.presets.get(preset)[0])
199                 self.g_value.set(self.presets.get(preset)[1])
200                 self.b_value.set(self.presets.get(preset)[2])
201                 self.r_scale.set(self.presets.get(preset)[3])
202                 self.g_scale.set(self.presets.get(preset)[4])
```

```
203                 self.b_scale.set(self.presets.get(preset)[5])
204             else:
205                 self.r_value.set("")
206                 self.g_value.set("")
207                 self.b_value.set("")
208                 self.r_scale.set(0.5)
209                 self.g_scale.set(0.5)
210                 self.b_scale.set(0.5)
211 
212     # Sets the current formula to the formula associated with the key shown in ...
            the value box
213     # Also sets the text entries to the corresponding name and formula
214     def set_as_current_preset(self) -> None:
215         if self.preset_box.get() != "":
216             self.set_current_formula(self.presets.get(self.preset_box.get()))
217             if self.current_page == "Page 1":
218                 if self.preset_box.get().__contains__("_RGB") == False:
219                     self.delete_name(e="")
220                     self.name_entry.insert(0, self.preset_box.get())
221                     self.delete_formula(e="")
222                     self.formula_entry.insert(0, ...
                            self.presets.get(self.preset_box.get()))
223                 else:
224                     mb.showerror("Error", "Invalid formula")
225             if self.current_page == "Page 3":
226                 if self.preset_box.get().__contains__("_RGB"):
227                     self.delete_name(e="")
228                     self.name_entry.insert(0, ...
                            self.preset_box.get().replace("_RGB", ""))
229                     self.r_value.set(self.presets.get(self.preset_box.get())[0])
230                     self.g_value.set(self.presets.get(self.preset_box.get())[1])
231                     self.b_value.set(self.presets.get(self.preset_box.get())[2])
232                     self.r_scale.set(self.presets.get(self.preset_box.get())[3])
233                     self.g_scale.set(self.presets.get(self.preset_box.get())[4])
234                     self.b_scale.set(self.presets.get(self.preset_box.get())[5])
235                 else:
236                     mb.showerror("Error", "Invalid formula")
237 
238     # Removes the currently selected preset and any RGB preset using that preset
239     def remove_preset(self) -> None:
240         if self.current_page == "Page 1":
241             if self.presets.__contains__(self.preset_box.get()):
242                 if mb.askokcancel("Warning", "Are you sure you want to remove the ...
                        preset?"):
243                     self.presets.pop(self.preset_box.get())
244                     keys_to_pop = []
245                     for key in self.presets.keys():
246                         # Checks if any RGB preset uses the removed preset
247                         if self.presets.get(key)[0] == self.preset_box.get() or ...
                                self.presets.get(key)[1] == self.preset_box.get() or ...
                                self.presets.get(key)[2] == self.preset_box.get():
248                             keys_to_pop.append(key)
249                     for key in keys_to_pop:
250                         self.presets.pop(key)
251                     self.preset_box.destroy()
252                     self.preset_box = ttk.Combobox(self.window, ...
                            values=list(self.get_preset_without_rgb().keys()))
253                     self.preset_box.place(x=350, y=40)
254                     with open('presets.json', 'w') as fp:
255                         json.dump(self.presets, fp)
256                         fp.close()
257         if self.current_page == "Page 3":
258             if self.presets.__contains__(self.preset_box.get()):
259                 if mb.askokcancel("Warning", "Are you sure you want to remove the ...
                        preset?"):
```

```
260                     self.presets.pop(self.preset_box.get())
261                     self.preset_box.destroy()
262                     self.preset_box = ttk.Combobox(self.window, ...
                            values=list(self.get_preset_rgb().keys()))
263                     self.preset_box.place(x=350, y=40)
264                     with open('presets.json', 'w') as fp:
265                         json.dump(self.presets, fp)
266                         fp.close()
267
268     # Removes all presets
269     def clear_presets(self) -> None:
270             if mb.askokcancel("Warning", "Are you sure you want to remove all ...
                    presets?"):
271                 if self.current_page == "Page 1":
272                     for key in list(self.get_preset_without_rgb()):
273                         self.get_preset_without_rgb().pop(key)
274                     self.preset_box.destroy()
275                     self.preset_box = ttk.Combobox(self.window, ...
                            values=list(self.get_preset_without_rgb().keys()))
276                     self.preset_box.place(x=350, y=40)
277                     with open('presets.json', 'w') as fp:
278                         json.dump(self.presets, fp)
279                         fp.close()
280                 elif self.current_page == "Page 3":
281                     for key in list(self.get_preset_rgb()):
282                         self.get_preset_rgb().pop(key)
283                     self.preset_box.destroy()
284                     self.preset_box = ttk.Combobox(self.window, ...
                            values=list(self.get_preset_rgb().keys()))
285                     self.preset_box.place(x=350, y=40)
286                     with open('presets.json', 'w') as fp:
287                         json.dump(self.presets, fp)
288                         fp.close()
289
290     # Insert the representation for the currently selected wavelength into the ...
            formula
291     def insert_wavelength(self, text) -> None:
292         match text:
293             case "490nm":
294                 text = "self.n490"
295             case "500nm":
296                 text = "self.n500"
297             case "477nm":
298                 text = "self.n477"
299             case "478nm":
300                 text = "self.n478"
301             case "577nm":
302                 text = "self.n577"
303             case "591nm":
304                 text = "self.n591"
305             case "563nm":
306                 text = "self.n563"
307             case "553nm":
308                 text = "self.n553"
309             case "618nm":
310                 text = "self.n618"
311             case "616nm":
312                 text = "self.n616"
313             case "613nm":
314                 text = "self.n613"
315             case "600nm":
316                 text = "self.n600"
317             case "538nm":
318                 text = "self.n538"
319             case "549nm":
```

```
320                 text = "self.n549"
321             case "523nm":
322                 text = "self.n523"
323             case "511nm":
324                 text = "self.n511"
325 
326         self.formula_entry.insert(tk.INSERT, text)
327 
328     # Insert the numpy representation of each sign in the formula
329     def insert_sign(self, sign) -> None:
330         match sign:
331             case "-":
332                 self.formula_entry.insert(tk.INSERT, "np.subtract(  ,  )")
333             case "+":
334                 self.formula_entry.insert(tk.INSERT, "np.add(  ,  )")
335             case "/":
336                 self.formula_entry.insert(tk.INSERT, "np.divide(  ,  )")
337             case "x":
338                 self.formula_entry.insert(tk.INSERT, "np.multiply(  ,  )")
339 
340     # Delete all text from name entry
341     def delete_name(self, e) -> None:
342         self.name_entry.delete(0,tk.END)
343 
344     # Delete all text from formula entry
345     def delete_formula(self, e) -> None:
346         self.formula_entry.delete(0,tk.END)
```

Preset lists

Presets from one page can't be used in the other or problems will occur. That is why only presets corresponding to the respective page are shown in each page. These functions create the lists containing only presets of their pages. They do so by checking whether the names of the presets contain the '_RGB'-signature in their name.

```
    # Returns the list of presets containing "_RGB"
    def get_preset_rgb(self) -> dict[str, str]:
        only_rgb = self.presets.copy()
        for key in self.presets.keys():
            if key.__contains__("_RGB") == False:
                only_rgb.pop(key)
        return only_rgb
    # Returns the list of presets not containing "_RGB"
    def get_preset_without_rgb(self) -> dict[str, str]:
        no_rgb = self.presets.copy()
        for key in self.presets.keys():
            if key.__contains__("_RGB") == True:
                no_rgb.pop(key)
        return no_rgb
```

Button color

Makes the doctors preset buttons green if a preset is assigned to them (the name of the button is also the name of a preset in the list of presets: self.presets).

```
    # Changes the color of preset-buttons when presets are assigned
    def set_button_color(self, button) -> str:
        if self.presets.keys().__contains__(button):
            return "GREEN"
        else:
            return "ORANGE"
```

Saving/uploading images

The 'save_image' function is called when the user presses the respective button. This function then creates directories to save the image in (if they don't already exist). It then saves the image into the dedicated directory along with all the images of the image cube and a .json-file containing the preset used to create that image. Uploading an image is done by letting the user select an image from the directory and then setting that as the image to be displayed. It also sets the image cube to the images found in the respective folder and the current formula to the formula found in the preset belonging to the selected image.

```
    # Save the image currently displayed on screen to the directory of "Saved images" along with the image cube and its preset
    def save_image(self) -> None:
        # Freeze camera to capture current image
        if self.cam.CAM_OPEN:
            self.stop_camera()
        image = self.create_image()
        # Ask for a name
        try:
            name = simpledialog.askstring("Name", "Enter a name for the saved image")
        except:
            pass

        # Create directories if they don't yet exist
        if name != None:
            if os.path.exists("Saved images"):
                pass
            else:
                os.makedirs("Saved images")

            if os.path.exists("Saved images/Preset images"):
                pass
            else:
                os.makedirs("Saved images/Preset images")

            if os.path.exists("Saved images/RGB images"):
                pass
            else:
                os.makedirs("Saved images/RGB images")

            # Change the name as long as the name already exists, until it doesn't, then save it to the dedicated directory along with the preset
            if self.current_page == "Page 1" or self.current_page == "Page 2":
                while os.path.exists("Saved images/Preset images/"+name) == True:
                    name += "_"
                os.makedirs("Saved images/Preset images/"+name)
                image.save("Saved images/Preset images/"+name+"/"+name +".png")
            if self.current_page == "Page 3" or self.current_page == "Page 4":
                while os.path.exists("Saved images/RGB images/"+name) == True:
                    name += "_"
                os.makedirs("Saved images/RGB images/"+name)
                image.save("Saved images/RGB images/"+name+"/"+name +".png")

            if self.current_page == "Page 1":
                with open("Saved images/Preset images/"+name +"/Preset.json", 'w') as fp:
                    json.dump({self.name_entry.get():self.formula_entry.get()}, fp)
                    fp.close()
            elif self.current_page == "Page 2":
                with open("Saved images/Preset images/"+name +"/Preset.json", 'w') as fp:
                    json.dump({"":self.get_current_formula()}, fp)
                    fp.close()
```

```
50              elif self.current_page == "Page 3":
51                  with open("Saved images/RGB images/"+name +"/Preset.json", 'w') ...
                        as fp:
52                      json.dump({self.name_entry.get():(self.r_value.get(), ...
                            self.g_value.get(), self.b_value.get(), ...
                            self.r_scale.get(), self.g_scale.get(), ...
                            self.b_scale.get())}, fp)
53                      fp.close()
54              elif self.current_page == "Page 4":
55                  with open("Saved images/RGB images/"+name +"/Preset.json", 'w') ...
                        as fp:
56                      json.dump({"":(self.r_value.get(), self.g_value.get(), ...
                            self.b_value.get(), self.r_scale.get(), ...
                            self.g_scale.get(), self.b_scale.get())}, fp)
57                      fp.close()
58
59              # Save the image cube to the same directory as the file and preset
60              try:
61                  # Check if an image cube is captured (this line will throw an ...
                        error if it does not exist)
62                  self.n490 = self.n490
63                  if self.current_page == "Page 1" or self.current_page == "Page 2":
64                      os.makedirs("Saved images/Preset images/"+name+"/Image cube")
65
66                      nm477 = Image.fromarray(self.n477.astype(np.uint8))
67                      nm477.save("Saved images/Preset images/"+name+"/Image ...
                            cube/477nm.png")
68                      nm478 = Image.fromarray(self.n478.astype(np.uint8))
69                      nm478.save("Saved images/Preset images/"+name+"/Image ...
                            cube/478nm.png")
70                      nm490 = Image.fromarray(self.n490.astype(np.uint8))
71                      nm490.save("Saved images/Preset images/"+name+"/Image ...
                            cube/490nm.png")
72                      nm500 = Image.fromarray(self.n500.astype(np.uint8))
73                      nm500.save("Saved images/Preset images/"+name+"/Image ...
                            cube/500nm.png")
74
75                      nm511 = Image.fromarray(self.n511.astype(np.uint8))
76                      nm511.save("Saved images/Preset images/"+name+"/Image ...
                            cube/511nm.png")
77                      nm523 = Image.fromarray(self.n523.astype(np.uint8))
78                      nm523.save("Saved images/Preset images/"+name+"/Image ...
                            cube/523nm.png")
79                      nm538 = Image.fromarray(self.n538.astype(np.uint8))
80                      nm538.save("Saved images/Preset images/"+name+"/Image ...
                            cube/538nm.png")
81                      nm549 = Image.fromarray(self.n549.astype(np.uint8))
82                      nm549.save("Saved images/Preset images/"+name+"/Image ...
                            cube/549nm.png")
83
84                      nm553 = Image.fromarray(self.n553.astype(np.uint8))
85                      nm553.save("Saved images/Preset images/"+name+"/Image ...
                            cube/553nm.png")
86                      nm563 = Image.fromarray(self.n563.astype(np.uint8))
87                      nm563.save("Saved images/Preset images/"+name+"/Image ...
                            cube/563nm.png")
88                      nm577 = Image.fromarray(self.n577.astype(np.uint8))
89                      nm577.save("Saved images/Preset images/"+name+"/Image ...
                            cube/577nm.png")
90                      nm591 = Image.fromarray(self.n591.astype(np.uint8))
91                      nm591.save("Saved images/Preset images/"+name+"/Image ...
                            cube/591nm.png")
92
93                      nm600 = Image.fromarray(self.n600.astype(np.uint8))
```

```
94                      nm600.save("Saved images/Preset images/"+name+"/Image ...
                            cube/600nm.png")
95                      nm613 = Image.fromarray(self.n613.astype(np.uint8))
96                      nm613.save("Saved images/Preset images/"+name+"/Image ...
                            cube/613nm.png")
97                      nm616 = Image.fromarray(self.n616.astype(np.uint8))
98                      nm616.save("Saved images/Preset images/"+name+"/Image ...
                            cube/616nm.png")
99                      nm618 = Image.fromarray(self.n618.astype(np.uint8))
100                     nm618.save("Saved images/Preset images/"+name+"/Image ...
                            cube/618nm.png")
101
102                 if self.current_page == "Page 3" or self.current_page == "Page 4":
103                     os.makedirs("Saved images/RGB images/"+name+"/Image cube")
104
105                     nm477 = Image.fromarray(self.n477.astype(np.uint8))
106                     nm477.save("Saved images/RGB images/"+name+"/Image ...
                            cube/477nm.png")
107                     nm478 = Image.fromarray(self.n478.astype(np.uint8))
108                     nm478.save("Saved images/RGB images/"+name+"/Image ...
                            cube/478nm.png")
109                     nm490 = Image.fromarray(self.n490.astype(np.uint8))
110                     nm490.save("Saved images/RGB images/"+name+"/Image ...
                            cube/490nm.png")
111                     nm500 = Image.fromarray(self.n500.astype(np.uint8))
112                     nm500.save("Saved images/RGB images/"+name+"/Image ...
                            cube/500nm.png")
113
114                     nm511 = Image.fromarray(self.n511.astype(np.uint8))
115                     nm511.save("Saved images/RGB images/"+name+"/Image ...
                            cube/511nm.png")
116                     nm523 = Image.fromarray(self.n523.astype(np.uint8))
117                     nm523.save("Saved images/RGB images/"+name+"/Image ...
                            cube/523nm.png")
118                     nm538 = Image.fromarray(self.n538.astype(np.uint8))
119                     nm538.save("Saved images/RGB images/"+name+"/Image ...
                            cube/538nm.png")
120                     nm549 = Image.fromarray(self.n549.astype(np.uint8))
121                     nm549.save("Saved images/RGB images/"+name+"/Image ...
                            cube/549nm.png")
122
123                     nm553 = Image.fromarray(self.n553.astype(np.uint8))
124                     nm553.save("Saved images/RGB images/"+name+"/Image ...
                            cube/553nm.png")
125                     nm563 = Image.fromarray(self.n563.astype(np.uint8))
126                     nm563.save("Saved images/RGB images/"+name+"/Image ...
                            cube/563nm.png")
127                     nm577 = Image.fromarray(self.n577.astype(np.uint8))
128                     nm577.save("Saved images/RGB images/"+name+"/Image ...
                            cube/577nm.png")
129                     nm591 = Image.fromarray(self.n591.astype(np.uint8))
130                     nm591.save("Saved images/RGB images/"+name+"/Image ...
                            cube/591nm.png")
131
132                     nm600 = Image.fromarray(self.n600.astype(np.uint8))
133                     nm600.save("Saved images/RGB images/"+name+"/Image ...
                            cube/600nm.png")
134                     nm613 = Image.fromarray(self.n613.astype(np.uint8))
135                     nm613.save("Saved images/RGB images/"+name+"/Image ...
                            cube/613nm.png")
136                     nm616 = Image.fromarray(self.n616.astype(np.uint8))
137                     nm616.save("Saved images/RGB images/"+name+"/Image ...
                            cube/616nm.png")
138                     nm618 = Image.fromarray(self.n618.astype(np.uint8))
```

```
139                     nm618.save("Saved images/RGB images/"+name+"/Image ...
                            cube/618nm.png")
140             except:
141                 pass
142 
143     # Upload an image from the directory of "Saved images"
144     def upload_image(self) -> None:
145         # Freeze camera
146         if self.cam.CAM_OPEN:
147             self.stop_camera()
148         # Open directory based on type of preset
149         if self.current_page == "Page 1" or self.current_page == "Page 2":
150             self.current_file = askopenfilename(initialdir="Saved images/Preset ...
                images")
151         elif self.current_page == "Page 3" or self.current_page == "Page 4":
152             self.current_file = askopenfilename(initialdir="Saved images/RGB images")
153         if self.current_file != "" and self.current_file.__contains__(".png") and ...
            self.current_file.__contains__("Image cube") == False:
154             if self.current_page == "Page 1" or self.current_page == "Page 3":
155                 self.preset_box.set("")
156             try:
157                 # Upload image cube (if present) and preset (if present)
158                 self.n477 = ...
                        np.asarray(Image.open(os.path.dirname(self.current_file)+"/Image ...
                        cube/477nm.png"))
159                 self.n478 = ...
                        np.asarray(Image.open(os.path.dirname(self.current_file)+"/Image ...
                        cube/478nm.png"))
160                 self.n490 = ...
                        np.asarray(Image.open(os.path.dirname(self.current_file)+"/Image ...
                        cube/490nm.png"))
161                 self.n500 = ...
                        np.asarray(Image.open(os.path.dirname(self.current_file)+"/Image ...
                        cube/500nm.png"))
162 
163                 self.n511 = ...
                        np.asarray(Image.open(os.path.dirname(self.current_file)+"/Image ...
                        cube/511nm.png"))
164                 self.n523 = ...
                        np.asarray(Image.open(os.path.dirname(self.current_file)+"/Image ...
                        cube/523nm.png"))
165                 self.n538 = ...
                        np.asarray(Image.open(os.path.dirname(self.current_file)+"/Image ...
                        cube/538nm.png"))
166                 self.n549 = ...
                        np.asarray(Image.open(os.path.dirname(self.current_file)+"/Image ...
                        cube/549nm.png"))
167 
168                 self.n553 = ...
                        np.asarray(Image.open(os.path.dirname(self.current_file)+"/Image ...
                        cube/553nm.png"))
169                 self.n563 = ...
                        np.asarray(Image.open(os.path.dirname(self.current_file)+"/Image ...
                        cube/563nm.png"))
170                 self.n577 = ...
                        np.asarray(Image.open(os.path.dirname(self.current_file)+"/Image ...
                        cube/577nm.png"))
171                 self.n591 = ...
                        np.asarray(Image.open(os.path.dirname(self.current_file)+"/Image ...
                        cube/591nm.png"))
172 
173                 self.n600 = ...
                        np.asarray(Image.open(os.path.dirname(self.current_file)+"/Image ...
                        cube/600nm.png"))
```

```
174                 self.n613 = ...
                        np.asarray(Image.open(os.path.dirname(self.current_file)+"/Image ...
                        cube/613nm.png"))
175                 self.n616 = ...
                        np.asarray(Image.open(os.path.dirname(self.current_file)+"/Image ...
                        cube/616nm.png"))
176                 self.n618 = ...
                        np.asarray(Image.open(os.path.dirname(self.current_file)+"/Image ...
                        cube/618nm.png"))
177
178                 with ...
                        open(os.path.dirname(os.path.realpath(self.current_file))+"/Preset.json", ...
                        'r') as fp:
179                     preset = dict(json.load(fp))
180                     fp.close()
181                 if self.current_page == "Page 1":
182                     self.delete_name("e")
183                     self.name_entry.insert(0, list(preset.keys())[0])
184                     self.delete_formula("e")
185                     self.formula_entry.insert(0, preset.get(list(preset.keys())[0]))
186                     self.current_formula = preset.get(list(preset.keys())[0])
187                 elif self.current_page == "Page 2":
188                     self.current_formula = preset.get(list(preset.keys())[0])
189                 elif self.current_page == "Page 3":
190                     self.delete_name(e="")
191                     self.name_entry.insert(0, ...
                            list(preset.keys())[0].replace("_RGB", ""))
192                     self.r_value.set(preset.get(list(preset.keys())[0])[0])
193                     self.g_value.set(preset.get(list(preset.keys())[0])[1])
194                     self.b_value.set(preset.get(list(preset.keys())[0])[2])
195                     self.r_scale.set(preset.get(list(preset.keys())[0])[3])
196                     self.g_scale.set(preset.get(list(preset.keys())[0])[4])
197                     self.b_scale.set(preset.get(list(preset.keys())[0])[5])
198                 elif self.current_page == "Page 4":
199                     self.r_value.set(preset.get(list(preset.keys())[0])[0])
200                     self.g_value.set(preset.get(list(preset.keys())[0])[1])
201                     self.b_value.set(preset.get(list(preset.keys())[0])[2])
202                     self.r_scale.set(preset.get(list(preset.keys())[0])[3])
203                     self.g_scale.set(preset.get(list(preset.keys())[0])[4])
204                     self.b_scale.set(preset.get(list(preset.keys())[0])[5])
205             except:
206                 if self.current_page == "Page 1" or self.current_page == "Page 3":
207                     self.delete_name("e")
208                 if self.current_page == "Page 1":
209                     self.delete_formula("e")
210                 if self.current_page == "Page 3" or self.current_page == "Page 4":
211                     self.r_value.set("")
212                     self.g_value.set("")
213                     self.b_value.set("")
214                     self.r_scale.set(0.5)
215                     self.g_scale.set(0.5)
216                     self.b_scale.set(0.5)
217                 self.current_formula = "No valid formula" == True
218             self.mode = "IM"
```

Start variables

```
# Start a tkinter window with the display-app
root = tk.Tk()

# Run the app as a different user with different access depending on the ...
    parameter given
try:
    user = str(sys.argv[1])
except:
    user = "Tester"
app = XIMEA_DisplayApp(root, user)
# Window cannot be expanded
root.resizable(False,False)
# Start the loop and therefore the program
root.mainloop()
```