# Implementation of Algorithms for the Restoration of Retinal Images

David Samir Toro Hoyos

La Salle University
Faculty of Engineering
Automation Engineering
Bogotá
2016

# Implementation of Algorithms for the Restoration of Retinal Images

David Samir Toro Hoyos

Degree work to obtain the title of engineer in automation

Supervised by
José Antonio Tumialan Borja
Automation engineering

La Salle University
Faculty of Engineering
Automation Engineering
Bogotá
2016

Thesis approval

_____
_____
_____
_____
_____

_____
Jury's signature

_____
Jury's signature

## Acknowledgements

It is always difficult to thank enough to all the people that helped me in this daunting process. Of course, this piece of work would not be possible without the blessing of God so it is for his glory. Also, My work would not be realised without the worry and unconditional help of my family, my dear mother and father, Nayibe and Alonso, and brothers Ignacio, Jorge and Yuri. There were even kind people who offered their help when I needed it, Maria Santos and Lina Maria. I want to thank the family of Lina Maria and my brother Jorge for accompanying and encouraging me when my determination wavered and I needed strengths to continue my work.

## Contents

## List of Figures

## List of Tables

## List of Equations

## List of Algorithms

## List of Codes

## Introduction

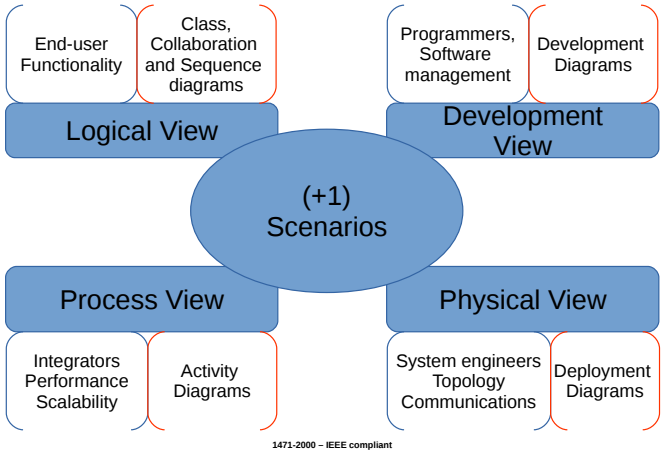
This document was conceived as a phase of a project at the AVARC research group of the Optometry Faculty at La Salle University. The phase consists in the implementation of an algorithm for the restoration of retinal images taken by mobile devices so that it can be further analysed and diagnosed to identify potential patients suffering diabetic retinopathy. It was taken into account the following objectives to ensure the implementation of the application:

1. **Develop algorithm(s) for image restoration:** In which a set of tools (algorithms) were developed for the realization of partial goals dividing the problem into several stages and processing each stage's result to get near the desired application.
2. **Quantitatively validate the application of the restoration algorithm(s):** Each algorithm was validated with its theory and in some cases, a comparison was carried out between its estimation and the desired output.
3. **Implement a restoration application for images:** All the tools were organized and packed into one module according to their category and with some of them a program was developed to offer an automated solution for the restoration of retinal images.

In the world of computer science, some old methods remain robust and effective to solve their intended problems, so in this context some archaic algorithms were included in the main distribution of the program and were not redesigned, as a saying says "there is no need to reinvent the wheel" and thus these algorithms were wrapped into tools. Some algorithms are named in this document but not demonstrated either by its simplicity or because they are thoroughly documented and can be found easily in other sources. All the algorithms are documented and the main ones are tested by means of demonstrations, comparisons or using "unittest" (i.e. Unit testing framework) to ensure robustness and reliability of the code.

Generally the "4+1" View Model (see Figure 1) is widely used to explain complex systems or to convey program models (Phillippe Kruchten, 1995) not only because it complies to the IEEE standard (IEEE, 2000) but because of its simplicity. Despite that, to ensure a clear explanation and understanding of the algorithms this document uses mainly pseudo-codes with some concepts while leaving the code implementation out in the annexes for demonstration purposes. So the "4+1" View Model is partially used here showing in some cases the use classes, some diagrams and leaving out other models. As a plus, additional non-standard tests and demonstrations are given for the reader to confirm that each algorithm works and can be used in practical examples.

*Figure 1*. "4+1" View Model




Due to that it is not the main purpose of this study to show the complete functionalities of the algorithms but just their implementation some of them are explained using pseudo-code (used to code to actual algorithm or to abstract long ones), others

implemented in their language (to emphasize details and functionality) and in some cases other resources (e.g. explanation, class diagrams, outputs, etc.). In this way, codes that explains themselves, are too short or depend heavily in the programming language definitions are briefly discussed with a Minimal Working Example (MWE ) or their pseudo-code not necessarily corresponding to the implemented one. More information is in the appendix and the reader is invited to review the external sources for completeness and the algorithms not explained in this document.

## Glossary

The concepts used in this document are a bit esoteric due to the vast majority of specialized procedures used in the algorithms but the intent of this is to make them as simple as possible to let the reader understand the document more easily. Below is a list of key words used throughout the document to encapsulate special concepts that may clarify and convey a better understanding of their meanings:

**data persistence**

In the context of a program, it consists in storing processed data in a "persistent" form to a non-volatile storage so that the next time the program is run it does not process the data again. This is useful for information that is frequently accessed but that is difficult to obtain e.g. Objects can be stored as byte streams (serialized) and then recreated back (de-serialized) when needed in a program.

**De-serialization**

It is the contrary process of serialization.

**Serialization**

It is the process of converting an object into a stream of bytes in order to store the object or transmit it to memory, a database, or a file. Its main purpose is to save the state of an object in order to be able to recreate it when needed.

**Level**

It represents a value that any pixel can take on an image i.e. there are usually 256 levels in an gray image and 2 levels in a binary image.

**Channel**

It refers to the color band in an image i.e. BGRA images have 4 channels compared to the single channel in GRAY images.

**Histogram**

Graphical representation (or its data equivalent) of the intensity distribution or the number of pixels in each level (intensity value) of a channel in an image.

**Intensity**

Pixel-level or the pixel's value.

**BGR image**

Refers to a RGB image with its channels reversed, thus channel 0 is for blue color conversely to Red in RGB images.

**BGRA image**

It is an RGB image that contains an additional A channel to support additional information like transparency for PNG formats.

**Descriptor**

Is the description of a feature in an image where elementary characteristics such us shape, color, texture, etc are taken to describe the feature.

**Threshold**

It is a type of image segmentation at which an image is partitioned into a foreground and background to isolate objects by converting from a gray-scale image to a binary image. Threshold can also refer to the value used to partition the image but for the sake of clarity it is fully used as threshold value.

**Segmentation**

It is the partitioning of an image into simpler and meaningful segments for the simplification of its analysis.

**Mask**

**Perspective transform**

**Filter**

**hashable**

**hashing**

**Numpy Arrays**

(SciPy, 2015b) In python one of the best ways to represent an array or matrix is using the numpy module. For mathematical matrices numpy provides a class called "matrix" and for arrays with element-wise operation support it provides the ndarray class. Throughout all the algorithms, arrays of the ndarray class are used extensively so it is important to understand its basic structure.

**ndarray.ndim.** The number of axes (dimensions or rank in python) of the array. Arrays of more than one dimension are like matrices but with a different behavior from their mathematical counterparts.

**ndarray.size.** The total number of elements of the array.

**ndarray.shape.** A tuple indicating the size of the array in each dimension. For a matrix with $n$ rows and $m$ columns the shape would be $(n, m)$ therefore the length of the shape is the ndarray.ndim and the product of each element in it is ndarray.size.

**ndarray.dtype.** A python object describing the type of the elements in the array (i.e. all elements in an array must be of the same type). Some examples are numpy.int32 or numpy.int16 for for integers, and numpy.float32 or numpy.float64 for floats, with the numbers at the end of these types being the bytes in which they are structured or can allocate data in memory but be aware that some of them differ to this naming convention (more can be read in the numpy documentation).

Numpy arrays and matrices work a little different than the operations in MATLAB but with some relations than can be easily adapted from one way to the other. These relations can be better explained in (SciPy, 2015a).

To see the common terms used addressing python code see (Python Software Foundation, 2015).

## Art State

Diabetes is a chronic disease that occurs when the pancreas does not produce enough insulin or the body cannot effectively use the insulin it produces to regulate blood's sugar causing severe complications over time. These complications affects especially the nerves and blood vessels causing macro and micro vascular changes leading to increases in the risk of heart disease and stroke (50% of people with diabetes die of cardiovascular disease), renal problems, neuropathy (nerve damage) and diabetic retinopathy (DR) (Faust, Acharya U., Ng, Ng, & Suri, 2012). Type 1 diabetes (T1D, previously known as insulin-dependent, juvenile or childhood-onset) is characterized by deficient insulin production in the pancreas requiring daily administration of insulin to the patient; currently its cause is not known and it is not preventable (this does not mean that some measurements should not be taken). Type 2 diabetes (T2D, formerly called non-insulin-dependent or adult-onset) is the most chronic worldwide which comprises 90% of people with diabetes around the world (WHO, 2016; WebMD, 2014).

The rate of diabetes is increasing, according to The World Health Organization (WHO) and International Diabetes Federation (IDF) in the year 2000 the number of people with diabetes were of 171 million and it was estimated to increase to 366 million by 2030 being by then the 7th leading cause of death (WHO & IDF, 2006), but just in 2014 the number of people with diabetes rose to 422 million and caused over 1.5 million deaths worldwide in 2012. This is probably going to continue to raise as it is consistent with the global prevalence of diabetes among adults (i.e. over 18 years of age) which has risen from 4.7% in 1980 to 8.5% in 2014. This only worsens in low and middle-income countries where it is estimated that more than 80% of diabetes deaths occur and diabetes prevalence has been rising more rapidly in the last years (WHO, 2016).

Many studies concord that people with diabetes are at risk of developing diabetic retinopathy (DR) ultimately leading to blindness as a result of long-term accumulated damage to the small blood vessels in the retina. Even though people with diabetes are 25 times more likely to develop blindness and DR is considered to have caused one percent of global blindness only one-half of the patients are aware of this disease. The most effective treatment for it can only be administered at the first stages of the disease thus regular DR screening is of paramount importance and recommended annually, but most developing countries lack the ability to fully record these DR cases.

To prevent it, many models has been considered to address this problem (Askew et al., 2012). Thanks to the resent years advancements in technology and the access to it have facilitated the acquisition of advanced appliances that allow fast and reliable DR screenings in which digital imaging technology plays a key role allowing to employ state-of-the-art processing techniques to automate the detection of abnormalities in retinal images (Faust et al., 2012).

One method in particular can take huge advantages from telemedicine and digital imaging for diagnosing DR and that is the utilization of tele-ophthalmology by obtaining digital retinal images with specialized cameras (fundus cameras) and electronically transmitting them to an expert for their assessment and patient diagnosis. As a solution, it is an efficient alternative for early diagnostic and treatment of diabetic patients.

In addition, tele-ophthalmology involves the following benefits (Martínez Rubio, Moya Moya, Bellot Bernabé, & Belmonte Martínez, 2012):

- Unlike normal methods, providing healthcare to a high number of patients.
- Reducing waiting periods.
- Saving time in the diagnostic and treatment.
- Avoiding delays which can produce health problems for patients.
- Digital ocular fundus photographs are of low cost and of little discomfort for the patient.
- The application of telemedicine reduces the workload to experts letting them treat more outpatients efficiently.

Taking a particular subdivision from the ophthalmological screening methods are the non-mydriatic screening techniques. Numerous papers have demonstrated that it is one of the health interventions with the best cost-effectiveness ratio, high sensitivity and specificity. Non-mydriatic retinal cameras allow high-quality photographs through undilated pupils in digital format

offering benefits like facilitating medical compliance (i.e. patient correctly following medical advice), patients not requiring to be still at the assessment and images delivering a much broader field of view (Askew et al., 2012).

Specialized non-mydriatic fundus cameras can be cost-efficient in operational terms but exhibit a relative medium inversion cost, making them difficult to afford in some cases. Yet again, technology can offer solutions to further reduce costs while offering decent results that can be at pear with what would be obtained with professional fundus cameras and that is a mobile device (e.g. cellphones, tablets, etc) which is a ubiquitous device that offers many functionalities including the acquisition of digital images and though it is not cheap it is more likely that anyone have one than a non-mydriatic fundus camera, thus preventing the necessity to acquire one.

Screening of patients with adapted mobile devices can become a highly valid method for the detection and prevention of DR allowing early access to assessments while offering easiness of use and other benefits like availability.

## Programming and Research Approach

Before we discuss the algorithms, it is important to understand the context in which they were conceived. Many programming languages were taken into consideration for the implementation of the algorithms but few sufficed the requirements of high performance while keeping the readability, portability, modularity and scientific basis that a language needs. Compiled languages like C++ offer high speeds but at the cost of lengthy codes and long developing times while their contra parts, interpreted languages like MATLAB and Python, sacrifice execution speed to offer short and fast code prototyping (i.e. while developing) because of their simple yet readable languages and even offering in some cases means to improve speed by compiling or translating to other languages like C or using them as back-ends (Cython.org, n.d.; Pyzo, n.d.).

Table 1
*Comparison of programming languages*

| **Factor** | **C++** | **MATLAB®** | **Python** |
|---|---|---|---|
| **Interpreter type** | compiled | just-in-time | Interpreted |
| **Language abstraction** | High-level | High-level | High-level |
| **Standard** | ISO standard | de-facto standard | de-facto standard |
| **License** | Open source | Licensed | Open source |

Source: Author

Table 2
*Comparison example of "for loops" in C and Python*

C for loop

```c
#include <stdio.h>
#include <string.h>
int main() {
char s[5] = "Hello";
for(int i=0;i<strlen(s);i++){
    printf("%c", s[i]);
    return 0;
}
```

Python simulation of C for loop

```python
s = "Hello"
for i in range(len(s)):
    print s[i]
```

Pythonic for loop

```python
s = "Hello"
for letter in s:
    print letter
```

Source: Author

### Optimization Methods

Here some optimization techniques are explained which were used in the implementation of the algorithms.

**Process big data as small data.** This optimization method consist in reducing the quantity of elements in data without loosing relevant information. Some examples involve normalizing data to small values (i.e. normalize to 1) to reducing the size of arrays. In the case of images for example it can produce performance penalties if it is too big to process (e.g. $10000x8000$) or not for the convenience of the algorithm. The solution is just to resize it (e.g. $500x400$) using a good interpolation (though linear interpolation does fine for most cases), process the resized image and at the end resize to the original image. For data manipulation like points or domain transformations the process it similar (in fact resizing is a common transformation) and **?? ??** covers it in more detail.

**Lazy evaluation.** Lazy evaluations which in this case can be described as load on demand. This is advantageous to keep RAM memory free of data not in use and load them from disk when needed. In the case of structures, it is used to compute data only when it is going to be used. Python offers good solutions as the *generators* and in the case of classes the `@property` decorator to convert a method to a getter, setter or deleter useful to process a requested value on the go instead of processing everything at the class instantiation.

**Preventing circular references.** There is of up most importance to keep algorithms from producing object with circular references (object referenced by other objects keeping the object alive) which can produce memory leaks, instability and crashes due to exceeded memory use. In the case of python there exist a garbage collector (it can be imported as *gc*) to manage memory in the programs but currently objects with circular references cannot be collected and are kept alive until the whole program ends. To prevent it the *weakref* module can be use to provide weak references to an object which do not prevents it from being garbage collected.

**Multitasking.** Multitasking techniques for heavy tasks. Usually, multiprocessing was used to take advantage of multi-core computers and multithreading for lighter tasks or for task incompatible with the multiprocessing API.

**Memory mapped files.** It is another useful technique to keep memory free at expense of more processing time caused by memory being used not in RAM but in the local disk. This is also useful to compute big data which cannot be loaded to RAM memory and to let data be accessed for multiple tasks at the same time (Usually shared memory between programs is a really difficult achievement but it can be easily done using memory mapped files).

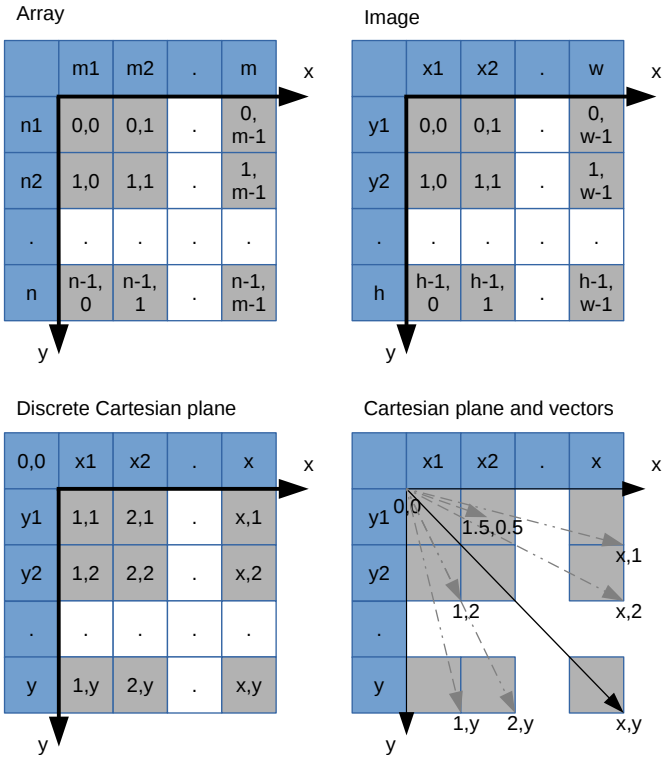
**Memoization and caching.** Saving data with expensive function calls or difficulty to obtain them for later use. In this way, values are cached to be used in the same session or persisted to disk to keep these computations between sessions. The persistence of data use a serialization and de-serialization process which involves saving and loading information from disk. In Python, this is achieved using the package 'pickle' and the caching using hashes to generate key-like values usually in dictionary structures (other structures can be used too).

### Images in openCV

An image can be represented as a matrix of width $w$ and height $h$ with its elements as pixels, where each pixel is a representation of a color in one point of a discrete plane (2D dimension). In the case of OpenCV and many other libraries for image manipulation, the use of Numpy arrays as the base for image representation is becoming the standard (Numpy is a fast and powerful library for array manipulation and one of the main modules for scientific development in python). A Numpy array is multidimensional container of items of the same type and size differing mainly to the mathematical matrix in which its operations are element-wise (i.e. corresponding index in each array). The Image shape, hight and width $(h, w)$ , then corresponds to a Numpy array with $n$ rows and $m$ columns $(n, m)$ which in a Cartesian plane would be the $(y, x)$ axes (Scipy, 2015).

Figure 2 shows the different representations used in this document and how they are accessed or indexed. Images and arrays are indexed from 0 to $N-1$. Notice that images are interchangeable with arrays and can be treated like so with pixels represented by the array elements. Where the first pixel $h, w = (0, 0)$ corresponds to the first element $n, m = (0, 0)$ spanning from the origin $y, x = (0, 0)$ to the end of it where other elements being. In the context of discrete planes arrays continue to be indexed natively from 0 to $N-1$ (the structure does not change) but are treated with an additional 1 so that arrays representing discrete planes start from 1 and finish in $N$. To use vectors arrays are referenced as in the discrete Cartesian case but indexes

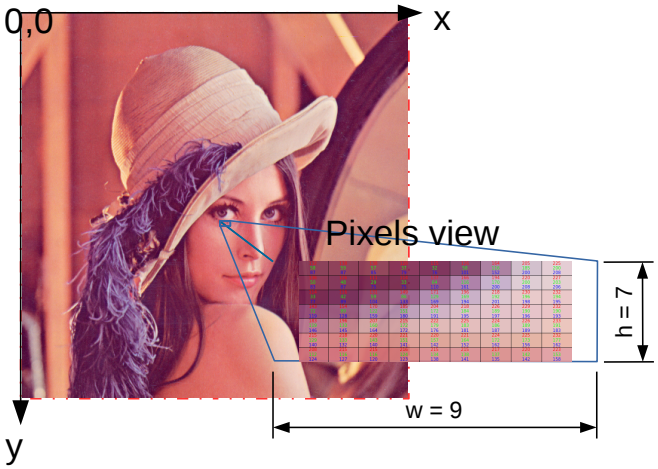
*Figure 2.* Used representations




are subdivided usually using interpolation so that vectors are represented using floating points but converted to integer types like in the previous cases to access the pixel's value.

Figure 3 better illustrates the notion of arrays representing images with the height as the number of rows and the width as the number of columns. Notice that from the Cartesian plane perspective the $x$ axis goes to the right and the $y$ axis downwards with the *origin* at the north-west of the image which corresponds to the $y, x = (0, 0)$.

*Figure 3.* Axes orientation of images



## Algorithms design

There always emerge important questions in the development of every project like how should it be made? perhaps this peace of code is not robust enough for every possible scenario? or is it the best solution to my problem?. Well, not all solutions are absolute and there could be better implementations but for the most of it I consider that experimentation is one of the best insurance methods that exists for scientific solutions. As I see that without experimentation, a solution may not be well tested and it could not be pushed forward to the limits of their potential. Another good reason is that it opens room for creativity that could lead to unexpected inventions not yet discovered. So in the next section is presented some of the experiments that led to the implementation of the main algorithm. The repository with all the codes is in (David, 2016b). Among the repository's folders there are two containing most of the algorithms, one called *tests* where experiments were carried out and a package called *RRtoolbox* with developed algorithm used in the project. Figure 4 shows the file structure of the *RRtoolbox* package to provide a better visualization of it.

There are more algorithms provided by the *RRtoolbox* package and other non-package codes left for tasting that is not explained in next sections but the list would go so long that a book could be written about them. The other algorithms not covered here are left as task for the reader to discover in the *RRtoolbox* package or reading the manual in David2016RRtoolboxDoc. Some of them are even more complex and useful than the presented in this document but were left out for they are not relevant to the project's objectives.

### General algorithms

There are many algorithms that prove useful in the manipulation of arrays that can help in the realization of more complex algorithms. These general algorithms present some of the most simple yet useful procedures for common array manipulation.

**The area under a polygon (*poligonArea*).** The area under a polygon can be calculated if the points of its vertices are known. This is described in (Darel Rex Finley, n.d.) which can be expressed by the general equation of the *poligonArea* as:

$$poligonArea = \frac{1}{2}\left|xsum - ysum\right|$$

Where *ysum* and *xsum* are defined as:

$$ysum = x_0 y_n + \sum_{i=0}^{n-1} x_{i+1} y_i$$

$$xsum = x_n y_0 + \sum_{i=0}^{n-1} x_i y_{i+1}$$

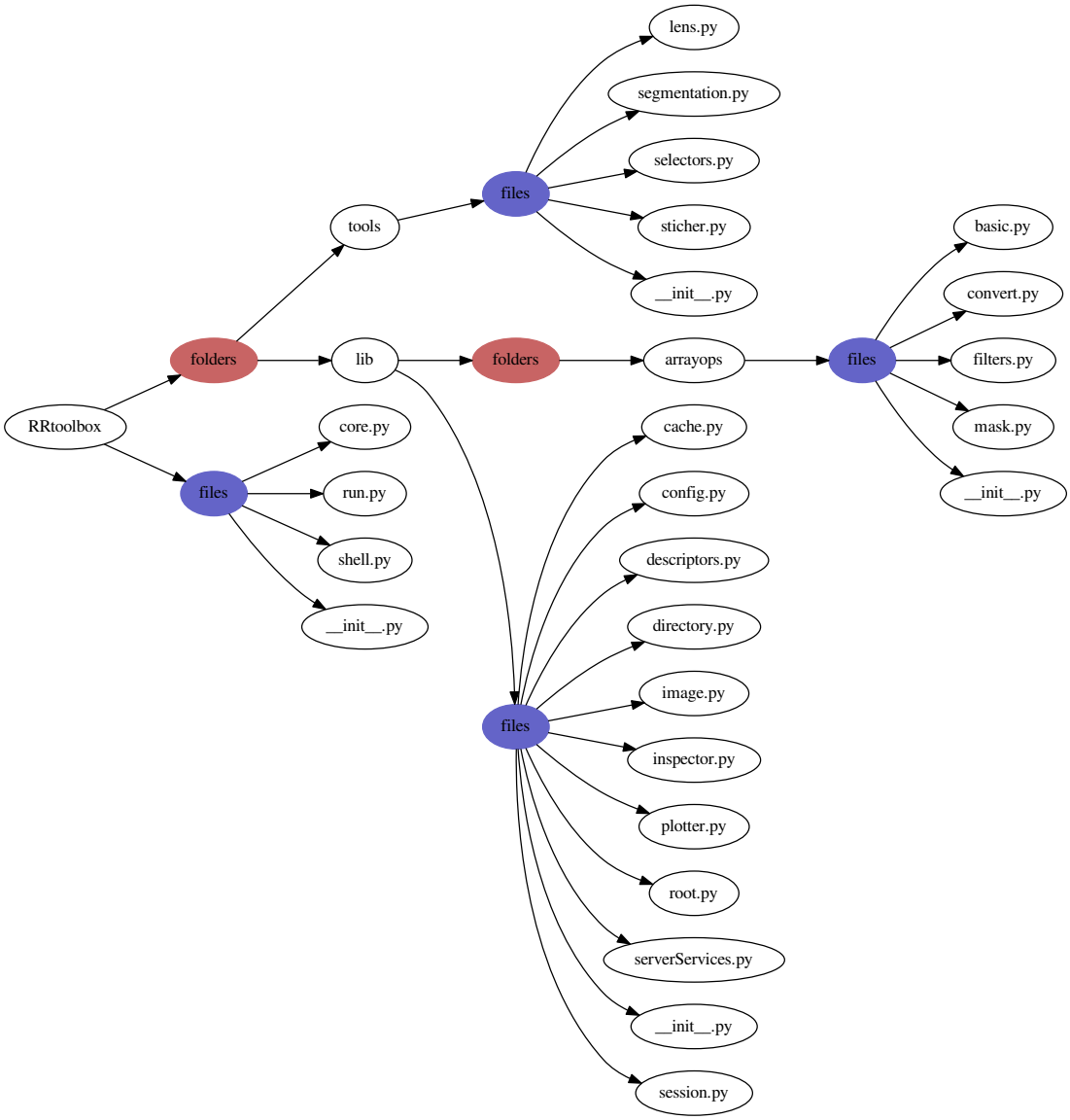
Then replacing *xsum* and *ysum* results in Equation 1:

$$poligonArea = \frac{1}{2}\left|-x_0 y_n + x_n y_0 + \sum_{i=0}^{n-1}(x_i y_{i+1} - x_{i+1} y_i)\right| \tag{1}$$

This equation can calculate the area of any arbitrary polygon but with the limitation that the points that conform it must be in adequate order (i.e. if lines are traced passing at each point in order then these lines must not intersect each other). If for example we have the following points (see Figure 5):

$$points = \begin{bmatrix} -3 & -2 \\ -1 & 4 \\ 6 & 1 \end{bmatrix}$$

*Figure 4*. File estructure of *RRtoolbox* package



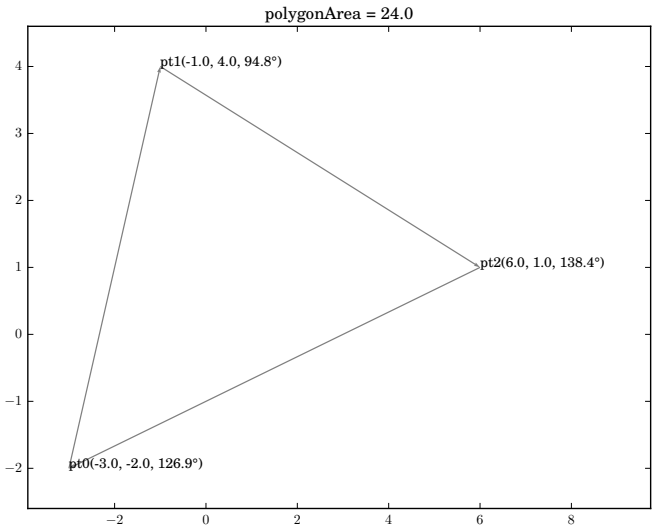

Then we solve it as in Equation 1 to obtain the individual summations.

$$
\begin{aligned}
ysum &= x_0y_2 + x_1y_0 + x_2y_1 \\
&= (-3)\cdot 1 + (-1)\cdot(-2) + 6\cdot 4 \\
&= 23 \\
xsum &= x_0y_1 + x_1y_2 + x_2y_0 \\
&= (-3)\cdot 4 + (-1)\cdot 1 + 6\cdot(-2) \\
&= -25
\end{aligned}
$$

Which is then replaced in *poligonArea* giving the result:

$$poligonArea = Abs((-25) - 23)/2 = 24$$

*Figure 5.* Find the area of a polygon using Equation 1




The *polygonArea* method in Equation 1 is imported in Python as `from RRtoolbox.lib.arrayops.basic import polygonArea_calcule, polygonArea` with `polygonArea_calcule` as the actual function and `polygonArea` the alias. There are other implementations that calculates the area of a polygon and where used to provide more options and test them with each other and can be imported using `from RRtoolbox.lib.arrayops.basic import polygonArea_contour, polygonArea_fill`. The `polygonArea_contour` is a wrapper over the OpenCV method `cv2.contourArea` but with more functionalities and `polygonArea_fill` calculates the area filling the objects and summing up the number of pixels which is comparable to an invariant moment.

**Overlay.** This method as its name implies is used to overlay a foreground ($Fore$) over background ($Back$) array with an $alpha$ transparency which can be either and array or a number. The following equation is used to with an $alpha$ array:

$$Overlay_{i,j} = Fore_{i,j} \cdot alpha_{i,j} + Back_{i,j} \cdot (1 - alpha_{i,j}) \qquad \forall \quad ||alpha_{i,j} \leq 1|| \tag{2}$$

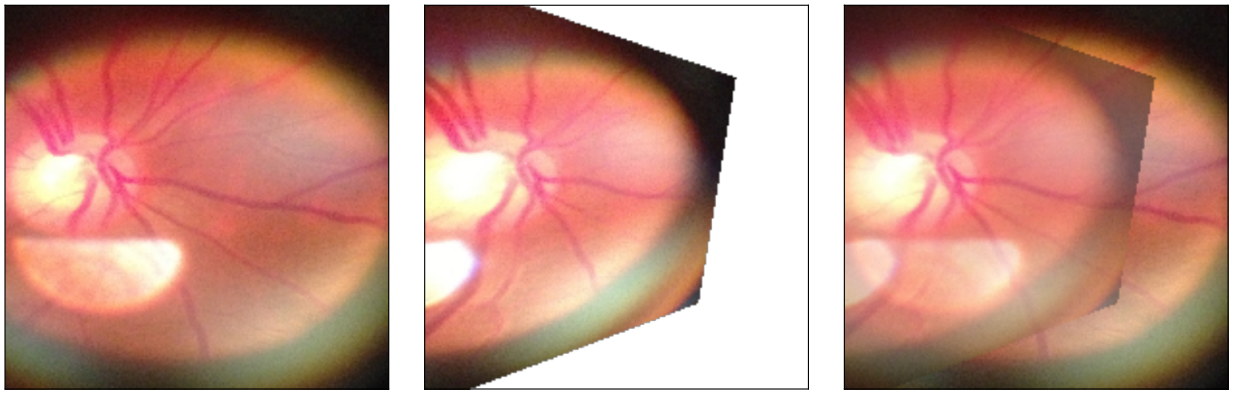
Where $alpha$ is an array with numbers constrained from 0 to 1. Notice that values less than 1 attenuates $Fore$ array and values more than 0 attenuates the $Back$ array. This can lead to the notion of $alpha$ being a transparency parameter. Equation 2 is extended using *Numpy* and is not necessarily True to operate over $Fore$ and $Back$ arrays of different dimensions because it uses propagation to handle arrays which uses a different mechanism complying to certain rules.

Figure 6 shows an overlay example of the implemented Equation 2 using *Numpy*. In this case $alpha = 0.5$ which is a floating number instead of an array. This function can be imported using `from RRtoolbox.lib.arrayops.basic import overlay` which offers more functionalities than in Equation 2.

**Real image – Rendered image.** There are situations in which we want to 'view' inside an image to show or select the parts of the image with more detail (Zooming-in) or that the image is too big to render in a visualization program and the solution it to resize it to a small image but conserving the coordinates of where the colors are in the original image. This algorithm was developed to comply to those needs using pure mathematics with familiar equations and some constraints.

Suppose that there exists a real image $I$ with shape $(H, W)$ for height and width respectively which has a rendered image $I_r$ with shape $(H_r, W_r)$ of any view inside $I$ (i.e. any part of $I$ can be selected and cropped with a different resolution than $I$ that is useful for zooming-in). To calculate the coordinate system in $x, y$ of the real image $I$ to the coordinate system $x_r, y_r$ of the rendered image $I_r$ where $x_r, y_r$ is a point in the domain of $I_r$ and $I_r$ is a ROI contained between $x_2 - x_1$ and $y_2 - y_1$ in image

*Figure 6*. Overlay example using Equation 2



(a) background (b) foreground (c) Overlay with alpha = 0.5

$I$ (i.e. $I_r$ is a re-scaled ROI of $I$ with a cropping box from the top-left point $x_1, x_2$ to the bottom-right point $x_2, y_2$) then the following equations apply:

To convert from $I$ to $I_r$ coordinates:

$$x_r(x),\ y_r(y) = W_r \cdot \frac{(x - x_1)}{(x_2 - x_1)},\ H_r \cdot \frac{(y - y_1)}{(y_2 - y_1)} \tag{3}$$

To convert from $I_r$ to $I$ coordinates:

$$x(x_r),\ y(y_r) = x_1 + x_r \cdot \frac{(x_2 - x_1)}{W_r},\ y_1 + y_r \cdot \frac{(y_2 - y_1)}{H_r} \tag{4}$$

With the constraints:

$$(H_r,\ W_r) = (y_{rmax} - y_{r0},\ x_{rmax} - x_{r0}) \begin{cases} x_{r0} < x_{rmax} & \in I_r \\ y_{r0} < y_{rmax} & \in I_r \\ x_{max} \geq x_2 > x_1 \geq x_0 & \in I \\ y_{max} \geq y_2 > y_1 \geq y_0 & \in I \end{cases}$$

A complete implementation of a plotter class called *Plotim* was implemented using Equation 3 and Equation 4 for the rendering of images. This class can be used by importing it in Python with `from RRtoolbox.lib.plotter import Plotim`. Other inherited classes from *Plotim* are found in `from RRtoolbox.lib.plotter` and demonstrate the capabilities of these simple equations.

**Domain transformations: scaling to original results.** As explained in Real image – Rendered image section , images can be rendered afferently for some needs but still processing them as the original. This is one of the techniques used to reduce processing times and standardize input images by resizing the original image $I_o$ to a scaled image $I_s$ which is used in the actual algorithm's processing and then adapting the results to be applied back in the original image. But there surges a pretty obvious problem, how to convert the processed results of $I_s$ back to $I_o$? In the case of coordinates or points this can be solved using linear relations to convert from one domain to the other (explained in Real image – Rendered image section ) but if the result is a transformation matrix (TM) then additional transformation matrices (TMs) must be applied to adjust it to work with the original sample.

***Perspective transformation rules.*** Before going into mathematical detail lets define some rules and conventions useful for perspectives transformations. These conventions are used when dealing with transformations matrices:

- When the transformations are with respect to the absolute Cartesian system it is referred as xyz and all TMs must be pre-multiplied at each step.
- When the transformations are relative with respect to the previous position it is referred as uvw and all TMs are post-multiplied at each step.

The perspective transform maps from $x, y$ coordinates to $u, v$ coordinates using transformation matrices. To transform any $x, y$ point in an image a $3x3$ transformation matrix $M$ is multiplied with the column vector $\begin{bmatrix} x & y & 1 \end{bmatrix}$ which yields another column vector $\begin{bmatrix} X' & Y' & c \end{bmatrix}$:

$$\begin{bmatrix} M_{0,0} & M_{0,1} & M_{0,2} \\ M_{1,0} & M_{1,1} & M_{1,2} \\ M_{2,0} & M_{2,1} & M_{2,2} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ c \end{bmatrix}$$

Then the resulting $u, v$ point where $x, y$ is moved is calculated as:

$$u, v = \frac{x'}{c}, \frac{y'}{c}$$

Suppose that a scaled transformation matrix ($M_s$) is found to convert the perspective from a scaled image A ($A_s$) to the perspective of a scaled image B ($B_s$). So $M_s$ is used to convert $A_s$ to the domain of $B_s$ but the objective is to convert the original image A ($A_o$) to the domain of the original image B ($B_o$) with a original transformation matrix ($M_o$). Several steps are summarized to achieve this:

1. Use a scaling transformation $A_{os}$ to transform the points from $A_o$ to the domain of $A_s$.
2. Use $M_s$ to transform the points of $A_s$ to the domain of $B_s$.
3. Use a scaling transformation $B_{so}$ to transform the points from $B_s$ to the domain of $B_o$.

Now for the mathematical abstraction.

$$\begin{aligned} M_o &= B_{so} \cdot M_s \cdot A_{os} \\ &= \begin{bmatrix} Bx_{so} & 0 & 0 \\ 0 & By_{so} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} M_{s11} & M_{s12} & M_{s13} \\ M_{s21} & M_{s22} & M_{s23} \\ M_{s31} & M_{s32} & M_{s33} \end{bmatrix} \begin{bmatrix} Ax_{os} & 0 & 0 \\ 0 & Ay_{os} & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} Ax_{os} \cdot Bx_{so} \cdot M_{s11} & Ay_{os} \cdot Bx_{so} \cdot M_{s12} & Bx_{so} \cdot M_{s13} \\ Ax_{os} \cdot By_{so} \cdot M_{s21} & Ay_{os} \cdot By_{so} \cdot M_{s22} & By_{so} \cdot M_{s23} \\ Ax_{os} \cdot M_{s31} & Ay_{os} \cdot M_{s32} & M_{s33} \end{bmatrix} \end{aligned}$$

Where

$$Ay_{os} = \frac{Ah_s}{Ah_o},\ Ax_{os} = \frac{Aw_s}{Aw_o},\ Bx_{so} = \frac{Bw_o}{Bw_s},\ By_{so} = \frac{Bh_o}{Bh_s}$$

With $(Aw_o, Ah_o)$ as the shape of $A_o$, $(Aw_s, Ah_s)$ the shape of $A_s$, $(Bw_o, Bh_o)$ the shape of $B_o$ and $(Bw_s, Bh_s)$ the shape of $B_s$. Then $M_o$ can be rewritten as:

$$M_o = \begin{bmatrix} \left(\frac{Aw_s}{Aw_o}\right)\left(\frac{Bw_o}{Bw_s}\right) M_{s11} & \left(\frac{Ah_s}{Ah_o}\right)\left(\frac{Bw_o}{Bw_s}\right) M_{s12} & \left(\frac{Bw_o}{Bw_s}\right) M_{s13} \\ \left(\frac{Aw_s}{Aw_o}\right)\left(\frac{Bh_o}{Bh_s}\right) M_{s21} & \left(\frac{Ah_s}{Ah_o}\right)\left(\frac{Bh_o}{Bh_s}\right) M_{s22} & \left(\frac{Bh_o}{Bh_s}\right) M_{s23} \\ \left(\frac{Aw_s}{Aw_o}\right) M_{s31} & \left(\frac{Ah_s}{Ah_o}\right) M_{s32} & M_{s33} \end{bmatrix} \tag{5}$$

A family of applications and functions are found in the *RRtoolbox* package using the concept of converting data for convenience and the reconverting results to the intended target. In particular many of these methods can be found in `RRtoolbox.lib.arrayops.convert` where the particular case presented in Equation 5 can be imported in Python as `from RRtoolbox.lib.arrayops.convert import sh2oh`.

**Normalization.** Array normalization is useful to convert all the values to a more comfortable representation or to standardize them. An array $A$ is normalized to the range $[0, 1]$ with the simple function:

$$N(A) = \frac{A - min(A)}{max(A)} \tag{6}$$

Where *min* and *max* are functions that return the minimum and maximum element of $A$ respectively. The normalization is imported in Python using `from RRtoolbox.lib.arrayops.basic import normalize, normalize2, rescale` with `normalize2` and `rescale` being variants of the normalization.

### Load functions

In python there are several ways to load an image with functions from packages that provide different features making it easy to provide support to load images from different sources, formats and specifications. This capability was exploited by coding a way to seamlessly load images when provided the path to the source and determine the means to retrieve them. A functional implementation can be imported using `from RRtoolbox.lib.image import loadFunc` which creates a loader function to use when quickly loading many images with the same loading configuration or to keep images in disk and load them on the fly with the desired configuration (i.e. with certain shape, any supported color transformation and from desired sources like URLs, sockets, memmaped files or supported formats). The `loadFunc` in most of the implementations of the *RRtoolbox* package and has become kind of a standard in it. More useful functions and classes are found in `RRtoolbox.lib.image`.

**Load from sockets.** Loading an image from a socket is achieved using the the socket package from python by defining a server and a client with a custom simple protocol to transfer pickled data.

**Load from URLs.** To load images from an Uniform Resource Locator the urllib family of modules from python can be used. In python there is urllib, urllib2 and urllib3 with some differences between python 2 and python 3. For the implementation it was used urlopen from urllib3 and urllib.request from python 2 and 3 respectably which returns a handle similar to a file-like object created by the built-in `open` object used to open image's URLs. These two APIs being similar facilitates more the implementation.

**Load from files.** It can be achieved directly using the OpenCV function `cv2.imread` or loading images' files using the build-in `open` function, retrieving the binary string, converting it to an array with `np.fromstring` and then decoding the resulting array with `cv2.imdecode` to obtain the image.

**Load from memmaped files.** This is one of the ways to save memory to a file and access that file directly from the disk instead of keeping it in the RAM. To create and retrieve a binary file in *NumPy* format (i.e. extension .npy) the save and load functions are provided in the `numpy.lib` module which creates a binary representation of a *NumPy* array that can be simply loaded to memory, pickled (technically it is the pickled file itself different to the string representation used for other purposes e.g. store in databases as a string and not as a external file), transferred remotely, accessed with standard read and write modes and still behave like a normal *NumPy* object which can be referenced or shared between processes (useful for multitasking).

### Pre-processing, Filters and Enhancing Methods

**Histogram equalization.** Histogram equalization ($HE$) distributes the colors equally in an image based from its histogram, so if an image is bright most pixels will be confined to the highest levels or if an image is dark most pixels will be at the lowest levels and $HE$ will distribute this levels to the levels that are lacking pixels improving the contrast of the image. That way images of the same scenery with different light conditions will be almost the same after image equalization or histograms confined to particular regions will be distributed to all regions (K & Mordvintsev, 2013).

To cover the full spectrum a transformation function or lookup table is needed to map from the input pixel of the specific regions to the output pixels of the full region. This is accomplished using the cumulative distribution function ($CDF$), in a sense *HE* helps standardize the $CDF$ of an image when distributing all the colors equally throughout it. Code 1 is a MWE which demonstrates the histogram equalization. Additional explanations in (Gonzalez, Woods, & Eddins, 2004, section 3.3.2).

Code 1: Histogram equalization MWE

```
import cv2
import numpy as np
img = cv2.imread('image.jpg',0) # load the image
hist,levels = np.histogram(img.flatten(),256,[0,256]) # get histogram
cdf = hist.cumsum() # get CDF by accumulating histogram
cdf_m = np.ma.masked_equal(cdf,0) # mask to convert to range [0,1]
cdf_m = (cdf_m - cdf_m.min())*255/(cdf_m.max()-cdf_m.min()) # scale to range [0,255]
cdf = np.ma.filled(cdf_m,0).astype('uint8') # mask as a look-up table
img2 = cdf[img] # apply the mask to transform the inputs pixel to output pixels
```

OpenCV has a function to apply *HE* called `cv2.equalizeHist` where its input is a grayscale image and the output is the equalized image.

A drawback is that it won't work well where there are large intensity variations, if for example a histogram is bimodal with the two peaks apart of each other (i.e. one peak at low levels and the other at high levels) or histogram is not confined to a particular region then these will be equalized causing lost of details in some cases due to saturation (where there is saturation lost of information is present too). To prevent it openCV provides a Contrast Limited Adaptive Histogram Equalization (*CLAHE*) function that is applied to sub-regions in the image. This approach does not consider the global contrast of the image as in *HE* preventing the problems that this implies, in contrast *CLAHE* divides the image into small blocks called *tiles* of typically $8x8$ (the parameter is *tileGridSize*) to apply *HE* in each one and that way the histogram would be confined to small regions (solving the particular problems in *HE*) but this brings up another problem, if noise is present it will be amplified. To avoid this, a contrast limit is applied to any level above a specified contrast (the parameter is *clipLimit* with 40 as its default value) in a process where the pixels are clipped and distributed uniformly to other levels before *HE* is applied. Bilinear interpolation is applied to remove artefacts created at the borders of each tile after each individual equalization (K & Mordvintsev, 2013).

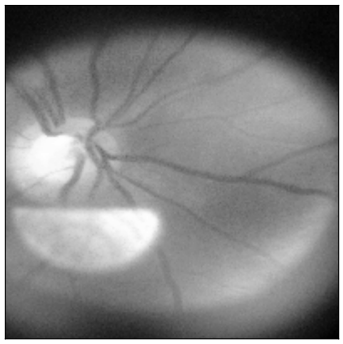
In general *HE* techniques are used in pre-processing operations to usually normalize the image patterns and lighting conditions (Médioni, 2005) but it has been observed that it intensifies noise (Antal & Hajdu, 2012) and for that noise attenuation is recommended before using *HE* (Sopharak, Uyyanonvara, & Barman, 2013). For further optimizations using C++ and CUDA to process in the computer's GPU you can follow the advices in (Fierval, 2015).

Figure 7 shows a *HE* and *CLAHE* comparison with their respective histogram of colors. Notice the input image (Figure 7a) and how the *HE* equalizes the input (Figure 7c) by converting its non-linear $CDF$ (Figure 7b) in an uniform $CDF$ (Figure 7d). This leads to some uneven parts in the image but *CLAHE* solves it as explained before (Figure 7e).

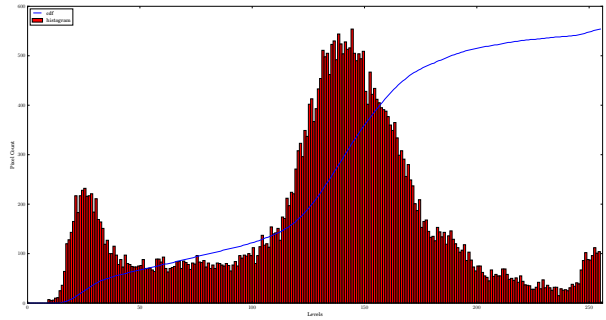
Figure 8 shows additional experiments with the $CLAHE$ method using color images. The default application with parameters $tileGridSize = (8, 8)$ and $clipLimit = 2.0$ is found in Figure 8b. In Figure 8c the parameters $tileGridSize = (30, 30)$ and $clipLimit = 2.0$ are used and in Figure 8d changed to $tileGridSize = (8, 8)$ and $clipLimit = 15.0$. This example demonstrates that in color images the application of *CLAHE* with an incorrect selection of parameters can lead to unappealing results.

**Matrix decomposition to reduce levels.** Matrix decomposition offers a different representation of the original image array and lets the information to be changed easily while affecting to all pixels instead to a single one. This can reduce the diversity of image levels to offer less colors than the original and reduce file storage but at an unwanted price, less details and added artefacts like edges that can prevent the image to be adequately computed. Code 2 shows an implementation using *NumPy* and the developed package *RRtoolbox* explained in other sections.

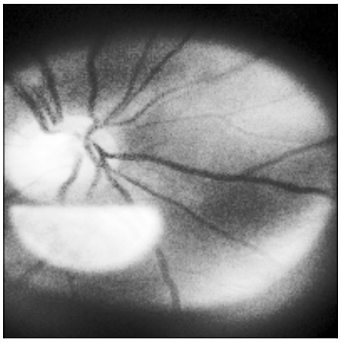
*Figure 7*. Histogram equalization example



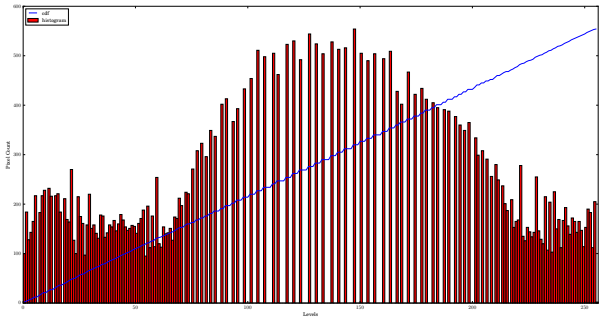
(a) Input



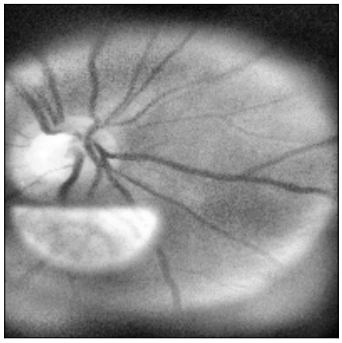
(b) Histogram of input



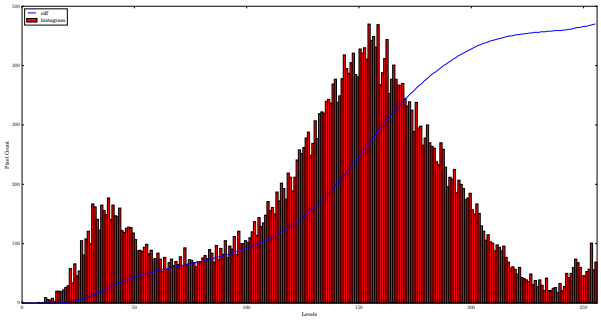
(c) global Equalization



(d) Histogram of global Equalization



(e) CLAHE with clipLimit=2.0 and tileGridSize=(8,8)



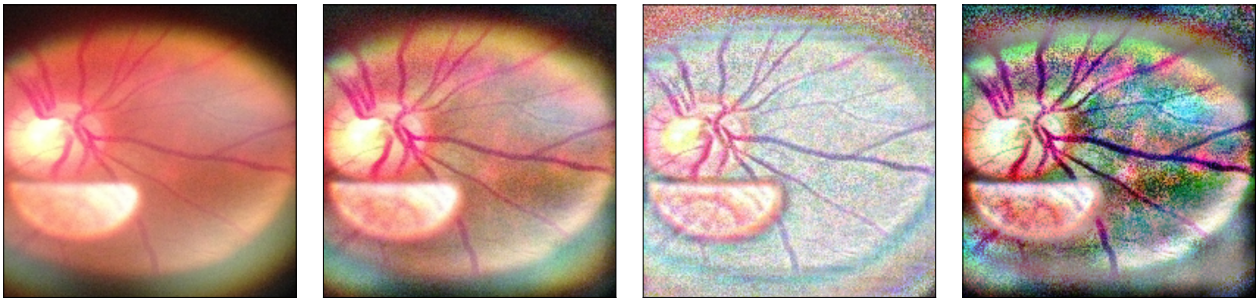
(f) Histogram of CLAHE with clipLimit=2.0 and tileGridSize=(8,8)

Figure 9 shows two decomposition tests for a gray image with different patterns and intensities (Figure 9a). Notice that the shape of the *s* array holding the singular values is s.shape = (200,) corresponding to the minimum value of the image's shape im.shape = (200, 200). The recomposed image with slice s[50:]=0 produces an almost complete original image (Figure 9b) while with slice s[1:]=0 produces an background-like image without patterns (Figure 9c) and with slice s[:1]=0 only the patterns or edges of the original image are preserved (Figure 9d).

**Smoothing with 1D-filters.** There is an filter algorithm in (SciPy, n.d.) presenting the 'flat', 'hanning', 'hamming', 'bartlett' and 'blackman' windows that are used to convolve with the input data. This operation smooths out data which exhibit noisy like behaviour but with trends to certain values. One of their applications is to smooth histograms to obtain their local minima and maxima values.

As a reference filter Figure 10 shows the Savgol or Savitzky-Golay filter which uses a method much like a fitting algorithm combined with a convolution. This filter is used to compare the other filters due that it fits almost perfectly to the histogram

*Figure 8.* CLAHE



(a) Input image

(b) CLAHE with tileGridSize=(8, 8) and clipLimit=2.0

(c) CLAHE with tileGridSize=(30, 30) and clipLimit=2.0

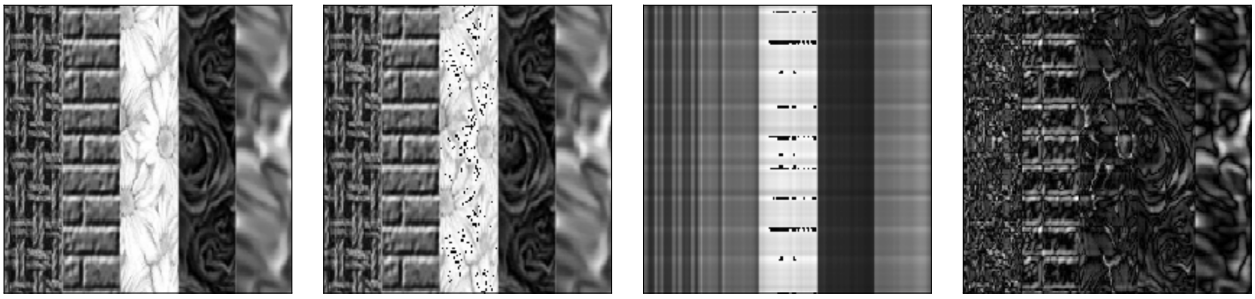
(d) CLAHE with tileGridSize=(8, 8) and clipLimit=15.0

Code 2: Decomposition MWE

```
from RRtoolbox.lib.image import loadcv, np
W,H = 400,300 # This is how shape works with Numpy H,W == im.shape = rows,cols
shape = W,H # this is how functions receive shape

# load the image
im = loadcv("my_image.jpg",0,shape)

# get the Singular Value Decomposition (SVD) of the image
U, s, V = np.linalg.svd(im, full_matrices=False)
# U - Unitary matrices. The actual shape depends on the value of
# s - The singular values for every matrix, sorted in descending order.
# V - Unitary matrices

# reduce information in decomposition
print s.shape # this correspond to np.min(shape) and s rows go from 0 to s.shape-1

# recompose original image
im = np.abs(np.dot(U, np.dot(np.diag(s), V))) # complete SVD

# copy s array for two tests
s1 = s.copy()
s2 = s.copy()

s1[:1]=0 # destroy the values at the most significant row 0

# recompose image 1
im1 = np.abs(np.dot(U, np.dot(np.diag(s1), V))) # reduced SVD

s2[50:]=0 # destroy the values  of the less significant
          # rows from rows 50 to the last row s.shape-1

# recompose image 2
im2 = np.abs(np.dot(U, np.dot(np.diag(s2), V))) # reduced SVD
```

sample. Notice that this prevents the filtered data to not be smoothed as desired to reduce the number of local minima and maxima values and the incorrect parametrization of the filter can lead to reduce the fidelity of the filtered data (Figure 10b).

*Figure 9.* Decomposition tests



(a) Normal: s.shape = (200,)

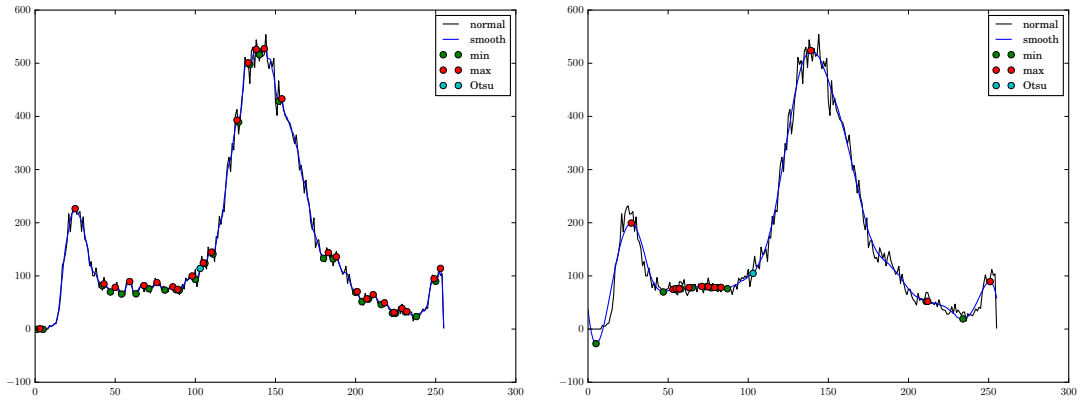
(b) Recomposed svd with s[50:]=0

(c) Recomposed svd with s[1:]=0

(d) Recomposed svd with s[:1]=0

This filter can be found in the *Scipy* Python package using `from scipy.signal import savgol_filter`. $window_{len}$ is the length of the filter window (i.e. the number of coefficients in the polynomial) and must be a positive odd integer. *polyorder* is the order of the polynomial used to fit the samples and must be less than $window_{len}$.

*Figure 10.* Savgol filter




(a) Savgol with $window_{len}$ = 11 and polyorder = 5

(b) Savgol with $window_{len}$ = 51 and polyorder = 5

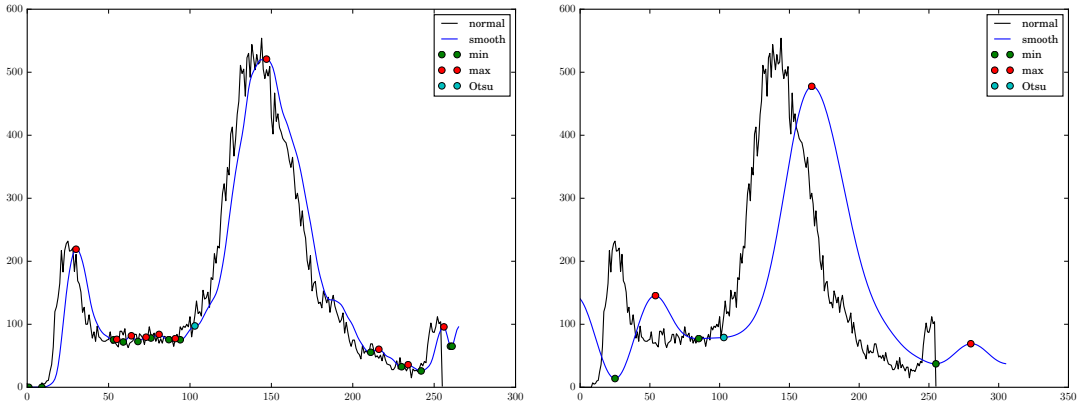
Figure 11 shows the filter as implemented in (SciPy, n.d.) using the Hanning window. Filtered data is moved proportional to $window_{len}$ creating an offset between the input and the filtered result. This produce a wrong result for the minima and maxima values obtained using the filtered data which should be aligned with the input data.

Figure 12 shows the improved filter using the Hanning window with shifted convolution to correct filtered data offset presented in Figure 11. Notice that filtered data does not present offset between it and the input and the minima and maxima values correspond to the actual values of the input. The proposed filter is found in the developed *RRtoolbox* package and it can be imported using `from RRtoolbox.lib.arrayops.filters import smooth`.

**Gaussian Filter.** It is one of the most widely used filters to reduce Gaussian noise in images but it produce a blur-like effect that eliminates details along with its application. Nonetheless, it is known that this can be a desirable effect used to simulate the loss of detail produced when an scenery is photographed from a farther viewpoint for applications where scale invariance is important (Rey Otero & Delbracio, 2014).

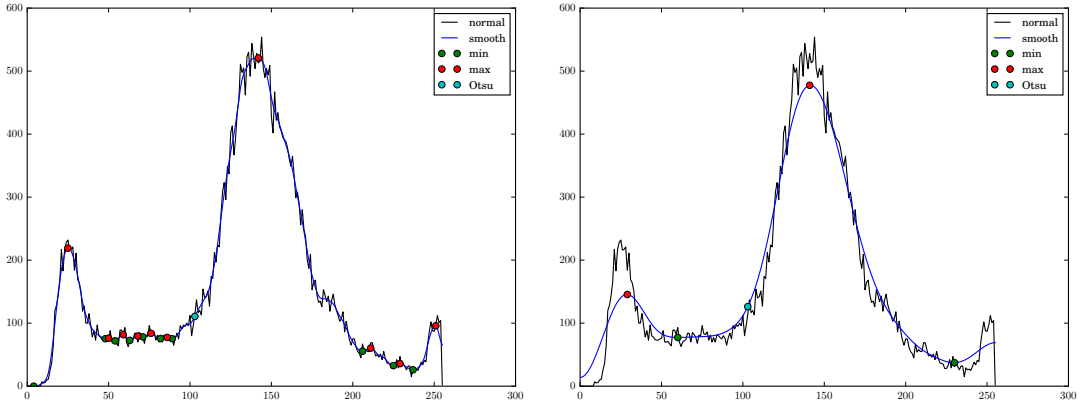
**Bilateral Filter.** The *bilateralFilter* can reduce unwanted noise very well while keeping edges fairly sharp. However, it is very slow compared to most filters so it is not recommended for real-time applications where the hardware is limited and can't cope with fast processing times giving retarded results. In the other hand, this filter does very well in applications where the

*Figure 11.* Filter with Hanning window




(a) Hanning with $window_{len}$ = 11 (b) Hanning with $window_{len}$ = 51

*Figure 12.* Filter with Hanning window and shifted convolution




(a) Hanning with $window_{len}$ = 11 (b) Hanning with $window_{len}$ = 51

time is not the most important factor for its functionality. The OpenCV package offers the function *bilateralFilter* (OpenCV, n.d.-b) which applies bilateral filtering to the input image as described in (Paris, Kornprobst, Tumblin, & Durand, 2008) with the following arguments:

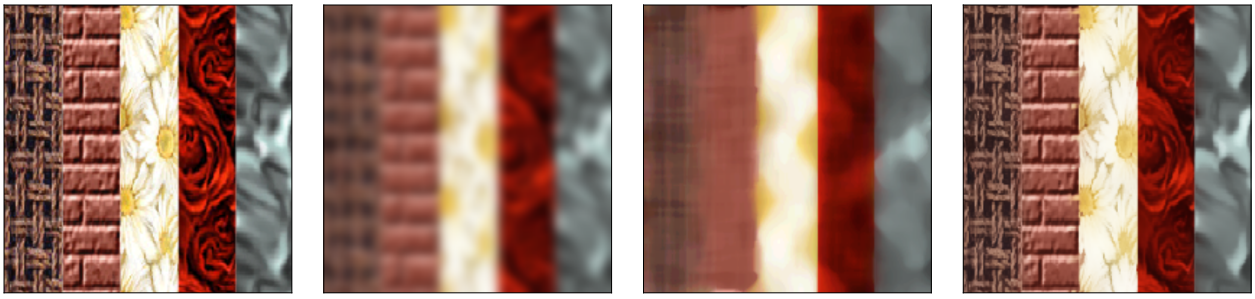
- *src*– Source 8-bit or floating-point, 1-channel or 3-channel image.
- *dst*– Destination image of the same size and type as *src*.
- *d*– Diameter of each pixel neighbourhood that is used during filtering. If it is non-positive, it is computed from *sigmaS pace*.
- *sigmaColor*– Filter sigma in the color space. A larger value of the parameter means that farther colors within the pixel neighbourhood (see *sigmaS pace* ) will be mixed together, resulting in larger areas of semi-equal color.
- *sigmaS pace* – Filter sigma in the coordinate space. A larger value of the parameter means that farther pixels will influence each other as long as their colors are close enough (see *sigmaColor* ). When $d > 0$ , it specifies the neighbourhood size regardless of *sigmaS pace* . Otherwise, $d$ is proportional to *sigmaS pace* .

***Filter size.*** Large filters (d>5) are very slow, so it is recommended to use d=5 for real-time applications, and perhaps d=9 for off-line applications that need heavy noise filtering.

***Sigma values.*** For simplicity, you can set the 2 sigma values to be the same. If they are small (<10), the filter will not have much effect, whereas if they are large ( >150), they will have a very strong effect, making the image look "cartoonish".

Figure 13 shows a comparison between the bilateral filter and the most common spacial filters, the Gaussian and Mean filter. As it is known the Gaussian filter (Figure 13b) produces a blurry-like result from the original (Figure 13a) and the Mean filter eliminates most of the edges from it (Figure 13c) while the bilateral filter preserves the edges and retains most of the features (Figure 13d)

*Figure 13*. Spacial filters comparison



(a) Original (b) Gauss: kernel = (21, 21) (c) Median: ksize = 21 (d) Bilateral: d = 21, sigmaC = 82, sigmaS = 57

The bilateral filter tries to preserve the edges but not always retains the features if it is not well parametrized which could be an unwanted result for the user. To prevent this some tests were made using the filter and getting good parameters to make a selector function for the correct bilateral filter parameters based in the image shape and the 'mild', 'normal' and 'heavy' modes of noise in the image. Figure 14 shows the selection for the parameters in the Bilateral filter according to the image's shape and mode. Notice that the mode only changes the diameter parameter $d$ and the others continue to variate equally when the shape of the image changes too. The selector function is found in the developed *RRtoolbox* package imported as `from RRtoolbox.lib.arrayops.filters import getBilateralParameters`.

Figure 15 shows the results of the Bilateral filter for an noisy image with different shapes with the parameters selected for the algorithm as shown in Figure 14 in 'heavy' mode. Notice here that when images are bigger and bilateral filters compensate for it to filter noise it consume a considerate amount of time to filter.

**SigmoidImageFilter.** The sigmoid filter is based in the sigmoid function which is characterized by having an "S" shape with a dampening behavior when tending to $-\infty$ and an approximation to the value "1" when tending to $\infty$ (i.e. an adequate offset can be used to determine the threshold to damper or let through certain values). In particular this filter was inspired from one of the tools of Mevislab, the $SigmoidImageFilter$ which uses the Insight Segmentation and Registration Toolkit (ITK) described in (ITK, n.d.). The equation is as follows:

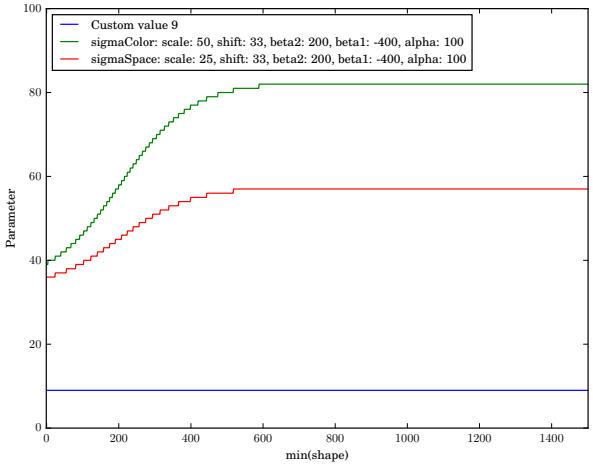
$$Sigmoid(x_i, \alpha, \beta, Max, Min) = (Max - Min)\left(\frac{1}{1 + e^{-(x_i-\beta)/\alpha}}\right) + Min \tag{7}$$

where $Max$ is the maximum value at the output, $Min$ is the minimum value at the output, $\beta$ the offset and $\alpha$ a scaling factor. The $\beta$ parameter can be thought as the offset on the pixel value that you are trying to isolate i.e. if the object you are trying to segment is at a pixel intensity above 150, you would choose a $\beta$ value that is around that value and the $\alpha$ parameter can be thought of as the scaling or variance of the sigmoid (CMISS, n.d.).

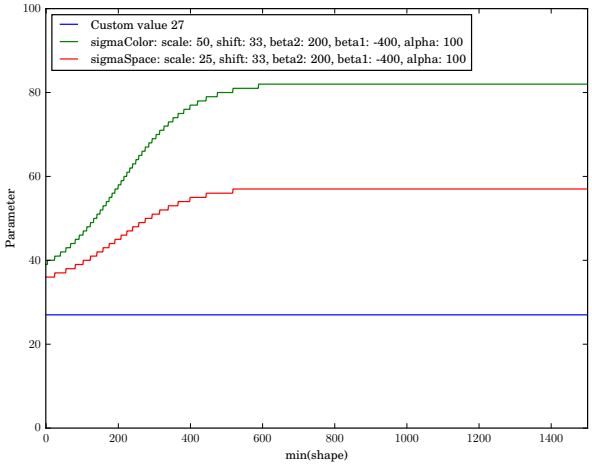
**Normalized SigmoidImageFilter.** The normalized sigmoid filter is extracted from the component in Equation 7 that computes any $x$ value (for discrete it is $x_i$) with an offset $\beta$ and a scaling factor $\alpha$ that its result is inside the range of $[0, 1]$. This is:

$$nS_i = normSigmoid(x_i, \alpha, \beta) = \frac{1}{1 + e^{(\beta-x_i)/\alpha}} \qquad \forall \quad x_i, \alpha, \beta \in \mathbb{R} \quad \wedge \quad 0 \leqslant nS_i \leqslant 1 \tag{8}$$

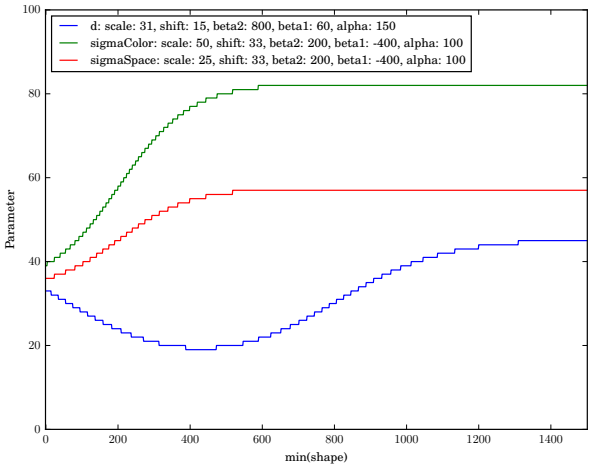
*Figure 14*. Selection of bilateral parameters according to image shape




(a) Bilateral parameters in 'mild' mode




(b) Bilateral parameters in 'normal' mode




(c) Bilateral parameters in 'heavy' mode

which stabilizes at the limits:

$$\lim_{x\to-\infty} nS(x) = 0 \qquad \forall \quad \alpha > 0$$
$$\lim_{x\to\infty} nS(x) = 1$$

This facilitates the calculation of $\alpha$ and $\beta$. For example, solving for $\alpha$ yields:

$$\alpha(\beta) = \frac{\beta - x}{\ln\left(\frac{1}{nS} - 1\right)}$$

And solving for $\beta$ yields:

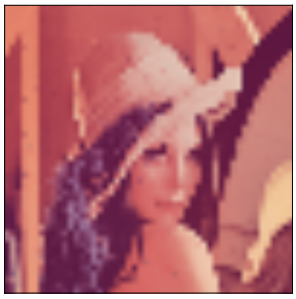
$$\beta(\alpha) = \alpha * \ln\left(\frac{1}{nS} - 1\right) + x$$

Also, Equation 7 can be expressed in therm of the Equation 8 as follows:

$$Sigmoid(x, \alpha, \beta, Max, Min) = (Max - Min) * normSigmoid(x, \alpha, \beta) + Min \tag{9}$$

*Figure 15.* Bilateral filter exposed to different shapes

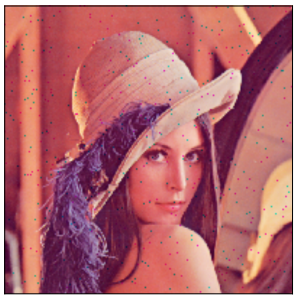
(a) Input shape = (50, 50)



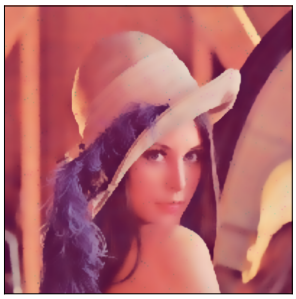
(b) Filtered with d = 31, sigmaC = 42, sigmaS = 37 and time = 0.573 secs

(c) Input shape = (100, 100)

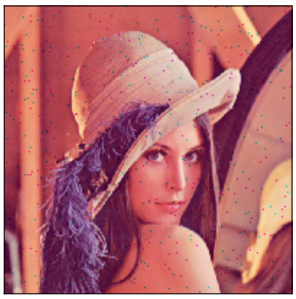
(d) Filtered with d = 28, sigmaC = 46, sigmaS = 39 and time = 1.337 secs



(e) Input shape = (600, 600)



(f) Filtered with d = 22, sigmaC = 82, sigmaS = 57 and time = 3.414 secs



(g) Input shape = (1500, 1500)

(h) Filtered with d = 45, sigmaC = 82, sigmaS = 57 and time = 48.979 secs

which stabilizes at the limits:

$$\begin{aligned} \lim_{x\to-\infty} Sigmoid(x) &= Min \\ \lim_{x\to\infty} Sigmoid(x) &= Max \end{aligned} \qquad \forall \quad \alpha > 0$$

The sigmoid function and Normalized sigmoid are in the *RRtoolbox* package and can be imported using `from RRtoolbox.lib.arrayops.filters import sigmoid, normsigmoid`.

**Custom filters using *normSigmoid*.** Observing that the *normSigmoid* function is similar to a Butterworth filter it can be used to make custom filters as in signal processing allowing an image to be filtered (as in non-spacial or color filters) with common filters such as *high-pass*, *low-pass*, *band-stop*, *band-pass* or any custom filter created by the combination of the *normSigmoid* function and normalizations (see Equation 6).

The simplest filters can be made using a single *normSigmoid* function. These are the high-pass and low-pass filters created when $\alpha$ is positive and negative respectively. The others can be seen as a compound of these two.

$$Lowpass(x,\alpha,\beta) = normSigmoid(x,\alpha,\beta) \qquad \forall \quad \alpha < 0 \tag{10}$$

$$Highpass(x,\alpha,\beta) = normSigmoid(x,\alpha,\beta) \qquad \forall \quad \alpha > 0 \tag{11}$$

$$Bandstop(x,\alpha,\beta) = Lowpass(x,\alpha,\beta_1) - Lowpass(x,\alpha,\beta_2) + 1 \qquad \forall \quad \beta_1 > \beta_2 \tag{12}$$

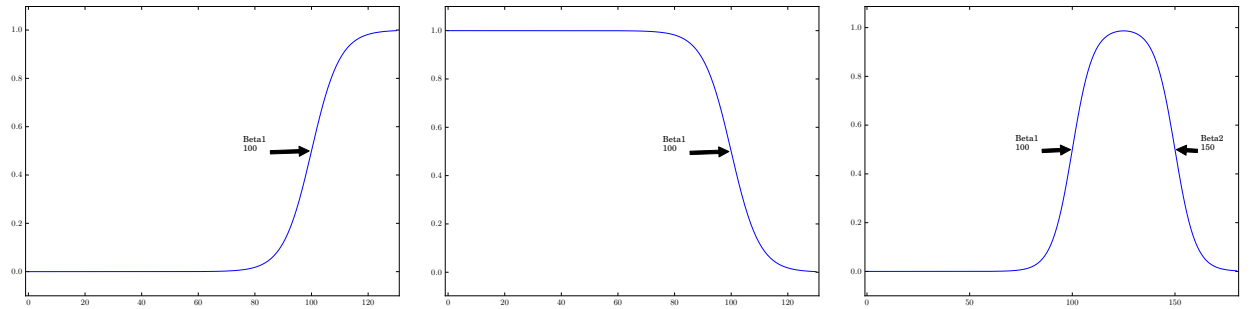
$$Bandpass(x,\alpha,\beta) = Highpass(x,\alpha,\beta_1) - Highpass(x,\alpha,\beta_2) \qquad \forall \quad \beta_1 > \beta_2 \tag{13}$$

$$Bandstop_{Inverted}(x,\alpha,\beta) = Lowpass(x,\alpha,\beta_2) - Lowpass(x,\alpha,\beta_1) - 1 \qquad \forall \quad \beta_1 > \beta_2 \tag{14}$$

$$Bandpass_{Inverted}(x,\alpha,\beta) = Highpass(x,\alpha,\beta_2) - Highpass(x,\alpha,\beta_1) \qquad \forall \quad \beta_1 > \beta_2 \tag{15}$$

Figure 16 shows the response of each normalized filter and Figure 17 the comparison among them. Notice how all the filters converge if they share the same $\alpha$ and $\beta$ parameters. These filters are imported using `from RRtoolbox.lib.arrayops import FilterBase, Lowpass, Highpass, Bandstop, Bandpass, InvertedBandstop, InvertedBandpass, filterFactory`. The additional import, `ffilterFactory`, is a function used to create the necessary filter from the provided arguments.

*Figure 16.* Common filters response using *normSigmoid*




(a) Highpass with alpha=5, beta1 = 100and beta2=None

(b) Lowpass with alpha=5, beta1 = 100and beta2=None

(c) Bandpass with alpha=5, beta1 = 100and beta2=150

(d) Bandstop with alpha=5, beta1 = 100and beta2=150

(e) InvertedBandstop with alpha=5, beta1 = 100and beta2=150

(f) InvertedBandpass with alpha=5, beta1 = 100and beta2=150

All the filters which range are between $[0, 1]$ are called normalized filters and can be derived from *normSigmoid* or the *high-pass* and *low-pass* filters. This inheritance-like behaviour led to the normalized filters being implemented using classes. The class diagrams can be seen in Figure 18. It shows how the filter classes inherit from the *FilterBase* class, their methods (represented by the m icon) and fields (f icon). The C icon is for classes, P deleter, P getter and P setter properties. As the normalized filters suggest the color levels in the image $I$ are saturated when passed through them:

$$filteredI_{saturated} = filter_{normalized}(I)$$

To filter the image $I$ without saturation (only dampened values are saturated to zero) a simple element-wise matrix multiplication can be carried out as follows:

$$filteredI_{un-saturated} = mul(I, filter_{normalized}(I)) = I_{i,j} * filter_{normalized}(I_{i,j})$$

*Figure 17*. Common filters comparison

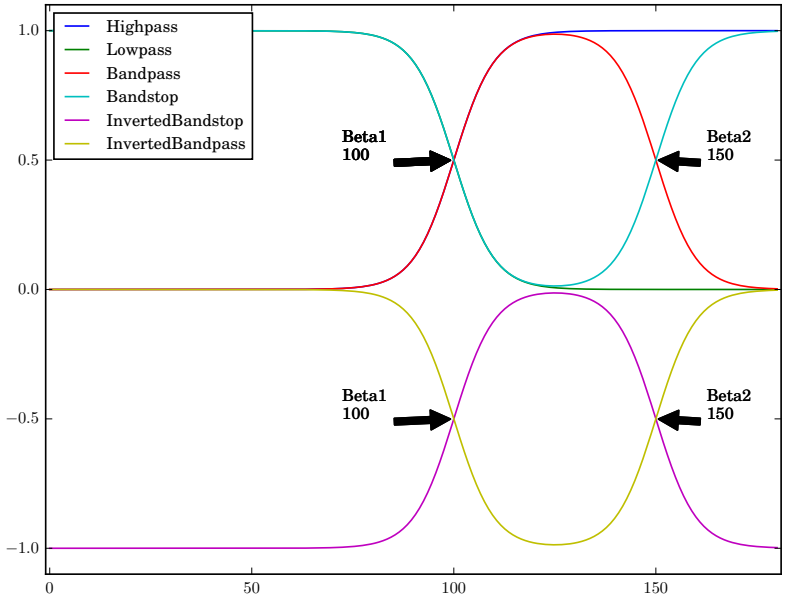
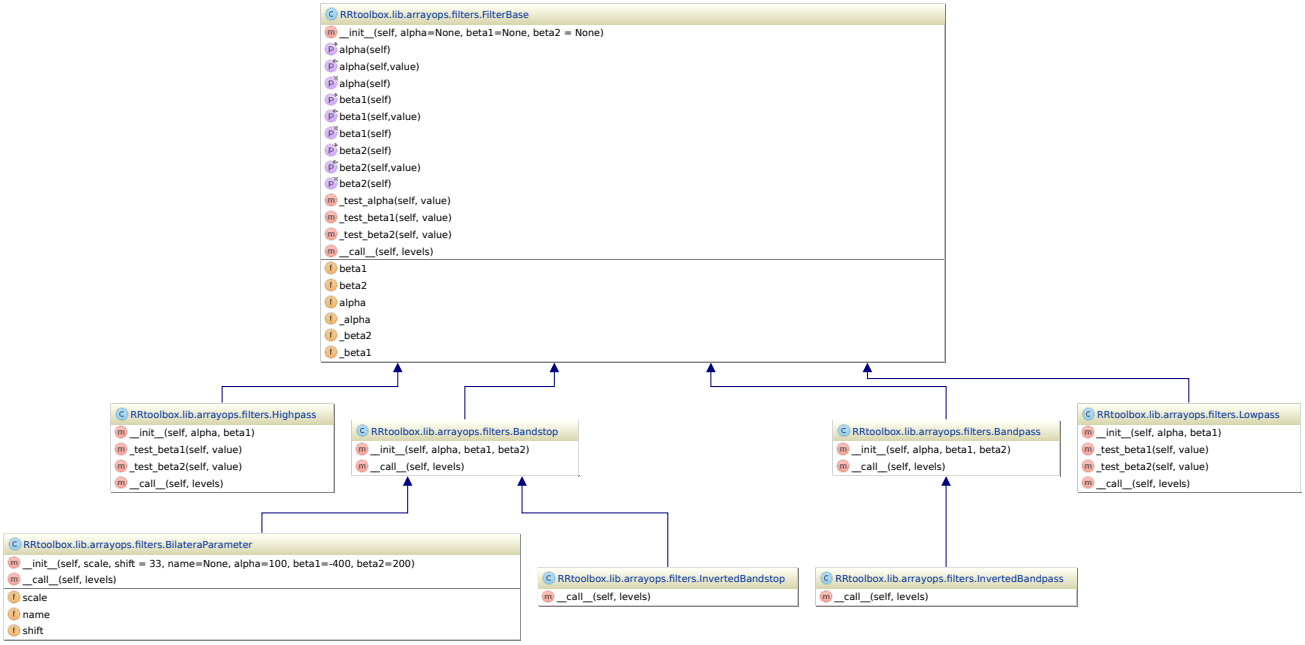


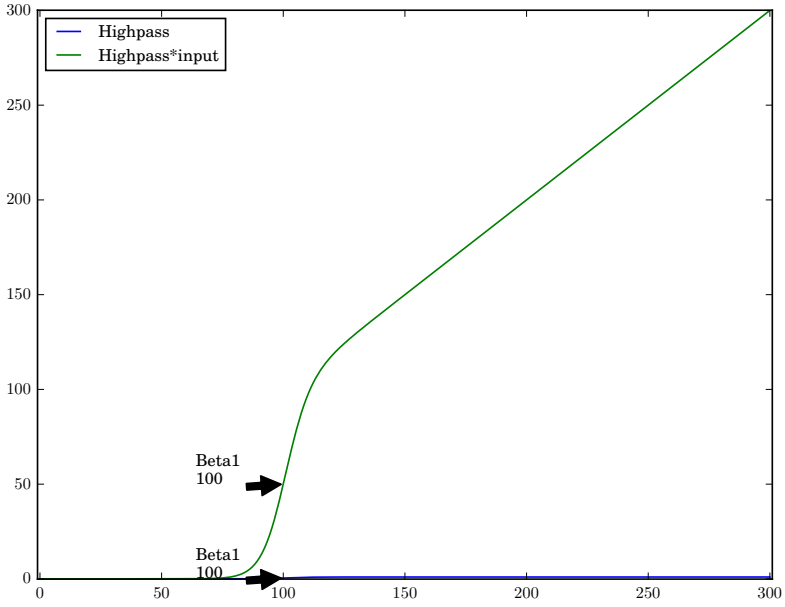

*Figure 18*. Filters class diagrams




Where the *mul*() function denotes element-wise multiplication or index by index operation which is different to matrix multiplication in mathematics.

Figure 19 shows an example of the normalized *high-pass* filter customized to prevent saturation. Filters that are not comprised between 0 and 1 are referred as custom filters. Notices that the response tents to infinite when input tents to infinite too. Because images of *uint8* type values have 256 levels ranging from 0 to 255 the filter only would filter those values and would not have problems with the unstable behaviour.

*Figure 19*. Custom filtering without saturation




**Sigmoid filtering and saturation.**

## Segmentations

Several segmenting methods were taken into account but only threshold methods were developed for simplicity. The principal idea in segmenting retinal images is to over-thresh and further process them according to some general observations in the experimentation to obtain the final segmentation. Some methods were compared against an expert (i.e. what is considered to be a right threshold) and others to a well known automated threshold method, in this case the *Otsu* method. This with the intention of examining the algorithm robustness and reliability when applying to an unknown retinal image. Next subsections assume that a general threshold is applied to an image and an specific problem wants to be solved.

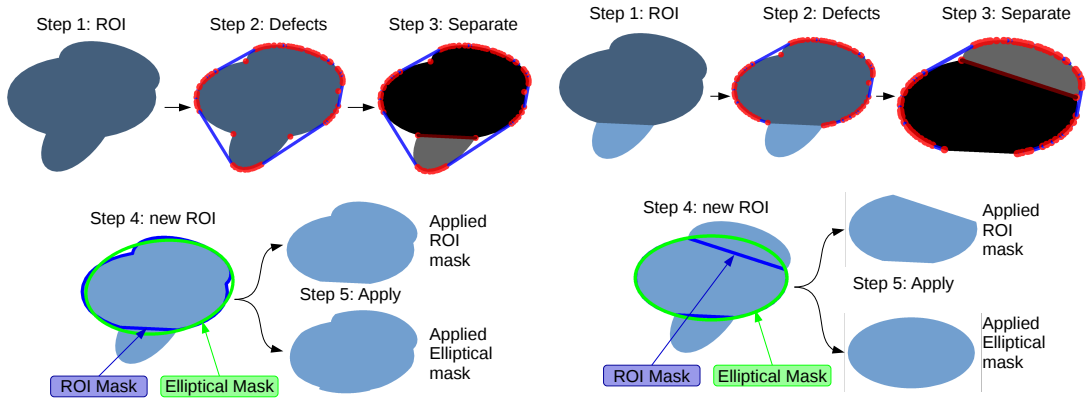
**Convex hull with line cuts.** This method separates an irregular object where two convexity defects are dominant, that is were the distances between the object and the convex hull are more pronounced. This is intended to be applied on objects with irregularities or protuberances where what is wanted are regular shapes like squares, circles and ellipses or objects with few defects.

The iterative splitting of the object can be stopped using the convexity ratio $r_{convexity}$ explained in the Convexity ratio section .

The pseudo code is as shown in **??**. Its implementation is not part of the *RRtoolbox* package as it is still a concept not adequate for robust operations. It can be found under *tests / hypothesis2_5_masks_defect_lines2.py* in the repository.

This concept in *tests / hypothesis2_5_masks_defect_lines2.py* is demonstrated using a model object as in Figure 20. The region of interest (ROI) is an irregular object with an obvious round main body and some humps sticking from it (it 1, step 1). These humps form defects with respect to the convex hull of the ROI (it 1, step 2) so the more pronounced the defect is the greater the distance from the ROI and the convex hull is. This way it can be detected where the biggest irregularity is by selecting the two most biggest distances from the ROI and the convex hull and finding their points in the binary image. After this, the main body can be easily segmented by cutting the ROI in two by these two points (it 1, step 3). When the ROI is divided in two it is the choice of user to select the new ROI but for the sake of automation here and because what is wanted is easy enough as to know which object is bigger then a simple comparison is made to select the main body. As expected the main body is the object with biggest area (it 1, step3, colored in black) and the other (in gray) is a protuberance which is discarded. Once the new ROI is selected it can be used directly as a mask to apply it in the object (it 1, step 5, above), refined it to better match the object (it 1, step 5, below) or it can be further developed by applying the same algorithm in another

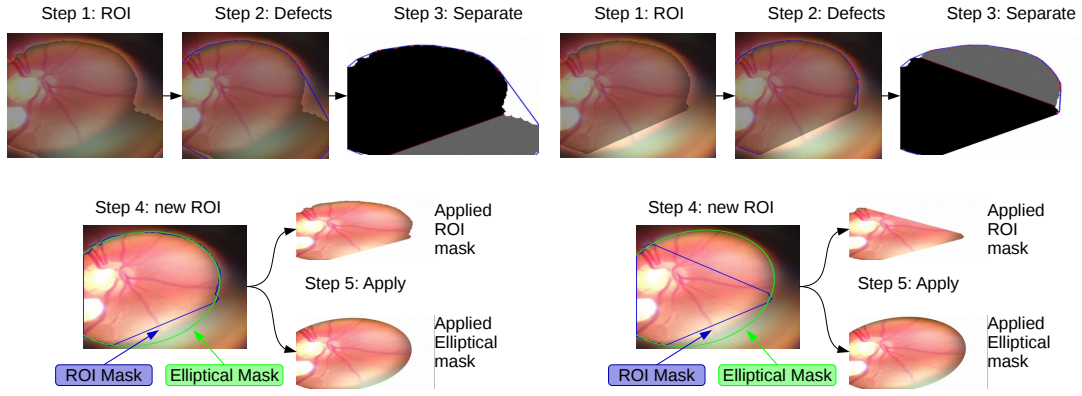
*Figure 20.* Ideal threshold with defect lines




(a) Iteration 0: $C_r$ = 0.945903192491 (b) Iteration 1: $C_r$ = 0.987807853408

iteration and sub-segment it until the desired ROI is reached (i.e. one way to stop the iterations is by using a goal with $r_{convexity}$ which approach to 1 the more it approach to its convex hull). Using the new ROI and iterating over it a second time though the same steps eliminates the second hump in the original ROI (it 2, step 4, ROI mask) producing an accumulative effect each time a new iteration is made. Likewise, as stated, the $r_{convexity}$ which was 0.945 in the iteration 1 and now is 0.987 in the iteration 2 is approaching to a ratio of 1 giving a good indication that the object has been successfully approached to its convex hull or a more regular shape. It can be seen that though the ROI mask (it 2, step 4) is edgy and unrefined it can be applied "as is" onto the original object to produce a segmentation without humps (it 2, step 5, above) but if an ellipse is fitted in the ROI mask it produces an Elliptical mask that can segment exactly the main body which is also of a well defined elliptical shape (it 2, step 5, below) inside the original ROI (it 1, step 1). Because the object was ideal the main elliptical body could be segmented precisely without problems producing staggering result for a simple method as it is.

*Figure 21.* Practical threshold with defect lines




(a) Iteration 0: $C_r$ = 0.958214053977 (b) Iteration 1: $C_r$ = 0.987951675768

But in actual practice not all the objects are ideal, nonetheless this algorithm works for some of those cases too. Figure 21 Shows an example applying the convex hull with line cuts algorithm in a real case problem. Here the retinal image is over-threshed to give a full segmentation of the retinal area (It 1, step 1), but as it is seen this not only segmented the retinal area but a part affected by flares and noise produced by the bad focus of the camera plus the effects of the flash light. Because the retina is known to have a round shape it can be assumed that anything attached to it as that big protuberance is not part of it but something else. Applying the convex hull (It 1, step 2) and separating the objects by its defects (It 1, step 3) as in the ideal case previously explained gives as expected the main body of the object as shown in (It 1, step 3 and step 4), which a the end can be used to apply the original produced mask onto the over-threshed object to produce a under-threshed object with a better

segmentation of the retinal area. But as it is known the retina is not formed by straight lines so a smooth version is produced by fitting an ellipse onto the ROI mask to produce the Elliptical mask which gives a more appealing result (it 1, step 5).

**Binary masks.** There are common methods to threshold images which produce binary mask to use when processing restricted areas in an image. Unfortunately for retinal applications common methods do not work and even the *Otsu* threshold does not work well for the used cases in this project. To overcome this several developed methods were tested but proved inadequate or failed to the task. Below an experimental methods is presented which currently fails in some cases but an additional developed method has been successful so far and performs well in the application with retinal images with severe noise cases. More successful methods are found in `RRtoolbox.lib.arrayops.mask`.

***Experimental watershed method to find optic disk area.*** There is a test that uses watershed to segment bright areas with the objective of obtaining the optic disk and the body of the retina but this proved to be unstable for many cases. The seed points were provided using a histogram analysis to obtain stable points taking into account the background, retinal area and bright areas like flares and the optic disk.

*Figure 22*. Histogram analysis to find threshold values to use in watershed method

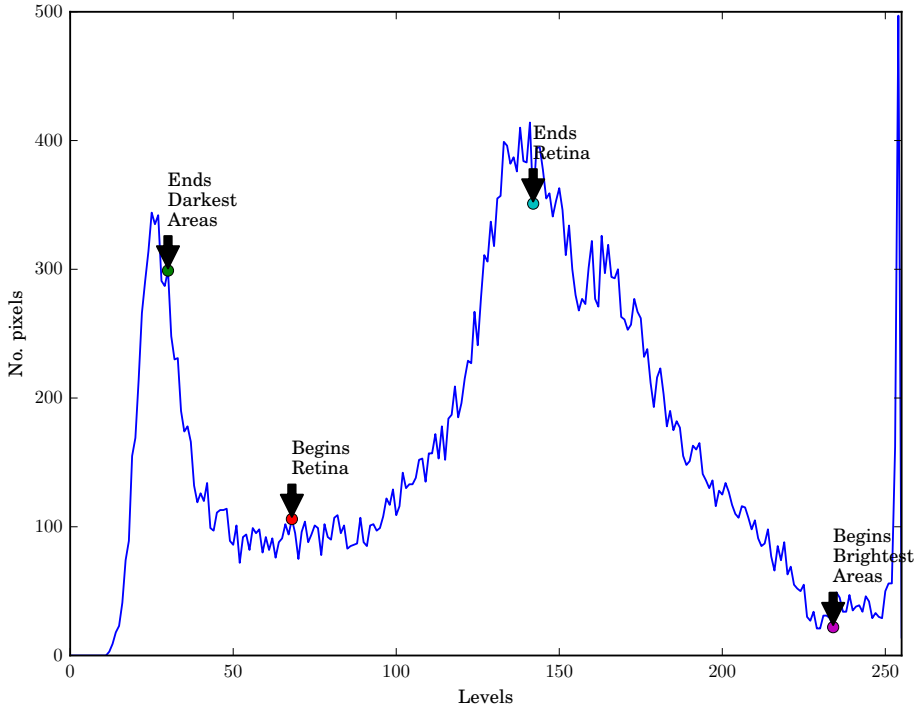
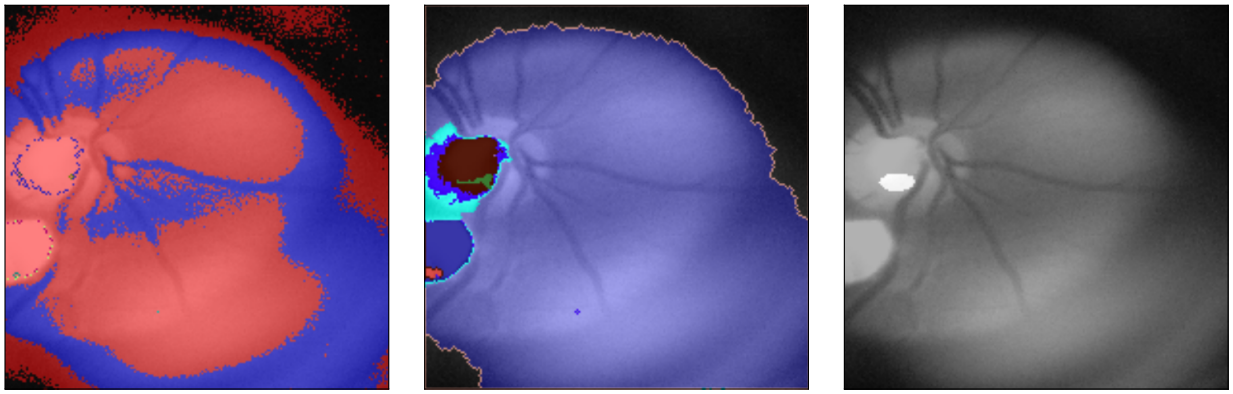



Figure 22 shows a case with the result of the automatic histogram analysis applied to provide the necessary information in the watershed method. The result of feeding these parameters in the watershed method are shown in Figure 23. In Figure 23a the seeds are placed in the sample image which produce the watershed in Figure 23b. It is observed that the watershed not only segmented the retinal area but the blurry area around it. Lastly the brightest areas are analysed to guess what is the optic disk in of these segmentations but as there is a flare present near it the algorithm is confused presenting a foul result trying to not select the flare (Figure 23c).

This algorithm is still in the experimental stage and presents many issues to solve. Both algorithms used to analyse histograms (`retina_markers_thresh`) and apply the watershed method (`find_optic_disc_watershed`) can be imported using `from RRtoolbox.tools.segmentation import retina_markers_thresh, find_optic_disc_watershed` in Python.

*Figure 23.* Brightest areas in image using watershed method



(a) im1_2 markers (b) im1_2 segmentations (c) im1_2 optic disc

***Proposed threshold method.*** A method using the *Otsu* threshold was develop to proximate the segmentation to an object at each iteration. The algorithm was implemented in a function called `multiple_otsu` and it was adapted for concrete applications in the `background` and `foreground` functions and as their names imply these are sued to threshold the background and foreground in an image. These functions can be imported in Python with the *RRtoolbox* package using `from RRtoolbox.lib.arrayops.mask import multiple_otsu, background, foreground`.

Figure 24 shows the comparison between threshold methods. In it column 3 shows the overlaid mask obtained using the background mask over the input image and in column 4 it can be seen the result for the foreground threshold. Wait! but the foreground threshold performs worse than the Otsu method in column 2. Well it is not totally true, the `foreground` method is just the inverse of the `background` method so that these two can be compatible. Notice that the `background` method yields what it promises and produces a better background mask than the Otsu method presented in column 2 and it happens that the `foreground` method yields the inverse of it.

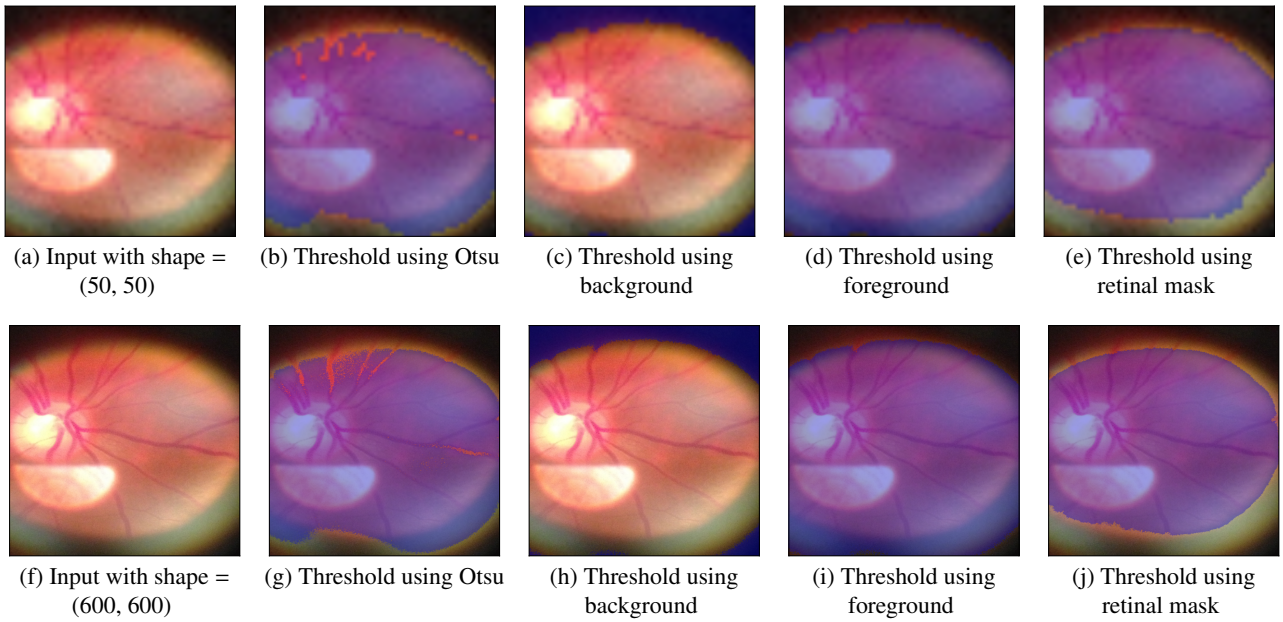
***Developed method to find retinal area.*** There is another developed algorithm that can find 'solid' objects in an image without getting confused by blurry areas. This algorithm is called `layeredfloods` because it uses a weighted sum with the flooding algorithm but it has the condition of using color images for better results. If a gray image is fed to the algorithm it tends to yield the same results as the normal Otsu method. This algorithm does not yield exactly a binary mask but an alpha mask which is discussed in more detail in the Alpha masks section . The `layeredfloods` algorithm then was adapted to a new function called `retinal_mask` effectively used to threshold retinal images. Both `layeredfloods` and `retinal_mask` are found in the *RRtoolbox* package and can be imported using `from RRtoolbox.tools.segmentation import layeredfloods, retinal_mask`. The source code of these algorithms are also found there.

Column 5 in Figure 24 shows the overlaid mask over an input image obtained using the `retinal_mask` function. Notice that the algorithm is robust for different conditions like image shape, illumination, color changes (notice how the Otsu method is opened in some areas because the color there was not of covered by the threshold value) and blurry areas.

**Alpha masks.** Though binary masks can accomplish some objectives one begins to realize that they present some limitations as they are exactly designed to segment and nothing more. A better way can be offered using Alpha masks which represent levels of transparency in images. This of course at the expense working with more information, using complex methods to process them and being more resource intensive. Nonetheless, today computers allow to quickly and easily use these masks opening new ways of computing images.

Alpha masks add a layer of information to images by appending an additional dimension to its array. Then BGR images must be transformed to BGRA or RGB to RGBA corresponding to the Red, Green, Blue and Alpha channels used in the image. This is usually used in the PNG image format allowing its images to have transparency when overlying with others to create composites. A similar example was shown in Figure 6 using the alpha parameter in the overlay algorithm. Next paragraphs explains how to create two alpha masks for the segmentation of retinal images and how these same masks can be used for overlay operations, adding more information to segmentations, adjusting image colors and filtering them.

*Figure 24*. Spacial filters comparison



(a) Input with shape = (50, 50) (b) Threshold using Otsu (c) Threshold using background (d) Threshold using foreground (e) Threshold using retinal mask

(f) Input with shape = (600, 600) (g) Threshold using Otsu (h) Threshold using background (i) Threshold using foreground (j) Threshold using retinal mask

***Create an alpha mask using filters.*** The Custom filters using *normSigmoid* section explained the use of the *normSigmoid* in Equation 8 to create custom filters. These filters create a normalized output when fed with an image which can be treated as an *alpha* array and do some useful operations with it. For instance *alpha* can be multiplied (point to point) to the image attenuating it (if *alpha* is normalized this operation does not amplifies it).

There are cases where custom filters produce non-normalized results due to adding and subtracting operations. This is solved using Equation 6 from the Normalization section ensuring data integrity (*uint8* data types in *Numpy* are truncated when trying to surpass the 255 value).

Without more further adieu I will explain a simple code that can be imported using `from RRtoolbox.tools.segmentation import get_bright_alpha` which uses filters to create an alpha mask that can be used to merge a foreground and background image. The working example is presented in Code 3 and Figure 25 shows the histogram analysis done by `get_beta_params_hist` to find the beta values and parametrize the filters (blue for background and green for foreground). Additionally the lateral view of the normalized filter response is presented too (in red) obtained with a simulation of the 256 levels in a gray image.

A clearer representation of what the Code 3 does is shown in Figure 26 with the 3D filter response of the simulated 0-255 grey levels of the background and foreground images respectively instead of the actual real images. This demonstrates that the algorithm in Code 3 acts in the color channels and not in the space domain. Notice that if a lateral slice is made across the graph from origin to origin at each axis, that slice corresponds to the merged red plot in Figure 25.

Once the algorithm in Code 3 is used with the real images it creates the alpha mask `normalizedmask` from their gray color channels. This mask can be used with the same background and foreground images to merge them which shown in Figure 28c.

***Create an alpha mask for the space domain.*** There is an studier method using the `layeredfloods` algorithm and it is implemented in the *RRtoolbox* package. This algorithm can be imported using `from RRtoolbox.tools.segmentation import get_layered_alpha`. Figure 27 shows the 3D representation of the alpha mask with row and column axes obtained using `get_layered_alpha`. Notice that the rows and columns indicate that any array operating the alpha mask must have the same dimensions and this mask acts in the space domain. The overlaid result of using this mask is shown in Figure 28d.

Code 3: MWE to create alpha mask with filters

```
# import necessary functions
import cv2
from RRtoolbox.lib.arrayops import normalize, Bandpass, Bandstop
from RRtoolbox.tools.segmentation import get_beta_params_hist

# load images
back ="im1_1.jpg"
fore ="asift2fore.png"
backgray = cv2.imread(back,0) # load the background image
foregray = cv2.imread(fore,0) # load the foreground image

# get background beta parameters
beta1B, beta2B = get_beta_params_hist(backgray)

# process backgray in Bandstop filter with automatic parameters
# for example beta1 = 50, beta2 = 190
backmask = Bandstop(alpha = 3, beta1 = beta1B, beta2 = beta2B)(backgray)

# get foreground beta parameters
beta1F, beta2F = get_beta_params_hist(foregray)

# process foregray in Bandpass filter with automatic parameters
# for example beta1 = 50, beta2 = 220
foremask = Bandpass(alpha = 3, beta1 = beta1F, beta2 = beta2F)(foregray)

# merge masks
mergedmask = foremask * backmask

# normalize to [0,1]
normalizedmask = normalize(mergedmask)
```

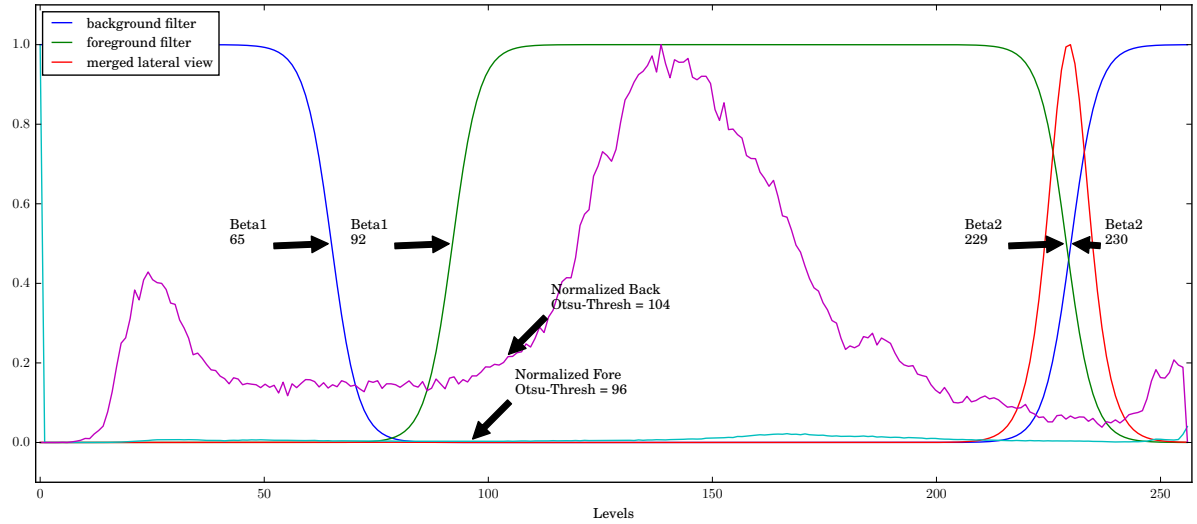
### Object Recognition and Matching algorithms

**Entropy.** It is a classification algorithm from images with multi-focus differences based from (Liu & Yu, 2015). The images must be of the same scene differing only the focus of the camera's lens when taking the picture. They used this algorithm to create high detailed images by combining with the wavelet transform multiple images with diverse focuses into a single focused one. The entropy algorithm based from it can be imported from the *RRtoolbox* package using `from RRtoolbox.tools.selectors import entropy` and Figure 29 shows an example sorting images using this function to classify a set of unfocused images of the same object.

**Histogram comparison.** The algorithm was implemented using OpenCV funtion `cv2.compareHist` and based from (Rosebrock, 2014). See the source code and documentation in the repository (David, 2016b, 2016a). Import it for use in Python with `from RRtoolbox.tools.selectors import hist_comp`. Figure 30 shows an application sorting images using the histogram comparison algorithm.

**Histogram matching.** The purpose of this algorithm is as its name indicates match the histogram of two image to convert the color of one images to the other. This algorithm was conceived trying to recreate imhistmatch function from MATLAB (MathWorks, n.d.-a) and based from an stackoverflow answer in (Ali_m, n.d.). Import the created algorithm with `from RRtoolbox.lib.image import hist_match` or see the source code and documentation in the repository (David, 2016b, 2016a). Additional explanations found in (Gonzalez et al., 2004, section 3.3.3). Figure 31 shows an example using the matching algorithm.

*Figure 25.* Alpha mask histogram analysis and filters response




**Detecting and computing features.** OpenCV offers a variety of detectors and descriptors (OpenCV, n.d.-a) as listed below:

- *BRISK*, Binary Robust Invariant Scalable Keypoints (Leutenegger, Chli, & Siegwart, 2011).
- *FAST*, Features from Accelerated Segment Test (Rosten & Drummond, 2006; Rosten, Porter, & Drummond, 2010).
- *FREAK*, Fast Retina Keypoint (Alahi, Ortiz, & Vandergheynst, 2012).
- *MSER*, Maximally Stable Extremal Regions (Forssen, 2007; Nistér & Stewénius, 2008).
- *ORB*, Oriented FAST and Rotated BRIEF (Rublee, Rabaud, Konolige, & Bradski, 2011).
- *SIFT*, Scale Invariant Feature Transform (Rey Otero & Delbracio, 2014).
- *SURFT*, Speeded-Up Robust Features (Zhou & Asari, 2011).

SIFT is preferable and selected as the default to apply in the algorithms requiring pattern recognition as it is one of the oldest and successful algorithms for object recognition, image stitching and other applications (Rublee et al., 2011). But because it is licensed and only can be used freely for research purposes (SURFT is licensed too) the other matchers can be used as well.

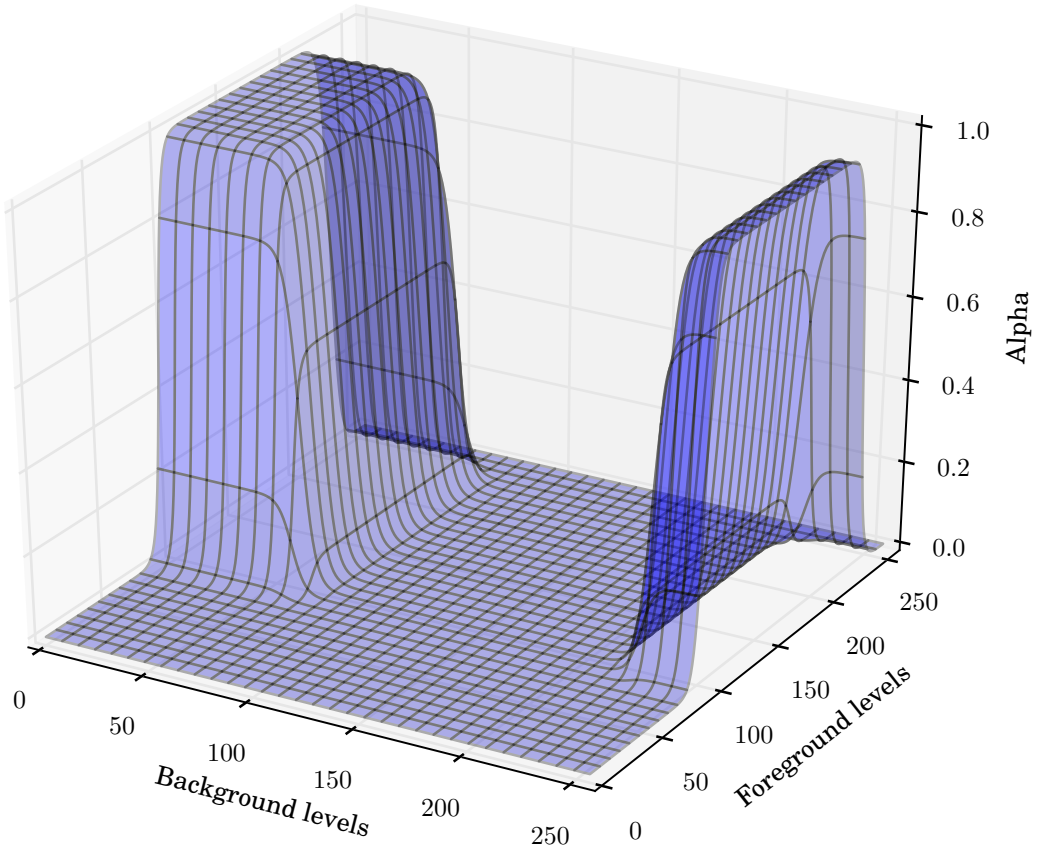
Figure 32 shows a feature detection example using the SIFT algorithm. The colourful circles represent the computed key-points. where the lines from their origin represents the orientation and the radius the size of the key-point. Notice how the rotation and image resolution does not affects much the generation of robust key-points.

**ASIFT.** ASIFT is a method used to improve on top of a feature-based image matching algorithm by creating various affine transformations of the image that is being computed for its descriptors with the normal matching program (G. Yu & Morel, 2011). The matching algorithm must be scale and rotation invariant to local image features for it to work with ASIFT. The implementation in Python was based from an OpenCV example at (Alexander Alekhin, n.d.) and the code is in the repository with path *RRtoolbox/lib/descriptors.py*. The implementation to use with any matching algorithm provided by OpenCV is the class `Feature` which can be imported using `from RRtoolbox.lib.descriptors import Feature`.

### Using rates and probabilities

There are may ways that results can be rated and that is using probabilities and rates based from robust values. The *RRtoolbox* use them regularly and this section treats some of them. For example the number of descriptors can be used to determine if an image is detailed with many edges, an inlier tests consist of obtaining the rate between inline key-pints and number of lines in a homography, rectangularity the measure of how much an object resembles a rectangle, regularity Ratio of

*Figure 26*. Alpha mask obtained using filters




forms with similar measurements and angles and the convexity ratio the measure of how much an object is convex or resembles it convex hull. The implementation of some of these rates are encapsulated in the `Imcoors` class which can be imported using `from RRtoolbox.lib.image import ImCoors`.

**Convexity ratio.** One way to determine if an object is concave is to compare its area from the convex hull by finding the ratio between these two (Bozeman & Pilling, 2013, Definition 5). This is what the convexity ratio $r_{convexity}$ does:

$$r_{convexity} = \frac{Area_{object}}{Area_{hull}} \tag{16}$$

where $Area_{object}$ is the area of the object (endogon) and $Area_{hull}$ is the area of its convex hull (exogon) which can be calculated using Equation 1 found in The area under a polygon (*poligonArea*) section . Notice that a convex object will give a convexity ratio of 1 and the more an object is concave the rate will tend to 0. This definition differs a litter from other variations (Kindratenko, 1997, Part 2: Shape Analysis,pp. 36–37).

Figure 33 shows an example using the convexity ratio equation. In this example the object points are:

*Figure 27.* Alpha mask obtained using layered masks

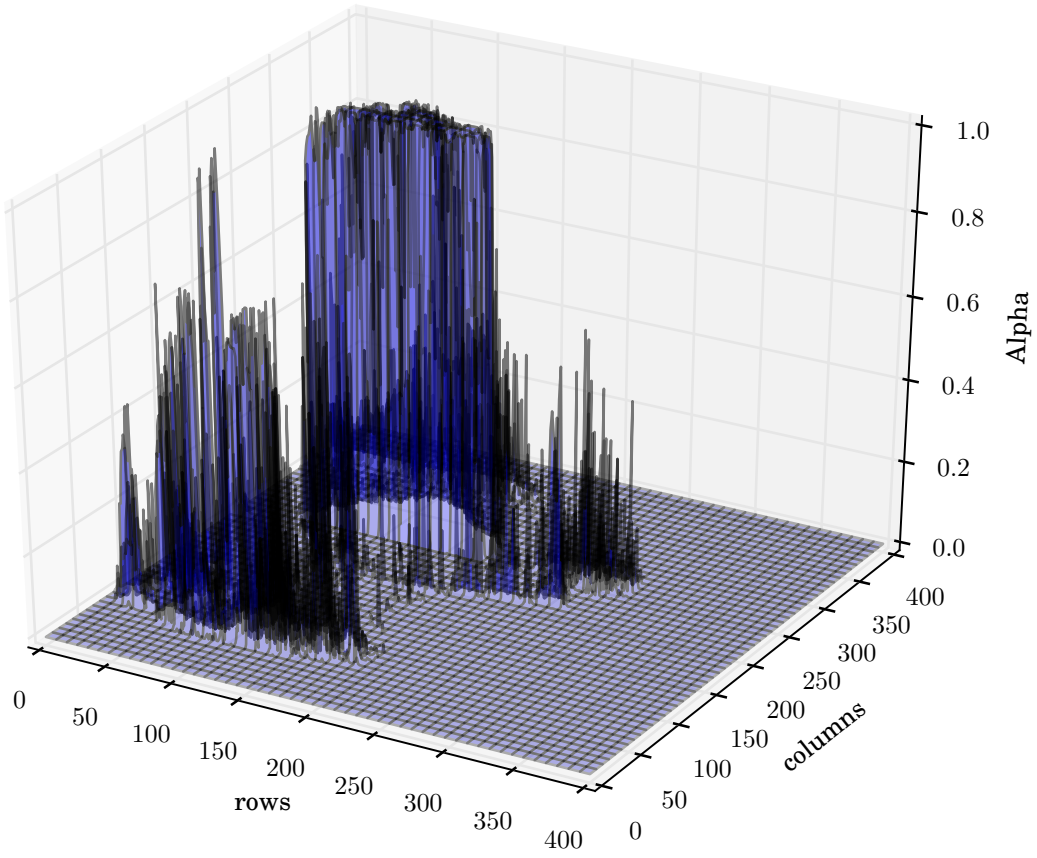
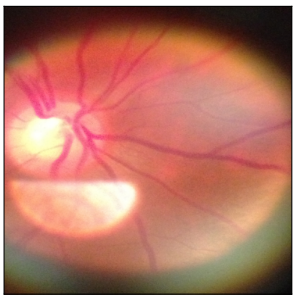
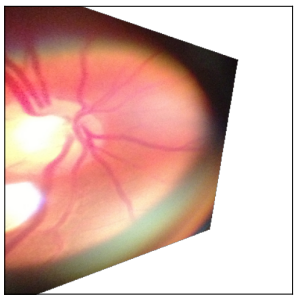


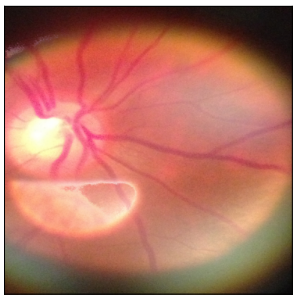

*Figure 28.* Overlay example using alpha masks



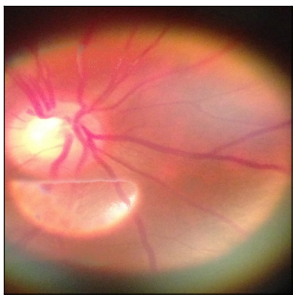
(a) background



(b) foreground



(c) Overlay with alpha mask in Figure 26



(d) Overlay with alpha mask in Figure 27

$$object_{points} = \begin{bmatrix} 6.0 & -3.0 \\ 3.0 & 0.0 \\ 5.0 & 4.0 \\ 0.0 & 4.0 \\ -5.0 & 5.0 \\ -2.0 & 0.0 \\ -6.0 & 0.0 \\ -1.0 & -3.0 \\ 0.0 & -6.0 \\ 2.0 & -1.0 \end{bmatrix}$$

*Figure 29*. Ordered images using entropy

(a) IMG_20160519_172655 with entropy 0.000 (b) IMG_20160519_172645 with entropy 0.014 (c) IMG_20160519_172650 with entropy 0.022 (d) IMG_20160519_172654 with entropy 0.030 (e) IMG_20160519_172637 with entropy 0.037

and the points from its Convex hull:

$$hull_{points} = \begin{bmatrix} 6.0 & -3.0 \\ 5.0 & 4.0 \\ -5.0 & 5.0 \\ -6.0 & 0.0 \\ 0.0 & -6.0 \end{bmatrix}$$

then using Equation 16 and Equation 1 yields:

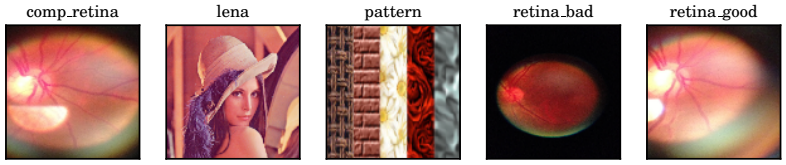
$$\begin{aligned} r_{convexity} &= \frac{Area_{object_{points}}}{Area_{hull_{points}}} \\ &= \frac{53.5}{93.0} \\ &= 0.575268817204 \end{aligned}$$

**Rectangularity.** This defines the rate to compare how much an object resembles a rectangle. The way to determine if an object resembles a rectangle is to compare its area from the area of a fitted rotated rectangle in the object and finding the ratio between these two.

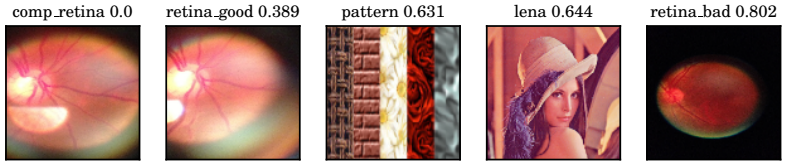
$$rectangularity = \frac{Area_{object}}{Area_{rectangular}} \tag{17}$$

Where $Area_{object}$ is the area of the object and $Area_{rectangular}$ the area of the fitted rotated rectangle that can be provided by OpenCV with the `BoxPoints` function (see OpenCV documentation for applications). A rectangular object will give a rectangularity of 1 while the rate of an object with a different shape will tend to 0. A simple example is show in Figure 34.

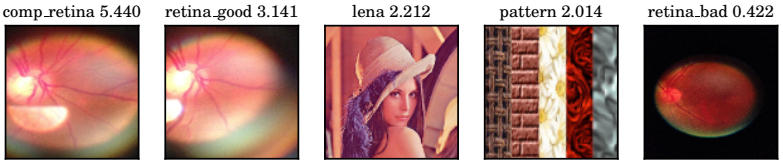
*Figure 30.* Ordered images using histogram comparison



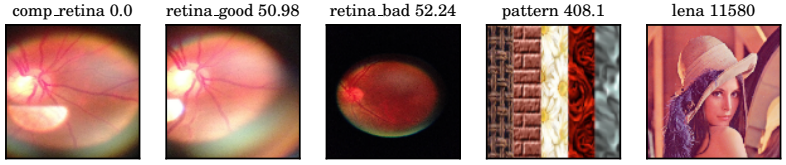

(a) Raw ordered images to sort using histogram comparison



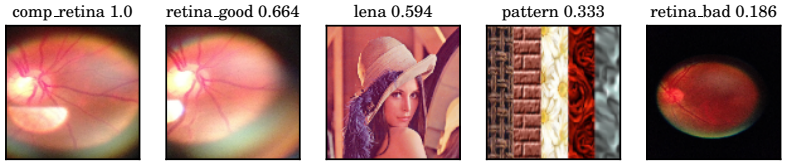

(b) Ordered images using Hellinger




(c) Ordered images using Intersection




(d) Ordered images using Chi-Squared




(e) Ordered images using Correlation

*Figure 31*. Matching example

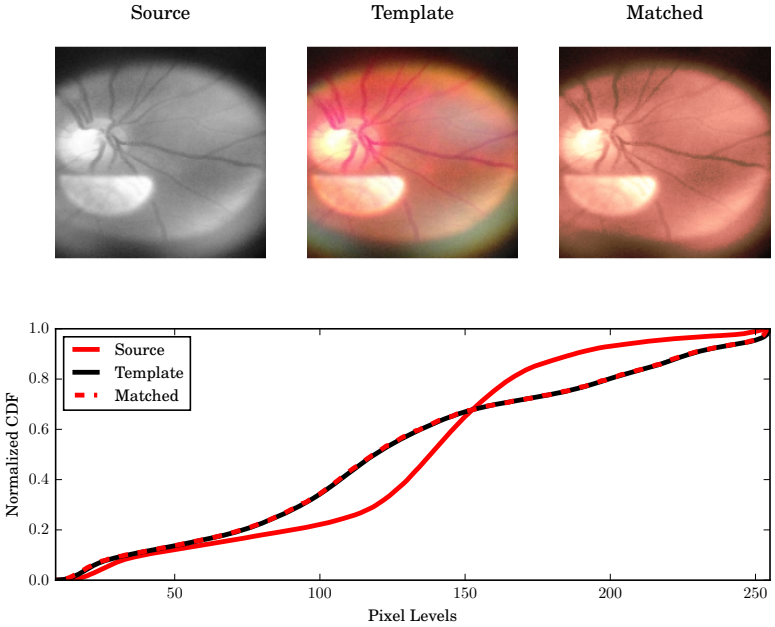


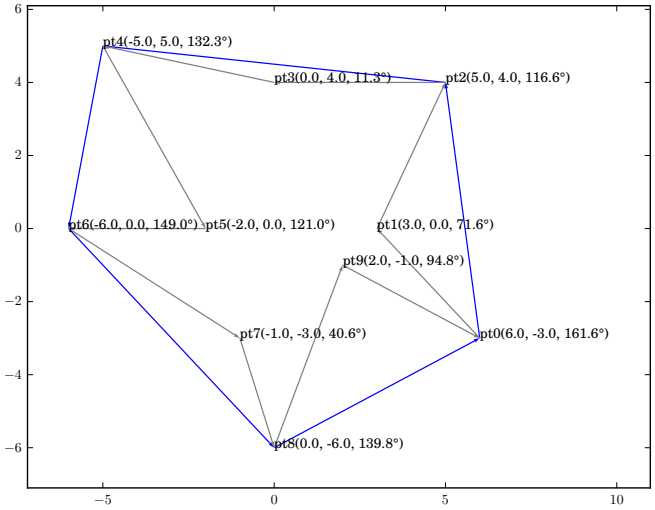

*Figure 32*. Features using SIFT

(a) SIFT in image with shape = (100,100) and rotated -30 degrees

(b) SIFT in image with shape = (300,300)

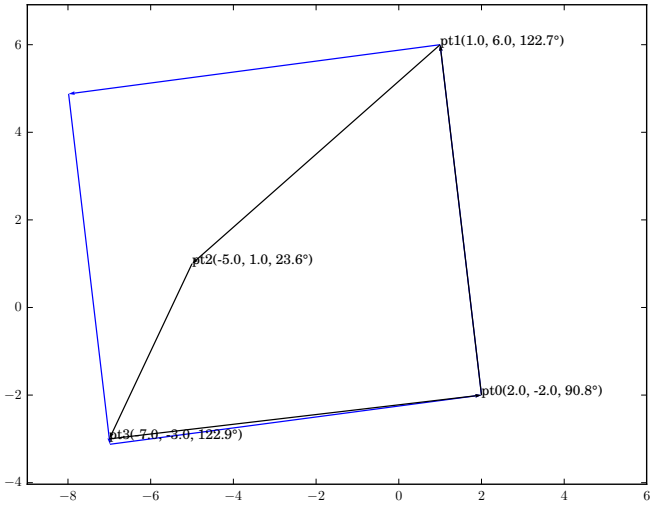
(c) SIFT in image with shape = (300,300) and rotated 30 degrees

*Figure 33*. Convexity ratio of an object




(a) ConvexityRatio = 0.575268817204

*Figure 34*. Rectangularity example




(a) Rectangularity: 0.595890402794

## Implementation

The *RRtoolbox* package has many algorithms in development but it already can create some useful applications. This section presents some of them compiled in a complete software called *imrestore* which is an application to restore images in general but in this case is configured to restore retinal images.

The *imrestore* program was developed using a prototyping method which consisted in building step by step the objectives of the restoration tool. Each finished stage of the program was followed by a exhaustive procedure of testing and debugging techniques using profilers, debugging messages, inspection of variables at each line of code, visualization of the processed images, optimizations (both in CPU performance and memory usage) and any needed corrections which leaded to the repetition of the cycle until no further bugs were found. After a stable stage, only then new features were added consisting of additional options to customize the results as well as the coding of the proceeding steps in the program.

Figure 35 shows the class diagram for the main classes of the *imrestore* program. This classes are in the *imrestore.py* script and are called using a *shell* invoked from a terminal or directly from a python environment using the *shell* function to interpret commands as in the terminal, instantiating *ImRestore* for general images or the *RetinalRestore* class for retinal images. The icon is for classes, fields, methods, deleter properties, getter properties and for setter properties.

- Local features: key-points and descriptors:
  - (1.1) SIFT, SURF, ORB, etc.
  - ASIFT*
- Select main or base image from set for merging: raw, sorting, user input
- Matching (spacial):
  - Filter 0.7 below Hamming distance
  - Key points classification
- Selection in matching set: (pre-selection of good matches)
  - Best matches: for general purpose
  - Entropy: used when set is ensured to be of the same object (The program ensures that, if it is not the case).
  - Histogram comparison: use if set contains unwanted perspectives or images that do not correspond to image.
  - Custom function
- Calculate Homography
- Probability tests: (ensures that the matches images correspond to each other)
- Stitching and merging
  - Histogram matching* (color)
  - Segmentation*
  - Alpha mask calculation*
  - Overlay
- Overall filtering*: bilateral filtering
- Lens simulation for retinal photos*

*Figure 35*. *imrestore.py* classes: *ImRestore* and its inheritor *RetinalRestore*


* optional

Everything related to the code is hosted in (David, 2016b) which contains the main source code package *RRtoolbox*, a development tool using sequential function charts called *RRtoolFC* (*FC* stands for Function Chart), documentation, tests, examples and the implemented *imrestore.py* script along with some supporting files. *RRtoolbox* documentation file is in (David, 2016a). The current algorithm needs more refining but has improved over precise segmentation of objects under irregular cases to offer robustness to noise areas like blurred areas,

flares, low brightness, low contrast between actual object and background (normal algorithms need clear distinctions between background and foreground where the histogram is bimodal and not multi-modal and are not prepared for any possible case).

Code 4: Imrestore's script code

```
#!/usr/local/bin/python
# -*- coding: utf-8 -*-
'''
imrestore (oriented to retinal images):

Restore images by merging and stitching techniques.

Optimization techniques:
    resize to smaller versions*

    memoization*:
        -persistence
        -serialization and de-serialization
        -caching

    multitasking*:
        -multiprocessing
        -multithreading

    lazy evaluations:
        -load on demand
        -use of weak references

    Memory mapped files*

STEPS:

    (1) Local features: Key-points and descriptors:
        -(1.1) SIFT, SURF, ORB, etc
        -ASIFT*

    (2) Select main or base image from set for merging:
        -Raw, Sorting, User input

    (3) Matching (spacial):
        -filter 0.7 below Hamming distance
        -key points classification

    (4) selection in matching set: (pre selection of good matches)
        (4.1) Best matches: for general purpose
        (4.2) Entropy: used when set is ensured to be of the same object
            (The program ensures that, if it is not the case).
        (4.3) Histogram comparison: use if set contains unwanted
            perspectives or images that do not correspond to image.
        (4.4) Custom function

    (5) Calculate Homography

    (6) Probability tests: (ensures that the matches images
    correspond to each other)

    (7) Stitching and Merging
        (7.1) Histogram matching* (color)
```

```
        (7.2) Segmentation*
        (7.3) Alpha mask calculation*
        (7.4) Overlay

    (8) Overall filtering*:
        Bilateral filtering

    (9) Lens simulation for retinal photos*

* optional

Notes:

    Optimization techniques:

        Resize to smaller versions: process smaller versions of the
        inputs and convert the result back to the original versions.
        This reduces processing times, standardize the way data is
        processed (with fixed sizes), lets limited memory to be used,
        allows to apply in big images without breaking down algorithms
        that cannot do that.

        Memoization:
            Persistence: save data to disk for later use.

            serialization and de-serialization: (serialization, in
            python is refereed as pickling) convert live objects into
            a format that can be recorded; (de-serialization, in python
            referred as unpickling) it is used to restore serialized
            data into "live" objects again as if the object was created
            in program conserving its data from precious sessions.

            Caching: saves the result of a function depending or not in
            the inputs to compute data once and keep retrieving it from
            the cached values if asked.

        Multitasking:
            Multiprocessing: pass tasks to several processes using the
                computer's cores to achieve concurrency.
            Multithreading: pass tasks to threads to use "clock-slicing"
                of a processor to achieve "concurrency".

        Lazy  evaluations:
            load on demand: if data is from an external local file, it is
            loaded only when it is needed to be computed otherwise it is
            deleted from memory or cached in cases where it is extensively
            used. For remotes images (e.g. a server, URL) or in a inadequate
            format, it is downloaded and converted to a numpy format in a
            temporal local place.

            Use of weak references: in cases where the data is cached or
            has not been garbage collected, data is retrieved through
            weak references and if it is needed but has been garbage
            collected it is load again and assigned to the weak reference.

        Memory mapped files:
        Instantiate an object and keep it not in memory but in a file and
        access it directly there. Used when memory is limited or data is
```

```
        too big to fit in memory. Slow downs are negligible for read only
        mmaped files (i.e. "r") considering the gain in free memory, but
        it is a real drawback for write operations (i.e. "w","r+","w+").

    Selection algorithms:
        Histogram comparison - used to quickly identify the images that
            most resemble a target
        Entropy - used to select the best enfoqued images of the same
            perspective of an object

    Local features: Key-points and descriptors:
        ASIFT: used to add a layer of robustness onto other local
        feature methods to cover all affine transformations. ASIFT
        was conceived to complete the invariance to transformations
        offered by SIFT which simulates zoom invariance using gaussian
        blurring and normalizes rotation and translation. This by
        simulating a set of views from the initial image, varying the
        two camera axis orientations: latitude and longitude angles,
        hence its acronym Affine-SIFT. Whereas SIFT stands for Scale
        Invariant Feature Transform.

    Matching (spacial):
        Calculate Homography: Used to find the transformation matrix
        to overlay a foreground onto a background image.

    Filtering:
        Bilateral filtering: used to filter noise and make the image
        colors more uniform (in some cases more cartoonist-like)

    Histogram matching (color): used to approximate the colors from the
        foreground to the background image.

    Segmentation: detect and individualize the target objects (e.g. optic
        disk, flares) to further process them or prevent them to be altered.

    Alfa mask calculation: It uses Alfa transparency obtained with sigmoid
        filters and binary masks from the segmentation to specify where an
        algorithm should have more effect or no effect at all
        (i.e. intensity driven).

    Stitching and Merging:
        This is an application point, where all previous algorithms are
        combined to stitch images so as to construct an scenery from the
        parts and merge them if overlapped or even take advantage of these
        to restore images by completing lacking information or enhancing
        poorly illuminated parts in the image. A drawback of this is that
        if not well processed and precise information is given or calculated
        the result could be if not equal worse than the initial images.

    Lens simulation for retinal photos: As its name implies, it is a
        post-processing method applied for better appeal of the image
        depending on the tastes of the user.
'''
from __future__ import division
# TODO install openCV 2.4.12 as described in http://stackoverflow.com/a/37283690/5288758
# to solve the error Process finished with exit code 139
# UPDATE: openCV 2.4.12 does not solve the error Process finished with exit code 139

```

```
__author__ = 'Davtoh'
# needed for installing executable
# program imports
import os
import cv2
import warnings
import numpy as np
from RRtoolbox.tools.lens import simulateLens
from RRtoolbox.lib.config import MANAGER, FLOAT
from RRtoolbox.lib.image import hist_match
from RRtoolbox.lib.directory import getData, getPath, mkPath, increment_if_exits
from RRtoolbox.lib.cache import MemoizedDict, LazyDict
from RRtoolbox.lib.image import loadFunc, ImCoors
from RRtoolbox.lib.arrayops.mask import brightness, foreground, thresh_biggestCnt
from multiprocessing.pool import ThreadPool as Pool
from RRtoolbox.tools.selectors import hist_map, hist_comp, entropy
from RRtoolbox.tools.segmentation import retinal_mask, get_layered_alpha
from RRtoolbox.lib.root import TimeCode, glob, lookinglob, Profiler, VariableNotSettable, globFilter
from RRtoolbox.lib.descriptors import Feature, inlineRatio
from RRtoolbox.tools.segmentation import get_bright_alpha, Bandpass, Bandstop
from RRtoolbox.lib.plotter import MatchExplorer, Plotim, fastplt
from RRtoolbox.lib.arrayops.filters import getBilateralParameters
from RRtoolbox.lib.arrayops.convert import getSOpointRelation, dict2keyPoint
from RRtoolbox.lib.arrayops.basic import getTransformedCorners, transformPoint, \
    im2shapeFormat,normalize, getOtsuThresh, contours2mask, pad_to_fit_H, overlay


def check_valid(fn):
    """
    checks that a file is valid for loading.
    :param fn: filename
    :return: True for valid, False for invalid.
    """
    test = os.path.isfile(fn)
    if test and getData(fn)[-2].startswith("_"):
        return False
    return test

class ImRestore(object):
    """
    Restore images by merging and stitching techniques.

    :param filenames: list of images or string to path which uses glob filter in path.
            Loads image array from path, url, server, string
            or directly from numpy array (supports databases)
    :param debug: (0) flag to print messages and debug data.
            0 -> do not print messages.
            1 -> print normal messages.
            2 -> print normal and debug messages.
            3 -> print all messages and show main results.
                (consumes significantly more memory).
            4 -> print all messages and show all stage results.
                (consumes significantly more memory).
            5 -> print all messages, show all results and additional data.
                (consumes significantly more memory).
    :param feature: (None) feature instance. It contains the configured
            detector and matcher.
    :param pool: (None) use pool Ex: 4 to use 4 CPUs.
```

```
228     :param cachePath: (None) saves memoization to specified path. This is
229             useful to save some computations and use them in next executions.
230             If True it creates the cache in current path.
231
232             .. warning:: Cached data is not guaranteed to work between different
233                          configurations and this can lead to unexpected program
234                          behaviour. If a different configuration will be used it
235                          is recommended to clear the cache to recompute values.
236     :param clearCache: (0) clear cache flag.
237             * 0 do not clear.
238             * 1 check data integrity of previous session before use
239             * 2 re-compute data but other cache data is left intact.
240             * 3 All CachePath is cleared before use.
241             Notes: using cache can result in unexpected behaviour
242                 if some configurations does not match to the cached data.
243     :param loader: (None) custom loader function used to load images.
244             If None it loads the original images in color.
245     :param process_shape: (400,400) process shape, used to load pseudo images
246             to process features and then results are converted to the
247             original images. The smaller the image more memory and speed gain
248             If None it loads the original images to process the features but it
249             can incur to performance penalties if images are too big and RAM
250             memory is scarce.
251     :param load_shape: (None) custom shape used load images which are being merged.
252     :param baseImage: (None) First image to merge to.
253             * None -> takes first image from raw list.
254             * True -> selects image with most features.
255             * Image Name.
256     :param selectMethod: (None) Method to sort images when matching. This
257             way the merging order can be controlled.
258             * (None) Best matches.
259             * Histogram Comparison: Correlation, Chi-squared,
260                 Intersection, Hellinger or any method found in hist_map
261             * Entropy.
262             * custom function of the form: rating,fn <-- selectMethod(fns)
263     :param distanceThresh: (0.75) filter matches by distance ratio.
264     :param inlineThresh: (0.2) filter homography by inlineratio.
265     :param rectangularityThresh: (0.5) filter homography by rectangularity.
266     :param ransacReprojThreshold: (5.0) maximum allowed reprojection error
267             to treat a point pair as an inlier.
268     :param centric: (False) tries to attach as many images as possible to
269             each matching. It is quicker since it does not have to process
270             too many match computations.
271     :param hist_match: (False) apply histogram matching to foreground
272             image with merge image as template
273     :param grow_scene: If True, allow the restored image to grow in shape if
274             necessary at the merging process.
275     :param expert: Path to an expert database. If provided it will use this data
276             to generate the mask used when merging to the restored image.
277     :param maskforeground:(False)
278             * True, limit features area using foreground mask of input images.
279                 This mask is calculated to threshold a well defined object.
280             * Callable, Custom function to produce the foreground image which
281                 receives the input gray image and must return the mask image
282                 where the keypoints will be processed.
283     :param noisefunc: True to process noisy images or provide function.
284     :param save: (False)
285             * True, saves in path with name _restored_{base_image}
```

```
286             * False, does not save
287             * Image name used to save the restored image.
288     :param overwrite: If True and the destine filename for saving already
289             exists then it is replaced, else a new filename is generated
290             with an index "{filename}_{index}.{extension}"
291     """
292
293     def __init__(self, filenames, **opts):
294         self.profiler = opts.pop("profiler",None)
295         if self.profiler is None:
296             self.profiler = Profiler("ImRestore init")
297
298         self.log_saved = [] # keeps track of last saved file.
299
300         # for debug
301         self.verbosity = opts.pop("verbosity", 1)
302
303         ################################## GET IMAGES ###################################
304         if filenames is None or len(filenames)==0: # if images is empty use demonstration
305             #test = MANAGER["TESTPATH"]
306             #if self.verbosity: print "Looking in DEMO path {}".format(test)
307             #fns = glob(test + "*",check=check_valid)
308             raise Exception("List of filenames is Empty")
309         elif isinstance(filenames, basestring):
310             # if string assume it is a path
311             if self.verbosity: print "Looking as {}".format(filenames)
312             fns = glob(filenames,check=check_valid)
313         elif not isinstance(filenames, basestring) and \
314                         len(filenames) == 1 and "*" in filenames[0]:
315             filenames = filenames[0] # get string
316             if self.verbosity: print "Looking as {}".format(filenames)
317             fns = glob(filenames,check=check_valid)
318         else: # iterator containing data
319             fns = filenames # list file names
320
321         # check images
322         if not len(fns)>1:
323             raise Exception("list of images must be "
324                             "greater than 1, got {}".format(len(fns)))
325
326         # for multiprocessing
327         self.pool = opts.pop("pool",None)
328         if self.pool is not None: # convert pool count to pool class
329             NO_CPU = cv2.popNumberOfCPUs()
330             if self.pool <= NO_CPU:
331                 self.pool = Pool(processes = self.pool)
332             else:
333                 raise Exception("pool of {} exceeds the "
334                                 "number of processors {}".format(self.pool,NO_CPU))
335         # for features
336         self.feature = opts.pop("feature",None)
337         # init detector and matcher to compute descriptors
338         if self.feature is None:
339             self.feature = Feature(pool=self.pool, debug=self.verbosity)
340             self.feature.config(name='a-sift-flann')
341         else:
342             self.feature.pool = self.pool
343             self.feature.debug = self.verbosity
```

```
        # select method to order images to feed in superposition
        self.selectMethod = opts.pop("selectMethod",None)
        best_match_list = ("bestmatches", "best matches")
        entropy_list = ("entropy",)
        if callable(self.selectMethod):
            self._orderValue = 3
        elif self.selectMethod in hist_map:
            self._orderValue = 2
        elif self.selectMethod in entropy_list:
            self._orderValue = 1
        elif self.selectMethod in best_match_list or self.selectMethod is None:
            self._orderValue = 0
        else:
            raise Exception("selectMethod {} not recognized".format(self.selectMethod))

        # distance threshold to filter best matches
        self.distanceThresh = opts.pop("distanceThresh",0.75) # filter ratio

        # threshold for inlineRatio
        self.inlineThresh = opts.pop("inlineThresh",0.2) # filter ratio
        # ensures adequate value [0,1]
        assert self.inlineThresh<=1 and self.inlineThresh>=0

        # threshold for rectangularity
        self.rectangularityThresh = opts.pop("rectangularityThresh",0.5) # filter ratio
        # ensures adequate value [0,1]
        assert self.rectangularityThresh<=1 and self.rectangularityThresh>=0

        # threshold to for RANSAC reprojection
        self.ransacReprojThreshold = opts.pop("ransacReprojThreshold",5.0)

        self.centric = opts.pop("centric",False) # tries to attach as many images as possible
        # it is not memory efficient to compute descriptors from big images
        self.process_shape = opts.pop("process_shape", (400, 400)) # use processing shape
        self.load_shape = opts.pop("load_shape", None) # shape to load images for merging
        self.minKps = 3 # minimum len of key-points to find Homography
        self.histMatch = opts.pop("hist_match",False)
        self.denoise=opts.pop("denoise", None)

        ############################## OPTIMIZATION MEMOIZEDIC ##########################
        self.cachePath = opts.pop("cachePath",None)
        self.clearCache = opts.pop("clearCache",0)

        self.expert = opts.pop("expert",None)
        if self.expert is not None:
            self.expert = MemoizedDict(self.expert) # convert path

        # to select base image ahead of any process
        baseImage = opts.pop("baseImage",None)
        if isinstance(baseImage,basestring):
            if baseImage not in fns:
                base, path, name, ext = getData(baseImage)
                if not path: # if name is incomplete look for it
                    base, path, name, ext = getData(fns[0])
                try: # tries user input
                    # selected image must be in fns
                    baseImage = lookinglob(baseImage,
```

```
                                        path= "".join((base, path)),
                                        filelist=fns, raiseErr=True)#,ext=".*"
                except Exception as e: # tries to find image based in user input
                    # generate informative error for the user
                    try:
                        # look in the file pattern path
                        baseImage = lookinglob(baseImage,raiseErr=True)#,ext=".*"
                        # append new image
                        fns.append(baseImage)
                    except Exception as e2:
                        e.args = e.args + e2.args + \
                                 ("A pattern could be '{}'".format("".join((name,".*"))),)
                        raise e

        self.baseImage = baseImage

        if self.verbosity: print "No. images {}...".format(len(fns))

        # assign filenames
        self.filenames = fns

        # make loader
        self.loader = opts.pop("loader",None) # BGR loader
        if self.loader is None: self.loader = loadFunc(1)
        self._loader_cache = None # keeps last image reference
        self._loader_params = None # keeps last track of last loading options to reload

        self.save = opts.pop("save",False)
        self.grow_scene = opts.pop("grow_scene",True)
        self.maskforeground = opts.pop("maskforeground",False)
        self.overwrite = opts.pop("overwrite",False)

        # do a check of the options
        if opts:
            raise Exception("Unknown keyword(s) {}".format(opts.keys()))

        # processing variables
        self._feature_list = None
        self._feature_dic = None
        self.used = None
        self.failed = None
        self.restored = None
        self.kps_base,self.desc_base = None,None

    @property
    def denoise(self):
        return self._noisefunc
    @denoise.setter
    def denoise(self, value):
        if value is False:
            value = None
        if value is True:
            value = "mild"
        if value in ("mild","heavy","normal",None) or callable(value):
            self._noisefunc = value
        else:
            raise Exception("denoise '{}' not recognised".format(value))
    @denoise.deleter
```

```
460     def denoise(self):
461         del self._noisefunc
462 
463     @property
464     def feature_list(self):
465         if self._feature_list is None:
466             return self.compute_keypoints()
467         return self._feature_list
468     @feature_list.setter
469     def feature_list(self,value):
470         raise VariableNotSettable("feature_list is not settable")
471     @feature_list.deleter
472     def feature_list(self):
473         self._feature_list = None
474 
475     @property
476     def feature_dic(self):
477         if self._feature_dic is None:
478             if self.cachePath is not None:
479                 if self.cachePath is True:
480                     self.cachePath = os.path.abspath(".") # MANAGER["TEMPPATH"]
481                 if self.cachePath == "{temp}":
482                     self.cachePath = self.cachePath.format(temp=MANAGER["TEMPPATH"])
483                 memoized = MemoizedDict(os.path.join(self.cachePath, "descriptors"))
484                 if self.verbosity: print "Cache path is in {}".format(memoized._path)
485                 self._feature_dic = LazyDict(getter=self.compute_keypoint,
486                                              dictionary=memoized)
487                 if self.clearCache==3:
488                     self._feature_dic.clear()
489                     if self.verbosity: print "Cache path cleared"
490             else:
491                 self._feature_dic = LazyDict(getter=self.compute_keypoint)
492             if self.clearCache==1 or self.clearCache==2:
493                 # tell LazyDict to recompute data if key is requested
494                 self._feature_dic.cached = False
495         return self._feature_dic
496     @feature_dic.setter
497     def feature_dic(self,value):
498         self._feature_dic = value
499     @feature_dic.deleter
500     def feature_dic(self):
501         self._feature_dic = None
502 
503     def load_image(self, path=None, shape=None):
504         """
505         load image from source
506 
507         :param path: filename, url, .npy, server, image in string
508         :param shape: shape to convert image
509         :return: BGR image
510         """
511         params = (path, shape)
512         if self._loader_cache is None or params != self._loader_params:
513             # load new image and cache it
514             img = self.loader(path) # load image
515             if shape is not None:
516                 img = cv2.resize(img,shape)
517             self._loader_cache = img # this keeps a reference
```

```
            self._loader_params = params
            return img
        else: # return cached image
            return self._loader_cache

    def compute_keypoint(self,path):
        img = self.load_image(path, self.load_shape)
        lshape = img.shape[:2]
        try:
            if self.cachePath is None:
                point = Profiler(msg=path, tag="cached")
            else:
                point = Profiler(msg=path, tag="memoized")

            # compare safely if path is in dictionary, this works for LazyDic,
            # MemoizeDic or normal dictionaries
            if path not in self.feature_dic or self.clearCache==2 and path in self.feature_dic:
                raise KeyError # clears entry from cache
            kps, desc, pshape = self.feature_dic[path] # thread safe
            if pshape is None:
                raise ValueError
            if self.verbosity: print "{} is cached...".format(path)
        except (KeyError, ValueError) as e: # not memorized
            point = Profiler(msg=path, tag="processed")
            if self.verbosity: print "Processing features for {}...".format(path)
            if lshape != self.process_shape:
                img = cv2.resize(img, self.process_shape)
            img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
            # get features
            if self.maskforeground is None:
                kps, desc = self.feature.detectAndCompute(img)
            else:
                mask = None
                if callable(self.maskforeground):
                    mask = self.maskforeground(img)
                if self.maskforeground is True:
                    mask = foreground(img)

                if mask is not None and self.verbosity > 4:
                    try:
                        fastplt(overlay(img.copy(),mask*255,alpha=mask*0.5),block=True,
                                title="{} mask to detect features".format(getData(path)[-2]))
                    except:
                        pass

                kps, desc = self.feature.detectAndCompute(img, mask)
            pshape = img.shape[:2] # get process shape
            # to memoize
            self.feature_dic[path] = kps, desc, pshape
        # re-scale keypoints to original image
        if lshape != pshape:
            # this necessarily does not produce the same result
            """
            # METHOD 1: using Transformation Matrix
            H = getSOpointRelation(process_shape, lshape, True)
            for kp in kps:
                kp["pt"]=tuple(cv2.perspectiveTransform(
                    np.array([[kp["pt"]]]), H).reshape(-1, 2)[0])
```

```
            """
            # METHOD 2:
            rx,ry = getSOpointRelation(pshape, lshape)
            for kp in kps:
                x,y = kp["pt"]
                kp["pt"] = x*rx,y*ry
                kp["path"] = path
        else:
            for kp in kps: # be very carful, this should not appear in self.feature_dic
                kp["path"] = path # add paths to key-points
        # for profiling individual processing times
        if self.profiler is not None: self.profiler._close_point(point)
        return kps, desc, pshape

    def compute_keypoints(self):
        """
        Computes key-points from file names.

        :return: self.feature_list
        """
        ################### Local features: Key-points and descriptors #################
        with TimeCode("Computing features...\n", profiler=self.profiler,
                      profile_point=("Computing features",),
                      endmsg="Computed feature time was {time}\n",
                      enableMsg=self.verbosity) as timerK:
            fns = self.filenames
            self._feature_list = [] # list of key points and descriptors
            for index,path in enumerate(fns):
                if self.verbosity: print "\rFeatures {}/{}...".format(index + 1, len(fns)),
                kps, desc, pshape = self.compute_keypoint(path)
                # number of key-points, index, path, key-points, descriptors
                self._feature_list.append((len(kps),index,path,kps,desc))

        return self._feature_list

    def pre_selection(self):
        """
        This method selects the first restored image so that self.restored is initialized
        with a numpy array and self.used should specify the used image preferably in
        self.feature_list.

        :return: None
        """
        ########################### Pre-selection from a set ############################
        baseImage = self.baseImage # baseImage option should not be update
        # initialization and base image selection
        if baseImage is None: # select first image as baseImage
           _,_,baseImage,self.kps_base,self.desc_base = self.feature_list[0]
        elif isinstance(baseImage,basestring):
            self.kps_base,self.desc_base,_ = self.feature_dic[baseImage]
        elif baseImage is True: # sort images
            self.feature_list.sort(reverse=True) # descendant: from bigger to least
            # select first for most probable
            _,_,baseImage,self.kps_base,self.desc_base = self.feature_list[0]
        else:
            raise Exception("baseImage must be None, True or String")

        if self.verbosity: print "baseImage is", baseImage
```

```
        self.used = [baseImage] # select first image path
        # load first image for merged image
        self.restored = self.load_image(baseImage, self.load_shape)

    def restore(self):
        """
        Restore using file names (self.file_names) with base image (self.baseImage
        calculated from self.pre_selection()) and other configurations.

        :return: self.restored
        """
        self.pre_selection()
        self.failed = [] # registry for failed images
        fns = self.filenames # in this process fns should not be changed
        ########################## Order set initialization ############################
        if self._orderValue: # obtain comparison with structure (value, path)
            if self._orderValue == 1: # entropy
                comparison = zip(*entropy(fns, loadfunc=loadFunc(1, self.process_shape),
                                          invert=False)[:2])
                if self.verbosity: print "Configured to sort by entropy..."
            elif self._orderValue == 2: # histogram comparison
                comparison = hist_comp(fns, loadfunc=loadFunc(1, self.process_shape),
                                       method=self.selectMethod)
                if self.verbosity:
                    print "Configured to sort by {}...".format(self.selectMethod)
            elif self._orderValue == 3:
                comparison = self.selectMethod(fns)
                if self.verbosity: print "Configured to sort by Custom Function..."
            else:
                raise Exception("DEBUG: orderValue {} does "
                    "not correspond to {}".format(self._orderValue,self.selectMethod))
        elif self.verbosity: print "Configured to sort by best matches"

        with TimeCode("Restoring ...\n",profiler=self.profiler,
                      profile_point=("Restoring",),
                      endmsg= "Restoring overall time was {time}\n",
                      enableMsg= self.verbosity) as timerR:

            while True:
                with TimeCode("Matching ...\n",profiler=self.profiler,
                              profile_point=("Matching",),
                              endmsg= "Matching overall time was {time}\n",
                              enableMsg= self.verbosity) as timerM:
                    ################### remaining keypoints to match ###################
                    # initialize key-point and descriptor base list
                    kps_remain,desc_remain = [],[]
                    for _,_,path,kps,desc in self.feature_list:
                        # append only those which are not in the base image
                        if path not in self.used:
                            kps_remain.extend(kps)
                            desc_remain.extend(desc)

                    if not kps_remain: # if there is not image remaining to stitch break
                        if self.verbosity:
                            if self.failed:
                                print "No image remains to merge..."
                            else:
                                print "All images have been merged..."
```

```
                    break

                desc_remain = np.array(desc_remain) # convert descriptors to array

                ############################ Matching ################################
                # select only those with good distance (hamming, L1, L2)
                raw_matches = self.feature.matcher.knnMatch(desc_remain,
                                trainDescriptors = self.desc_base, k = 2) #2
                # If path=2, it will draw two match-lines for each key-point.
                classified = {}
                for m in raw_matches:
                    # filter by Hamming, L1 or L2 distance
                    if m[0].distance < m[1].distance * self.distanceThresh:
                        m = m[0]
                        kp1 = kps_remain[m.queryIdx]  # keypoint in query image
                        kp2 = self.kps_base[m.trainIdx]  # keypoint in train image

                        key = kp1["path"] # ensured that key is not in used
                        if key in classified:
                            classified[key].append((kp1,kp2))
                        else:
                            classified[key] = [(kp1,kp2)]

                ########################## Order set ##################################
                # use only those in classified of histogram or entropy comparison
                if self._orderValue:
                    ordered = [(val,path) for val,path
                               in comparison if path in classified]
                else: # order with best matches
                    ordered = sorted([(len(kps),path)
                                for path,kps in classified.items()],reverse=True)


            with TimeCode("Merging ...\n",profiler=self.profiler,
                          profile_point=("Merging",),
                          endmsg= "Merging overall time was {time}\n",
                          enableMsg= self.verbosity) as timerH:

                # feed key-points in order according to order set
                for rank, path in ordered:
                    point = Profiler(msg=path) # profiling point
                    ######################### Calculate Homography ##################
                    mkp1,mkp2 = zip(*classified[path]) # probably good matches
                    if len(mkp1)>self.minKps and len(mkp2)>self.minKps:

                        # get only key-points
                        p1 = np.float32([kp["pt"] for kp in mkp1])
                        p2 = np.float32([kp["pt"] for kp in mkp2])
                        if self.verbosity > 4:
                            print 'Calculating Homography for {}...'.format(path)

                        # Calculate homography of fore over back
                        H, status = cv2.findHomography(p1, p2,
                                        cv2.RANSAC, self.ransacReprojThreshold)
                    else: # not sufficient key-points
                        if self.verbosity > 1:
                            print 'Not enough key-points for {}...'.format(path)
                        H = None
```

```
                    # test that there is homography
                    if H is not None: # first test
                        # load fore image
                        fore = self.load_image(path, self.load_shape)
                        h,w = fore.shape[:2] # image shape

                        # get corners of fore projection over back
                        projection = getTransformedCorners((h,w),H)
                        c = ImCoors(projection) # class to calculate statistical data
                        lines, inlines = len(status), np.sum(status)

                        # ratio to determine how good fore is in back
                        inlineratio = inlineRatio(inlines,lines)

                        Test = inlineratio>self.inlineThresh \
                                and c.rotatedRectangularity>self.rectangularityThresh

                        text = "inlines/lines: {}/{}={}, " \
                               "rectangularity: {}, test: {}".format(
                            inlines, lines, inlineratio, c.rotatedRectangularity,
                            ("failed","succeeded")[Test])

                        if self.verbosity>1: print text

                        if self.verbosity > 3: # show matches
                            MatchExplorer("Match " + text, fore,
                                          self.restored, classified[path], status, H)

                        ######################## probability test ####################
                        if Test: # second test

                            if self.verbosity>1: print "Test succeeded..."
                            while path in self.failed: # clean path in fail registry
                                try: # race-conditions safe
                                    self.failed.remove(path)
                                except ValueError:
                                    pass

                            ################### merging and stitching ################
                            self.merge(path, H)

                            # used for profiling
                            if self.profiler is not None:
                                self.profiler._close_point(point)

                            if not self.centric:
                                break
                        else:
                            self.failed.append(path)
                    else:
                        self.failed.append(path)

                # if all classified have failed then end
                if set(classified.keys()) == set(self.failed):
                    # in the classification there could have been some filtered keys that
                    # did not pass the distance test, these are assigned to the failed list
                    self.failed.extend(list(set(self.used) ^
```

```
                                        set(self.failed) ^
                                        set(self.filenames)))
                    if self.verbosity:
                        print "Restoration finished, these images do not fit: "
                        for p in self.failed:
                            print p

                    break

        with TimeCode("Post-processing ...\n",profiler=self.profiler,
                      profile_point=("Post-processing",),
                      endmsg= "Post-processing overall time was {time}\n",
                      enableMsg= self.verbosity) as timerP:
            processed = self.post_process_restoration(self.restored)
            if processed is not None:
                self.restored = processed

        # profiling post-processing
        self.time_postprocessing = timerP.time_end

        ################################### Save image ###################################
        if self.save:
            self.save_image()

        return self.restored # return merged image

    def save_image(self, path = None, overwrite = None):
        """
        save restored image in path.

        :param path: filename, string to format or path to save image.
                if path is not a string it would be replaced with the string
                "{path}restored_{name}{ext}" to format with the formatting
                "{path}", "{name}" and "{ext}" from the baseImage variable.
        :param overwrite: If True and the destine filename for saving already
            exists then it is replaced, else a new filename is generated
            with an index "{filename}_{index}.{extension}"
        :return: saved path, status (True for success and False for fail)
        """
        if path is None:
            path = self.save
        if overwrite is None:
            overwrite = self.overwrite

        bbase, bpath, bname, bext = getData(self.used[0])
        if isinstance(path,basestring):
            # format path if user has specified so
            data = getData(self.save.format(path="".join((bbase, bpath)),
                                            name=bname, ext=bext))
            # complete any data lacking in path
            for i,(n,b) in enumerate(zip(data,(bbase, bpath, bname, bext))):
                if not n: data[i] = b
        else:
            data = bbase,bpath,"_restored_",bname,bext
        # joint parts to get string
        fn = "".join(data)
        mkPath(getPath(fn))

```

```
866         if not overwrite:
867             fn = increment_if_exits(fn)
868 
869         if cv2.imwrite(fn,self.restored):
870             if self.verbosity: print "Saved: {}".format(fn)
871             self.log_saved.append(fn)
872             return fn, True
873         else:
874             if self.verbosity: print "{} could not be saved".format(fn)
875             return fn, False
876 
877     def merge(self, path, H, shape = None):
878         """
879         Merge image to main restored image.
880 
881         :param path: file name to load image
882         :param H: Transformation matrix of image in path over restored image.
883         :param shape: custom shape to load image in path
884         :return: self.restored
885         """
886         alpha = None
887 
888         if shape is None:
889             shape = self.load_shape
890 
891         fore = self.load_image(path, shape) # load fore image
892 
893         if self.histMatch: # apply histogram matching
894             fore = hist_match(fore, self.restored)
895 
896         if self.verbosity > 1: print "Merging..."
897 
898         # process expert alpha mask if alpha was not provided by the user
899         if self.expert is not None:
900 
901             # process _restored_mask if None
902             if not hasattr(self,"_restored_mask"):
903                 # from path/name.ext get only name.ext
904                 bname = "".join(getData(self.used[-1])[-2:])
905                 try:
906                     bdata = self.expert[bname]
907                     bsh = bdata["shape"]
908                     bm_retina = contours2mask(bdata["coors_retina"],bsh)
909                     bm_otic_disc = contours2mask(bdata["coors_optic_disc"],bsh)
910                     bm_defects = contours2mask(bdata["coors_defects"],bsh)
911 
912                     self._restored_mask = np.logical_and(np.logical_or(np.logical_not(bm_retina),
913                                             bm_defects), np.logical_not(bm_otic_disc))
914                 except Exception as e:
915                     #exc_type, exc_value, exc_traceback = sys.exc_info()
916                     #lines = traceback.format_exception(exc_type, exc_value, exc_traceback)
917                     warnings.warn("Error using expert {} to create self._restored_mask:"
918                                   " {}{}".format(bname,type(e),e.args))
919 
920             # only if there is a _restored_mask
921             if hasattr(self,"_restored_mask"):
922                 fname = "".join(getData(path)[-2:])
923                 try:
```

```
                    fdata = self.expert[fname]
                    fsh = fdata["shape"]
                    fm_retina = contours2mask(fdata["coors_retina"],fsh)
                    #fm_otic_disc = contours2mask(fdata["coors_optic_disc"],fsh)
                    fm_defects = contours2mask(fdata["coors_defects"],fsh)

                    fmask = np.logical_and(fm_retina,np.logical_not(fm_defects))

                    self._restored_mask = maskm = np.logical_and(self._restored_mask,fmask)

                    h, w = self.restored.shape[:2]
                    alpha = cv2.warpPerspective(maskm.copy().astype(np.uint8), H, (w, h))
                except Exception as e:
                    warnings.warn("Error using expert {} to create alpha mask:"
                                  " {}{}".format(fname,type(e),e.args))

        if alpha is None:
            # pre process alpha mask
            alpha = self.pre_process_fore_Mask(self.restored,fore,H)

        ################################### SUPERPOSE ###################################

        # fore on top of back
        alpha_shape = fore.shape[:2]
        if self.grow_scene: # this makes the images bigger if possible
            # fore(x,y)*H = fore(u,v) -> fore(u,v) + back(u,v)
            ((left,top),(right,bottom)) = pad_to_fit_H(fore.shape, self.restored.shape, H)
             # moved transformation matrix with pad
            H_back = FLOAT([[1,0,left],[0,1,top],[0,0,1]]) # in back
            H_fore = H_back.dot(H) # in fore
            # need: top_left, bottom_left, top_right,bottom_right
            h2,w2 = self.restored.shape[:2]
            w,h = int(left + right + w2),int(top + bottom + h2)
            # this memory inefficient, image is copied to prevent cross-references
            self.restored = cv2.warpPerspective(self.restored.copy(), H_back, (w, h))
            fore = cv2.warpPerspective(fore.copy(), H_fore, (w, h))
        else: # this keeps back shape
            H_fore = H
            H_back = np.eye(3)
            h, w = self.restored.shape[:2]
            fore = cv2.warpPerspective(fore.copy(), H_fore, (w, h))

        if alpha is None: # if no pre-processing function for alpha implemented
            alpha = self.post_process_fore_Mask(self.restored,fore)

        if alpha is None: # create valid mask for stitching
            alpha = cv2.warpPerspective(np.ones(alpha_shape), H_fore, (w, h))

        if self.verbosity > 3: # show merging result
            fastplt(alpha, title="alpha mask from {}".format(path),block=True)

        self.restored = overlay(self.restored, fore, alpha) # overlay fore on top of back

        if H_fore is None: # H is not modified use itself
            H_fore = H

        if self.verbosity > 4: # show merging result
            Plotim("Last added from {}".format(path), self.restored).show()
```

```
982
983         ###################### update base features ######################
984         # make projection to test key-points inside it
985         if self.verbosity > 1: print "Updating key-points..."
986         # fore projection in restored image
987         projection = getTransformedCorners(fore.shape[:2],H_fore)
988         # update key-points
989         newkps, newdesc = [], []
990         for _,_,p,kps,desc in self.feature_list:
991             # append all points in the base image and update their position
992             if p in self.used: # transform points in back
993                 for kp,dsc in zip(kps,desc): # kps,desc
994                     pt = kp["pt"] # get point
995                     if H_back is not None: # update point
996                         pt = tuple(transformPoint(pt,H_back))
997                         kp["pt"] = pt
998                     # include only those points outside foreground
999                     if cv2.pointPolygonTest(projection, pt, False) == -1:
1000                        newkps.append(kp)
1001                        newdesc.append(dsc)
1002            elif p == path: # transform points in fore
1003                # include only those points inside foreground
1004                for kp,dsc in zip(kps,desc): # kps,desc
1005                    kp["pt"] = tuple(transformPoint(kp["pt"],H_fore))
1006                    newkps.append(kp)
1007                    newdesc.append(dsc)
1008        # update self.kps_base and self.desc_base
1009        self.kps_base = newkps
1010        self.desc_base = np.array(newdesc)
1011
1012        if self.verbosity > 4: # show keypints in merging
1013            Plotim("merged Key-points",  # draw key-points in image
1014                   cv2.drawKeypoints(
1015                       im2shapeFormat(self.restored,self.restored.shape[:2]+(3,)),
1016                              [dict2keyPoint(index) for index in self.kps_base],
1017                              flags=4, color=(0,0,255))).show()
1018        if self.verbosity: print "This image has been merged: {}...".format(path)
1019        self.used.append(path) # update used
1020
1021        return self.restored
1022
1023    def post_process_restoration(self, image):
1024        """
1025        Post-process a merged retinal image.
1026
1027        :param image: retinal image
1028        :return: filtered and with simulated lens
1029        """
1030        if callable(self.denoise):
1031            return self.denoise(image)
1032        # detect how much noise to process and convert it to beta parameters
1033        if self.denoise is not None:
1034            # slower but interactive for heavy noise
1035            # filter using parameters and bilateral filter
1036            params = getBilateralParameters(image.shape, self.denoise)
1037            return cv2.bilateralFilter(image, *params)
1038
1039    def post_process_fore_Mask(self, back, fore):
```

```
        """
        Method to post-process fore mask used after fore and back are transformed
        to a new domain for merging. This method is called by the merge method in the
        event that an alpha mask has not been created by self.pre_process_fore_Mask
        method or obtained with the self.expert variables.

        :param back: background image. This is called by method self.merge
                with self.restored
        :param fore: fore ground image.
        :return: alpha mask with shape (None,None)
        """
        pass

    def pre_process_fore_Mask(self, back, fore, H):
        """
        Method to pre-process fore mask used before fore and back are transformed
        to a new domain for merging.

        :param back:
        :param fore:
        :param H:
        :return: alpha mask with shape (None,None)
        """
        pass

class RetinalRestore(ImRestore):
    """
    Restore retinal images by merging and stitching techniques. These parameters are
    added to :class:`ImRestore`:

    :param lens: flag to determine if lens are applied. True
        to simulate lens, False to not apply lens.
    :param enclose: flag to enclose and return only retinal area.
        True to return ROI, false to leave image "as is".
    """
    def __init__(self, filenames, **opts):
        # overwrite variables
        opts["denoise"]=opts.pop("denoise", True)
        opts["maskforeground"] = opts.pop("maskforeground",lambda img: retinal_mask(img,True))
        # create new variables
        self.lens = opts.pop("lens",False)
        self.enclose = opts.pop("enclose",False)
        # call super class
        super(RetinalRestore,self).__init__(filenames, **opts)

    #__init__.__doc__ = ImRestore.__init__.__doc__+__init__.__doc__

    def post_process_fore_Mask(self, back, fore):
        """
        Function evaluated just after superposition and before overlay.

        :param back: background image
        :param fore: foreground image
        :return: alpha mask
        """

        def _mask(back,fore):
            """
```

```
            get bright alpha mask (using histogram method)

            :param back: BGR background image
            :param fore: BGR foreground image
            :return: alpha mask
            """
            # TODO, not working for every retinal scenario
            foregray = brightness(fore)
            # get window with Otsu to prevent expansion
            thresh,w = cv2.threshold(foregray,0,1,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
            return get_bright_alpha(brightness(back).astype(float),
                                    foregray.astype(float), window=w)


        pshape = (400,400) # process shape
        # rescaling of the image to process mask
        if pshape is not None:
            oshape = back.shape[:2]
            back = cv2.resize(back,pshape)
            fore = cv2.resize(fore,pshape)

        # get alpha mask
        alphamask = get_layered_alpha(back,fore)

        # rescaling mask to original shape
        if pshape is not None:
            alphamask = cv2.resize(alphamask,oshape[::-1])

        return alphamask

    def post_process_restoration(self, image):
        """
        Post-process a merged retinal image.

        :param image: retinal image
        :return: filtered and with simulated lens
        """
        image_ = super(RetinalRestore,self).post_process_restoration(image)
        if image_ is not None:
            image = image_

        # simulation of lens
        if self.lens:
            # call function to overlay lens on image
            try:
                image = simulateLens(image)
            except Exception as e:
                warnings.warn("simulate lens failed with error {}".format(e))

        # crop retinal area only
        if self.enclose:
            # convert to gray
            gray = brightness(image)
            # get object mask
            mask = foreground(gray)
            # get contour of biggest area
            cnt = thresh_biggestCnt(mask)
            # get enclosure box
```

```
            x,y,w,h = cv2.boundingRect(cnt)
            # crop image
            if len(image.shape)>2:
                image = image[x:x+w,y:y+h,:]
            else:
                image = image[x:x+w,y:y+h]

        return image

def feature_creator(string):
    """
    Converts a string to a feature object.

    :param string: any supported feature detector in openCV. the format is
            "[a-]<sift|surf|orb>[-flann]" (str) Ex: "a-sift-flann" where
            "a-" or "-flann" are optional.
    :return: feature object
    """
    return Feature().config(string)

def tuple_creator(string):
    """
    Process string to get tuple.

    :param string: string parameters with "," (colon) as separator
            Ex: param1,param2,...,paramN
    :return: tuple
    """
    tp = []
    func = string_interpreter()
    for i in string.split(","):
        try:
            tp.append(func(i))
        except:
            tp.append(i)
    return tuple(tp)

def loader_creator(string):
    """
    creates an image loader.

    :param string: flag, x size, y size. Ex 1: "0,100,100" loads gray images of shape
            (100,100) in gray scale. Ex 2: "1" loads images in BGR color and with
            original shapes. Ex 3: "0,200,None" loads gray images of shape (200,None)
            where None is calculated to keep image ratio.
    :return: loader
    """
    params = tuple_creator(string)
    try:
        flag = params[0]
    except:
        flag=1
    try:
        x = params[1]
    except:
        x=None
    try:
        y = params[2]
```

```
    except:
        y=None
    return loadFunc(flag,(x,y))

def denoise_creator(string):
    """
    creates an function to de-noise images using bilateral filter.

    :param string: d, sigmaColor, sigmaSpace. Ex: 27,75,75 creates the
            filter to de-noise images.
    :return: denoiser
    """
    d, sigmaColor, sigmaSpace = tuple_creator(string)
    def denoiser(image):
        return cv2.bilateralFilter(image, d, sigmaColor, sigmaSpace)
    return denoiser

def string_interpreter(empty=None, commahandler=None, handle=None):
    """
    create a string interpreter
    :param empty: (None) variable to handle empty strings
    :param commahandler: (tuple_creator) function to handle comma separated strings
    :return: interpreter function
    """
    def interprete_string(string):
        if string == "": # in argparse this does not applies
            return empty
        if "," in string:
            if commahandler is None:
                return tuple_creator(string)
            else:
                return commahandler(string)
        if string.lower() == "none":
            return None
        if string.lower() == "true":
            return True
        if string.lower() == "false":
            return False
        if handle is None:
            try:
                return int(string)
            except:
                return string
        else:
            return handle(string)
    interprete_string.__doc__="""
        Interpret strings.

        :param string: string to interpret.
        :return: interpreted string. If empty string (i.e. '') it returns {}.
                If 'None' returns None. If 'True' returns True. If 'False' returns False.
                If comma separated it applies {} else applies {}.
        """.format(empty, commahandler, handle)
    return interprete_string

class NameSpace(object):
    """
    used to store variables
```

```
    """

def shell(args=None, namespace=None):
    """
    Shell to run in terminal the imrestore program

    :param args: (None) list of arguments. If None it captures the
            arguments in sys.
    :param namespace: (None) namespace to place variables. If None
            it creates a namespace.
    :return: namespace
    """
    if namespace is None:
        namespace = NameSpace()
    import argparse
    parser = argparse.ArgumentParser(formatter_class=argparse.RawDescriptionHelpFormatter,
                                     description="Restore images by merging and stitching "
                                                 "techniques.",
                                     epilog=__doc__ +
                                            "\nContributions and bug reports are appreciated."
                                            "\nauthor: David Toro"
                                            "\ne-mail: davsamirtor@gmail.com"
                                            "\nproject: https://github.com/davtoh/RRtools")
    parser.add_argument('filenames', nargs='*',
                        help='List of images or path to images. Glob operations can be '
                             'achieved using the wildcard sign "*". '
                             'It can load image from files, urls, servers, strings'
                             'or directly from numpy arrays (supports databases)'
                             'Because the shell process wildcards before it gets '
                             'to the parser it creates a list of filtered files in '
                             'the path. Use quotes in shell to prevent this behaviour '
                             'an let the restorer do it instead e.g. "/path/to/images/*.jpg". '
                             'if "*" is used then folders and filenames that start with an '
                             'underscore "_" are ignored by the restorer')
    parser.add_argument('-v','--verbosity',type=int,default=1,
                        help="""(0) flag to print messages and debug data.
                                0 -> do not print messages.
                                1 -> print normal messages.
                                2 -> print normal and debug messages.
                                3 -> print all messages and show main results.
                                    (consumes significantly more memory).
                                4 -> print all messages and show all results.
                                    (consumes significantly more memory).
                                5 -> print all messages, show all results and additional data.
                                    (consumes significantly more memory).""")
    parser.add_argument('-f','--feature', type=string_interpreter(commahandler=feature_creator),
                        help='Configure detector and matcher')
    parser.add_argument('-u','--pool', action='store', type=int,
                        help='Use pool Ex: 4 to use 4 CPUs')
    parser.add_argument('-c','--cachePath',default=None,
                        help="""
                           saves memoization to specified path. This is useful to save
                           some computations and use them in next executions.
                           Cached data is not guaranteed to work between different
                           configurations and this can lead to unexpected program
                           behaviour. If a different configuration will be used it
                           is recommended to clear the cache to recompute values.
                           If True it creates the cache in current path.
```

```
                            """)
    parser.add_argument('-e','--clearCache', type=int, default=0,
                        help='clear cache flag.'
                            '* 0 do not clear.'
                            '* 1 check data integrity of previous session before use'
                            '* 2 re-compute data but other cache data is left intact.'
                            '* 3 All CachePath is cleared before use.'
                            'Notes: using cache can result in unexpected behaviour '
                            'if some configurations does not match to the cached data.')
    parser.add_argument('--loader', type=string_interpreter(commahandler=loader_creator),
                        nargs='?', help='Custom loader function used to load images. '
                            'By default or if --loader flag is empty it loads the '
                            'original images in color. The format is "--loader colorflag, '
                            'x, y" where colorflag is -1,0,1 for BGRA, gray and BGR images '
                            'and the load shape are represented by x and y. '
                            'Ex 1: "0,100,100" loads gray images of shape (100,100) in '
                            'gray scale. Ex 2: "1" loads images in BGR color and with '
                            'original shapes. Ex 3: "0,200,None" loads gray images of shape '
                            '(200,None) where None is calculated to keep image ratio.')
    parser.add_argument('-p','--process_shape', default=(400,400), type=string_interpreter(),
                        nargs='?', help='Process shape used to convert to pseudo images '
                            'to process features and then convert to the '
                            'original images. The smaller the image more memory and speed '
                            'gain. By default process_shape is 400,400'
                            'If the -p flag is empty it loads the original '
                            'images to process the features but it can incur to performance'
                            ' penalties if images are too big and RAM memory is scarce')
    parser.add_argument('-l','--load_shape', default=None, type=string_interpreter(),
                        nargs='?', help='shape used to load images which are beeing merged.')
    parser.add_argument('-b','--baseImage', default=True, type=string_interpreter(), nargs='?',
                        help='Specify image''s name to use from path as first image to merge '
                            'in the empty restored image. By default it selects the image '
                            'with most features. If the -b flag is empty it selects the '
                            'first image in filenames as base image')
    parser.add_argument('-m','--selectMethod',
                        help='Method to sort images when matching. This '
                            'way the merging order can be controlled.'
                            '* (None) Best matches'
                            '* Histogram Comparison: Correlation, Chi-squared,'
                            'Intersection, Hellinger or any method found in hist_map'
                            '* Entropy'
                            '* custom function of the form: rating,fn <-- selectMethod(fns)')
    parser.add_argument('-d','--distanceThresh', type = float, default=0.75,
                        help='Filter matches by distance ratio')
    parser.add_argument('-i','--inlineThresh', type = float, default=0.2,
                        help='Filter homography by inlineratio')
    parser.add_argument('-r','--rectangularityThresh', type = float, default=0.5,
                        help='Filter homography by rectangularity')
    parser.add_argument('-j','--ransacReprojThreshold', type = float, default=10.0,
                        help='Maximum allowed reprojection error '
                            'to treat a point pair as an inlier')
    parser.add_argument('-n','--centric', action='store_true',
                        help='Tries to attach as many images as possible to '
                            'each matching. It is quicker since it does not have to process '
                            'too many match computations')
    parser.add_argument('-t','--hist_match', action='store_true',
                        help='Apply histogram matching to foreground '
                            'image with merge image as template')
```

```
    parser.add_argument('-s','--save', default=True, nargs='?', action="store",
                        const=False, type = string_interpreter(False),
                        help='Customize image name used to save the restored image.'
                            'By default it saves in path with name "_restored_{base_image}".'
                            'if the -s flag is specified empty it does not save. Formatting '
                            'is supported so for example the default name can be achived as '
                            '"-s {path}_restored_{name}{ext}"')
    parser.add_argument('-o','--overwrite', action='store_true',
                        help = 'If True and the destine filename for saving already '
                            'exists then it is replaced, else a new filename is generated '
                            'with an index "{filename}_{index}.{extension}"')
    parser.add_argument('-g','--grow_scene', action='store_true',
                        help='Flag to allow image to grow the scene so that that the final '
                            'image can be larger than the base image')
    parser.add_argument('-y','--denoise', nargs='?', action="store", const=True,
                        type=string_interpreter(True,commahandler=denoise_creator),
                        help="Flag to process noisy images. Use mild, normal, heavy or "
                            "provide parameters for a bilateral filter as "
                            "'--denoise d,sigmaColor,sigmaSpace' as for example "
                            "'--denoise 27,75,75'. By default it is None which can be "
                            "activated according to the restorer, if an empty flag is "
                            "provided as '--denoise' it deactivates de-noising images.")
    parser.add_argument('-a','--lens', action='store_true',
                        help='Flag to apply lens to retinal area. Else do not apply lens')
    parser.add_argument('-k','--enclose', action='store_true',
                        help='Flag to enclose and return only retinal area. '
                            'Else leaves image "as is"')
    parser.add_argument('-z','--restorer',choices = ['RetinalRestore','ImRestore'],
                        default='RetinalRestore',
                        help='imrestore is for images in general but it can be parametrized. '
                            'By default it has the profile "retinalRestore" for retinal '
                            'images but its general behaviour can be restorerd by '
                            'changing it to "imrestore"')
    parser.add_argument('-x','--expert', default=None,help='path to the expert variables')
    parser.add_argument('-q','--console', action='store_true',
                        help='Enter interactive mode to let user execute commands in console')
    parser.add_argument('-w','--debug', action='store_true', # https://pymotw.com/2/pdb/
                        help='Enter debug mode to let programmers find bugs. In the debugger '
                             'type "h" for help and know the supported commands.')
    parser.add_argument('--onlykeys', action='store_true',
                        help='Only compute keypoints. This is useful when --cachePath is '
                            'used and the user wants to have the keypoints cached beforehand')

    # parse sys and get argument variables
    args = vars(parser.parse_args(args=args, namespace=namespace))

    # shell variables
    debug = args.pop('debug')
    console = args.pop('console')
    onlykeys =  args.pop('onlykeys')

    # debugger
    if debug:
        print "debug ON."
        import pdb; pdb.set_trace()

    # this is needed because the shell process wildcards before it gets to argparse
    # creating a list in the path thus it must be filtered. Use quotes in shell
```

```
    # to prevent this behaviour
    if len(args['filenames'])>1:
        args['filenames'] = [p for p in args['filenames'] if check_valid(p) or "*" in p]
    else:
        args['filenames'] = args['filenames'][0]

    # print parsed arguments
    if args['verbosity']>1:
        print "Parsed Arguments\n",args

    # use configuration
    use_restorer = args.pop("restorer")
    if use_restorer == 'RetinalRestore':
        if args['verbosity']: print "Configured for retinal restoration..."
        self = RetinalRestore(**args)
    elif use_restorer == 'ImRestore':
        if args['verbosity']: print "Configured for general restoration..."
        for key in ['enclose', 'lens']:
            args.pop(key) # clean up unused key
        self = ImRestore(**args)
    else:
        raise Exception("no restoration class called {}".format(use_restorer))

    if namespace is not None:
        # update namespace from early stages so it can have access to the restorer
        namespace.restorer = self

    if console:
        print "interactive ON."
        print "restoring instance is 'namespace.restorer' or 'self'"
        print "Ex: type 'self.restore()' to proceed with restoration."
        import code; code.interact(local=locals())
    elif onlykeys:
        self.compute_keypoints()
    else:
        # start restoration
        self.restore()

    return namespace # return given namespace

if __name__ == "__main__":
    shell() # run the shell
    # TODO visualizator for alpha mask
    # TODO implement standard deviation in bright areas to detect optic disk
```

After the script is completed the creation of the executable can be done using a shell interface or a graphical interface can be added on top of it. This is further explained in User Interface section .

## User Interface

This section gives an introduction to the *imrestore* interfaces which can be resumed in its classes, command-line interface using a shell and the developing GUI more detailed discussed in Future Work.

### Command-line interface

Usually the following command format is used to run the *imrestore* script:

```
python imrestore.py <path to folder with images>
```

`python imrestore.py` calls Python interpreter to run *imrestore.py* script but if the file is a bash or executable `./<path to executable>` is used. For example, to test *imrestore* once Pyinstaller finishes creating the executable as explained in subsection *Executable* just enter the command:

```
./dist/imrestore tests/im1*
```

The `./` is used to signify that the operation is from respect the current path. So if the terminal's path is in *dist* folder use:

```
./imrestore tests/im1*
```

For further reading the capabilities of *imrestore* and configurations just type in:

```
./imrestore -h
```

or its equivalent

```
./imrestore --help
```

### Executable

There are several options to create executables and installers from python sources ("deployment - Python Wiki," n.d.). From them Pyinstaller suffices for all the needs to create them. It is a python package so it must be installed after python, these procedures are found in subsubsection A.1.3 and further reading in (Cortesi, Bajo, Caban, & McMillan, n.d.). To create one-file executable from a script the following command format is used:

```
pyinstaller -p <path to sources> -n <name of program> -F <path to script> --version-file=<path to version
↪  file>
```

The `-p` option is to add a path to search for imports, without this the executable would not have all the features. `-n` Name to assign to the bundled *app* and *spec* file. `-F` Create a one-file bundled executable.

Lets create *imrestore* executable then, so open the Terminal where *RRtoolbox* folder and *imrestore.py* are (i.e. like in the repository) then type the following command to create the executable:

```
pyinstaller -p ./ -n imrestore -F ./imrestore.py/ --version-file=version
```

This will create the folders *build* where temporal files are compiled to be used in the creation of the executable and *dist* where the executable is placed once pyinstaller finishes building it. Under windows 7 the msvcr100.dll is needed so it is advised to install Microsoft Visual C++ 2010 before running pyinstaller to let it find the necessary files. Notice that the executable is only compatible for computers with the same operating system as the one used to create it.

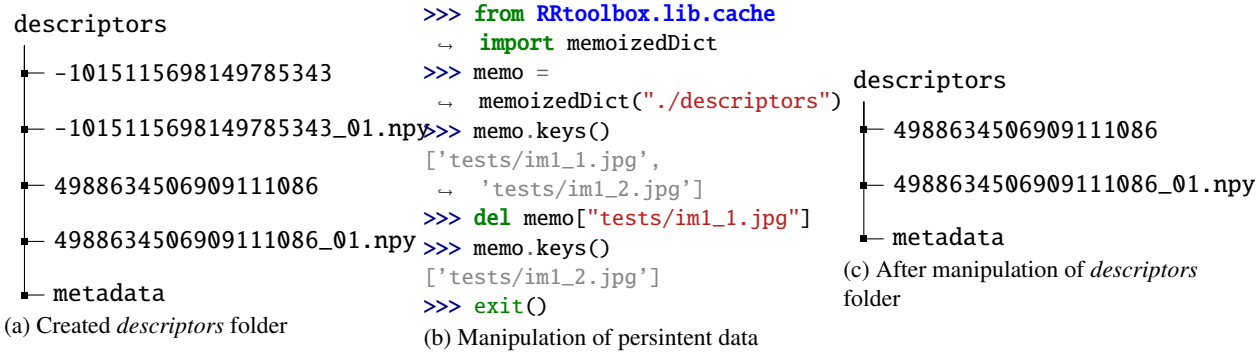
## Results

The majority of results were taken using *imrestore.py* script with generic options (see Code 5) where samples represented worse-case scenarios that could be obtained in a non-professional way, ensuring that a professional ophthalmologist can obtain better results than the ones here presented.

To show an example of the output that *imrestore* generates lets run the program like a normal user would do if he downloaded the repository from (David, 2016b) to use it just by placing in the console the command '`python imrestore.py tests/im1* --lens --overwrite --cachePath .`'.

Notice that the option `--cachedPath .` is provided to create the cache in the current path of the terminal which generates a folder called *descriptors* with structure shown in Figure 36a. In the first try it executes everything normally but caching data for subsequent executions (see the output messages generated by imrestore inTable 3). In the second execution it finds that *descriptors* folder is already created so it looks for cached data when needed (see Table 4) to not process it and if data is missing in the cache it simply recalculates it. For the sake of demonstrating it lets manipulate the persisted data using *memoizedDict* provided by *RRtoolbox* package which is the developed cache manager used by *imrestore*. In a python environment the user only has to import the *memoizedDict* class and instantiate it to delete the desired key(s) as it is shown in Figure 36b. Figure 36c shows how the file structure in the *descriptors* folder has changed after its manipulation. If imrestore is run once again it encounters that there are no features for 'tests/im1_1.jpg' so it calculates them and because it had cached 'tests/im1_2.jpg' this data is not re-evaluated saving time and computer power (see the output messages in Table 5 to confirm this behaviour). Notice that, because customization is a common procedure when restoring images of a same set, *imrestore* provides the `--onlykeys` option to process the key-points of images without restoring them which only makes sense in combination with cache options and if the user wants to keep using images of the same set many times.

*Figure 36*. Persistence to *descriptors* folder and its manipulation




(a) Created *descriptors* folder

(b) Manipulation of persintent data

(c) After manipulation of *descriptors* folder

Now that we know the *imrestore* basic use we proceed to run the generic command for 27 sets that contain retinal images with severe noise cases hoping that *imrestore* can give us the best automated results. Of course each set presents different challenges containing images with different perspectives, illumination, flares, unfocused blurry areas and others from where *imrestore* has to automatically choose the best images of the set with the first as the *base image* for merging and stitching procedures. A pattern is not fed to *imrestore*, in fact the idea is to test it with any possible scenario and let it take automated decisions. After this, we analyse all the results and re-run the program with fitting options to customise how *imrestore* process some sets that could offer better results.

The general command to process the results is in Code 5. The `--lens` flag is used to simulate the retinal camera lens at the end of the restoration process, this with the intention of eliminating the blurry areas around the retinal area which is a common issue when images are obtained with mobile devices. The `--overwrite` option replaces images with the same name when saving the restoration result and `--cachePath {temp}` option is used to tell *imrestore* to cache data in the *RRtoolbox's temp* folder allowing the use of Code 5 in any path while saving the cache in the same place always. Naturally a custom path could also be provided as `--cachePath <my custom path>`.

Only the images that where used are shown in the result figures, so that the first image is the *base image* (e.g. Figure 38a and Figure 39a) used for its restoration and the subsequent images except the last the ones used to improve it (e.g. Table 6 does not have more images and Table 7 has Figure 39b). The last image is the result for the set using the *base image* and it is what *imrestore* saves as output (e.g. Figure 38b and Figure 39c). In addition to it, a profiling which is like a report is presented for each set to let the user see the times of each process in the program. Notice that this differs to the output messages presented in Tables 3, 4 and 5 in that it is generated by the internal *imrestore* profiler that can be imported using `from RRtoolbox.lib.root import Profiler`.

Code 5: General command to process the retinal sets

```
python imrestore.py <path to folder with images> --lens --overwrite -c <cache file>
```

Table 3
*Output messages for first execution of* `python imrestore.py tests/im1* --lens --overwrite --cachePath . command`

```
Configured for retinal restoration...
Cache path is in ./descriptors
No. images 2...
Computing features...

Features 1/2... Processing features for
 ↪  tests/im1_1.jpg...
affine sampling: 1 / 43
affine sampling: 2 / 43
affine sampling: 3 / 43
affine sampling: 4 / 43
affine sampling: 5 / 43
affine sampling: 6 / 43
affine sampling: 7 / 43
affine sampling: 8 / 43
affine sampling: 9 / 43
affine sampling: 10 / 43
affine sampling: 11 / 43
affine sampling: 12 / 43
affine sampling: 13 / 43
affine sampling: 14 / 43
affine sampling: 15 / 43
affine sampling: 16 / 43
affine sampling: 17 / 43
affine sampling: 18 / 43
affine sampling: 19 / 43
affine sampling: 20 / 43
affine sampling: 21 / 43
affine sampling: 22 / 43
affine sampling: 23 / 43
affine sampling: 24 / 43
affine sampling: 25 / 43
affine sampling: 26 / 43
affine sampling: 27 / 43
affine sampling: 28 / 43
affine sampling: 29 / 43
affine sampling: 30 / 43
affine sampling: 31 / 43
affine sampling: 32 / 43
affine sampling: 33 / 43
affine sampling: 34 / 43
affine sampling: 35 / 43
affine sampling: 36 / 43
affine sampling: 37 / 43
affine sampling: 38 / 43
affine sampling: 39 / 43
affine sampling: 40 / 43
affine sampling: 41 / 43
affine sampling: 42 / 43
affine sampling: 43 / 43

Features 2/2... Processing features for
 ↪  tests/im1_2.jpg...
affine sampling: 1 / 43
affine sampling: 2 / 43
affine sampling: 3 / 43
affine sampling: 4 / 43
affine sampling: 5 / 43
affine sampling: 6 / 43
affine sampling: 7 / 43
affine sampling: 8 / 43
affine sampling: 9 / 43
affine sampling: 10 / 43
affine sampling: 11 / 43
affine sampling: 12 / 43
affine sampling: 13 / 43
affine sampling: 14 / 43
affine sampling: 15 / 43
affine sampling: 16 / 43
affine sampling: 17 / 43
affine sampling: 18 / 43
affine sampling: 19 / 43
affine sampling: 20 / 43
affine sampling: 21 / 43
affine sampling: 22 / 43
affine sampling: 23 / 43
affine sampling: 24 / 43
affine sampling: 25 / 43
affine sampling: 26 / 43
affine sampling: 27 / 43
affine sampling: 28 / 43
affine sampling: 29 / 43
affine sampling: 30 / 43
affine sampling: 31 / 43
affine sampling: 32 / 43
affine sampling: 33 / 43
affine sampling: 34 / 43
affine sampling: 35 / 43
affine sampling: 36 / 43
affine sampling: 37 / 43
affine sampling: 38 / 43
affine sampling: 39 / 43
affine sampling: 40 / 43
affine sampling: 41 / 43
affine sampling: 42 / 43
affine sampling: 43 / 43
Computed feature time was 2.561066  seconds
baseImage is tests/im1_1.jpg
Configured to sort by best matches
Restoring ...
Matching ...
Matching overall time was 0.016191  seconds
Merging ...
This image has been merged: tests/im1_2.jpg...
Merging overall time was 0.856332  seconds
Matching ...
All images have been merged...
Matching overall time was 0.000033  seconds
Restoring overall time was 0.872662  seconds
Post-processing ...
Post-processing overall time was 0.234878  seconds
Saved: tests/_restored_im1_1.jpg
```

Table 4
*Output messages for second execution of* `python imrestore.py tests/im1* --lens --overwrite --cachePath .` *command*

```
Configured for retinal restoration...
Cache path is in ./descriptors
No. images 2...
Computing features...

Features 1/2... tests/im1_1.jpg is cached...

Features 2/2... tests/im1_2.jpg is cached...
Computed feature time was 0.190396  seconds
baseImage is tests/im1_1.jpg
Configured to sort by best matches
Restoring ...
Matching ...
Matching overall time was 0.015764  seconds
Merging ...
This image has been merged: tests/im1_2.jpg...
Merging overall time was 0.829140  seconds
Matching ...
All images have been merged...
Matching overall time was 0.000030  seconds
Restoring overall time was 0.845043  seconds
Post-processing ...
Post-processing overall time was 0.226974  seconds
Saved: tests/_restored_im1_1.jpg
```

Table 5
*Output messages for execution of* `python imrestore.py tests/im1* --lens --overwrite --cachePath .` *command with modified cache*

```
Configured for retinal restoration...
Cache path is in ./descriptors
No. images 2...
Computing features...

Features 1/2... Processing features for
 ↪  tests/im1_1.jpg...
affine sampling: 1 / 43
affine sampling: 2 / 43
affine sampling: 3 / 43
affine sampling: 4 / 43
affine sampling: 5 / 43
affine sampling: 6 / 43
affine sampling: 7 / 43
affine sampling: 8 / 43
affine sampling: 9 / 43
affine sampling: 10 / 43
affine sampling: 11 / 43
affine sampling: 12 / 43
affine sampling: 13 / 43
affine sampling: 14 / 43
affine sampling: 15 / 43
affine sampling: 16 / 43
affine sampling: 17 / 43
affine sampling: 18 / 43
affine sampling: 19 / 43
affine sampling: 20 / 43
affine sampling: 21 / 43
affine sampling: 22 / 43
affine sampling: 23 / 43
affine sampling: 24 / 43
affine sampling: 25 / 43
affine sampling: 26 / 43
affine sampling: 27 / 43
affine sampling: 28 / 43
affine sampling: 29 / 43
affine sampling: 30 / 43
affine sampling: 31 / 43
affine sampling: 32 / 43
affine sampling: 33 / 43
affine sampling: 34 / 43
affine sampling: 35 / 43
affine sampling: 36 / 43
affine sampling: 37 / 43
affine sampling: 38 / 43
affine sampling: 39 / 43
affine sampling: 40 / 43
affine sampling: 41 / 43
affine sampling: 42 / 43
affine sampling: 43 / 43

Features 2/2... tests/im1_2.jpg is cached...
Computed feature time was 1.246475  seconds
baseImage is tests/im1_1.jpg
Configured to sort by best matches
Restoring ...
Matching ...
Matching overall time was 0.016467  seconds
Merging ...
This image has been merged: tests/im1_2.jpg...
Merging overall time was 0.856636  seconds
Matching ...
All images have been merged...
Matching overall time was 0.000033  seconds
Restoring overall time was 0.873242  seconds
Post-processing ...
Post-processing overall time was 0.232622  seconds
Saved: tests/_restored_im1_1.jpg
```

**Result for set 1**

*Figure 37*. Images in set1

(a) IMG_0173 (b) IMG_0176 (c) IMG_0179 (d) IMG_0180 (e) IMG_0182 (f) IMG_0184 (g) IMG_0185

Table 6
*Profiling for set1 using command* '`imrestore ../results/set1/*.* --lens --overwrite --cachePath {temp}`'

| | |
|---|---|
| 'ImRestore init' -> 11.4868 secs | 'ImRestore init' -> 4.2917 secs |
| 'Computing features' -> 11.0730 secs | 'Computing features' -> 2.7749 secs |
| (processed) 'IMG_0173' -> 1.7930 secs | (memoized) 'IMG_0173' -> 0.6027 secs |
| (processed) 'IMG_0176' -> 1.5659 secs | (memoized) 'IMG_0176' -> 0.5151 secs |
| (processed) 'IMG_0179' -> 1.5224 secs | (memoized) 'IMG_0179' -> 0.3121 secs |
| (processed) 'IMG_0180' -> 1.5101 secs | (memoized) 'IMG_0180' -> 0.2485 secs |
| (processed) 'IMG_0182' -> 1.5386 secs | (memoized) 'IMG_0182' -> 0.2894 secs |
| (processed) 'IMG_0184' -> 1.5334 secs | (memoized) 'IMG_0184' -> 0.2105 secs |
| (processed) 'IMG_0185' -> 1.5580 secs | (memoized) 'IMG_0185' -> 0.4565 secs |
| 'Restoring' -> 0.1188 secs | 'Restoring' -> 0.3652 secs |
| 'Matching' -> 0.0535 secs | 'Matching' -> 0.1424 secs |
| 'Merging' -> 0.0650 secs | 'Merging' -> 0.2225 secs |
| 'Post-processing' -> 0.2951 secs | 'Post-processing' -> 1.1516 secs |

*Figure 38*. Result for set1 using command '`imrestore ../results/set1/*.* --lens --overwrite --cachePath {temp}`'

(a) IMG_0176 (b) Restored image

This set have 7 images as shown in Figure 37 and all of them are processed when Code 5 is used to find their features but only a few are selected by the algorithm in the restoration. This is evidenced in the entries 'Computing features' and 'Restoring' in Table 6 which shows the profiling of the execution. It demonstrates the advantage of using the cache option which reduces the un-cached time from 11.49 to 4.29 seconds when cached, this means that 7.20 seconds seconds were spared with almost 3 times the gain.

Figure 38 shows the result for set 1. In it *imrestore* selects Figure 38a as the *base image* assuming that it is the clearest image with best features. Nonetheless this image has a flare that must be corrected but it does not find a good image from the set to achieve this so no additional images are used in the merging. Because of this Figure 38a is left unchanged and only post-processed to generate the restored image shown in Figure 38b.

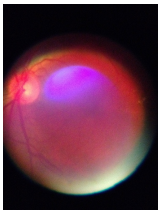
Table 7
*Profiling for set2 using command ‘*`imrestore ../results/set2/*.* --lens --overwrite --cachePath {temp}`*’*

```
'ImRestore init' -> 18.0600 secs
└─ 'Computing features' -> 16.2941 secs
   ├─ (processed) 'IMG_0405' -> 2.9245 secs
   ├─ (processed) 'IMG_0406' -> 2.8483 secs
   ├─ (processed) 'IMG_0407' -> 2.6455 secs
   ├─ (processed) 'IMG_0408' -> 2.5008 secs
   ├─ (processed) 'IMG_0409' -> 2.6195 secs
   └─ (processed) 'IMG_0410' -> 2.6894 secs
└─ 'Restoring' -> 1.3122 secs
   ├─ 'Matching' -> 0.0396 secs
   └─ 'Merging' -> 1.2721 secs
      └─ 'IMG_0408' -> 1.2253 secs
└─ 'Post-processing' -> 0.4537 secs
```

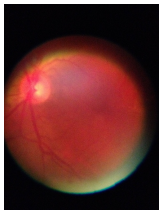
```
'ImRestore init' -> 7.4845 secs
└─ 'Computing features' -> 2.4074 secs
   ├─ (memoized) 'IMG_0405' -> 0.3392 secs
   ├─ (memoized) 'IMG_0406' -> 0.1877 secs
   ├─ (memoized) 'IMG_0407' -> 0.1871 secs
   ├─ (memoized) 'IMG_0408' -> 0.4202 secs
   ├─ (memoized) 'IMG_0409' -> 0.4340 secs
   └─ (memoized) 'IMG_0410' -> 0.2727 secs
└─ 'Restoring' -> 4.3245 secs
   ├─ 'Matching' -> 0.0950 secs
   └─ 'Merging' -> 4.2291 secs
      └─ 'IMG_0408' -> 4.1309 secs
└─ 'Post-processing' -> 0.7526 secs
```

**Result for set 2**

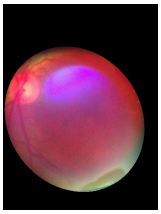
*Figure 39*. Result for set2 using command ‘`imrestore ../results/set2/*.* --lens --overwrite --cachePath {temp}`’



(a) IMG_0409



(b) IMG_0408



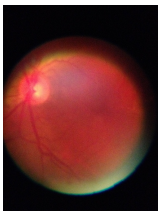
(c) Restored image

*Figure 40*. Images in set2

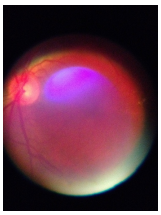
(a) IMG_0405

(b) IMG_0406

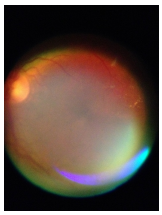
(c) IMG_0407



(d) IMG_0408



(e) IMG_0409



(f) IMG_0410

As in the Result for set 1 the generic Code 5 is used for set 2 with images in Figure 40 and presenting the profiling results in Table 7 for 6 processed images. It continues to show the advantage of using the cache option reducing the time from 18.06 to 7.48 seconds which is the 41.44% of the total with 10.58 seconds seconds of gained time.

Images used in the restoration are shown in Figure 39 and the ones not used in it are in **??**. This time it does finds the merging image shown in Figure 39b to try to restore the *base image* shown in Figure 39a resulting in the restored image in Figure 39c. As it can be seen, to our judgement, the *base image* is not that appealing, in fact the *base image* could have been better the merged image in Figure 39b or the one which was not used in Figure 40c. This is due that in the form that *imrestore* is currently configured it assumes that the image with most key-points is the most likely to restore from.

Table 8
*Profiling for set2 using command 'imrestore ../results/set2/*.* -b IMG_0407.* -y --lens --overwrite --cachePath {temp}'*

'ImRestore init' -> 40.1164 secs
- (processed) 'IMG_0407' -> 7.6018 secs
- 'Restoring' -> 30.0773 secs
  - 'Matching' -> 29.9244 secs
    - 'Computing features' -> 29.8293 secs
      - (processed) 'IMG_0405' -> 6.4433 secs
      - (processed) 'IMG_0406' -> 4.2656 secs
      - (cached) 'IMG_0407' -> 0.0004 secs
      - (processed) 'IMG_0408' -> 4.1587 secs
      - (processed) 'IMG_0409' -> 7.5317 secs
      - (processed) 'IMG_0410' -> 7.2634 secs
  - 'Merging' -> 0.1523 secs
- 'Post-processing' -> 2.4374 secs

'ImRestore init' -> 2.4718 secs
- 'Restoring' -> 1.1325 secs
  - 'Matching' -> 0.8040 secs
    - 'Computing features' -> 0.7667 secs
      - (memoized) 'IMG_0405' -> 0.1199 secs
      - (memoized) 'IMG_0406' -> 0.0449 secs
      - (memoized) 'IMG_0407' -> 0.0984 secs
      - (memoized) 'IMG_0408' -> 0.0936 secs
      - (memoized) 'IMG_0409' -> 0.1362 secs
      - (memoized) 'IMG_0410' -> 0.1200 secs
  - 'Merging' -> 0.3282 secs
- 'Post-processing' -> 1.3393 secs

*Figure 41.* Customized result for set2 using command 'imrestore ../results/set2/*.* -b IMG_0407.* -y --lens --overwrite --cachePath {temp}'

(a) IMG_0407

(b) Restored image

Lets try to enhance the result showed by Figure 39 created when using the generic command in Code 5. To do this I will introduce two more options, the `--denoise` and `--baseImage` flags. The flag `--denoise` is pretty obvious, it uses an automated algorithm to select the best parameters to apply filters to the image so that it can be smoother to the user. `--baseImage` also explains itself as it is used to select a custom *base image* or configure its automated selection. Now, lets select the new *base image* as IMG_0407 using the pattern 'IMG_0407.*' where the point and wildcat '.*' is used to specify that it can find the image in the set with any extension. Notice that without '.*' in the pattern the image cannot be found because there is no image in the set without extension leading to an error and also if there are more than one image with the same name but with different extensions the program would find more than one image with this pattern throwing an error indicating this. The customised command is then 'imrestore ../results/set2/*.* -b IMG_0407.* -y --lens --overwrite --cachePath {temp}' which uses abbreviations of `--baseImage` and `--denoise` as `--b` and `--y` respectively. Figure 41 shows the new result, with this *imrestore* produced one of the best results that could be obtained from set2 and though it did not find good matches for Figure 41a it applied the post processing methods to generate the restored image in Figure 41b. In a way the algorithm did not find more matches to merge in the new *base image* due that it was sufficient and had no problems whatsoever for the other images in the set to enhance, these images were not that good (Figure 40).

Table 8 shows the profiling for both the processed and cached case of the custom command 'imrestore ../results/set2/*.* -b IMG_0407.* -y --lens --overwrite --cachePath {temp}'. There shouldn't be nothing abnormal about it, as always the cached profiling register less time than the un-cached one reducing the time from 40.12 to 2.47 seconds, only the 6.16%. Wait! there is something different about this profiling than the presented in Table 7, the profiling for the session without `--cachePath` (left) processed IMG_0407 features ahead of time, what could have happened? well, it is due to the capacity of the program to perform Lazy evaluations and because the user asked the program to use IMG_0407 as the *base image* it had to process its features before the restoration process began. More intriguing is that this same image was requested when all the features were computed in the 'Computing features' entry (Table 8, left) but because it was all-ready processed the program returned the cached data. Though the option for caching was not specified the program by default is configured to use cache internally but not between sessions (recognized by the 'memoized' tag in the features when cached, Table 8 right), this behaviour can be changed using the `--clearCache` flag which offers more caching options. Why the cached session in the right of Table 8 did not register the same as its processed counter part? it was due that when the cache option is enabled (e.g. `--cachePath` is used) the program shares the information of previous sessions and no further processing of the quested data is needed, this can be dangerous

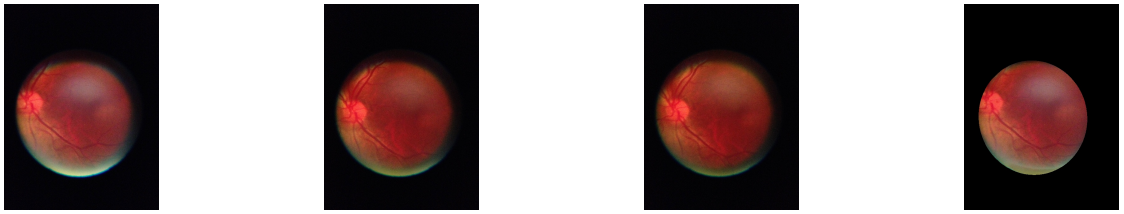
as previous data could change the normal results of the current session, this can be prevented using `--clearCache 1` option to check data integrity each time it is requested. These caching functionalities are provided by the `LazyDict` and `MemoizedDict` classes found in RRtoolbox/lib/cache.py and can be imported in Python with `from RRtoolbox.lib.cache import LazyDict, MemoizedDict`.

**Result for set 3**

Table 9
*Profiling for set3 using command 'imrestore ../results/set3/*.* --lens --overwrite --cachePath {temp}'*

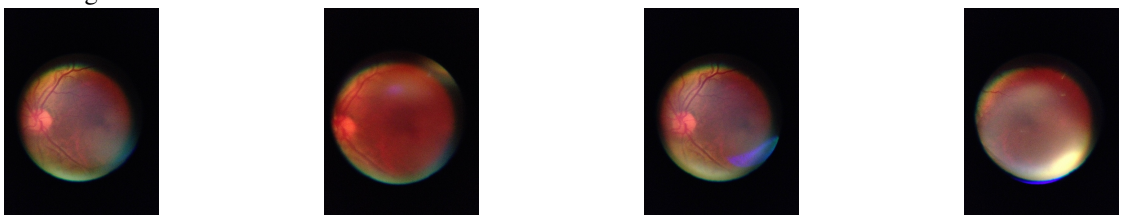
| | |
|---|---|
| 'ImRestore init' -> 25.5017 secs | 'ImRestore init' -> 9.4361 secs |
| 'Computing features' -> 21.8672 secs | 'Computing features' -> 2.1685 secs |
| (processed) 'IMG_0428' -> 3.5337 secs | (memoized) 'IMG_0428' -> 0.1825 secs |
| (processed) 'IMG_0429' -> 3.2418 secs | (memoized) 'IMG_0429' -> 0.3714 secs |
| (processed) 'IMG_0430' -> 2.9368 secs | (memoized) 'IMG_0430' -> 0.3066 secs |
| (processed) 'IMG_0431' -> 2.9219 secs | (memoized) 'IMG_0431' -> 0.3074 secs |
| (processed) 'IMG_0432' -> 2.9320 secs | (memoized) 'IMG_0432' -> 0.3854 secs |
| (processed) 'IMG_0433' -> 2.8682 secs | (memoized) 'IMG_0433' -> 0.1596 secs |
| (processed) 'IMG_0434' -> 3.0255 secs | (memoized) 'IMG_0434' -> 0.1585 secs |
| 'Restoring' -> 3.1824 secs | 'Restoring' -> 6.5249 secs |
| 'Matching' -> 0.1331 secs | 'Matching' -> 0.2359 secs |
| 'Merging' -> 3.0486 secs | 'Merging' -> 6.2884 secs |
| 'IMG_0429' -> 1.4398 secs | 'IMG_0429' -> 2.8025 secs |
| 'IMG_0430' -> 1.5079 secs | 'IMG_0430' -> 3.2572 secs |
| 'Post-processing' -> 0.4521 secs | 'Post-processing' -> 0.7427 secs |

*Figure 42*. Result for set3 using command 'imrestore ../results/set3/*.* --lens --overwrite --cachePath {temp}'



(a) IMG_0431 (b) IMG_0429 (c) IMG_0430 (d) Restored image

*Figure 43*. Images not used in restoration of set3



(a) IMG_0434 (b) IMG_0428 (c) IMG_0433 (d) IMG_0432

In Table 9 there are 7 computed images for their features and 2 in the 'Merging' entry. This is confirmed in Figure 42 with the *base image* in Figure 42a, the restored in Figure 42d and the remaining merged images. In the restored image we can see that the bright part at the bottom in Figure 42a was replaced by the one in Figure 42b which is more convenient. The lens also did a good job in segmenting correctly the retinal area despite the blurry-like aura enclosing the restoring image. The 3 used images were very similar and though the selected *base*

*image* was not the best of them this restoration can be considered good as the other images were not suitable presenting blurriness and violet lights with more whitish borders in the retinal area.

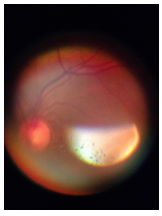
The explanation for the profiling tables are becoming superfluous as it seems that the cache will always reduce the processing time of an *imrestore* session compared to one that is not cached. In Table 9 is evidenced that it proudly reduced the time in a 63.00% from 25.50 to 9.44 seconds. The rate of used images also remains low in 0.43 with 3 used of 7 in the set (all images in Figure 42 except the restored in Figure 42d plus the 4 in Figure 43).

### Result for set 4

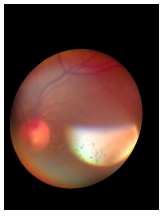
Table 10
*Profiling for set4 using command* '`imrestore ../results/set4/*.* --lens --overwrite --cachePath {temp}`'

| 'ImRestore init' -> 37.1245 secs | 'ImRestore init' -> 5.2054 secs |
|---|---|
| 'Computing features' -> 36.5185 secs | 'Computing features' -> 3.7392 secs |
| (processed) 'IMG_0411' -> 3.2775 secs | (memoized) 'IMG_0411' -> 0.3590 secs |
| (processed) 'IMG_0412' -> 3.1809 secs | (memoized) 'IMG_0412' -> 0.3026 secs |
| (processed) 'IMG_0413' -> 2.9239 secs | (memoized) 'IMG_0413' -> 0.1917 secs |
| (processed) 'IMG_0418' -> 2.9996 secs | (memoized) 'IMG_0418' -> 0.3780 secs |
| (processed) 'IMG_0419' -> 2.8301 secs | (memoized) 'IMG_0419' -> 0.1714 secs |
| (processed) 'IMG_0420' -> 2.8677 secs | (memoized) 'IMG_0420' -> 0.0978 secs |
| (processed) 'IMG_0421' -> 2.9634 secs | (memoized) 'IMG_0421' -> 0.0921 secs |
| (processed) 'IMG_0423' -> 2.8928 secs | (memoized) 'IMG_0423' -> 0.2538 secs |
| (processed) 'IMG_0424' -> 2.8513 secs | (memoized) 'IMG_0424' -> 0.3392 secs |
| (processed) 'IMG_0425' -> 2.9598 secs | (memoized) 'IMG_0425' -> 0.2417 secs |
| (processed) 'IMG_0426' -> 2.9678 secs | (memoized) 'IMG_0426' -> 0.8015 secs |
| (processed) 'IMG_0427' -> 3.0886 secs | (memoized) 'IMG_0427' -> 0.1747 secs |
| 'Restoring' -> 0.1900 secs | 'Restoring' -> 0.4057 secs |
| 'Matching' -> 0.0540 secs | 'Matching' -> 0.1222 secs |
| 'Merging' -> 0.1356 secs | 'Merging' -> 0.2830 secs |
| 'Post-processing' -> 0.4161 secs | 'Post-processing' -> 1.0605 secs |

*Figure 44*. Result for set4 using command '`imrestore ../results/set4/*.* --lens --overwrite --cachePath {temp}`'



(a) IMG_0411

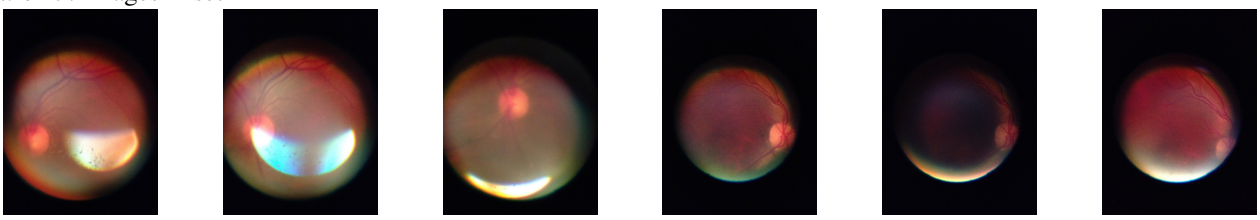


(b) Restored image

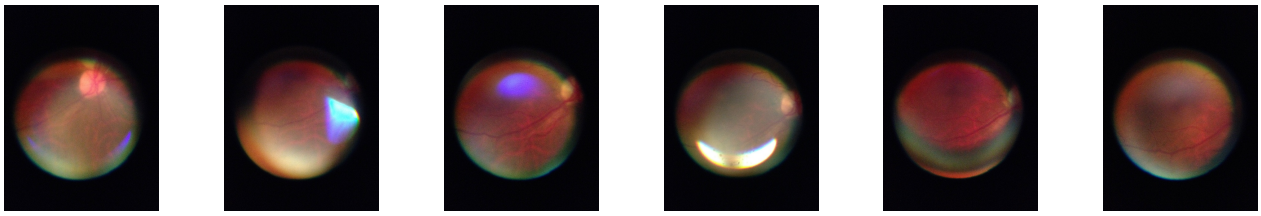
This set presents similar results as the previous sets with 12 computed images and all of them with different affected areas as shown in Figure 44. Despite all those images only the image in <function <lambda> at 0x7f086072b320> got to be used without being merged with the others and only applying the post-processing step with lens simulation as shown in Figure 44b. There really is not a preferable image as all of them are not good and no image have desired parts that could be used to restore the *base image* but at least the enhancing of it demonstrated the capabilities of *imrestore* in situations were there is nothing to do.

Table 10 shows again a reduction in time of 31.92 seconds from 37.12 to 5.21 when using cache. Despite that the rate of used images is low with 0.08 as 0 images where merged leaving 1 used and 11 not used in the set of 12 images (Figure 45).

*Figure 45*. Images in set4



(a) IMG_0411 (b) IMG_0412 (c) IMG_0413 (d) IMG_0418 (e) IMG_0419 (f) IMG_0420



(g) IMG_0421 (h) IMG_0423 (i) IMG_0424 (j) IMG_0425 (k) IMG_0426 (l) IMG_0427

Table 11
*Profiling for set5 using command '*`imrestore ../results/set5/*.* --lens --overwrite --cachePath {temp}`*'*

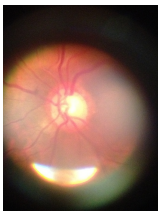
| | |
|---|---|
| 'ImRestore init' -> 23.8650 secs | 'ImRestore init' -> 5.9907 secs |
| 'Computing features' -> 21.7444 secs | 'Computing features' -> 3.4419 secs |
| (processed) 'IMG_0249' -> 3.2839 secs | (memoized) 'IMG_0249' -> 0.4656 secs |
| (processed) 'IMG_0250' -> 3.1891 secs | (memoized) 'IMG_0250' -> 0.6815 secs |
| (processed) 'IMG_0251' -> 3.0490 secs | (memoized) 'IMG_0251' -> 0.4374 secs |
| (processed) 'IMG_0252' -> 2.8878 secs | (memoized) 'IMG_0252' -> 0.2954 secs |
| (processed) 'IMG_0253' -> 2.9675 secs | (memoized) 'IMG_0253' -> 0.8914 secs |
| (processed) 'IMG_0254' -> 3.0005 secs | (memoized) 'IMG_0254' -> 0.2775 secs |
| (processed) 'IMG_0256' -> 2.9874 secs | (memoized) 'IMG_0256' -> 0.1578 secs |
| 'Restoring' -> 1.6591 secs | 'Restoring' -> 2.0090 secs |
| 'Matching' -> 0.0738 secs | 'Matching' -> 0.0854 secs |
| 'Merging' -> 1.5848 secs | 'Merging' -> 1.9230 secs |
| 'IMG_0254' -> 1.5285 secs | 'IMG_0254' -> 1.8020 secs |
| 'Post-processing' -> 0.4615 secs | 'Post-processing' -> 0.5398 secs |

## Result for set 5

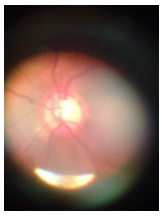
In this set 2 images were used from 7 images leading to a rate of 0.29 which is evidenced in Figure 46. The *base image* is in Figure 46a an the other in Figure 46b. Both of them presented the same flare in the same position and though they could be perfectly matched they produced a similar flare which can be seen in Figure 46c. This led to the *base image* to not being adequately corrected due that all the other images presented the same problem but the lens simulation produced a more appealing result in the merged images. Notice that the images not used in the restoring process were blurrier and safely could be ignored (Figure 47).

For this set the profiling presented in Table 11 keeps generating satisfactory results of the *imrestore* performance. Even when the set have 7 images the restoration takes only 23.86 seconds in process their features, match all the images with the automatically selected *base image*, merge those which are convenient for the restoration and post-process them with a simulation of lens. But off course, this is when a basic configuration is applied, after all *imrestore* can be configured to apply filtering, histogram matching between merged images, change the behaviour of how the images are restored and more will be included in the future (see Future Work section ). Even better the uncached time is bested when caching techniques are applied saving 17.87 seconds with its astonishing processing time of 5.99 seconds. The profiling puts in evidence that computing the feature is what takes most of the time in the application.

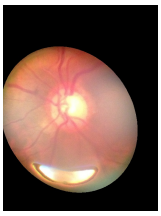
*Figure 46*. Result for set5 using command '`imrestore ../results/set5/*.* --lens --overwrite --cachePath {temp}`'



(a) IMG_0253

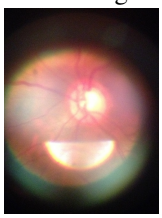


(b) IMG_0254

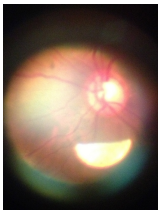


(c) Restored image

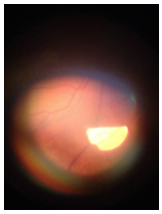
*Figure 47*. Images not used in restoration of set5



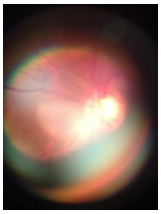
(a) IMG_0250



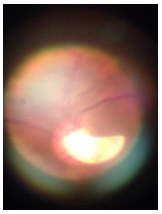
(b) IMG_0252



(c) IMG_0256



(d) IMG_0249



(e) IMG_0251

**Result for set 6**

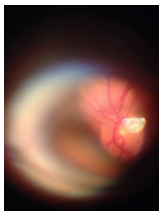
Table 12
*Profiling for set6 using command '`imrestore ../results/set6/*.* --lens --overwrite --cachePath {temp}`'*

| 'ImRestore init' -> 22.2521 secs | 'ImRestore init' -> 3.1205 secs |
|---|---|
| 'Computing features' -> 21.6767 secs | 'Computing features' -> 1.8374 secs |
| (processed) 'IMG_0257' -> 3.2036 secs | (memoized) 'IMG_0257' -> 0.1445 secs |
| (processed) 'IMG_0258' -> 3.1040 secs | (memoized) 'IMG_0258' -> 0.1330 secs |
| (processed) 'IMG_0259' -> 2.9702 secs | (memoized) 'IMG_0259' -> 0.1316 secs |
| (processed) 'IMG_0260' -> 2.9582 secs | (memoized) 'IMG_0260' -> 0.2068 secs |
| (processed) 'IMG_0262' -> 2.9436 secs | (memoized) 'IMG_0262' -> 0.3962 secs |
| (processed) 'IMG_0263' -> 3.0434 secs | (memoized) 'IMG_0263' -> 0.2603 secs |
| (processed) 'IMG_0264' -> 3.1115 secs | (memoized) 'IMG_0264' -> 0.3681 secs |
| 'Restoring' -> 0.0625 secs | 'Restoring' -> 0.3728 secs |
| 'Matching' -> 0.0289 secs | 'Matching' -> 0.0326 secs |
| 'Merging' -> 0.0334 secs | 'Merging' -> 0.3399 secs |
| 'Post-processing' -> 0.5129 secs | 'Post-processing' -> 0.9103 secs |

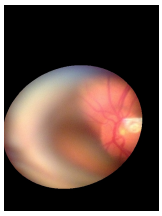
In Figure 48 there is an interesting case, from the 7 images in the set the one in Figure 48a got selected as the *base image* but it has some parts of the *Sclera* which is clearly a sign that this image should not be in the retinal set. Nonetheless the algorithm does not check for this and in fact selects the image proceeding as if nothing had happened. This case is similar to set5 in that all images are not good, none are suitable for restoring and most of them present parts of the eye that should not be in there.

As usual, the profiling of this set can be found in Table 12. It shows the 7 processed images in set6 shown in Figure 49. There was 0 images merged in the *base image* resulting in only one image used in total leaving a toll of 6 images not used for the set6. It is evidenced that the time was reduced by 7 times of the uncached session. In this way time passed from 22.25 to 3.12 seconds saving in total 19.13 seconds. This express a gain of the 85.98% with cached time only being 14.02%, this means that an excess in time of the 713.09% would happen if passed from cached to a not cached session. The statistics are not that encouraging with respect to the used images giving a rate of 0.14, 0.00 for merged images and 0.86 for failed images, a high rate for not used images.

*Figure 48*. Result for set6 using command '`imrestore ../results/set6/*.* --lens --overwrite --cachePath {temp}`'

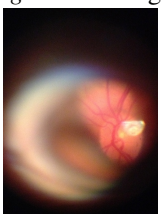


(a) IMG_0257

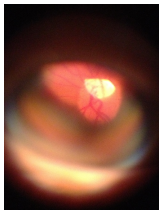


(b) Restored image

*Figure 49*. Images in set6

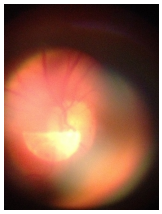


(a) IMG_0257



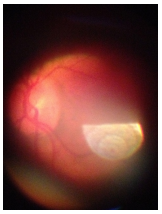
(b) IMG_0258

(c) IMG_0259

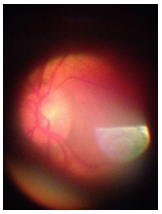


(d) IMG_0260

(e) IMG_0262



(f) IMG_0263



(g) IMG_0264

In this set it is confirmed that the algorithm currently is not good dealing with parts that do not correspond to the retinal area nonetheless it applies the restoration method yielding still better results than the input images even if they are not satisfactory.

## Result for set 7

Table 13
*Profiling for set7 using command '`imrestore ../results/set7/*.* --lens --overwrite --cachePath {temp}`'*

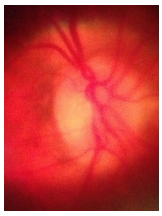
| 'ImRestore init' -> 17.3964 secs | 'ImRestore init' -> 6.2594 secs |
|---|---|
| 'Computing features' -> 15.7344 secs | 'Computing features' -> 1.8640 secs |
| (processed) 'IMG_0267' -> 3.4596 secs | (memoized) 'IMG_0267' -> 0.2475 secs |
| (processed) 'IMG_0269' -> 3.1190 secs | (memoized) 'IMG_0269' -> 0.3427 secs |
| (processed) 'IMG_0270' -> 3.1111 secs | (memoized) 'IMG_0270' -> 0.3933 secs |
| (processed) 'IMG_0271' -> 2.9327 secs | (memoized) 'IMG_0271' -> 0.4071 secs |
| (processed) 'IMG_0273' -> 3.0540 secs | (memoized) 'IMG_0273' -> 0.3494 secs |
| 'Restoring' -> 1.3234 secs | 'Restoring' -> 3.0931 secs |
| 'Matching' -> 0.0821 secs | 'Matching' -> 0.2248 secs |
| 'Merging' -> 1.2408 secs | 'Merging' -> 2.8679 secs |
| 'IMG_0271' -> 1.2214 secs | 'IMG_0271' -> 2.7820 secs |
| 'Post-processing' -> 0.3386 secs | 'Post-processing' -> 1.3023 secs |

Using again Code 5 in this set *imrestore* processed 2 of the 5 images in it. The results can be appreciated in Figure 50 and the images not used in the restoration in Figure 51. There was a total of 1 merged images with the *base image* in Figure 50a and 3 unused images for the set7. The restored image in Figure 50c does not evidence any change from the initial image but the lens simulation did a good job. There was nothing that the merged image in Figure 50b could provide to it an so nothing was introduced. The other images where not selected obviously for their lack of information and affected areas that could worsen the restoring process of the set.

Table 13 presents the profiling produced for set7. It is evidenced that the time was reduced by 3 times of the uncached session. In this way time passed from 17.40 to 6.26 seconds saving in total 11.14 seconds. This is translated in a gain of the 64.02% with cache time being

*Figure 50.* Result for set7 using command '`imrestore ../results/set7/*.* --lens --overwrite --cachePath {temp}`'

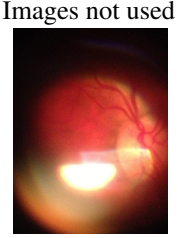
(a) IMG_0270



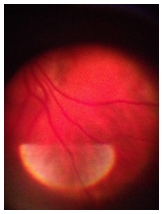
(b) IMG_0271

(c) Restored image

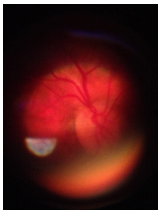
*Figure 51.* Images not used in restoration of set7



(a) IMG_0267



(b) IMG_0273



(c) IMG_0269

35.98% and this means that an excess in time of the 277.92% would happen if passed from cached to a not cached session. The rate of used images is 0.40, merged images 0.20 and failed images 0.60.

**Result for set 8**

Table 14
*Profiling for set8 using command '`imrestore ../results/set8/*.* --lens --overwrite --cachePath {temp}`'*

```
'ImRestore init' -> 23.3291 secs
└─ 'Computing features' -> 22.8403 secs
   ├─ (processed) 'IMG_0306' -> 3.5407 secs
   ├─ (processed) 'IMG_0307' -> 3.3176 secs
   ├─ (processed) 'IMG_0308' -> 3.1198 secs
   ├─ (processed) 'IMG_0309' -> 3.0648 secs
   ├─ (processed) 'IMG_0310' -> 3.1863 secs
   ├─ (processed) 'IMG_0311' -> 3.2326 secs
   └─ (processed) 'IMG_0357' -> 3.0689 secs
└─ 'Restoring' -> 0.0757 secs
   ├─ 'Matching' -> 0.0285 secs
   └─ 'Merging' -> 0.0470 secs
└─ 'Post-processing' -> 0.4130 secs
```

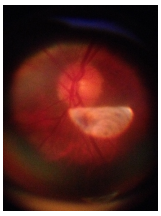
```
'ImRestore init' -> 4.5408 secs
└─ 'Computing features' -> 2.7663 secs
   ├─ (memoized) 'IMG_0306' -> 0.4620 secs
   ├─ (memoized) 'IMG_0307' -> 0.2225 secs
   ├─ (memoized) 'IMG_0308' -> 0.1624 secs
   ├─ (memoized) 'IMG_0309' -> 0.2730 secs
   ├─ (memoized) 'IMG_0310' -> 0.4763 secs
   ├─ (memoized) 'IMG_0311' -> 0.2752 secs
   └─ (memoized) 'IMG_0357' -> 0.2454 secs
└─ 'Restoring' -> 0.3333 secs
   ├─ 'Matching' -> 0.0767 secs
   └─ 'Merging' -> 0.2561 secs
└─ 'Post-processing' -> 1.4412 secs
```

Figure 53 shows all the images presented in set8.

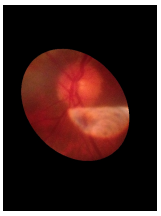
Figure 52 presents the results for set8 processing only 1 of the 7 images in it where the image in Figure 52a was selected as the *base image* leaving 6 images not used in the restoration of the set, this is shown in **??**. The restored image in Figure 52b shows a huge improvement over the input image. It can be observed that though most of it was blurry around the retinal area it got segmented correctly and the lens simulation fitted perfectly. This would have not being achieved with normal shareholding methods that evidently would have added all the area around the retinal area.

Table 14 presents the profiling produced for set8 where we can observe the times which took the program to run each step of the restoration process. In it the uncached session time passed from 23.33 seconds to 4.54 seconds in the cached session saving in total 18.79

*Figure 52*. Result for set8 using command ‘`imrestore ../results/set8/*.* --lens --overwrite --cachePath {temp}`’



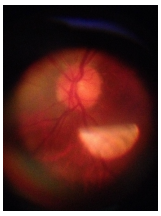
(a) IMG_0306



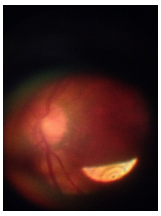
(b) Restored image

*Figure 53*. Images in set8

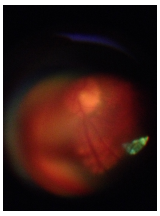
(a) IMG_0306



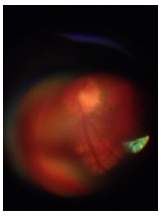
(b) IMG_0307



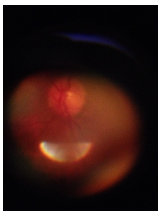
(c) IMG_0308



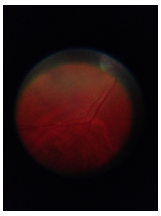
(d) IMG_0309



(e) IMG_0310



(f) IMG_0311



(g) IMG_0357

seconds that is 5 times faster than the uncached session and cache time only being the 19.46% from the total un-cached time.

The rate of used images is 0.14 with respect to the total of images in the set, this signifies that most images were no used but at the same time tells how well the algorithm left out the unwanted images. This is truth, as images in **??** present many problems and lack of information that cannot be used for the restoration of the *base image* in Figure 52a. The rate of images not used in the set is 0.86, completing the total with the used rate to 1. In a way these rates tell how good a set was taken for restoration, if it is low then images were not adequately taken but if the used rate reaches 1 or unused rate 0 it means that images present high quality from good perspectives of the same retinal image.

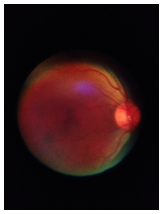
**Result for set 9**

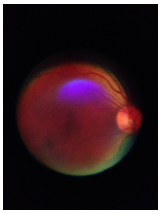
Table 15
*Profiling for set9 using command ‘`imrestore ../results/set9/*.* --lens --overwrite --cachePath {temp}`’*

- ’ImRestore init’ -> 11.8629 secs
  - ’Computing features’ -> 9.8378 secs
    - (processed) ’IMG_0358’ -> 3.4007 secs
    - (processed) ’IMG_0359’ -> 3.2114 secs
    - (processed) ’IMG_0360’ -> 3.0808 secs
  - ’Restoring’ -> 1.5534 secs
    - ’Matching’ -> 0.0342 secs
    - ’Merging’ -> 1.5187 secs
      - ’IMG_0359’ -> 1.5023 secs
  - ’Post-processing’ -> 0.4718 secs

- ’ImRestore init’ -> 7.0251 secs
  - ’Computing features’ -> 1.6750 secs
    - (memoized) ’IMG_0358’ -> 0.5123 secs
    - (memoized) ’IMG_0359’ -> 0.7498 secs
    - (memoized) ’IMG_0360’ -> 0.3352 secs
  - ’Restoring’ -> 4.1770 secs
    - ’Matching’ -> 0.0747 secs
    - ’Merging’ -> 4.1017 secs
      - ’IMG_0359’ -> 4.0529 secs
  - ’Post-processing’ -> 1.1731 secs

*Figure 54*. Result for set9 using command ‘`imrestore ../results/set9/*.* --lens --overwrite --cachePath {temp}`’


(a) IMG_0358


(b) IMG_0359

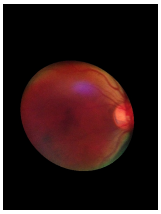
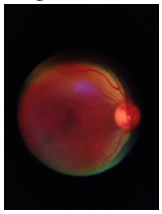
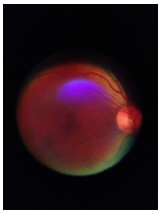
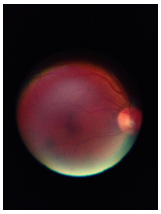

(c) Restored image

*Figure 55*. Images in set9


(a) IMG_0358


(b) IMG_0359


(c) IMG_0360

This set is only conformed by 3 images that are shown in Figure 55. Here no image present good quality and there is some kind of blue light in two of them (Figure 55a and Figure 55b), the other is too bright to be selected appreciated in Figure 55a. In spite of that 2 were used for the restoration with 1 used in Figure 54b to merge in the *base image* in Figure 54a and the other in Figure 55c was wisely left out from the restoration for its brightness. Figure 54 shows the merging process with the restored image in Figure 54c. The restored image is really similar to the *base image* with the difference of the lens simulation but it is ok as the merging image did not contribute good features to merge. In effect it was a good choice not to use any feature of the image in Figure 54b preventing the *base image* to be ruined.

The profiling produced for set9 is presented in Table 15 that shows prove of the restoration process and that it has not been altered. From it can extrated that the unchaed time was reduced 2 times with the cached session because the process took 11.86 seconds in the uncached session (lets assign it the 100%) and then 7.03 seconds in the cached session (the 59.22%), taking 4.84 seconds of difference (40.78%). If the cached session consumed the 100% of the processed time then the uncached time would be 168.87%.

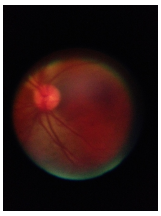
This set performed a good used rate of 0.67 and of failed images as 0.33, indicating that in general the set was adequate for restoration. Notice however that these rates do not provide useful information regarding details like if the restoration was satisfactory or if the selected images did not present bad features.

**Result for set 10**

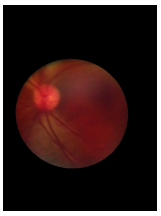
Table 16
*Profiling for set10 using command* '`imrestore ../results/set10/*.* --lens --overwrite --cachePath {temp}`'

| 'ImRestore init' -> 19.7932 secs | 'ImRestore init' -> 5.1132 secs |
|---|---|
| 'Computing features' -> 19.2640 secs | 'Computing features' -> 2.8103 secs |
| (processed) 'IMG_0361' -> 1.6647 secs | (memoized) 'IMG_0361' -> 0.3951 secs |
| (processed) 'IMG_0362' -> 1.6796 secs | (memoized) 'IMG_0362' -> 0.1611 secs |
| (processed) 'IMG_0363' -> 1.6930 secs | (memoized) 'IMG_0363' -> 0.2822 secs |
| (processed) 'IMG_0364' -> 1.6233 secs | (memoized) 'IMG_0364' -> 0.3484 secs |
| (processed) 'IMG_0365' -> 1.6466 secs | (memoized) 'IMG_0365' -> 0.1970 secs |
| (processed) 'IMG_0366' -> 1.7183 secs | (memoized) 'IMG_0366' -> 0.2956 secs |
| (processed) 'IMG_0367' -> 2.4281 secs | (memoized) 'IMG_0367' -> 0.0681 secs |
| (processed) 'IMG_0368' -> 1.9748 secs | (memoized) 'IMG_0368' -> 0.2667 secs |
| (processed) 'IMG_0369' -> 2.2068 secs | (memoized) 'IMG_0369' -> 0.2751 secs |
| (processed) 'IMG_0370' -> 2.0565 secs | (memoized) 'IMG_0370' -> 0.2951 secs |
| 'Restoring' -> 0.1149 secs | 'Restoring' -> 0.6961 secs |
| 'Matching' -> 0.0338 secs | 'Matching' -> 0.1171 secs |
| 'Merging' -> 0.0809 secs | 'Merging' -> 0.5787 secs |
| 'Post-processing' -> 0.4142 secs | 'Post-processing' -> 1.6067 secs |

*Figure 56*. Result for set10 using command '`imrestore ../results/set10/*.* --lens --overwrite --cachePath {temp}`'

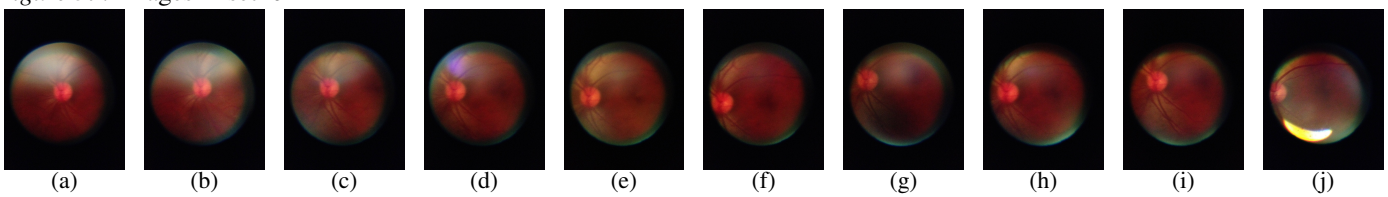


(a) IMG_0369



(b) Restored image

*Figure 57*. Images in set10



(a) IMG_0361 (b) IMG_0362 (c) IMG_0363 (d) IMG_0364 (e) IMG_0365 (f) IMG_0366 (g) IMG_0367 (h) IMG_0368 (i) IMG_0369 (j) IMG_0370

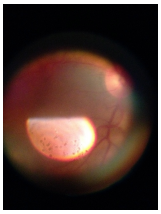
Using Code 5 in set10 *imrestore* processed 1 of the 10 images in set10, this is shown in Figure 57. 0 is the number of images merged in the *base image*. 9 is the number of images not used for the set10. Figure 56 presents the results for set10. **??** shows the images not used in set10. Figure 56a shows the *base image*, Figure 56b shows the restored image and the remaining merged image. Figure 57a shows the first image in set. **??** shows the first image not used in the merging. Table 16 presents the profiling produced for set10. It is evidenced that the time was reduced by 4 times of the uncached session. In this way time passed from 19.79 to 5.11 seconds saving in total 14.68 seconds. This is translated in a gain of the 74.17% with cache time being 25.83% and this means that an excess in time of the 387.10% would happen if passed from cached to a not cached session the rate of used images is 0.10. the rate of merged images is 0.00. the rate of failed images is 0.90.

**Result for set 11**

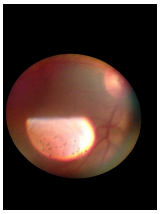
Table 17
*Profiling for set11 using command* '`imrestore ../results/set11/*.* --lens --overwrite --cachePath {temp}`'

| | |
|---|---|
| 'ImRestore init' -> 11.4920 secs | 'ImRestore init' -> 3.2793 secs |
| 'Computing features' -> 11.1180 secs | 'Computing features' -> 1.6029 secs |
| (processed) 'IMG_0373' -> 1.7432 secs | (memoized) 'IMG_0373' -> 0.2413 secs |
| (processed) 'IMG_0374' -> 1.7374 secs | (memoized) 'IMG_0374' -> 0.1325 secs |
| (processed) 'IMG_0375' -> 1.7869 secs | (memoized) 'IMG_0375' -> 0.1747 secs |
| (processed) 'IMG_0377' -> 1.6624 secs | (memoized) 'IMG_0377' -> 0.2360 secs |
| (processed) 'IMG_0379' -> 1.7335 secs | (memoized) 'IMG_0379' -> 0.4524 secs |
| (processed) 'IMG_0380' -> 1.6463 secs | (memoized) 'IMG_0380' -> 0.2016 secs |
| 'Restoring' -> 0.0518 secs | 'Restoring' -> 0.2934 secs |
| 'Matching' -> 0.0206 secs | 'Matching' -> 0.0865 secs |
| 'Merging' -> 0.0310 secs | 'Merging' -> 0.2067 secs |
| 'Post-processing' -> 0.3222 secs | 'Post-processing' -> 1.3831 secs |

*Figure 58*. Result for set11 using command '`imrestore ../results/set11/*.* --lens --overwrite --cachePath {temp}`'



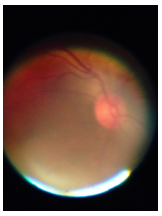
(a) IMG_0379



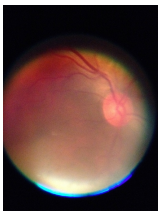
(b) Restored image

*Figure 59*. Images in set11

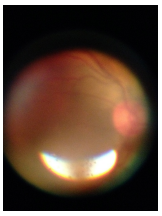
(a) IMG_0373



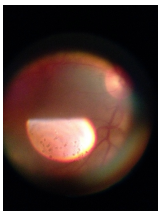
(b) IMG_0374



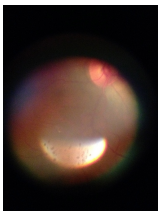
(c) IMG_0375



(d) IMG_0377



(e) IMG_0379



(f) IMG_0380

Using Code 5 in set11 *imrestore* processed 1 of the 6 images in set11, this is shown in Figure 59. 0 is the number of images merged in the *base image*. 5 is the number of images not used for the set11. Figure 58 presents the results for set11. **??** shows the images not used in set11. Figure 58a shows the *base image*, Figure 58b shows the restored image and the remaining merged image. Figure 59a shows the first image in set. **??** shows the first image not used in the merging. Table 17 presents the profiling produced for set11. It is evidenced that the time was reduced by 4 times of the uncached session. In this way time passed from 11.49 to 3.28 seconds saving in total 8.21 seconds. This is translated in a gain of the 71.46% with cache time being 28.54% and this means that an excess in time of the 350.44% would happen if passed from cached to a not cached session the rate of used images is 0.17. the rate of merged images is 0.00. the rate of failed images is 0.83.

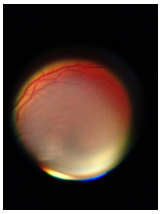
**Result for set 12**

Table 18
*Profiling for set12 using command '*`imrestore ../results/set12/*.* --lens --overwrite --cachePath {temp}`*'*

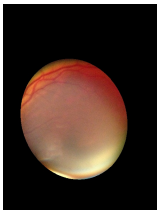
'ImRestore init' -> 9.7331 secs
- 'Computing features' -> 9.1343 secs
  - (processed) 'IMG_0381' -> 1.8524 secs
  - (processed) 'IMG_0382' -> 2.0073 secs
  - (processed) 'IMG_0383' -> 2.3685 secs
  - (processed) 'IMG_0384' -> 2.7038 secs
- 'Restoring' -> 0.1217 secs
  - 'Matching' -> 0.0199 secs
  - 'Merging' -> 0.1015 secs
- 'Post-processing' -> 0.4771 secs

'ImRestore init' -> 3.4769 secs
- 'Computing features' -> 1.9565 secs
  - (memoized) 'IMG_0381' -> 0.2367 secs
  - (memoized) 'IMG_0382' -> 0.5939 secs
  - (memoized) 'IMG_0383' -> 0.2115 secs
  - (memoized) 'IMG_0384' -> 0.3576 secs
- 'Restoring' -> 0.2917 secs
  - 'Matching' -> 0.0875 secs
  - 'Merging' -> 0.1941 secs
- 'Post-processing' -> 1.2287 secs

*Figure 60*. Result for set12 using command '`imrestore ../results/set12/*.* --lens --overwrite --cachePath {temp}`'

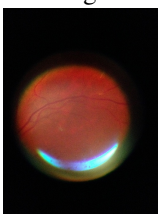


(a) IMG_0382

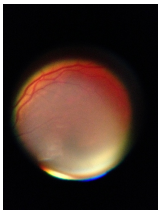


(b) Restored image

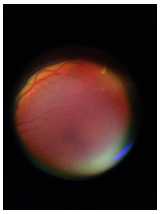
*Figure 61*. Images in set12



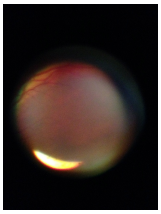
(a) IMG_0381



(b) IMG_0382



(c) IMG_0383



(d) IMG_0384

Using Code 5 in set12 *imrestore* processed 1 of the 4 images in set12, this is shown in Figure 61. 0 is the number of images merged in the *base image*. 3 is the number of images not used for the set12. Figure 60 presents the results for set12. **??** shows the images not used in set12. Figure 60a shows the *base image*, Figure 60b shows the restored image and the remaining merged image. Figure 61a shows the first image in set. **??** shows the first image not used in the merging. Table 18 presents the profiling produced for set12. It is evidenced that the time was reduced by 3 times of the uncached session. In this way time passed from 9.73 to 3.48 seconds saving in total 6.26 seconds. This is translated in a gain of the 64.28% with cache time being 35.72% and this means that an excess in time of the 279.94% would happen if passed from cached to a not cached session the rate of used images is 0.25. the rate of merged images is 0.00. the rate of failed images is 0.75.

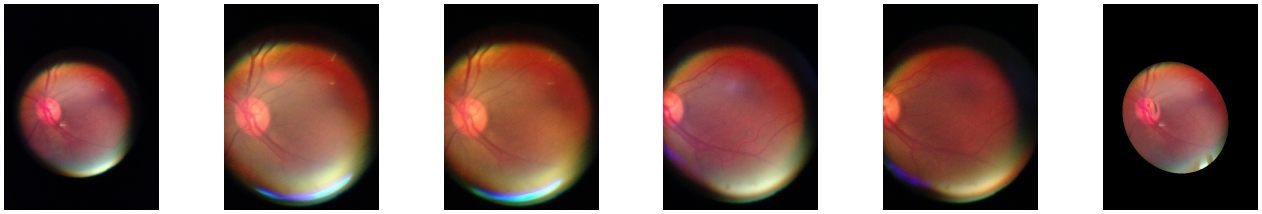
**Result for set 13**

Table 19
*Profiling for set13 using command* '`imrestore ../results/set13/*.* --lens --overwrite --cachePath {temp}`'

- 'ImRestore init' -> 18.3997 secs
  - 'Computing features' -> 12.8125 secs
    - (processed) 'IMG_0386' -> 2.7297 secs
    - (processed) 'IMG_0387' -> 2.6766 secs
    - (processed) 'IMG_0388' -> 2.6601 secs
    - (processed) 'IMG_0389' -> 2.1324 secs
    - (processed) 'IMG_0390' -> 2.4107 secs
  - 'Restoring' -> 5.2833 secs
    - 'Matching' -> 0.0669 secs
    - 'Merging' -> 5.2157 secs
      - 'IMG_0387' -> 1.4573 secs
      - 'IMG_0388' -> 1.1390 secs
      - 'IMG_0390' -> 1.4722 secs
      - 'IMG_0389' -> 1.1470 secs
  - 'Post-processing' -> 0.3039 secs

- 'ImRestore init' -> 11.0310 secs
  - 'Computing features' -> 1.8227 secs
    - (memoized) 'IMG_0386' -> 0.3536 secs
    - (memoized) 'IMG_0387' -> 0.6477 secs
    - (memoized) 'IMG_0388' -> 0.1994 secs
    - (memoized) 'IMG_0389' -> 0.2435 secs
    - (memoized) 'IMG_0390' -> 0.2834 secs
  - 'Restoring' -> 8.5249 secs
    - 'Matching' -> 0.0759 secs
    - 'Merging' -> 8.4483 secs
      - 'IMG_0387' -> 1.8513 secs
      - 'IMG_0388' -> 2.0259 secs
      - 'IMG_0390' -> 1.9489 secs
      - 'IMG_0389' -> 2.6221 secs
  - 'Post-processing' -> 0.6833 secs

*Figure 62*. Result for set13 using command '`imrestore ../results/set13/*.* --lens --overwrite --cachePath {temp}`'



(a) IMG_0386 (b) IMG_0387 (c) IMG_0388 (d) IMG_0390 (e) IMG_0389 (f) Restored image

Using Code 5 in set13 *imrestore* processed 5 of the 5 images in set13, this is shown in **??**. 4 is the number of images merged in the *base image*. 0 is the number of images not used for the set13. Figure 62 presents the results for set13. **??** shows the images not used in set13. Figure 62a shows the *base image*, Figure 62f shows the restored image and the remaining merged images. **??** shows the first image in set. **??** shows the first image not used in the merging. Table 19 presents the profiling produced for set13. It is evidenced that the time was reduced by 2 times of the uncached session. In this way time passed from 18.40 to 11.03 seconds saving in total 7.37 seconds. This is translated in a gain of the 40.05% with cache time being 59.95% and this means that an excess in time of the 166.80% would happen if passed from cached to a not cached session the rate of used images is 1.00. the rate of merged images is 0.80. the rate of failed images is 0.00.

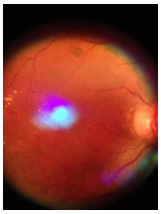
**Result for set 14**

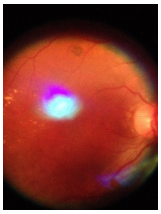
Table 20
*Profiling for set14 using command 'imrestore ../results/set14/*.* --lens --overwrite --cachePath {temp}'*

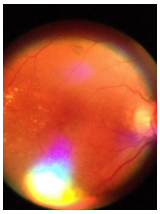
'ImRestore init' -> 15.8853 secs
- 'Computing features' -> 12.4801 secs
  - (processed) 'IMG_0401' -> 2.7119 secs
  - (processed) 'IMG_0402' -> 3.7434 secs
  - (processed) 'IMG_0403' -> 3.1613 secs
  - (processed) 'IMG_0404' -> 2.6146 secs
- 'Restoring' -> 2.9744 secs
  - 'Matching' -> 0.0961 secs
  - 'Merging' -> 2.8776 secs
    - 'IMG_0403' -> 1.4089 secs
    - 'IMG_0404' -> 1.4464 secs
- 'Post-processing' -> 0.4309 secs

'ImRestore init' -> 9.8135 secs
- 'Computing features' -> 1.5526 secs
  - (memoized) 'IMG_0401' -> 0.2395 secs
  - (memoized) 'IMG_0402' -> 0.5507 secs
  - (memoized) 'IMG_0403' -> 0.3295 secs
  - (memoized) 'IMG_0404' -> 0.3061 secs
- 'Restoring' -> 7.0761 secs
  - 'Matching' -> 0.1532 secs
  - 'Merging' -> 6.9223 secs
    - 'IMG_0403' -> 3.2824 secs
    - 'IMG_0404' -> 3.6093 secs
- 'Post-processing' -> 1.1848 secs

*Figure 63*. Result for set14 using command 'imrestore ../results/set14/*.* --lens --overwrite --cachePath {temp}'


(a) IMG_0402


(b) IMG_0403


(c) IMG_0404

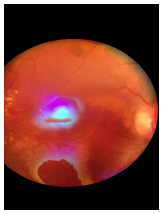
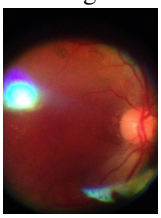
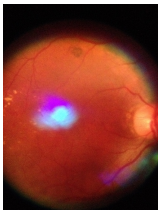
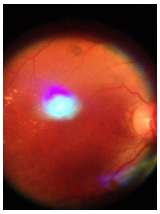
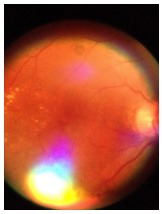

(d) Restored image

*Figure 64*. Images in set14


(a) IMG_0401


(b) IMG_0402


(c) IMG_0403


(d) IMG_0404

Using Code 5 in set14 *imrestore* processed 3 of the 4 images in set14, this is shown in Figure 64. 2 is the number of images merged in the *base image*. 1 is the number of images not used for the set14. Figure 63 presents the results for set14. **??** shows the images not used in set14. Figure 63a shows the *base image*, Figure 63d shows the restored image and the remaining merged images. Figure 64a shows the first image in set. **??** shows the first image not used in the merging. Table 20 presents the profiling produced for set14. It is evidenced that the time was reduced by 2 times of the uncached session. In this way time passed from 15.89 to 9.81 seconds saving in total 6.07 seconds. This is translated in a gain of the 38.22% with cache time being 61.78% and this means that an excess in time of the 161.87% would happen if passed from cached to a not cached session the rate of used images is 0.75. the rate of merged images is 0.50. the rate of failed images is 0.25.

**Result for set 15**

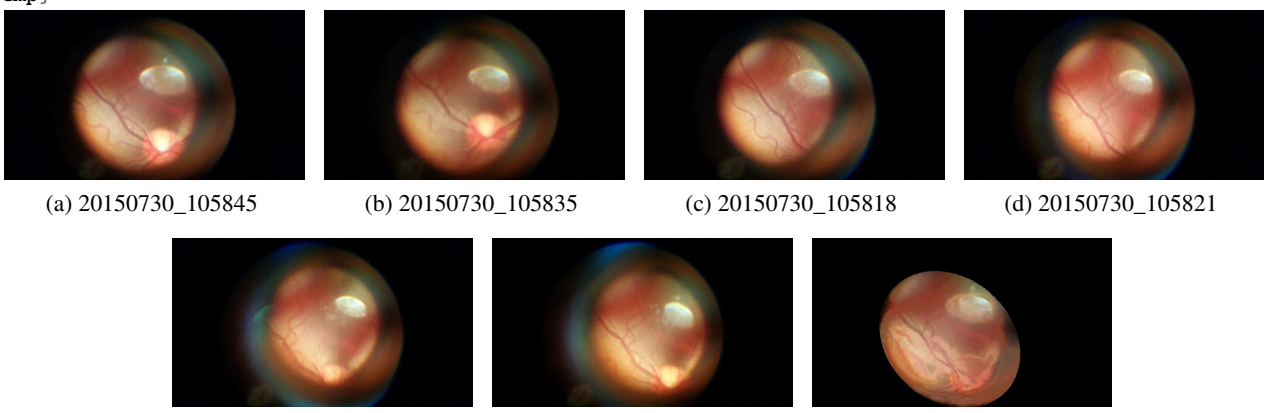
Table 21
*Profiling for set15 using command '`imrestore ../results/set15/*.* --lens --overwrite --cachePath {temp}`'*

- 'ImRestore init' -> 64.7092 secs
  - 'Computing features' -> 56.7453 secs
    - (processed) '20150730_092749' -> 2.2607 secs
    - (processed) '20150730_092751' -> 2.0070 secs
    - (processed) '20150730_092752' -> 1.9977 secs
    - (processed) '20150730_092756' -> 2.0434 secs
    - (processed) '20150730_092757' -> 2.0984 secs
    - (processed) '20150730_092758(0)' -> 2.1579 secs
    - (processed) '20150730_092758' -> 2.1397 secs
    - (processed) '20150730_092843' -> 2.2116 secs
    - (processed) '20150730_092846' -> 2.2935 secs
    - (processed) '20150730_092934' -> 2.2503 secs
    - (processed) '20150730_105818' -> 2.4197 secs
    - (processed) '20150730_105821' -> 2.4038 secs
    - (processed) '20150730_105822' -> 2.3793 secs
    - (processed) '20150730_105828' -> 2.5072 secs
    - (processed) '20150730_105830' -> 2.4340 secs
    - (processed) '20150730_105835' -> 2.5065 secs
    - (processed) '20150730_105842' -> 2.5571 secs
    - (processed) '20150730_105845' -> 2.6880 secs
    - (processed) '20150730_105901' -> 1.9463 secs
    - (processed) '20150730_105907' -> 1.8988 secs
    - (processed) '20150730_105912' -> 2.0264 secs
    - (processed) '20150730_105915' -> 3.2715 secs
    - (processed) '20150730_105921' -> 2.7538 secs
    - (processed) '20150730_105925' -> 2.6166 secs
  - 'Restoring' -> 7.5406 secs
    - 'Matching' -> 0.7458 secs
    - 'Merging' -> 6.7934 secs
      - '20150730_105835' -> 1.4151 secs
      - '20150730_105818' -> 1.3202 secs
      - '20150730_105821' -> 1.2991 secs
      - '20150730_105828' -> 1.3112 secs
      - '20150730_105822' -> 1.4006 secs
  - 'Post-processing' -> 0.4233 secs

- 'ImRestore init' -> 23.7269 secs
  - 'Computing features' -> 6.2923 secs
    - (memoized) '20150730_092749' -> 0.2724 secs
    - (memoized) '20150730_092751' -> 0.1949 secs
    - (memoized) '20150730_092752' -> 0.1478 secs
    - (memoized) '20150730_092756' -> 0.1412 secs
    - (memoized) '20150730_092757' -> 0.1987 secs
    - (memoized) '20150730_092758(0)' -> 0.4324 secs
    - (memoized) '20150730_092758' -> 0.3082 secs
    - (memoized) '20150730_092843' -> 0.3421 secs
    - (memoized) '20150730_092846' -> 0.6019 secs
    - (memoized) '20150730_092934' -> 0.1055 secs
    - (memoized) '20150730_105818' -> 0.1703 secs
    - (memoized) '20150730_105821' -> 0.2121 secs
    - (memoized) '20150730_105822' -> 0.1325 secs
    - (memoized) '20150730_105828' -> 0.1290 secs
    - (memoized) '20150730_105830' -> 0.1368 secs
    - (memoized) '20150730_105835' -> 0.6456 secs
    - (memoized) '20150730_105842' -> 0.3335 secs
    - (memoized) '20150730_105845' -> 0.3172 secs
    - (memoized) '20150730_105901' -> 0.1887 secs
    - (memoized) '20150730_105907' -> 0.1632 secs
    - (memoized) '20150730_105912' -> 0.3109 secs
    - (memoized) '20150730_105915' -> 0.2279 secs
    - (memoized) '20150730_105921' -> 0.1918 secs
    - (memoized) '20150730_105925' -> 0.1654 secs
  - 'Restoring' -> 16.4187 secs
    - 'Matching' -> 1.4788 secs
    - 'Merging' -> 14.9385 secs
      - '20150730_105835' -> 2.5812 secs
      - '20150730_105818' -> 2.0717 secs
      - '20150730_105821' -> 2.0427 secs
      - '20150730_105828' -> 4.3992 secs
      - '20150730_105822' -> 3.6504 secs
  - 'Post-processing' -> 1.0160 secs

Using Code 5 in set15 *imrestore* processed 6 of the 24 images in set15, this is shown in **??**. 5 is the number of images merged in the *base image*. 18 is the number of images not used for the set15. Figure 65 presents the results for set15. Figure 66 shows the images not used in set15. Figure 65a shows the *base image*, Figure 65g shows the restored image and the remaining merged images. **??** shows the first image in set. Figure 66a shows the first image not used in the merging. Table 21 presents the profiling produced for set15. It is evidenced that the time was reduced by 3 times of the uncached session. In this way time passed from 64.71 to 23.73 seconds saving in total 40.98 seconds. This is translated in a gain of the 63.33% with cache time being 36.67% and this means that an excess in time of the 272.72% would happen if passed from cached to a not cached session the rate of used images is 0.25. the rate of merged images is 0.21. the rate of failed images is 0.75.

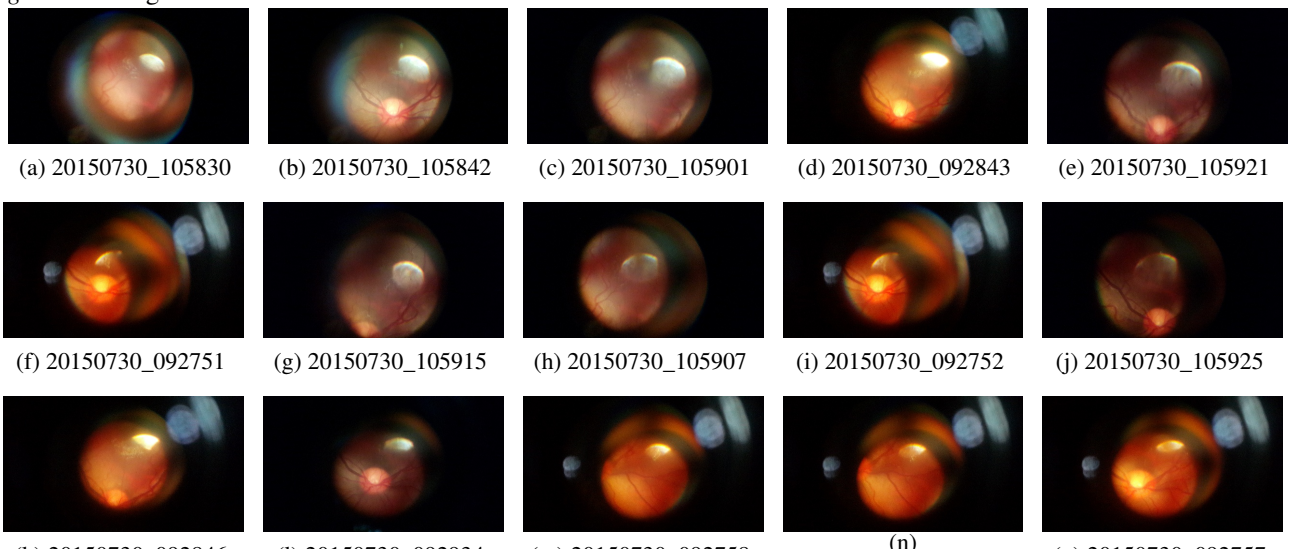
*Figure 65*. Result for set15 using command '`imrestore ../results/set15/*.* --lens --overwrite --cachePath {temp}`'



(a) 20150730_105845 (b) 20150730_105835 (c) 20150730_105818 (d) 20150730_105821

(e) 20150730_105828 (f) 20150730_105822 (g) Restored image

*Figure 66*. Images not used in restoration of set15



(a) 20150730_105830 (b) 20150730_105842 (c) 20150730_105901 (d) 20150730_092843 (e) 20150730_105921

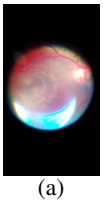
(f) 20150730_092751 (g) 20150730_105915 (h) 20150730_105907 (i) 20150730_092752 (j) 20150730_105925

(k) 20150730_092846 (l) 20150730_092934 (m) 20150730_092758 (n) 20150730_092758(0) (o) 20150730_092757

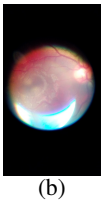
(p) 20150730_092756 (q) 20150730_105912 (r) 20150730_092749

**Result for set 16**

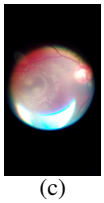
*Figure 67*. Result for set16 using command ‘`imrestore ../results/set16/*.* --lens --overwrite --cachePath {temp}`’



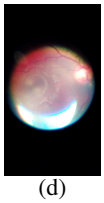
(a) IMG_20150730_115646_3



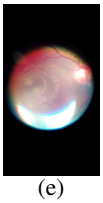
(b) IMG_20150730_115646_2



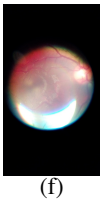
(c) IMG_20150730_115646_1



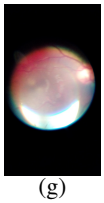
(d) IMG_20150730_115646



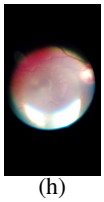
(e) IMG_20150730_115645_4



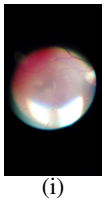
(f) IMG_20150730_115645_3



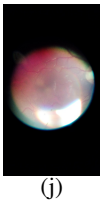
(g) IMG_20150730_115645_2



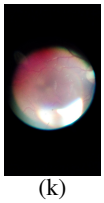
(h) IMG_20150730_115645_1



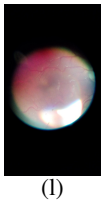
(i) IMG_20150730_115645



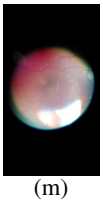
(j) IMG_20150730_115644_2



(k) IMG_20150730_115644_3



(l) IMG_20150730_115644



(m) IMG_20150730_115644_1

(n) Restored image

Using Code 5 in set16 *imrestore* processed 13 of the 41 images in set16, this is shown in **??**. 12 is the number of images merged in the *base image*. 28 is the number of images not used for the set16. Figure 67 presents the results for set16. **??** shows the images not used in set16. Figure 67a shows the *base image*, Figure 67n shows the restored image and the remaining merged images. **??** shows the first image in set. **??** shows the first image not used in the merging. Table 22 presents the profiling produced for set16. It is evidenced that the time was reduced by 2 times of the uncached session. In this way time passed from 117.93 to 51.41 seconds saving in total 66.52 seconds. This is translated in a gain of the 56.41% with cache time being 43.59% and this means that an excess in time of the 229.40% would happen if passed from cached to a not cached session the rate of used images is 0.32. the rate of merged images is 0.29. the rate of failed images is 0.68.

Table 22
*Profiling for set16 using command '`imrestore ../results/set16/*.* --lens --overwrite --cachePath {temp}`'*

```
'ImRestore init' -> 117.9345 secs
└── 'Computing features' -> 97.5388 secs
    ├── (processed) 'IMG_20150730_115646_2' -> 1.4922 secs
    ├── (processed) 'IMG_20150730_115649_2' -> 1.4647 secs
    ├── (processed) 'IMG_20150730_115642' -> 1.4554 secs
    ├── (processed) 'IMG_20150730_115644' -> 1.4881 secs
    ├── (processed) 'IMG_20150730_115644_1' -> 1.4671 secs
    ├── (processed) 'IMG_20150730_115644_2' -> 1.4397 secs
    ├── (processed) 'IMG_20150730_115644_3' -> 1.4666 secs
    ├── (processed) 'IMG_20150730_115645' -> 1.5455 secs
    ├── (processed) 'IMG_20150730_115645_1' -> 1.5194 secs
    ├── (processed) 'IMG_20150730_115645_2' -> 1.5564 secs
    ├── (processed) 'IMG_20150730_115645_3' -> 1.5483 secs
    ├── (processed) 'IMG_20150730_115645_4' -> 1.5988 secs
    ├── (processed) 'IMG_20150730_115646' -> 1.7309 secs
    ├── (processed) 'IMG_20150730_115646_1' -> 2.4020 secs
    ├── (processed) 'IMG_20150730_115646_3' -> 2.1936 secs
    ├── (processed) 'IMG_20150730_115646_4' -> 2.1286 secs
    ├── (processed) 'IMG_20150730_115646_5' -> 2.3441 secs
    ├── (processed) 'IMG_20150730_115647' -> 2.5707 secs
    ├── (processed) 'IMG_20150730_115647_1' -> 2.3986 secs
    ├── (processed) 'IMG_20150730_115647_2' -> 2.3909 secs
    ├── (processed) 'IMG_20150730_115647_3' -> 2.7939 secs
    ├── (processed) 'IMG_20150730_115647_4' -> 2.4781 secs
    ├── (processed) 'IMG_20150730_115647_5' -> 2.4089 secs
    ├── (processed) 'IMG_20150730_115648' -> 2.7425 secs
    ├── (processed) 'IMG_20150730_115648_1' -> 2.7123 secs
    ├── (processed) 'IMG_20150730_115648_2' -> 2.9234 secs
    ├── (processed) 'IMG_20150730_115648_3' -> 2.7251 secs
    ├── (processed) 'IMG_20150730_115648_4' -> 2.8502 secs
    ├── (processed) 'IMG_20150730_115649' -> 2.8096 secs
    ├── (processed) 'IMG_20150730_115649_1' -> 2.6288 secs
    ├── (processed) 'IMG_20150730_115649_3' -> 2.9014 secs
    ├── (processed) 'IMG_20150730_115649_4' -> 3.0283 secs
    ├── (processed) 'IMG_20150730_115649_5' -> 3.0345 secs
    ├── (processed) 'IMG_20150730_115650' -> 3.0509 secs
    ├── (processed) 'IMG_20150730_115650_1' -> 3.0572 secs
    ├── (processed) 'IMG_20150730_115650_2' -> 3.3036 secs
    ├── (processed) 'IMG_20150730_115650_3' -> 2.9822 secs
    ├── (processed) 'IMG_20150730_115650_4' -> 3.0594 secs
    ├── (processed) 'IMG_20150730_115651' -> 3.0236 secs
    ├── (processed) 'IMG_20150730_115651_1' -> 2.9403 secs
    └── (processed) 'IMG_20150730_115651_2' -> 3.0323 secs
└── 'Restoring' -> 19.9346 secs
    ├── 'Matching' -> 2.1919 secs
    └── 'Merging' -> 17.7395 secs
        ├── 'IMG_20150730_115646_2' -> 1.2835 secs
        ├── 'IMG_20150730_115646_1' -> 1.3223 secs
        ├── 'IMG_20150730_115646' -> 1.3512 secs
        ├── 'IMG_20150730_115645_4' -> 1.4103 secs
        ├── 'IMG_20150730_115645_3' -> 1.4844 secs
        ├── 'IMG_20150730_115645_2' -> 1.4223 secs
        ├── 'IMG_20150730_115645_1' -> 1.4780 secs
        ├── 'IMG_20150730_115645' -> 1.3184 secs
        ├── 'IMG_20150730_115644_2' -> 1.4152 secs
        ├── 'IMG_20150730_115644_3' -> 1.4898 secs
        ├── 'IMG_20150730_115644' -> 1.5109 secs
        └── 'IMG_20150730_115644_1' -> 1.6267 secs
└── 'Post-processing' -> 0.4611 secs
```

```
'ImRestore init' -> 51.4099 secs
└── 'Computing features' -> 2.9800 secs
    ├── (memoized) 'IMG_20150730_115646_2' -> 0.0569 secs
    ├── (memoized) 'IMG_20150730_115649_2' -> 0.0395 secs
    ├── (memoized) 'IMG_20150730_115642' -> 0.0501 secs
    ├── (memoized) 'IMG_20150730_115644' -> 0.0501 secs
    ├── (memoized) 'IMG_20150730_115644_1' -> 0.0381 secs
    ├── (memoized) 'IMG_20150730_115644_2' -> 0.0493 secs
    ├── (memoized) 'IMG_20150730_115644_3' -> 0.0400 secs
    ├── (memoized) 'IMG_20150730_115645' -> 0.0533 secs
    ├── (memoized) 'IMG_20150730_115645_1' -> 0.0420 secs
    ├── (memoized) 'IMG_20150730_115645_2' -> 0.0775 secs
    ├── (memoized) 'IMG_20150730_115645_3' -> 0.1020 secs
    ├── (memoized) 'IMG_20150730_115645_4' -> 0.1209 secs
    ├── (memoized) 'IMG_20150730_115646' -> 0.0653 secs
    ├── (memoized) 'IMG_20150730_115646_1' -> 0.0471 secs
    ├── (memoized) 'IMG_20150730_115646_3' -> 0.0686 secs
    ├── (memoized) 'IMG_20150730_115646_4' -> 0.0600 secs
    ├── (memoized) 'IMG_20150730_115646_5' -> 0.0434 secs
    ├── (memoized) 'IMG_20150730_115647' -> 0.0538 secs
    ├── (memoized) 'IMG_20150730_115647_1' -> 0.0777 secs
    ├── (memoized) 'IMG_20150730_115647_2' -> 0.0634 secs
    ├── (memoized) 'IMG_20150730_115647_3' -> 0.0406 secs
    ├── (memoized) 'IMG_20150730_115647_4' -> 0.0746 secs
    ├── (memoized) 'IMG_20150730_115647_5' -> 0.0536 secs
    ├── (memoized) 'IMG_20150730_115648' -> 0.0479 secs
    ├── (memoized) 'IMG_20150730_115648_1' -> 0.0669 secs
    ├── (memoized) 'IMG_20150730_115648_2' -> 0.0549 secs
    ├── (memoized) 'IMG_20150730_115648_3' -> 0.0473 secs
    ├── (memoized) 'IMG_20150730_115648_4' -> 0.0755 secs
    ├── (memoized) 'IMG_20150730_115649' -> 0.0457 secs
    ├── (memoized) 'IMG_20150730_115649_1' -> 0.0477 secs
    ├── (memoized) 'IMG_20150730_115649_3' -> 0.0533 secs
    ├── (memoized) 'IMG_20150730_115649_4' -> 0.0585 secs
    ├── (memoized) 'IMG_20150730_115649_5' -> 0.0535 secs
    ├── (memoized) 'IMG_20150730_115650' -> 0.0601 secs
    ├── (memoized) 'IMG_20150730_115650_1' -> 0.1502 secs
    ├── (memoized) 'IMG_20150730_115650_2' -> 0.0573 secs
    ├── (memoized) 'IMG_20150730_115650_3' -> 0.0566 secs
    ├── (memoized) 'IMG_20150730_115650_4' -> 0.0473 secs
    ├── (memoized) 'IMG_20150730_115651' -> 0.0609 secs
    ├── (memoized) 'IMG_20150730_115651_1' -> 0.0549 secs
    └── (memoized) 'IMG_20150730_115651_2' -> 0.0663 secs
└── 'Restoring' -> 47.8004 secs
    ├── 'Matching' -> 4.8422 secs
    └── 'Merging' -> 42.9549 secs
        ├── 'IMG_20150730_115646_2' -> 2.5186 secs
        ├── 'IMG_20150730_115646_1' -> 2.1790 secs
        ├── 'IMG_20150730_115646' -> 4.4143 secs
        ├── 'IMG_20150730_115645_4' -> 4.8933 secs
        ├── 'IMG_20150730_115645_3' -> 4.0004 secs
        ├── 'IMG_20150730_115645_2' -> 3.6895 secs
        ├── 'IMG_20150730_115645_1' -> 4.9356 secs
        ├── 'IMG_20150730_115645' -> 5.4561 secs
        ├── 'IMG_20150730_115644_2' -> 4.6674 secs
        ├── 'IMG_20150730_115644_3' -> 2.3625 secs
        ├── 'IMG_20150730_115644' -> 1.6333 secs
        └── 'IMG_20150730_115644_1' -> 1.5454 secs
└── 'Post-processing' -> 0.6295 secs
```

**Result for set 17**

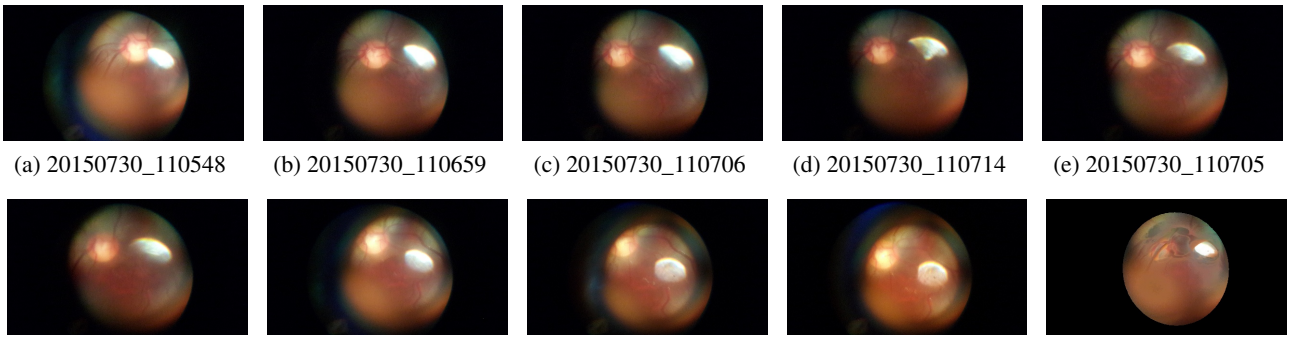
Table 23
*Profiling for set17 using command '*`imrestore ../results/set17/*.* --lens --overwrite --cachePath {temp}`*'*

```
'ImRestore init' -> 57.6812 secs
└─ 'Computing features' -> 43.7062 secs
   ├─ (processed) '20150730_110405' -> 2.1867 secs
   ├─ (processed) '20150730_110530' -> 2.2306 secs
   ├─ (processed) '20150730_110539' -> 2.6009 secs
   ├─ (processed) '20150730_110545' -> 2.1879 secs
   ├─ (processed) '20150730_110548' -> 2.2207 secs
   ├─ (processed) '20150730_110552' -> 2.1820 secs
   ├─ (processed) '20150730_110645' -> 2.3567 secs
   ├─ (processed) '20150730_110648' -> 2.4604 secs
   ├─ (processed) '20150730_110650' -> 2.3409 secs
   ├─ (processed) '20150730_110656' -> 2.5220 secs
   ├─ (processed) '20150730_110657' -> 2.9992 secs
   ├─ (processed) '20150730_110659' -> 2.8675 secs
   ├─ (processed) '20150730_110705' -> 2.9512 secs
   ├─ (processed) '20150730_110706' -> 2.6663 secs
   ├─ (processed) '20150730_110711' -> 2.9128 secs
   ├─ (processed) '20150730_110714' -> 2.7999 secs
   └─ (processed) '20150730_110717' -> 2.7908 secs
└─ 'Restoring' -> 13.5036 secs
   ├─ 'Matching' -> 0.6487 secs
   └─ 'Merging' -> 12.8529 secs
      ├─ '20150730_110659' -> 1.6110 secs
      ├─ '20150730_110706' -> 1.4996 secs
      ├─ '20150730_110714' -> 1.5324 secs
      ├─ '20150730_110705' -> 1.6003 secs
      ├─ '20150730_110711' -> 1.6473 secs
      ├─ '20150730_110545' -> 1.7304 secs
      ├─ '20150730_110530' -> 1.6097 secs
      └─ '20150730_110539' -> 1.5720 secs
└─ 'Post-processing' -> 0.4713 secs
```

```
'ImRestore init' -> 33.4125 secs
└─ 'Computing features' -> 4.6865 secs
   ├─ (memoized) '20150730_110405' -> 0.2474 secs
   ├─ (memoized) '20150730_110530' -> 0.2203 secs
   ├─ (memoized) '20150730_110539' -> 0.2126 secs
   ├─ (memoized) '20150730_110545' -> 0.4412 secs
   ├─ (memoized) '20150730_110548' -> 0.2425 secs
   ├─ (memoized) '20150730_110552' -> 0.1935 secs
   ├─ (memoized) '20150730_110645' -> 0.1071 secs
   ├─ (memoized) '20150730_110648' -> 0.2147 secs
   ├─ (memoized) '20150730_110650' -> 0.7490 secs
   ├─ (memoized) '20150730_110656' -> 0.1244 secs
   ├─ (memoized) '20150730_110657' -> 0.2560 secs
   ├─ (memoized) '20150730_110659' -> 0.3538 secs
   ├─ (memoized) '20150730_110705' -> 0.3229 secs
   ├─ (memoized) '20150730_110706' -> 0.1282 secs
   ├─ (memoized) '20150730_110711' -> 0.1760 secs
   ├─ (memoized) '20150730_110714' -> 0.1532 secs
   └─ (memoized) '20150730_110717' -> 0.3329 secs
└─ 'Restoring' -> 28.2917 secs
   ├─ 'Matching' -> 1.4724 secs
   └─ 'Merging' -> 26.8175 secs
      ├─ '20150730_110659' -> 2.5309 secs
      ├─ '20150730_110706' -> 3.8754 secs
      ├─ '20150730_110714' -> 2.7425 secs
      ├─ '20150730_110705' -> 3.6128 secs
      ├─ '20150730_110711' -> 3.8775 secs
      ├─ '20150730_110545' -> 3.6397 secs
      ├─ '20150730_110530' -> 3.5062 secs
      └─ '20150730_110539' -> 2.9516 secs
└─ 'Post-processing' -> 0.4343 secs
```

Using Code 5 in set17 *imrestore* processed 9 of the 17 images in set17, this is shown in **??**. 8 is the number of images merged in the *base image*. 8 is the number of images not used for the set17. Figure 68 presents the results for set17. Figure 69 shows the images not used in set17. Figure 68a shows the *base image*, Figure 68j shows the restored image and the remaining merged images. **??** shows the first image in set. Figure 69a shows the first image not used in the merging. Table 23 presents the profiling produced for set17. It is evidenced that the time was reduced by 2 times of the uncached session. In this way time passed from 57.68 to 33.41 seconds saving in total 24.27 seconds. This is translated in a gain of the 42.07% with cache time being 57.93% and this means that an excess in time of the 172.63% would happen if passed from cached to a not cached session the rate of used images is 0.53. the rate of merged images is 0.47. the rate of failed images is 0.47.

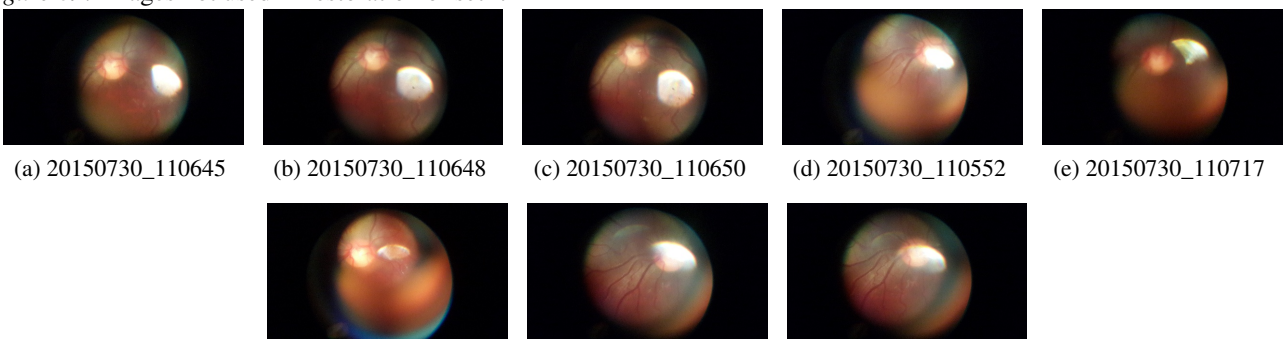
*Figure 68*. Result for set17 using command '`imrestore ../results/set17/*.* --lens --overwrite --cachePath {temp}`'



(a) 20150730_110548 (b) 20150730_110659 (c) 20150730_110706 (d) 20150730_110714 (e) 20150730_110705

(f) 20150730_110711 (g) 20150730_110545 (h) 20150730_110530 (i) 20150730_110539 (j) Restored image

*Figure 69*. Images not used in restoration of set17



(a) 20150730_110645 (b) 20150730_110648 (c) 20150730_110650 (d) 20150730_110552 (e) 20150730_110717

(f) 20150730_110405 (g) 20150730_110657 (h) 20150730_110656

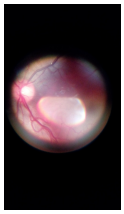
**Result for set 18**

Table 24
*Profiling for set18 using command '*`imrestore ../results/set18/*.* --lens --overwrite --cachePath {temp}`*'*

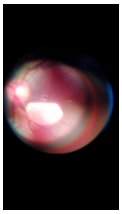
'ImRestore init' -> 40.2444 secs
- 'Computing features' -> 38.0787 secs
  - (processed) 'IMG_20150730_105626' -> 2.1840 secs
  - (processed) 'IMG_20150730_105632' -> 2.0535 secs
  - (processed) 'IMG_20150730_105655' -> 2.2025 secs
  - (processed) 'IMG_20150730_105748' -> 2.0853 secs
  - (processed) 'IMG_20150730_105753' -> 2.1469 secs
  - (processed) 'IMG_20150730_105800' -> 2.1074 secs
  - (processed) 'IMG_20150730_115251' -> 2.1841 secs
  - (processed) 'IMG_20150730_115254' -> 2.2393 secs
  - (processed) 'IMG_20150730_115301' -> 2.2283 secs
  - (processed) 'IMG_20150730_131444' -> 2.1268 secs
  - (processed) 'IMG_20150730_131454' -> 2.2785 secs
  - (processed) 'IMG_20150730_134219' -> 2.1742 secs
  - (processed) 'IMG_20150730_134225' -> 2.3129 secs
  - (processed) 'IMG_20150730_134231' -> 2.3135 secs
  - (processed) 'IMG_20150730_134235' -> 2.3683 secs
  - (processed) 'IMG_20150730_134254' -> 2.3516 secs
  - (processed) 'IMG_20150730_134300' -> 2.4463 secs
- 'Restoring' -> 1.7388 secs
  - 'Matching' -> 0.2569 secs
  - 'Merging' -> 1.4814 secs
    - 'IMG_20150730_105748' -> 1.3090 secs
- 'Post-processing' -> 0.4269 secs

'ImRestore init' -> 11.1511 secs
- 'Computing features' -> 6.1360 secs
  - (memoized) 'IMG_20150730_105626' -> 1.0331 secs
  - (memoized) 'IMG_20150730_105632' -> 0.2301 secs
  - (memoized) 'IMG_20150730_105655' -> 0.2322 secs
  - (memoized) 'IMG_20150730_105748' -> 0.1579 secs
  - (memoized) 'IMG_20150730_105753' -> 0.1915 secs
  - (memoized) 'IMG_20150730_105800' -> 0.3220 secs
  - (memoized) 'IMG_20150730_115251' -> 0.3197 secs
  - (memoized) 'IMG_20150730_115254' -> 0.5842 secs
  - (memoized) 'IMG_20150730_115301' -> 0.1764 secs
  - (memoized) 'IMG_20150730_131444' -> 0.1325 secs
  - (memoized) 'IMG_20150730_131454' -> 0.4424 secs
  - (memoized) 'IMG_20150730_134219' -> 0.2878 secs
  - (memoized) 'IMG_20150730_134225' -> 0.3203 secs
  - (memoized) 'IMG_20150730_134231' -> 0.1847 secs
  - (memoized) 'IMG_20150730_134235' -> 0.1742 secs
  - (memoized) 'IMG_20150730_134254' -> 0.1547 secs
  - (memoized) 'IMG_20150730_134300' -> 0.1961 secs
- 'Restoring' -> 4.2545 secs
  - 'Matching' -> 0.3605 secs
  - 'Merging' -> 3.8935 secs
    - 'IMG_20150730_105748' -> 3.1016 secs
- 'Post-processing' -> 0.7606 secs

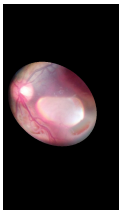
*Figure 70.* Result for set18 using command '`imrestore ../results/set18/*.* --lens --overwrite --cachePath {temp}`'



(a) IMG_20150730_105626



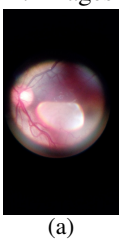
(b) IMG_20150730_105748



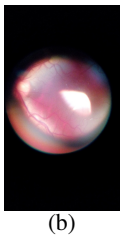
(c) Restored image

Using Code 5 in set18 *imrestore* processed 2 of the 17 images in set18, this is shown in Figure 71. 1 is the number of images merged in the *base image*. 15 is the number of images not used for the set18. Figure 70 presents the results for set18. **??** shows the images not used in set18. Figure 70a shows the *base image*, Figure 70c shows the restored image and the remaining merged images. Figure 71a shows the first image in set. **??** shows the first image not used in the merging. Table 24 presents the profiling produced for set18. It is evidenced that the time was reduced by 4 times of the uncached session. In this way time passed from 40.24 to 11.15 seconds saving in total 29.09 seconds. This is translated in a gain of the 72.29% with cache time being 27.71% and this means that an excess in time of the 360.90% would happen if passed from cached to a not cached session the rate of used images is 0.12. the rate of merged images is 0.06. the rate of failed images is 0.88.

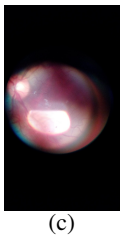
*Figure 71*. Images in set18



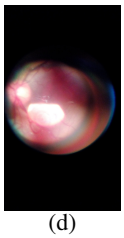
(a)
IMG_20150730_105626



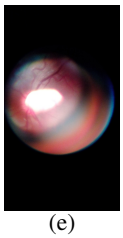
(b)
IMG_20150730_105632



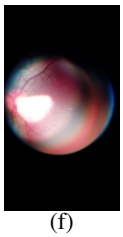
(c)
IMG_20150730_105655



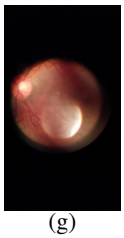
(d)
IMG_20150730_105748



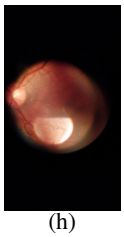
(e)
IMG_20150730_105753



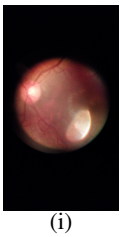
(f)
IMG_20150730_105800



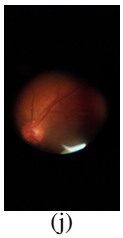
(g)
IMG_20150730_115251



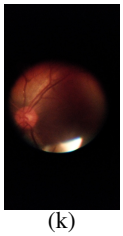
(h)
IMG_20150730_115254



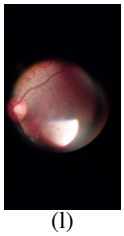
(i)
IMG_20150730_115301



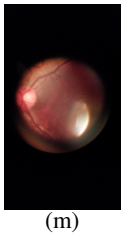
(j)
IMG_20150730_131444



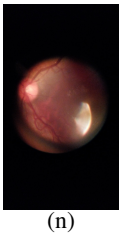
(k)
IMG_20150730_131454



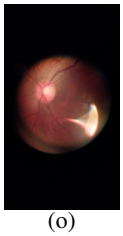
(l)
IMG_20150730_134219



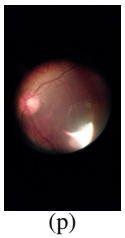
(m)
IMG_20150730_134225



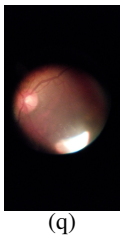
(n)
IMG_20150730_134231



(o)
IMG_20150730_134235



(p)
IMG_20150730_134254



(q)
IMG_20150730_134300

**Result for set 19**

Using Code 5 in set18 *imrestore* processed 2 of the 17 images in set18, this is shown in Figure 71. 1 is the number of images merged in the *base image*. 15 is the number of images not used for the set18. Figure 70 presents the results for set18. **??** shows the images not used in set18. Figure 70a shows the *base image*, Figure 70c shows the restored image and the remaining merged images. Figure 71a shows the first image in set. **??** shows the first image not used in the merging. Table 24 presents the profiling produced for set18. It is evidenced that the time was reduced by 4 times of the uncached session. In this way time passed from 40.24 to 11.15 seconds saving in total 29.09 seconds. This is translated in a gain of the 72.29% with cache time being 27.71% and this means that an excess in time of the 360.90% would happen if passed from cached to a not cached session the rate of used images is 0.12. the rate of merged images is 0.06. the rate of failed images is 0.88.

In[9]: print generic_paragraph(s="set19",s2=None)

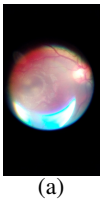
Using Code 5 in set19 *imrestore* processed 25 of the 31 images in set19, this is shown in **??**. 24 is the number of images merged in the *base image*. 6 is the number of images not used for the set19. Figure 72 presents the results for set19. Figure 73 shows the images not used in set19. Figure 72a shows the *base image*, Figure 72z shows the restored image and the remaining merged images. **??** shows the first image in set. Figure 73a shows the first image not used in the merging. Table 25 presents the profiling produced for set19. It is evidenced that the time was reduced by 1 times of the uncached session. In this way time passed from 120.48 to 92.78 seconds saving in total 27.70 seconds. This is translated in a gain of the 22.99% with cache time being 77.01% and this means that an excess in time of the 129.85% would happen if passed from cached to a not cached session the rate of used images is 0.81. the rate of merged images is 0.77. the rate of failed images is 0.19.

Table 25
*Profiling for set19 using command* '`imrestore ../results/set19/*.* --lens --overwrite --cachePath {temp}`'

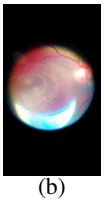
- 'ImRestore init' -> 120.4802 secs
  - 'Computing features' -> 86.0512 secs
    - (processed) 'IMG_20150730_115647_1' -> 2.8565 secs
    - (processed) 'IMG_20150730_115642' -> 2.8028 secs
    - (processed) 'IMG_20150730_115644' -> 2.6503 secs
    - (processed) 'IMG_20150730_115644_1' -> 2.5347 secs
    - (processed) 'IMG_20150730_115644_2' -> 2.6401 secs
    - (processed) 'IMG_20150730_115644_3' -> 2.5682 secs
    - (processed) 'IMG_20150730_115645_1' -> 2.6394 secs
    - (processed) 'IMG_20150730_115645_3' -> 2.6636 secs
    - (processed) 'IMG_20150730_115645_4' -> 2.6275 secs
    - (processed) 'IMG_20150730_115646' -> 2.5770 secs
    - (processed) 'IMG_20150730_115646_1' -> 2.6437 secs
    - (processed) 'IMG_20150730_115646_2' -> 2.6496 secs
    - (processed) 'IMG_20150730_115646_3' -> 2.7298 secs
    - (processed) 'IMG_20150730_115646_5' -> 2.7782 secs
    - (processed) 'IMG_20150730_115647' -> 2.8211 secs
    - (processed) 'IMG_20150730_115647_2' -> 2.7638 secs
    - (processed) 'IMG_20150730_115647_3' -> 2.6771 secs
    - (processed) 'IMG_20150730_115647_4' -> 2.7531 secs
    - (processed) 'IMG_20150730_115648' -> 2.7393 secs
    - (processed) 'IMG_20150730_115648_1' -> 2.7187 secs
    - (processed) 'IMG_20150730_115648_2' -> 2.7531 secs
    - (processed) 'IMG_20150730_115648_3' -> 2.8693 secs
    - (processed) 'IMG_20150730_115648_4' -> 2.6890 secs
    - (processed) 'IMG_20150730_115649' -> 2.9883 secs
    - (processed) 'IMG_20150730_115649_1' -> 3.1665 secs
    - (processed) 'IMG_20150730_115649_2' -> 2.7142 secs
    - (processed) 'IMG_20150730_115649_4' -> 2.7838 secs
    - (processed) 'IMG_20150730_115650' -> 2.8794 secs
    - (processed) 'IMG_20150730_115650_3' -> 2.8313 secs
    - (processed) 'IMG_20150730_115650_4' -> 2.7142 secs
    - (processed) 'IMG_20150730_115651' -> 2.8182 secs
  - 'Restoring' -> 33.9889 secs
    - 'Matching' -> 1.8534 secs
    - 'Merging' -> 32.1306 secs
      - 'IMG_20150730_115646_2' -> 1.4884 secs
      - 'IMG_20150730_115646_1' -> 1.3321 secs
      - 'IMG_20150730_115646' -> 1.2919 secs
      - 'IMG_20150730_115645_4' -> 1.1619 secs
      - 'IMG_20150730_115645_3' -> 1.2057 secs
      - 'IMG_20150730_115646_5' -> 1.2746 secs
      - 'IMG_20150730_115647_1' -> 1.2460 secs
      - 'IMG_20150730_115647' -> 1.2818 secs
      - 'IMG_20150730_115647_2' -> 1.2583 secs
      - 'IMG_20150730_115647_3' -> 1.2666 secs
      - 'IMG_20150730_115647_4' -> 1.2700 secs
      - 'IMG_20150730_115648_1' -> 1.3461 secs
      - 'IMG_20150730_115648' -> 1.2728 secs
      - 'IMG_20150730_115651' -> 1.4752 secs
      - 'IMG_20150730_115650_4' -> 1.4091 secs
      - 'IMG_20150730_115650_3' -> 1.3400 secs
      - 'IMG_20150730_115648_2' -> 1.3298 secs
      - 'IMG_20150730_115648_3' -> 1.4255 secs
      - 'IMG_20150730_115648_4' -> 1.4204 secs
      - 'IMG_20150730_115649' -> 1.3792 secs
      - 'IMG_20150730_115649_2' -> 1.4441 secs
      - 'IMG_20150730_115649_1' -> 1.4099 secs
      - 'IMG_20150730_115649_4' -> 1.3202 secs
      - 'IMG_20150730_115650' -> 1.3939 secs
  - 'Post-processing' -> 0.4401 secs

- 'ImRestore init' -> 92.7849 secs
  - 'Computing features' -> 8.0022 secs
    - (memoized) 'IMG_20150730_115647_1' -> 0.2936 secs
    - (memoized) 'IMG_20150730_115642' -> 0.2081 secs
    - (memoized) 'IMG_20150730_115644' -> 0.2033 secs
    - (memoized) 'IMG_20150730_115644_1' -> 0.2508 secs
    - (memoized) 'IMG_20150730_115644_2' -> 0.3342 secs
    - (memoized) 'IMG_20150730_115644_3' -> 0.3694 secs
    - (memoized) 'IMG_20150730_115645_1' -> 0.2020 secs
    - (memoized) 'IMG_20150730_115645_3' -> 0.1500 secs
    - (memoized) 'IMG_20150730_115645_4' -> 0.1451 secs
    - (memoized) 'IMG_20150730_115646' -> 0.1902 secs
    - (memoized) 'IMG_20150730_115646_1' -> 0.2722 secs
    - (memoized) 'IMG_20150730_115646_2' -> 0.1825 secs
    - (memoized) 'IMG_20150730_115646_3' -> 0.2998 secs
    - (memoized) 'IMG_20150730_115646_5' -> 0.2395 secs
    - (memoized) 'IMG_20150730_115647' -> 0.1999 secs
    - (memoized) 'IMG_20150730_115647_2' -> 0.2818 secs
    - (memoized) 'IMG_20150730_115647_3' -> 0.2831 secs
    - (memoized) 'IMG_20150730_115647_4' -> 0.5242 secs
    - (memoized) 'IMG_20150730_115648' -> 0.1178 secs
    - (memoized) 'IMG_20150730_115648_1' -> 0.1983 secs
    - (memoized) 'IMG_20150730_115648_2' -> 0.2691 secs
    - (memoized) 'IMG_20150730_115648_3' -> 0.1850 secs
    - (memoized) 'IMG_20150730_115648_4' -> 0.2002 secs
    - (memoized) 'IMG_20150730_115649' -> 0.1064 secs
    - (memoized) 'IMG_20150730_115649_1' -> 0.1482 secs
    - (memoized) 'IMG_20150730_115649_2' -> 0.3206 secs
    - (memoized) 'IMG_20150730_115649_4' -> 0.2173 secs
    - (memoized) 'IMG_20150730_115650' -> 0.1346 secs
    - (memoized) 'IMG_20150730_115650_3' -> 0.2424 secs
    - (memoized) 'IMG_20150730_115650_4' -> 0.2487 secs
    - (memoized) 'IMG_20150730_115651' -> 0.3232 secs
  - 'Restoring' -> 83.8567 secs
    - 'Matching' -> 4.6324 secs
    - 'Merging' -> 79.2189 secs
      - 'IMG_20150730_115646_2' -> 2.5882 secs
      - 'IMG_20150730_115646_1' -> 3.2101 secs
      - 'IMG_20150730_115646' -> 2.5621 secs
      - 'IMG_20150730_115645_4' -> 3.2147 secs
      - 'IMG_20150730_115645_3' -> 3.4402 secs
      - 'IMG_20150730_115646_5' -> 4.1303 secs
      - 'IMG_20150730_115647_1' -> 3.2987 secs
      - 'IMG_20150730_115647' -> 3.0160 secs
      - 'IMG_20150730_115647_2' -> 2.3737 secs
      - 'IMG_20150730_115647_3' -> 2.4315 secs
      - 'IMG_20150730_115647_4' -> 3.5084 secs
      - 'IMG_20150730_115648_1' -> 3.6602 secs
      - 'IMG_20150730_115648' -> 3.4772 secs
      - 'IMG_20150730_115651' -> 4.4735 secs
      - 'IMG_20150730_115650_4' -> 3.0858 secs
      - 'IMG_20150730_115650_3' -> 4.2219 secs
      - 'IMG_20150730_115648_2' -> 3.6904 secs
      - 'IMG_20150730_115648_3' -> 3.4017 secs
      - 'IMG_20150730_115648_4' -> 3.3823 secs
      - 'IMG_20150730_115649' -> 3.4723 secs
      - 'IMG_20150730_115649_2' -> 3.3438 secs
      - 'IMG_20150730_115649_1' -> 3.0954 secs
      - 'IMG_20150730_115649_4' -> 3.3002 secs
      - 'IMG_20150730_115650' -> 2.6352 secs
  - 'Post-processing' -> 0.9259 secs

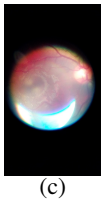
*Figure 72.* Result for set19 using command '`imrestore ../results/set19/*.* --lens --overwrite --cachePath {temp}`'



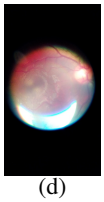
(a) IMG_20150730_115646_3



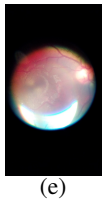
(b) IMG_20150730_115646_2



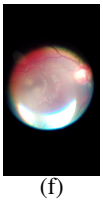
(c) IMG_20150730_115646_1



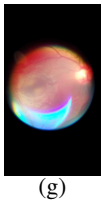
(d) IMG_20150730_115646



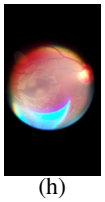
(e) IMG_20150730_115645_4



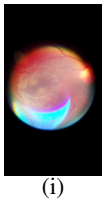
(f) IMG_20150730_115645_3



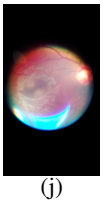
(g) IMG_20150730_115646_5



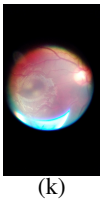
(h) IMG_20150730_115647_1



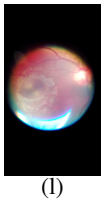
(i) IMG_20150730_115647



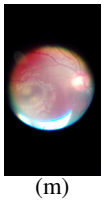
(j) IMG_20150730_115647_2



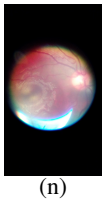
(k) IMG_20150730_115647_3



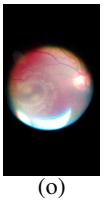
(l) IMG_20150730_115647_4



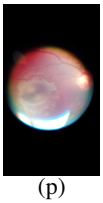
(m) IMG_20150730_115648_1



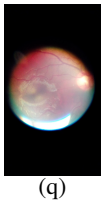
(n) IMG_20150730_115648



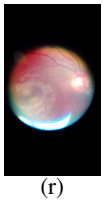
(o) IMG_20150730_115651



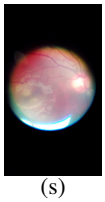
(p) IMG_20150730_115650_4



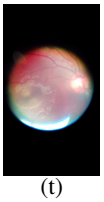
(q) IMG_20150730_115650_3



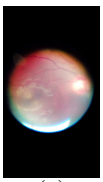
(r) IMG_20150730_115648_2



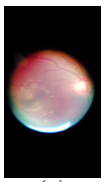
(s) IMG_20150730_115648_3



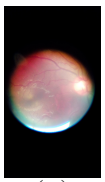
(t) IMG_20150730_115648_4



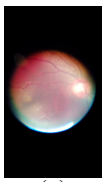
(u) IMG_20150730_115649



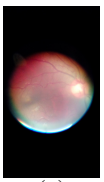
(v) IMG_20150730_115649_2



(w) IMG_20150730_115649_1



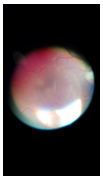
(x) IMG_20150730_115649_4



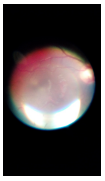
(y) IMG_20150730_115650

(z) Restored image

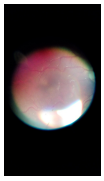
*Figure 73*. Images not used in restoration of set19



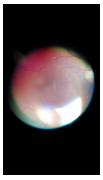
(a) IMG_20150730_115644_2



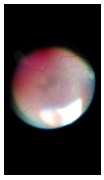
(b) IMG_20150730_115645_1



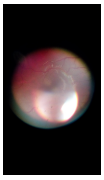
(c) IMG_20150730_115644_1



(d) IMG_20150730_115644_3



(e) IMG_20150730_115644



(f) IMG_20150730_115642

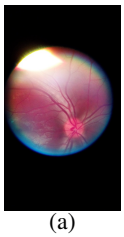
**Result for set 20**

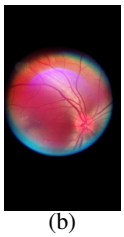
Table 26
*Profiling for set20 using command 'imrestore ../results/set20/*.* --lens --overwrite --cachePath {temp}'*

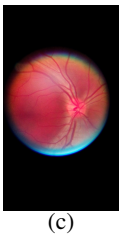
'ImRestore init' -> 54.0407 secs
- 'Computing features' -> 50.1011 secs
  - (processed) '20150730_132852' -> 2.4145 secs
  - (processed) '20150730_132857' -> 2.0729 secs
  - (processed) '20150730_132900' -> 2.3489 secs
  - (processed) '20150730_132904' -> 2.0129 secs
  - (processed) '20150730_132917' -> 1.7671 secs
  - (processed) '20150730_132924' -> 1.9242 secs
  - (processed) '20150730_132929' -> 1.8181 secs
  - (processed) '20150730_132932' -> 1.8655 secs
  - (processed) '20150730_132936' -> 1.7875 secs
  - (processed) 'IMG_20150730_122502' -> 1.7832 secs
  - (processed) 'IMG_20150730_122504' -> 1.8384 secs
  - (processed) 'IMG_20150730_122506' -> 1.8803 secs
  - (processed) 'IMG_20150730_122520' -> 1.7905 secs
  - (processed) 'IMG_20150730_130654' -> 2.0385 secs
  - (processed) 'IMG_20150730_130659' -> 2.1586 secs
  - (processed) 'IMG_20150730_130706' -> 2.3120 secs
  - (processed) 'IMG_20150730_130713' -> 2.2099 secs
  - (processed) 'IMG_20150730_130719' -> 2.0846 secs
  - (processed) 'IMG_20150730_130723' -> 2.4502 secs
  - (processed) 'IMG_20150730_130729' -> 2.3709 secs
  - (processed) 'IMG_20150730_130756' -> 2.2615 secs
  - (processed) 'IMG_20150730_130759' -> 2.0909 secs
  - (processed) 'IMG_20150730_130803' -> 2.0534 secs
  - (processed) 'IMG_20150730_130809' -> 1.8940 secs
- 'Restoring' -> 3.6517 secs
  - 'Matching' -> 0.4430 secs
  - 'Merging' -> 3.2080 secs
    - 'IMG_20150730_130659' -> 0.9521 secs
    - 'IMG_20150730_130723' -> 0.9611 secs
    - 'IMG_20150730_130713' -> 1.1127 secs
- 'Post-processing' -> 0.2879 secs

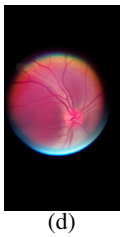
'ImRestore init' -> 26.4929 secs
- 'Computing features' -> 9.7973 secs
  - (memoized) '20150730_132852' -> 0.2608 secs
  - (memoized) '20150730_132857' -> 0.1316 secs
  - (memoized) '20150730_132900' -> 0.3009 secs
  - (memoized) '20150730_132904' -> 0.2884 secs
  - (memoized) '20150730_132917' -> 0.0986 secs
  - (memoized) '20150730_132924' -> 0.3143 secs
  - (memoized) '20150730_132929' -> 0.5799 secs
  - (memoized) '20150730_132932' -> 0.3983 secs
  - (memoized) '20150730_132936' -> 0.3612 secs
  - (memoized) 'IMG_20150730_122502' -> 0.4372 secs
  - (memoized) 'IMG_20150730_122504' -> 0.2810 secs
  - (memoized) 'IMG_20150730_122506' -> 0.2850 secs
  - (memoized) 'IMG_20150730_122520' -> 0.3053 secs
  - (memoized) 'IMG_20150730_130654' -> 0.4560 secs
  - (memoized) 'IMG_20150730_130659' -> 0.4691 secs
  - (memoized) 'IMG_20150730_130706' -> 0.3788 secs
  - (memoized) 'IMG_20150730_130713' -> 0.5561 secs
  - (memoized) 'IMG_20150730_130719' -> 0.4428 secs
  - (memoized) 'IMG_20150730_130723' -> 1.1865 secs
  - (memoized) 'IMG_20150730_130729' -> 0.7323 secs
  - (memoized) 'IMG_20150730_130756' -> 0.2676 secs
  - (memoized) 'IMG_20150730_130759' -> 0.1314 secs
  - (memoized) 'IMG_20150730_130803' -> 0.2384 secs
  - (memoized) 'IMG_20150730_130809' -> 0.2800 secs
- 'Restoring' -> 14.8907 secs
  - 'Matching' -> 1.6175 secs
  - 'Merging' -> 13.2723 secs
    - 'IMG_20150730_130659' -> 4.0912 secs
    - 'IMG_20150730_130723' -> 4.0746 secs
    - 'IMG_20150730_130713' -> 4.3786 secs
- 'Post-processing' -> 1.8048 secs

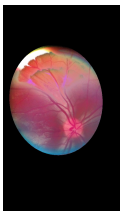
*Figure 74*. Result for set20 using command 'imrestore ../results/set20/*.* --lens --overwrite --cachePath {temp}'


(a) IMG_20150730_130654


(b) IMG_20150730_130659


(c) IMG_20150730_130723

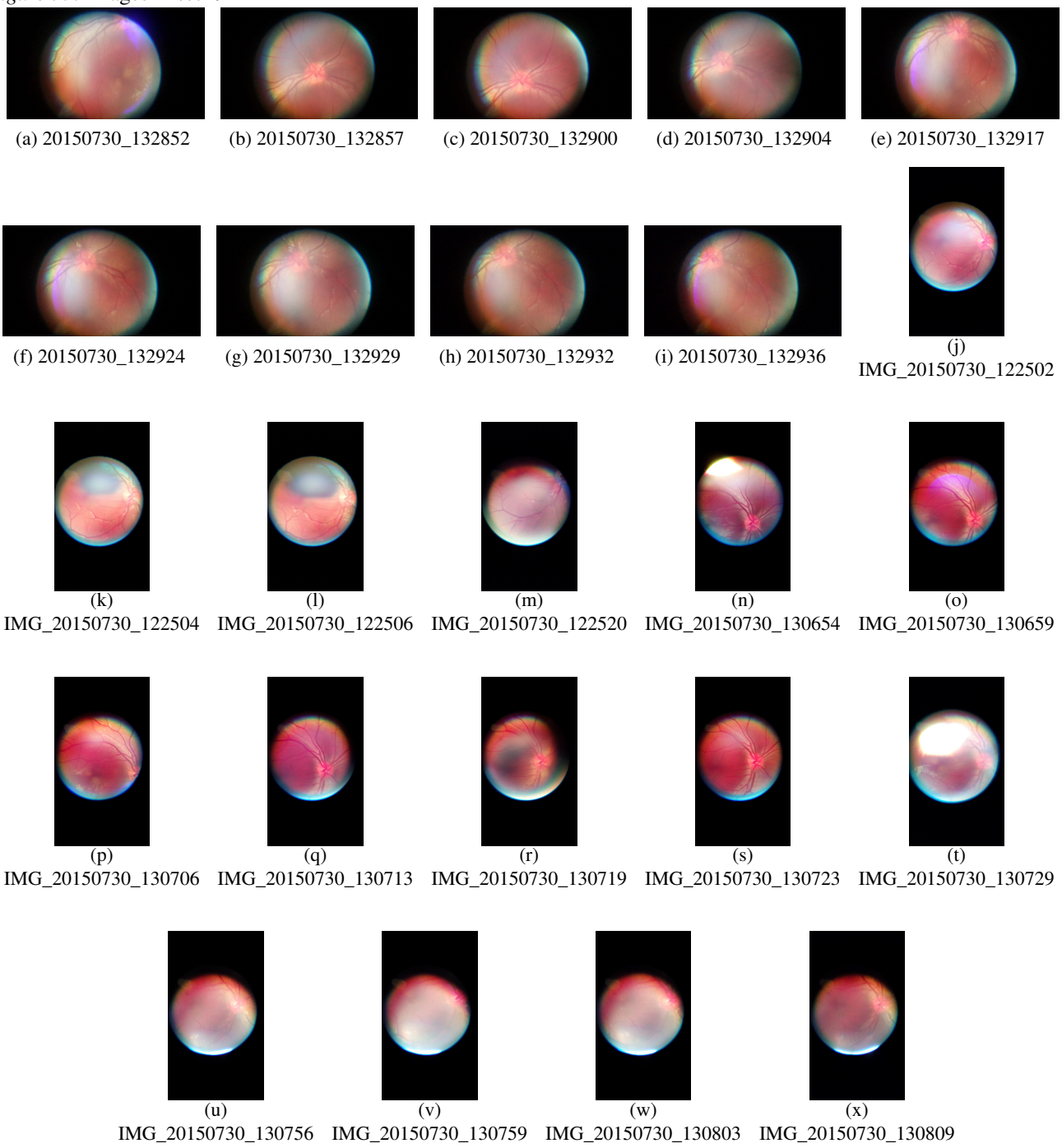

(d) IMG_20150730_130713


(e) Restored image

Using Code 5 in set20 *imrestore* processed 4 of the 24 images in set20, this is shown in Figure 75. 3 is the number of images merged in the *base image*. 20 is the number of images not used for the set20. Figure 74 presents the results for set20. **??** shows the images not used in

set20. Figure 74a shows the *base image*, Figure 74e shows the restored image and the remaining merged images. Figure 75a shows the first image in set. **??** shows the first image not used in the merging. Table 26 presents the profiling produced for set20. It is evidenced that the time was reduced by 2 times of the uncached session. In this way time passed from 54.04 to 26.49 seconds saving in total 27.55 seconds. This is translated in a gain of the 50.98% with cache time being 49.02% and this means that an excess in time of the 203.98% would happen if passed from cached to a not cached session the rate of used images is 0.17. the rate of merged images is 0.12. the rate of failed images is 0.83.

*Figure 75*. Images in set20



(a) 20150730_132852 (b) 20150730_132857 (c) 20150730_132900 (d) 20150730_132904 (e) 20150730_132917

(f) 20150730_132924 (g) 20150730_132929 (h) 20150730_132932 (i) 20150730_132936 (j) IMG_20150730_122502

(k) IMG_20150730_122504 (l) IMG_20150730_122506 (m) IMG_20150730_122520 (n) IMG_20150730_130654 (o) IMG_20150730_130659

(p) IMG_20150730_130706 (q) IMG_20150730_130713 (r) IMG_20150730_130719 (s) IMG_20150730_130723 (t) IMG_20150730_130729

(u) IMG_20150730_130756 (v) IMG_20150730_130759 (w) IMG_20150730_130803 (x) IMG_20150730_130809

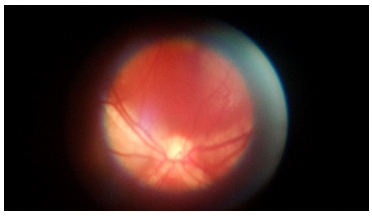
**Result for set 21**

Table 27
*Profiling for set21 using command 'imrestore ../results/set21/*.* --lens --overwrite --cachePath {temp}'*

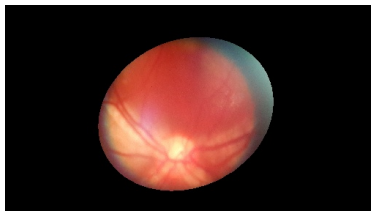
'ImRestore init' -> 29.1726 secs
- 'Computing features' -> 28.4758 secs
  - (processed) '20150730_131816' -> 2.3519 secs
  - (processed) '20150730_131823' -> 2.3904 secs
  - (processed) '20150730_131827' -> 2.0813 secs
  - (processed) '20150730_131834' -> 2.9239 secs
  - (processed) '20150730_131836' -> 2.7180 secs
  - (processed) '20150730_131839' -> 2.9072 secs
  - (processed) '20150730_131844' -> 3.6443 secs
  - (processed) '20150730_131913' -> 3.1464 secs
  - (processed) '20150730_131915' -> 2.8072 secs
  - (processed) '20150730_131916' -> 2.9230 secs
- 'Restoring' -> 0.0880 secs
  - 'Matching' -> 0.0340 secs
  - 'Merging' -> 0.0535 secs
- 'Post-processing' -> 0.6087 secs

'ImRestore init' -> 2.8818 secs
- 'Computing features' -> 1.7148 secs
  - (memoized) '20150730_131816' -> 0.4360 secs
  - (memoized) '20150730_131823' -> 0.2340 secs
  - (memoized) '20150730_131827' -> 0.3120 secs
  - (memoized) '20150730_131834' -> 0.1405 secs
  - (memoized) '20150730_131836' -> 0.1112 secs
  - (memoized) '20150730_131839' -> 0.1017 secs
  - (memoized) '20150730_131844' -> 0.0799 secs
  - (memoized) '20150730_131913' -> 0.0542 secs
  - (memoized) '20150730_131915' -> 0.0737 secs
  - (memoized) '20150730_131916' -> 0.0703 secs
- 'Restoring' -> 0.2236 secs
  - 'Matching' -> 0.0542 secs
  - 'Merging' -> 0.1691 secs
- 'Post-processing' -> 0.9434 secs

*Figure 76*. Result for set21 using command 'imrestore ../results/set21/*.* --lens --overwrite --cachePath {temp}'

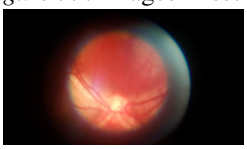


(a) 20150730_131816

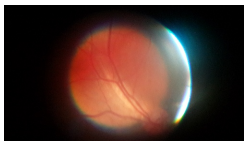


(b) Restored image

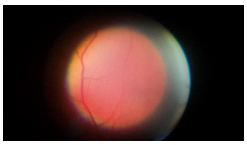
*Figure 77*. Images in set21



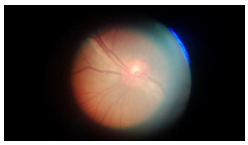
(a) 20150730_131816



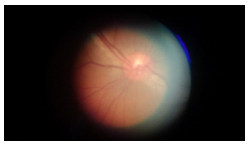
(b) 20150730_131823



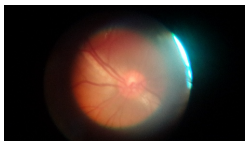
(c) 20150730_131827



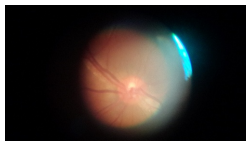
(d) 20150730_131834



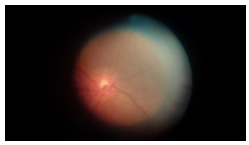
(e) 20150730_131836



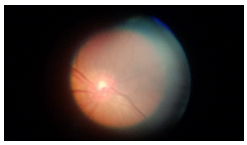
(f) 20150730_131839



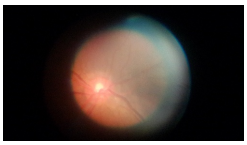
(g) 20150730_131844



(h) 20150730_131913



(i) 20150730_131915



(j) 20150730_131916

Using Code 5 in set21 *imrestore* processed 1 of the 10 images in set21, this is shown in Figure 77. 0 is the number of images merged in the *base image*. 9 is the number of images not used for the set21. Figure 76 presents the results for set21. **??** shows the images not used in set21. Figure 76a shows the *base image*, Figure 76b shows the restored image and the remaining merged image. Figure 77a shows the first image in set. **??** shows the first image not used in the merging. Table 27 presents the profiling produced for set21. It is evidenced that the

time was reduced by 10 times of the uncached session. In this way time passed from 29.17 to 2.88 seconds saving in total 26.29 seconds. This is translated in a gain of the 90.12% with cache time being 9.88% and this means that an excess in time of the 1012.29% would happen if passed from cached to a not cached session the rate of used images is 0.10. the rate of merged images is 0.00. the rate of failed images is 0.90.

**Result for set 22**

Table 28
*Profiling for set22 using command 'imrestore ../results/set22/*.* --lens --overwrite --cachePath {temp}'*

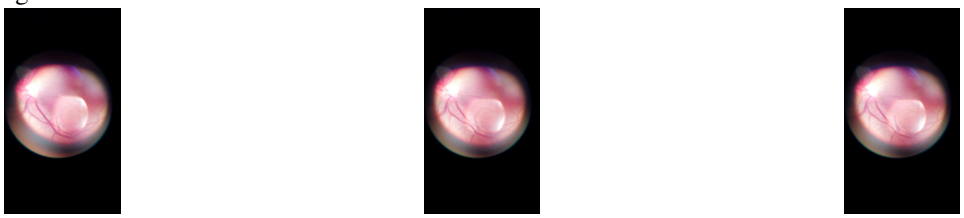
'ImRestore init' -> 21.4743 secs
- 'Computing features' -> 17.9448 secs
  - (processed) 'IMG_20150730_121750' -> 2.6784 secs
  - (processed) 'IMG_20150730_121756' -> 2.8147 secs
  - (processed) 'IMG_20150730_121801' -> 2.9608 secs
  - (processed) 'IMG_20150730_121804' -> 2.9499 secs
  - (processed) 'IMG_20150730_121804_1' -> 3.2179 secs
  - (processed) 'IMG_20150730_121804_2' -> 3.0732 secs
- 'Restoring' -> 3.1301 secs
  - 'Matching' -> 0.2217 secs
  - 'Merging' -> 2.9078 secs
    - 'IMG_20150730_121801' -> 1.4115 secs
    - 'IMG_20150730_121750' -> 1.4492 secs
- 'Post-processing' -> 0.3994 secs

'ImRestore init' -> 12.1105 secs
- 'Computing features' -> 1.9042 secs
  - (memoized) 'IMG_20150730_121750' -> 0.2615 secs
  - (memoized) 'IMG_20150730_121756' -> 0.3144 secs
  - (memoized) 'IMG_20150730_121801' -> 0.3144 secs
  - (memoized) 'IMG_20150730_121804' -> 0.2827 secs
  - (memoized) 'IMG_20150730_121804_1' -> 0.3336 secs
  - (memoized) 'IMG_20150730_121804_2' -> 0.2384 secs
- 'Restoring' -> 8.9032 secs
  - 'Matching' -> 0.4129 secs
  - 'Merging' -> 8.4896 secs
    - 'IMG_20150730_121801' -> 4.1347 secs
    - 'IMG_20150730_121750' -> 4.1933 secs
- 'Post-processing' -> 1.3031 secs

*Figure 78*. Result for set22 using command 'imrestore ../results/set22/*.* --lens --overwrite --cachePath {temp}'

(a) IMG_20150730_121756 (b) IMG_20150730_121801 (c) IMG_20150730_121750 (d) Restored image

*Figure 79*. Images not used in restoration of set22



(a) IMG_20150730_121804_2 (b) IMG_20150730_121804 (c) IMG_20150730_121804_1

Using Code 5 in set22 *imrestore* processed 3 of the 6 images in set22, this is shown in **??**. 2 is the number of images merged in the *base image*. 3 is the number of images not used for the set22. Figure 78 presents the results for set22. Figure 79 shows the images not used in set22. Figure 78a shows the *base image*, Figure 78d shows the restored image and the remaining merged images. **??** shows the first image in set. Figure 79a shows the first image not used in the merging. Table 28 presents the profiling produced for set22. It is evidenced that the time was reduced by 2 times of the uncached session. In this way time passed from 21.47 to 12.11 seconds saving in total 9.36 seconds. This is translated in a gain of the 43.60% with cache time being 56.40% and this means that an excess in time of the 177.32% would happen if passed from cached to a not cached session the rate of used images is 0.50. the rate of merged images is 0.33. the rate of failed images is 0.50.

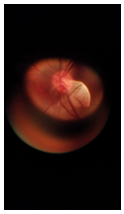
**Result for set 23**

Table 29
*Profiling for set23 using command '`imrestore ../results/set23/*.* --lens --overwrite --cachePath {temp}`'*

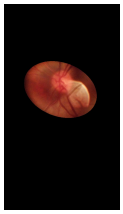
'ImRestore init' -> 27.8534 secs
- 'Computing features' -> 27.2165 secs
  - (processed) 'IMG_20150730_122140' -> 3.2541 secs
  - (processed) 'IMG_20150730_122154' -> 3.0911 secs
  - (processed) 'IMG_20150730_122200' -> 3.0967 secs
  - (processed) 'IMG_20150730_122216' -> 2.9525 secs
  - (processed) 'IMG_20150730_122225' -> 2.8391 secs
  - (processed) 'IMG_20150730_122227' -> 2.8896 secs
  - (processed) 'IMG_20150730_122229' -> 2.8071 secs
  - (processed) 'IMG_20150730_122235' -> 2.8947 secs
  - (processed) 'IMG_20150730_122237' -> 2.9100 secs
- 'Restoring' -> 0.1615 secs
  - 'Matching' -> 0.0795 secs
  - 'Merging' -> 0.0818 secs
- 'Post-processing' -> 0.4754 secs

'ImRestore init' -> 4.0939 secs
- 'Computing features' -> 2.7341 secs
  - (memoized) 'IMG_20150730_122140' -> 0.2125 secs
  - (memoized) 'IMG_20150730_122154' -> 0.2942 secs
  - (memoized) 'IMG_20150730_122200' -> 0.5929 secs
  - (memoized) 'IMG_20150730_122216' -> 0.3233 secs
  - (memoized) 'IMG_20150730_122225' -> 0.3009 secs
  - (memoized) 'IMG_20150730_122227' -> 0.3483 secs
  - (memoized) 'IMG_20150730_122229' -> 0.2184 secs
  - (memoized) 'IMG_20150730_122235' -> 0.1439 secs
  - (memoized) 'IMG_20150730_122237' -> 0.1124 secs
- 'Restoring' -> 0.4669 secs
  - 'Matching' -> 0.1127 secs
  - 'Merging' -> 0.3540 secs
- 'Post-processing' -> 0.8929 secs

*Figure 80.* Result for set23 using command '`imrestore ../results/set23/*.* --lens --overwrite --cachePath {temp}`'



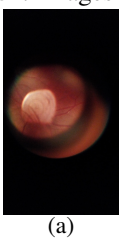
(a) IMG_20150730_122200



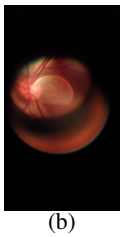
(b) Restored image

Using Code 5 in set23 *imrestore* processed 1 of the 9 images in set23, this is shown in Figure 81. 0 is the number of images merged in the *base image*. 8 is the number of images not used for the set23. Figure 80 presents the results for set23. **??** shows the images not used in set23. Figure 80a shows the *base image*, Figure 80b shows the restored image and the remaining merged image. Figure 81a shows the first image in set. **??** shows the first image not used in the merging. Table 29 presents the profiling produced for set23. It is evidenced that the time was reduced by 7 times of the uncached session. In this way time passed from 27.85 to 4.09 seconds saving in total 23.76 seconds. This is translated in a gain of the 85.30% with cache time being 14.70% and this means that an excess in time of the 680.36% would happen if passed from cached to a not cached session the rate of used images is 0.11. the rate of merged images is 0.00. the rate of failed images is 0.89.

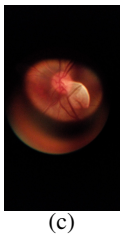
*Figure 81*. Images in set23



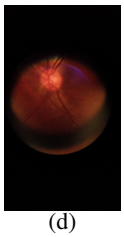
(a)
IMG_20150730_122140



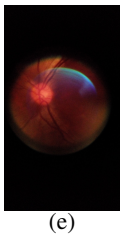
(b)
IMG_20150730_122154



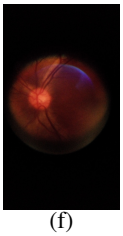
(c)
IMG_20150730_122200



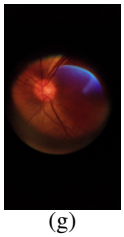
(d)
IMG_20150730_122216



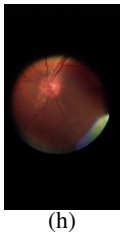
(e)
IMG_20150730_122225



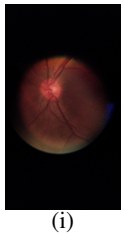
(f)
IMG_20150730_122227



(g)
IMG_20150730_122229



(h)
IMG_20150730_122235



(i)
IMG_20150730_122237

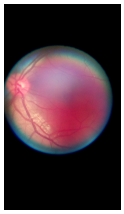
**Result for set 24**

Table 30
*Profiling for set24 using command ‘*`imrestore ../results/set24/*.* --lens --overwrite --cachePath {temp}`*’*

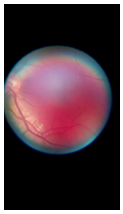
```
'ImRestore init' -> 17.4084 secs
└─ 'Computing features' -> 13.7685 secs
   ├─ (processed) 'IMG_20150730_125212' -> 3.6106 secs
   ├─ (processed) 'IMG_20150730_125216' -> 3.4603 secs
   ├─ (processed) 'IMG_20150730_125222' -> 3.3301 secs
   └─ (processed) 'IMG_20150730_125228' -> 3.1819 secs
└─ 'Restoring' -> 3.2164 secs
   ├─ 'Matching' -> 0.0737 secs
   └─ 'Merging' -> 3.1420 secs
      ├─ 'IMG_20150730_125212' -> 1.5209 secs
      └─ 'IMG_20150730_125222' -> 1.5719 secs
└─ 'Post-processing' -> 0.4235 secs
```

```
'ImRestore init' -> 6.2216 secs
└─ 'Computing features' -> 1.4231 secs
   ├─ (memoized) 'IMG_20150730_125212' -> 0.3641 secs
   ├─ (memoized) 'IMG_20150730_125216' -> 0.4247 secs
   ├─ (memoized) 'IMG_20150730_125222' -> 0.3686 secs
   └─ (memoized) 'IMG_20150730_125228' -> 0.1722 secs
└─ 'Restoring' -> 3.7034 secs
   ├─ 'Matching' -> 0.1339 secs
   └─ 'Merging' -> 3.5691 secs
      └─ 'IMG_20150730_125212' -> 3.4510 secs
└─ 'Post-processing' -> 1.0951 secs
```

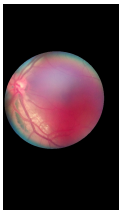
*Figure 82*. Result for set24 using command ‘`imrestore ../results/set24/*.* --lens --overwrite --cachePath {temp}`’



(a) IMG_20150730_125216

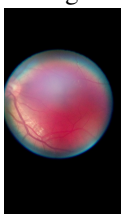


(b) IMG_20150730_125212

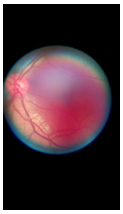


(c) Restored image

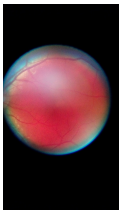
*Figure 83*. Images in set24



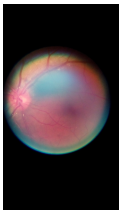
(a) IMG_20150730_125212



(b) IMG_20150730_125216



(c) IMG_20150730_125222



(d) IMG_20150730_125228

Using Code 5 in set24 *imrestore* processed 2 of the 4 images in set24, this is shown in Figure 83. 1 is the number of images merged in the *base image*. 2 is the number of images not used for the set24. Figure 82 presents the results for set24. **??** shows the images not used in set24. Figure 82a shows the *base image*, Figure 82c shows the restored image and the remaining merged images. Figure 83a shows the first image in set. **??** shows the first image not used in the merging. Table 30 presents the profiling produced for set24. It is evidenced that the time was reduced by 3 times of the uncached session. In this way time passed from 17.41 to 6.22 seconds saving in total 11.19 seconds. This is translated in a gain of the 64.26% with cache time being 35.74% and this means that an excess in time of the 279.81% would happen if passed from cached to a not cached session the rate of used images is 0.50. the rate of merged images is 0.25. the rate of failed images is 0.50.

**Result for set 25**

Table 31
*Profiling for set25 using command* '`imrestore ../results/set25/*.* --lens --overwrite --cachePath {temp}`'

'ImRestore init' -> 30.7527 secs
- 'Computing features' -> 25.3100 secs
  - (processed) 'IMG_20150730_125339' -> 3.3094 secs
  - (processed) 'IMG_20150730_125344' -> 3.1779 secs
  - (processed) 'IMG_20150730_125349' -> 2.9756 secs
  - (processed) 'IMG_20150730_125354' -> 2.9353 secs
  - (processed) 'IMG_20150730_125358' -> 2.9747 secs
  - (processed) 'IMG_20150730_125402' -> 3.1440 secs
  - (processed) 'IMG_20150730_125408' -> 3.2213 secs
  - (processed) 'IMG_20150730_125411' -> 3.0237 secs
- 'Restoring' -> 5.0108 secs
  - 'Matching' -> 0.1564 secs
  - 'Merging' -> 4.8534 secs
    - 'IMG_20150730_125354' -> 1.6178 secs
    - 'IMG_20150730_125358' -> 1.6166 secs
    - 'IMG_20150730_125344' -> 1.5631 secs
- 'Post-processing' -> 0.4318 secs

'ImRestore init' -> 11.2879 secs
- 'Computing features' -> 3.1808 secs
  - (memoized) 'IMG_20150730_125339' -> 0.8683 secs
  - (memoized) 'IMG_20150730_125344' -> 0.1919 secs
  - (memoized) 'IMG_20150730_125349' -> 0.1373 secs
  - (memoized) 'IMG_20150730_125354' -> 0.1748 secs
  - (memoized) 'IMG_20150730_125358' -> 0.1902 secs
  - (memoized) 'IMG_20150730_125402' -> 0.6316 secs
  - (memoized) 'IMG_20150730_125408' -> 0.5105 secs
  - (memoized) 'IMG_20150730_125411' -> 0.1528 secs
- 'Restoring' -> 7.3376 secs
  - 'Matching' -> 0.2048 secs
  - 'Merging' -> 7.1319 secs
    - 'IMG_20150730_125354' -> 1.9819 secs
    - 'IMG_20150730_125358' -> 2.4053 secs
    - 'IMG_20150730_125344' -> 2.6097 secs
- 'Post-processing' -> 0.7696 secs

*Figure 84*. Result for set25 using command '`imrestore ../results/set25/*.* --lens --overwrite --cachePath {temp}`'

(a) IMG_20150730_125349 (b) IMG_20150730_125354 (c) IMG_20150730_125358 (d) IMG_20150730_125344 (e) Restored image

*Figure 85*. Images not used in restoration of set25

(a) IMG_20150730_125339 (b) IMG_20150730_125411 (c) IMG_20150730_125402 (d) IMG_20150730_125408

Using Code 5 in set25 *imrestore* processed 4 of the 8 images in set25, this is shown in **??**. 3 is the number of images merged in the *base image*. 4 is the number of images not used for the set25. Figure 84 presents the results for set25. Figure 85 shows the images not used in set25. Figure 84a shows the *base image*, Figure 84e shows the restored image and the remaining merged images. **??** shows the first image in set. Figure 85a shows the first image not used in the merging. Table 31 presents the profiling produced for set25. It is evidenced that the time was reduced by 3 times of the uncached session. In this way time passed from 30.75 to 11.29 seconds saving in total 19.46 seconds.

This is translated in a gain of the 63.29% with cache time being 36.71% and this means that an excess in time of the 272.44% would happen if passed from cached to a not cached session the rate of used images is 0.50. the rate of merged images is 0.38. the rate of failed images is 0.50.

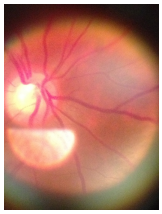
**Result for set 26**

Table 32
*Profiling for set26 using command 'imrestore ../results/set26/*.* --lens --overwrite --cachePath {temp}'*

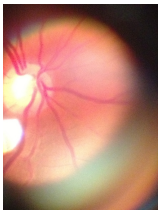
'ImRestore init' -> 22.7909 secs
- 'Computing features' -> 19.3321 secs
  - (processed) 'im1_1' -> 3.4209 secs
  - (processed) 'im1_2' -> 3.2761 secs
  - (processed) 'IMG_0243' -> 3.0074 secs
  - (processed) 'IMG_0244' -> 3.1421 secs
  - (processed) 'IMG_0247' -> 3.1072 secs
  - (processed) 'IMG_0248' -> 3.0257 secs
- 'Restoring' -> 3.0232 secs
  - 'Matching' -> 0.1249 secs
  - 'Merging' -> 2.8975 secs
    - 'im1_2' -> 1.3960 secs
    - 'IMG_0244' -> 1.4704 secs
- 'Post-processing' -> 0.4356 secs

'ImRestore init' -> 1.4906 secs
- 'Computing features' -> 0.2765 secs
  - (memoized) 'im1_1' -> 0.0688 secs
  - (memoized) 'im1_2' -> 0.0572 secs
  - (memoized) 'IMG_0243' -> 0.0239 secs
  - (memoized) 'IMG_0244' -> 0.0311 secs
  - (memoized) 'IMG_0247' -> 0.0305 secs
  - (memoized) 'IMG_0248' -> 0.0272 secs
- 'Restoring' -> 0.9699 secs
  - 'Matching' -> 0.0620 secs
  - 'Merging' -> 0.9076 secs
    - 'im1_2' -> 0.8774 secs
- 'Post-processing' -> 0.2441 secs

*Figure 86*. Result for set26 using command 'imrestore ../results/set26/*.* --lens --overwrite --cachePath {temp}'



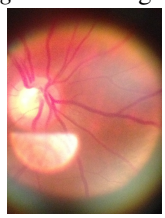
(a) im1_1



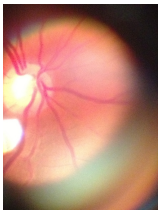
(b) im1_2

(c) Restored image

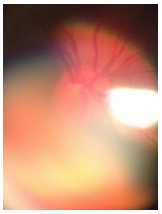
*Figure 87*. Images in set26



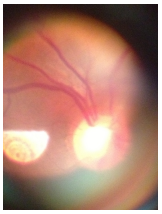
(a) im1_1



(b) im1_2

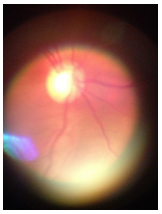


(c) IMG_0243



(d) IMG_0244

(e) IMG_0247



(f) IMG_0248

Using Code 5 in set26 *imrestore* processed 2 of the 6 images in set26, this is shown in Figure 87. 1 is the number of images merged in the *base image*. 4 is the number of images not used for the set26. Figure 86 presents the results for set26. **??** shows the images not used in set26. Figure 86a shows the *base image*, Figure 86c shows the restored image and the remaining merged images. Figure 87a shows the first image in set. **??** shows the first image not used in the merging. Table 32 presents the profiling produced for set26. It is evidenced that the time was reduced by 15 times of the uncached session. In this way time passed from 22.79 to 1.49 seconds saving in total 21.30 seconds. This is translated in a gain of the 93.46% with cache time being 6.54% and this means that an excess in time of the 1529.02% would happen if passed from cached to a not cached session the rate of used images is 0.33. the rate of merged images is 0.17. the rate of failed images is 0.67.

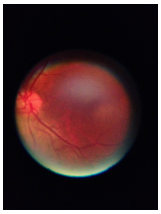
**Result for set 27**

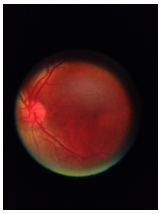
Table 33
*Profiling for set27 using command* '`imrestore ../results/set27/*.* --lens --overwrite --cachePath {temp}`'

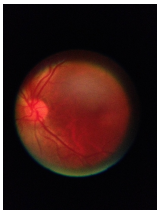
'ImRestore init' -> 13.4935 secs
- 'Computing features' -> 9.7324 secs
  - (processed) 'im5_1' -> 3.4046 secs
  - (processed) 'im5_2' -> 3.1560 secs
  - (processed) 'im5_3' -> 3.0355 secs
- 'Restoring' -> 3.2621 secs
  - 'Matching' -> 0.0385 secs
  - 'Merging' -> 3.2232 secs
    - 'im5_1' -> 1.6538 secs
    - 'im5_2' -> 1.5693 secs
- 'Post-processing' -> 0.4991 secs

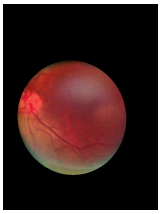
'ImRestore init' -> 8.8181 secs
- 'Computing features' -> 0.8755 secs
  - (memoized) 'im5_1' -> 0.3627 secs
  - (memoized) 'im5_2' -> 0.1645 secs
  - (memoized) 'im5_3' -> 0.2879 secs
- 'Restoring' -> 7.0875 secs
  - 'Matching' -> 0.0842 secs
  - 'Merging' -> 6.9966 secs
    - 'im5_1' -> 3.6402 secs
    - 'im5_2' -> 3.3563 secs
- 'Post-processing' -> 0.8551 secs

*Figure 88*. Result for set27 using command '`imrestore ../results/set27/*.* --lens --overwrite --cachePath {temp}`'


(a) im5_3


(b) im5_1


(c) im5_2


(d) Restored image

Using Code 5 in set27 *imrestore* processed 3 of the 3 images in set27, this is shown in **??**. 2 is the number of images merged in the *base image*. 0 is the number of images not used for the set27. Figure 88 presents the results for set27. **??** shows the images not used in set27. Figure 88a shows the *base image*, Figure 88d shows the restored image and the remaining merged images. **??** shows the first image in set. **??** shows the first image not used in the merging. Table 33 presents the profiling produced for set27. It is evidenced that the time was reduced by 2 times of the uncached session. In this way time passed from 13.49 to 8.82 seconds saving in total 4.68 seconds. This is translated in a gain of the 34.65% with cache time being 65.35% and this means that an excess in time of the 153.02% would happen if passed from cached to a not cached session the rate of used images is 1.00. the rate of merged images is 0.67. the rate of failed images is 0.00.

## Validation

## Discussion

Illumination is not a big issue whenever using descriptor methods like SIFT, SURF or ORB (see Figure 84) to match images but when these are merged they tend to have problems of color difference whenever a color is more vivid than the other producing unappealing results (see Figure 65 and Figure 70). A direct solution is to apply histogram matching so that one of the merging image's color approaches to the other, this solves the problem well when they have similar lighting conditions but begin to be noticeable when not. Then it is recommended that the user takes the set of images at similar lighting conditions.

There are factors that influence in the computation of key-points and matching like for example directly filtering desired key-points by applying masks for desired areas or adjusting the feature detector parameters. Though image resizing produce matching between images it is not guarantee that they will be merged equally when they are of different sizes. Also, blurriness in general affects the identification of candidate key-points. This is demonstrated in the application of filters that tend to produce blurry results. Opposite to this, color adjustment methods that do not produce consistent smooth results can create false points of interest that do not correspond to the original image scenery.

Because the alpha mask is calculated at each merging it depends on the previous results causing errors to be cumulative. Some of the solutions can be to keep track of the changes made to the merging image so that these can be taken into consideration when calculating the alpha mask. Of course this can lead to performance and memory penalties, additionally to making the code more complicated. These errors are being reduced with the improvement of the segmentation algorithm identifying regions to correct and leaving correct areas untouched. More is compensated using alpha masks instead of binary mask to produce smooth operations in segmentation procedures.

The restoration algorithm lacks the capability of classifying good and bad images according to a case, it just ranks them according to a evaluation criteria. Said cases like in section Result for set 6 hinders the capabilities of the restoration due that the algorithm does not checks for unwanted images and it is responsibility of the user. In fact, for retinal restoration the user could feed in images that not correspond to retinal areas and still they would be processed.

## Conclusions

Many algorithms were investigated to see if they could be of any use for the project (see Algorithms design section ). Those that were promising where tested and further developed for the realization of partial goals in the project dividing the problem into several stages that could be processed to get near the desired application.

The developed algorithms were validated with their corresponding theory and in some cases comparisons were carried out between their estimations and the desired outputs so that they could offer a gain in the processing of images.

These tools were organized according to their category and packed into the *RRtoolbox* package and then used to create the *imrestore* program to restores retinal images. The sources and tests are hosted in *GitHub's* repository and can be found in (David, 2016b) with *RRtoolbox's* manual specifically in (David, 2016a). The pre-release is hosted in GitHub as well following the instructions in subsection 0.3 of the appendix. The *RRtoolbox* package have many potentials as demonstrated with *imrestore* and the developing *RRtoolFC* GUI explained in the Future Work section .

It is clear that the current restoration method does not check the retinal images for unwanted cases but only the quality of the image and it is responsibility of the user to provide good sets (bad case in section Result for set 6). This was not contemplated in the objectives but future work could improve upon this. Although it can be a drawback *imrestore* compensates this with robust threshold methods for more precise segmentation of objects under irregular cases (i.e. illumination conditions, blurry areas, etc.) improving on robustness, something that normal algorithms do not offer (see lens simulation results in the figures of Results section ). Even more, *imrestore* replaces the use of binary masks with alpha masks to mitigate confusing areas and better control operations involving segmentations. Notice that these segmentation methods are in the color domain and they do not provide means to completely discriminate objects (i.e. the bright area of the optic disk in the retina could be treated as a normal flare). Currently the space domain is being worked on but it is still experimental (i.e. localization of custom objects like the optic disk for any case are not robust).

It was demonstrated that caching techniques improve considerably processing times (see profile tables in Results section ) but can be really dangerous if data is corrupt or is not checked for its integrity causing unexpected results.

## Recommendations

Though the objective is to restore set of images with defects and low resolution it is recommended that the user takes the set of images at similar lighting conditions and they should be focused preventing blurry areas. This ensures best results, remember quality images produce quality results.

When using *imrestore* it is a good practice to cache data to an absolute path using `--cachePath <my absolute custom path>` and prevent relative paths. This useful when *imrestore* is used multiple times in the same set of images to improve speed or continue from an advanced point of the program should an unexpected error crash it in a previous session.

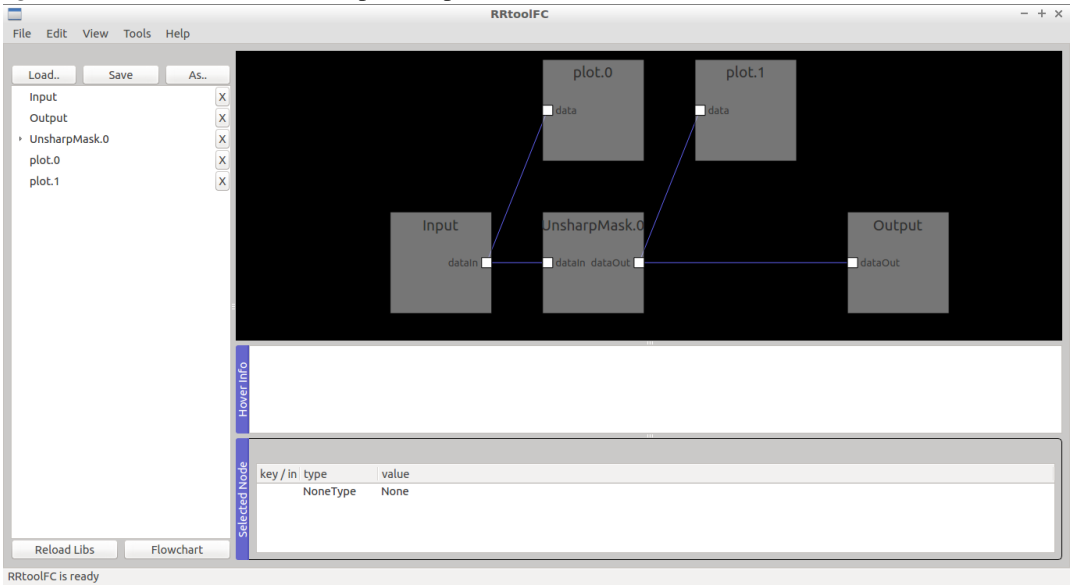
For retinal images it is recommended that only the retinal area should be in the image so that cases like in section Result for set 6 does not hinder the capabilities of the restoration. And although *imrestore* will try to change the perspective of an image so that it can fit in the restoring set it is recommended that all the images have similar perspectives (i.e. images are taken from similar positions and not too apart of the same angle).

## Future Work

So much work can be done to and on top of the *RRtoolbox* package which was demonstrated with the implementation of the *imrestore* program. This is the case contemplated in the *RRtools* repository (David, 2016b). In it an additional project is being developed under the name of *RRtoolFC* which is a graphical interface designed to ease the implementation of new tools regarding the application of *RRtoolbox* for images in general or any other package that wants to be ported to it. All this with the intention of providing users with the power to develop their own useful applications.

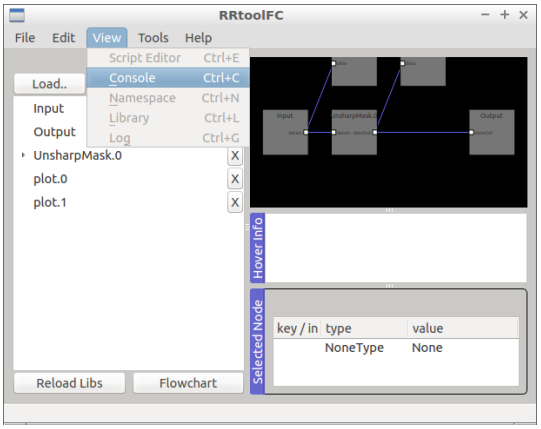
As the *RRtoolFC* acronyms implies it is the Function Chart utility of the Retinal Restoration tools. Running its main window `RRtoolFC/GUI/main.py` loads the GUI showed in Figure 89. This interface was mainly made using *Qt4 framework* and *PyQtGraph* for the support of the fast prototyping with *nodes* (Luke Campagnola, n.d.). Sadly *PyQtGraph* is not totally mature as other programs (MathWorks, n.d.-b) and does not provide the entire services needed for *RRtoolFC* slowing the development of it. This demanded the study of the *PyQtGraph* package to learn from its development and extend it to our needs. Some functionalities are being developed to support the transformation of any python code to generic *nodes* (the blocks showed in the canvas are of Figure 89) allowing the user to port their graphical program to and from pure python code. Additionally, though the nodes can be inherited from code it cannot be accessed from the graphical interface, so it is being developed a way to allow them to be accessed and modified if necessary. This obviously has the problem of how deep internal nodes of the current node can be accessed and modified or how much information can be included when saving programs using the *RRtoolFC* application. Furthermore, it is planned to add support for installations so that the user share and use their program as standalone applications.

*Figure 89. RRtoolFC* Main developed Graphical Interface

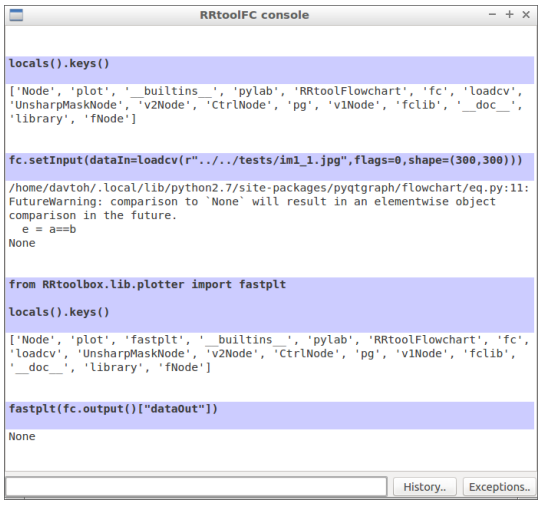



There is a console which opens the environment where the application runs. This console can be opened in 'View -> Console' (Figure 90a). In here we can make use of the *node* instances created in the graphical interface and even control internal variables not provided by it so that the advanced user can have more control over them or debug their graphical implementation should a difficult error appear to solve them. For example Figure 90b shows how to set and input and visualize an output in the console. Typing `locals().keys()` and pressing the Enter key evaluates the expression revealing the console local variables which correspond to what is in the graphical interface. "fc" is the flowchart and "loadcv" an image loader so evaluating `fc.setInput(dataIn=loadcv(r"../../tests/im1_1.jpg",flags=0,shape=(300,300)))` updates the flowchart and produce two windows corresponding to the nodes "plot.0" (Figure 91a) and "plot.1" (Figure 91b) placed at each step to visualize data. To show the output the *fastplt* function from the *RRtoolbox* package can be imported this is done evaluating `from RRtoolbox.lib.plotter import fastplt` and if `locals().keys()` is evaluated again it can be confirmed that *locals()* was updated and now we can use *fastplt*. Lastly evaluating `fastplt(fc.output()["dataOut"])` shows the flowchart's output (Figure 91c).

*Figure 90. RRtoolFC's* console

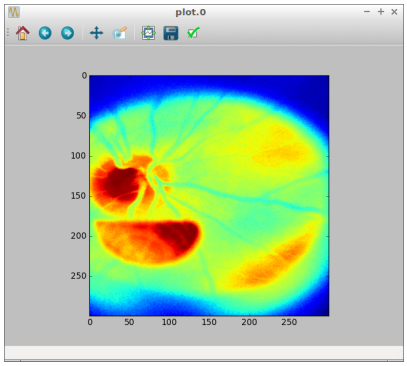



(a) How to open the console

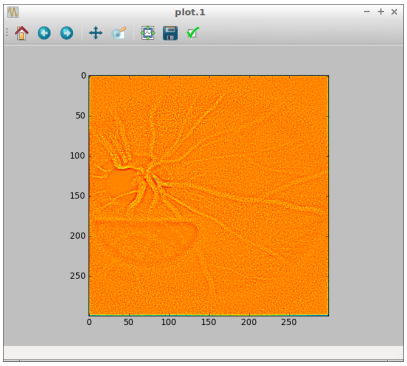



(b) Demonstration of useful commands in local variables

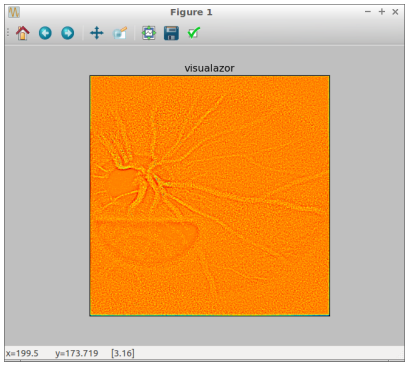
*Figure 91.* Functional application of the *RRtoolFC's* console when evaluating `fc.setInput(dataIn=loadcv(r"../../tests/im1_1.jpg",flags=0,shape=(300,300)))`




(a) Result showed by node plot.0




(b) Result showed by node plot.1




(c) Result of evaluating `fastplt(fc.output()["dataOut"])`

## References


Alahi, A., Ortiz, R., & Vandergheynst, P. (2012, June). FREAK: Fast Retina Keypoint. (pp. 510–517). doi:10.1109/CVPR.2012.6247715

Alexander Alekhin. (n.d.). Asift.py. Retrieved August 11, 2016, from https://github.com/opencv/opencv/blob/master/samples/python/asift.py

Ali_m. (n.d.). numpy - Histogram matching of two images in Python 2.x? - Stack Overflow. Retrieved August 11, 2016, from http://stackoverflow.com/a/33047048/5288758

Antal, B. & Hajdu, A. (2012, January). "Improving microaneurysm detection using an optimally selected subset of candidate extractors and preprocessing methods." *Pattern Recognition*, *45*(1), 264–270. doi:10.1016/j.patcog.2011.06.010

Askew, D. A., Crossland, L., Ware, R. S., Begg, S., Cranstoun, P., Mitchell, P., & Jackson, C. L. (2012, September). "Diabetic retinopathy screening and monitoring of early stage disease in general practice: Design and methods." *Contemporary Clinical Trials*, *33*(5), 969–975. doi:10.1016/j.cct.2012.04.011

Bozeman, J. R. & Pilling, M. (2013, January). "The convexity ratio and applications." *ResearchGate*, *76*(1). Retrieved from https://www.researchgate.net/publication/266541911_The_convexity_ratio_and_applications

CMISS. (n.d.). Sigmoid filter — Continuum Mechanics, Image analysis, Signal processing and System Identification. Retrieved April 20, 2016, from http://www.cmiss.org/cmgui/wiki/SigmoidFilter

Cortesi, D., Bajo, G., Caban, W., & McMillan, G. (n.d.). PyInstaller Manual — PyInstaller 3.2 documentation. Retrieved July 25, 2016, from http://pythonhosted.org/PyInstaller/

Cython.org. (n.d.). Cython: C-Extensions for Python. Retrieved March 9, 2016, from http://cython.org/

Darel Rex Finley. (n.d.). Area of a polygon algorithm - Math Open Reference. Retrieved from http://www.mathopenref.com/coordpolygonarea2.html

David, T. (2016a). RRtoolbox documentation. Retrieved from https://github.com/davtoh/RRtools/blob/master/documentation/_build/latex/RRtoolbox.pdf

David, T. (2016b). RRtools - Retinal Restoration Tools. Retrieved July 25, 2016, from https://github.com/davtoh/RRtools

Faust, O., Acharya U., R., Ng, E. Y. K., Ng, K. H., & Suri, J. S. (2012, April). "Algorithms for the automated detection of diabetic retinopathy using digital fundus images: A review." *Journal of Medical Systems*, *36*(1), 145–157. doi:10.1007/s10916-010-9454-7

Fierval. (2015). Fast Image Pre-processing with OpenCV 2.4, C++, CUDA: Memory, CLAHE. Retrieved June 3, 2016, from http://funcvis.org/blog/?p=54

Forssen, P. E. (2007, June). Maximally Stable Colour Regions for Recognition and Matching. (pp. 1–8). doi:10.1109/CVPR.2007.383120

Georg Brandl. (n.d.). Overview — Sphinx 1.4.5 documentation. Retrieved from http://www.sphinx-doc.org/en/stable/

Gonzalez, R. C., Woods, R. E., & Eddins, S. L. (2004). Digital Image Processing Using Matlab - Gonzalez Woods & Eddins.pdf. doi:10.1117/1.3115362

IEEE. (2000). IEEE SA - 1471-2000 - IEEE Recommended Practice for Architectural Description for Software-Intensive Systems. Retrieved March 31, 2016, from https://standards.ieee.org/findstds/standard/1471-2000.html

ITK. (n.d.). ITK SigmoidImageFilter. Retrieved April 20, 2016, from http://www.itk.org/Doxygen/html/classitk_1_1SigmoidImageFilter.html

K, A. R. & Mordvintsev, A. (2013). Histograms - 2: Histogram Equalization. Retrieved June 7, 2016, from http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_histograms/py_histogram_equalization/py_histogram_equalization.html

Kindratenko, V. (1997). *Development and Application of Image Analysis Techniques for Identification and Classification of Microscopic Particles*. UIA, Departement Scheikunde. Retrieved from https://books.google.com.co/books?id=PyEnrgEACAAJ

Leutenegger, S., Chli, M., & Siegwart, R. Y. (2011, November). BRISK: Binary Robust invariant scalable keypoints. (pp. 2548–2555). doi:10.1109/ICCV.2011.6126542

Liu, Y. & Yu, F. (2015, April). "An automatic image fusion algorithm for unregistered multiply multi-focus images." *Optics Communications*, *341*, 101–113. doi:10.1016/j.optcom.2014.12.015

Luke Campagnola. (n.d.). PyQtGraph - Scientific Graphics and GUI Library for Python. Retrieved from http://www.pyqtgraph.org/

Martínez Rubio, M., Moya Moya, M., Bellot Bernabé, A., & Belmonte Martínez, J. (2012, December). "Diabetic retinopathy screening and teleophthalmology." *Archivos de la Sociedad Española de Oftalmología*, *87*(12), 392–5. doi:10.1016/j.oftal.2012.04.004

MathWorks. (n.d.-a). Adjust histogram of image to match N-bin histogram of reference image - MATLAB imhistmatch. Retrieved August 11, 2016, from http://www.mathworks.com/help/images/ref/imhistmatch.html

MathWorks. (n.d.-b). Simulink - Simulation and Model-Based Design. Retrieved July 29, 2016, from http://www.mathworks.com/products/simulink/

Médioni, G. (2005). *Emerging Topics in Computer Version*. Prentice Hall Computer. Retrieved from https://books.google.com.co/books?id=qV90QgAACAAJ

Nistér, D. & Stewénius, H. (2008, October). Linear Time Maximally Stable Extremal Regions. In D. Forsyth, P. Torr, & A. Zisserman (Eds.), (pp. 183–196). Lecture Notes in Computer Science. Springer Berlin Heidelberg. Retrieved from http://link.springer.com/chapter/10.1007/978-3-540-88688-4_14

OpenCV. (n.d.-a). Feature Detection and Description — OpenCV 2.4.13.0 documentation. Retrieved August 6, 2016, from http://docs.opencv.org/2.4/modules/features2d/doc/feature_detection_and_description.html

OpenCV. (n.d.-b). Image Filtering — OpenCV 2.4.12.0 documentation. Retrieved April 1, 2016, from http://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html#bilateralfilter

deployment - Python Wiki. (n.d.). Retrieved July 25, 2016, from https://wiki.python.org/moin/deployment

Paris, S., Kornprobst, P., Tumblin, J., & Durand, F. (2008). "Bilateral Filtering: Theory and Applications." *Foundations and Trends® in Computer Graphics and Vision*, *4*(1), 1–75. doi:10.1561/0600000020

Phillippe Kruchten. (1995, November). "Architectural Blueprints—The "4+1" View Model of Software Architecture." *IEEE Software*, *12*(6), 540–555. doi:10.1145/216591.216611

Python Software Foundation. (2015). Glossary — Python 2.7.11 documentation. Retrieved February 16, 2016, from https://docs.python.org/2/glossary.html

Pyzo. (n.d.). Python vs Matlab — Pyzo - Python to the people. Retrieved from http://www.pyzo.org/python_vs_matlab.html

Rey Otero, I. & Delbracio, M. (2014, December). "Anatomy of the SIFT Method." *Image Processing On Line*, *4*, 370–396. doi:10.5201/ipol.2014.82

Rosebrock, A. (2014, July). How-To: 3 Ways to Compare Histograms using OpenCV and Python. Retrieved from http://www.pyimagesearch.com/2014/07/14/3-ways-compare-histograms-using-opencv-python/

Rosten, E. & Drummond, T. (2006, May). "Machine Learning for High Speed Corner Detection." *Computer Vision – ECCV 2006*. Lecture Notes in Computer Science, 430–443. doi:10.1007/11744023_34

Rosten, E., Porter, R., & Drummond, T. (2010, January). "Faster and better: A machine learning approach to corner detection." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *32*(1), 105–119. doi:10.1109/TPAMI.2008.275. arXiv: 0810.2434v1

Rublee, E., Rabaud, V., Konolige, K., & Bradski, G. (2011, November). ORB: An efficient alternative to SIFT or SURF. (pp. 2564–2571). doi:10.1109/ICCV.2011.6126544

SciPy. (n.d.). Smoothing of a 1D signal — SciPy Cookbook documentation. Retrieved August 10, 2016, from http://scipy-cookbook.readthedocs.io/items/SignalSmooth.html

SciPy. (2015a). Numpy for Matlab users — NumPy v1.11.dev0 Manual. Retrieved March 20, 2016, from https://docs.scipy.org/doc/numpy-dev/user/numpy-for-matlab-users.html

SciPy. (2015b). Quickstart tutorial — NumPy v1.11.dev0 Manual. Retrieved March 20, 2016, from https://docs.scipy.org/doc/numpy-dev/user/quickstart.html

Scipy. (2015). The N-dimensional array (ndarray) — NumPy v1.10 Manual. Retrieved March 20, 2016, from http://docs.scipy.org/doc/numpy/reference/arrays.ndarray.html

Sopharak, A., Uyyanonvara, B., & Barman, S. (2013, July). "Simple hybrid method for fine microaneurysm detection from non-dilated diabetic retinopathy retinal images." *Computerized Medical Imaging and Graphics*. Retinal Image Analysis, *37*(5–6), 394–402. doi:10.1016/j.compmedimag.2013.05.005

WebMD. (2014). Type 1 Diabetes - Prevention. Retrieved from http://www.webmd.com/diabetes/tc/type-1-diabetes-prevention

WHO. (2016). Diabetes: Fact sheet. Retrieved February 16, 2016, from http://www.who.int/mediacentre/factsheets/fs312/en/

WHO & IDF. (2006). Definition and diagnosis of diabetes mellitus and intermediate hyperglycaemia. doi:ISBN9241594934

Yu, G. & Morel, J.-M. (2011, February). "ASIFT: An Algorithm for Fully Affine Invariant Comparison." *Image Processing On Line*, *1*. doi:10.5201/ipol.2011.my-asift

Zhou, M. & Asari, V. K. (2011). Speeded-Up Robust Features Based Moving Object Detection on Shaky Video. In K. R. Venugopal & L. M. Patnaik (Eds.), (pp. 677–682). Communications in Computer and Information Science. Springer Berlin Heidelberg. Retrieved from http://link.springer.com/chapter/10.1007/978-3-642-22786-8_86

## Appendix A RRtoolbox: Python Implementation

Additional to the main sources of the *RRtoolbox* used to create *imrestore* it contains other algorithms used for testing, prototyping and demonstrating restoration tools. Some of the algorithm are explained here, how to set them up, their class diagrams and other program-related topics that are not in the main document.

### A.1 Procedures to set-up *RRtoolbox*

It was implemented in python using some of its build-in functions and others provided via modules. If the OS is Linux there is a high probability that python 2.7 is already installed with the distribution but if that is not the case it can be installed via the package manager. Here it will be assumed that the Linux is a Debian distribution and each time something is typed in the console it will be referenced with a "$" sign.

**A.1.1 Install Python.** Python is a famous programming language and support for it can be easily found in the web but for the sake of completeness it will be explained how to set it up. In Linux it is pretty straightforward:

```
$ sudo apt-get install python2.7
```

And that's it, even if the Linux distribution did have Python it should have one now. For windows, it is a few clicks away as showed below:

- Go to www.python.org
- Under Downloads select Windows
- Select the latest release for Python 2
- Download according to the Windows version and computer architecture
- Once downloaded run the installer and follow the Setup steps.
- Once installed register python executable path in a Windows variable.

***Register Python's path in a Windows' variable (Window 7).*** If for any reason the Python installation did not register its executable in a Windows variable for it to be used as a command in the cmd then this steps can be followed to solve the issue:

- Go to Control Panel (menu)
- System and security (icon)
- System (icon)
- Advanced system settings (left menu)
- Environment variables (button)
- Under the path variable append the python variable
  - Double click in Path
  - Once the "Edit System Variable" Dialog is opened
  - Under "Variable value" box: at the end of all its variables insert the following

    ```
    ; C:\Python27
    ```

  - Click OK button
- Click Ok button
- Now under the command Prompt the user should be able to execute a python command.

**A.1.2 Install pip.** Pip is a python module used as a package manager to install more packages. It is in Linux with the package manager:

```
$ sudo apt-get install python-pip
```

And for Windows it can be installed as follows

- Go to https://pip.pypa.io/en/stable/installing/
- Download the get-pip.py
- Run the script

  ```
  $ python get-pip.py
  ```

**A.1.3 Install packages to Python using pip.** To install a package to python using pip in Linux or Windows it just have to be typed in the console the command pip followed by the program. So, to install the principal dependencies of *RRtoolbox's* packages then type then one by one:

```
$ pip install numpy
$ pip install matplotlib
$ pip install dill
$ pip install joblib
$ pip install pyinstaller
```

Numpy and matplotlib pacakges also can be obtained installing Scipy which comes with them and other scientific modules or following the instructions at https://www.scipy.org/install.html:

```
$ pip install scipy
```

Code 6: Install OpenCV

```
# https://help.ubuntu.com/community/OpenCV
# http://docs.opencv.org/2.4/doc/tutorials/introduction/linux_install/linux_install.html
version="$(wget -q -O - http://sourceforge.net/projects/opencvlibrary/files/opencv-unix | egrep -m1 -o
 ↪  '\"[2](\.[0-9]+)+' | cut -c2-)"
echo "Preparing to install OpenCV" $version
# pre-requisites
mkdir OpenCV
cd OpenCV
echo "Removing any pre-installed ffmpeg and x264"
sudo apt-get -qq remove ffmpeg x264 libx264-dev
echo "Installing Dependenices"
sudo apt-get -qq install libopencv-dev build-essential checkinstall cmake pkg-config yasm libjpeg-dev
 ↪  libjasper-dev libavcodec-dev libavformat-dev libswscale-dev libdc1394-22-dev libxine2-dev
 ↪  libgstreamer0.10-dev libgstreamer-plugins-base0.10-dev libv4l-dev python-dev python-numpy libtbb-dev
 ↪  libqt4-dev libgtk2.0-dev libfaac-dev libmp3lame-dev libopencore-amrnb-dev libopencore-amrwb-dev
 ↪  libtheora-dev libvorbis-dev libxvidcore-dev x264 v4l-utils ffmpeg cmake qt5-default checkinstall
# downloading
if [ -f "OpenCV-$version.zip" ]
then
    echo "using OpenCV-$version.zip already in folder."
else
    echo "Downloading OpenCV" $version
    wget -O OpenCV-$version.zip
     ↪  http://sourceforge.net/projects/opencvlibrary/files/opencv-unix/$version/opencv-"$version".zip/download
fi
# installing
echo "Installing OpenCV" $version
unzip OpenCV-$version.zip
cd opencv-$version
mkdir build
cd build
cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local -D WITH_TBB=ON -D
 ↪  BUILD_NEW_PYTHON_SUPPORT=ON -D WITH_V4L=ON -D INSTALL_C_EXAMPLES=ON -D INSTALL_PYTHON_EXAMPLES=ON -D
 ↪  BUILD_EXAMPLES=ON -D WITH_QT=ON -D WITH_OPENGL=ON ..
make -j2
sudo checkinstall
sudo sh -c 'echo "/usr/local/lib" > /etc/ld.so.conf.d/opencv.conf'
sudo ldconfig
echo "OpenCV" $version "ready to be used"
```

To install OpenCV 2.4.13 (or newer versions from the 2.4 release) in Linux follow this instructions:

- Create an empty file "install_opencv.sh", open it and place Code 6.
- Save "install_opencv.sh" and close.
- Run in the terminal:

```
$ chmod +x install_opencv.sh
$ ./install_opencv.sh
```

The script install_opencv.sh will download and install the latest OpenCV release of 2.4 version. It builds the release from source which takes a considerate amount of time and computer power so an stable power connection and closing all unrelated programs is recommended.

In Windows, download the installer at https://sourceforge.net/projects/opencvlibrary/files/opencv-win/2.4.13/opencv-2.4.13.exe/download and install.

## 0.2 Procedures to generate *RRtoolbox* documentation

The *sphinx* Python module is used (Georg Brandl, n.d.). The command to generate *RRtoolbox* documentation is:

```
sphinx-apidoc -F -H RRtoolbox -A davtoh -V 0.1 -R 1 -o . ../RRtoolbox ../RRtoolbox/temp
```

This generates the folders *_build* used by sphinx to place its built documentations, *_static* to place additional sources (e.g images) and *_templates* used to place templates which modifies the look of created documents. Additionally it creates the files *Makefile*, *make.bat*, *conf.py* files used to configure *sphinx* to generate the documentation. The other *.rst* files are related to the documentation structure. To let *sphinx* find the *RRtoolbox* folder lines must be placed in the configuration file *conf.py*:

```
import sys,os
sys.path.insert(0, os.path.abspath("../")) # add RRtools path
```

Once sphinx is configured as desired the *make* command can be used to generate the documentation. So to generate the pdf just it has to be typed in:

```
make latexpdf
```

or to generate the html files:

```
make html
```

Additional options can be consulted asking the bash for help:

```
make -h
```

## 0.3 Procedures to download *imrestore* program and source

To download the execution files of *imrestore* just go to the *RRtools* repository found in (David, 2016b). Click in the link as shown at Figure 92 in the red box then when the page is loaded as in Figure 93a scroll down to the download section as in Figure 93b. Once there you can download the compiled versions for windows and Linux enclosed in the red boxes. The current source code can be downloaded too and its weight repents in the current sources, at the realization of this document the *.zip* file weights `92,2 MiB` and the contents can be appreciated in Figure 94.

*Figure 92*. Finding release link in *RRtools* repository

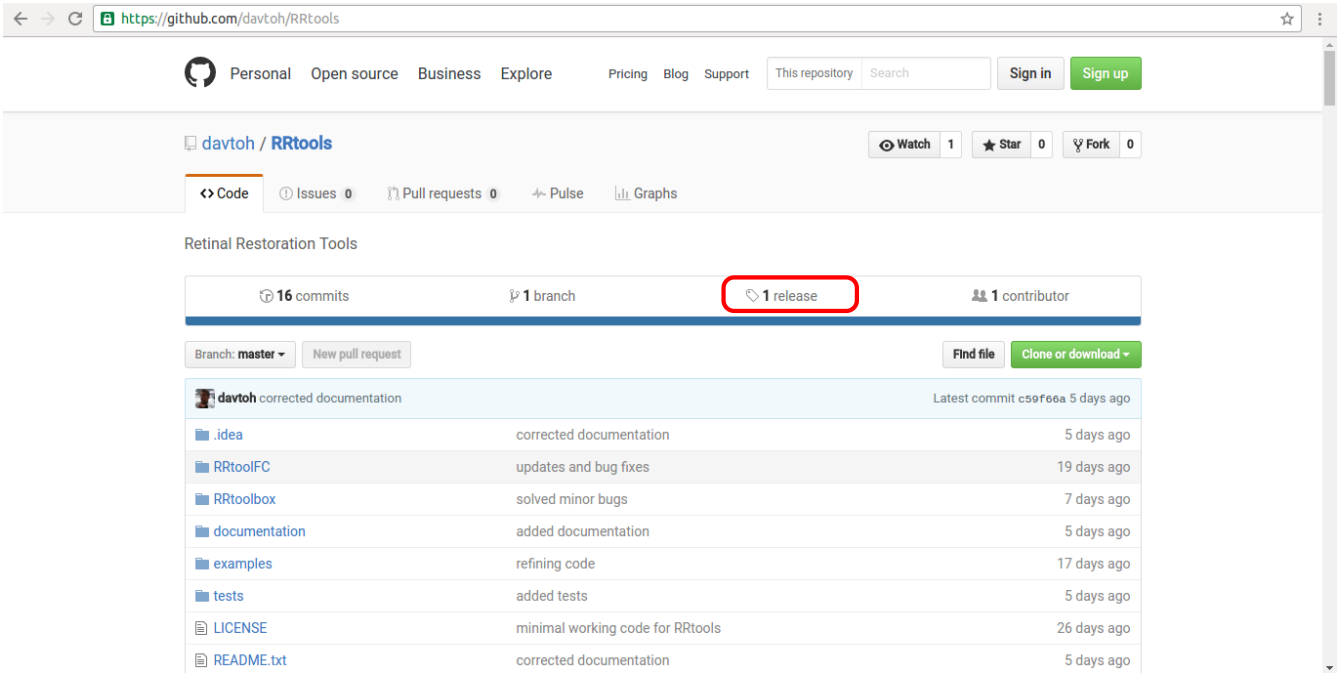
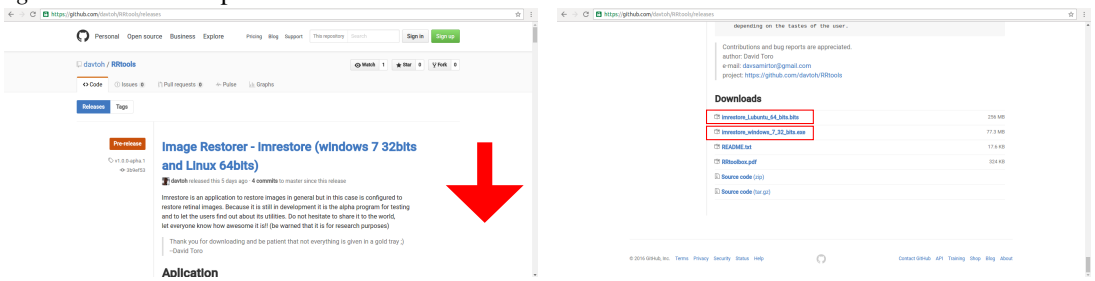


*Figure 93. Imrestore* pre-realease




(a) Pre-release page (b) *Downloads* section

*Figure 94. Imrestore* downloaded sources