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 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 and I' d like to thank them for accompanying and encouraging me when my determination wavered and needed strengths to continue my work, to them many many thanks, it certainly was not in vain.

Contents

List of Figures

List of Tables

List of Equations

List of Algorithms

List of Codes

1 Introduction

This document was conceived from a phase of a project at the AVARC (Grupo en Automatizacion, Vision artificial, Robotica y control) research group and CISVI (Centro de Investigación en Salud y Visión) at La Salle University (Universidad De La Salle, n.d.-a, n.d.-b). 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.

1.1 Case study

For the diagnosis and prevention of *diabetic retinopathy* (DR) patients must do certain tests that involve DR screening using very expensive cameras to acquire high quality photos of the retina. The AVARC and CISVI groups at La Salle University are implementing a system to address this problem using telemedicine services. The project was organized in several phases for the acquisition, transmission, restoration and processing of retinal images in potential patients that could develop DR. Because of the cost of fundus cameras and difficulty of their transportation it is necessary to implement a less expensive and portable system via mobile devices. The images are acquired using phones with pan tilt handles which have the problem of producing low quality photos with low resolution, noise, defocus, blurs due to movement, artefacts and light flares compared with those acquired by the fundus camera. The acquired images consist of 3 photos of the same retina in each eye which must be adapted to generate a restored image for the retina (e.g. it can be the best restored image or one image generated by combining the selection of other areas in each) to extract useful features that allow the diagnosis of patients. These images must be processed from a server so it must be taken into account the system specifications and resources available to process the restoration algorithms.

1.2 problem statement

Images captured by mobile devices are of low quality therefore we need to determine which images will adversely affect restoration (i.e. one or more images may contain a lot of noise, be very blurred or not be part of the sample group) to avoid losing precious time

processing and failure of the restoration. Developed algorithm should support and be focused in the restoration algorithms which are very critical for the project and are based on known problems (e.g. low resolution, nonuniform illumination, etc.). Also, There should be a criteria used to validate the restored images so that no valuable information is lost during the restoration process. This can be accomplished by comparing the restored image with a pattern image taken by a fundus camera. Once the restoration algorithms are developed and evaluated to produce concrete results a final implementation must be developed. The final application do not have to interact with the pattern images and must provide a way of interaction with the user. Said that, the implemented API or application does not need to use graphical interface.

To summarize, it is necessary to implement algorithms for restoring images that improves the quality of noisy images and to correct the problems arising on the acquisition. The algorithm for image restoration must be robust and its operation must be validated which can be done using images acquired by a fundus camera (image pattern).

1.3 Case objectives

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.

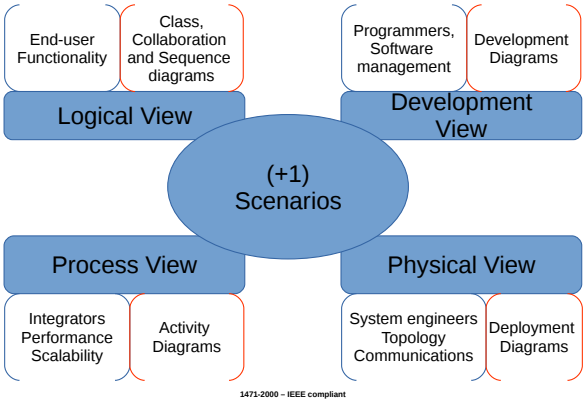
3. **Implement a restoration application for images:** All the tools were organized and packed into one module according to their category and with some of them a program was developed to offer an automated solution for the restoration of retinal images.

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 found in the documentation 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, all this can be found in the repository (David, 2016).

Generally the "4+1" View Model (see Figure 1) is widely used to explain complex systems or to convey program models (Phillippe Kruchten, Philippe Kruchten, Phillippe Kruchten, Kruchten, & 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 in (Toro, 2016) repository for the user to review, confirm and use them. So the "4+1" View Model is partially implemented showing in some cases the use classes and diagrams, leaving out other models. As a plus though, additional non-standard tests and demonstrations are given for the reader to confirm that each algorithm works and can be used in practical cases.

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 results some of them are briefly explained using pseudo-code (used to code the actual algorithm or to abstract long ones), others with examples in their native language (to emphasize details and functionality) and in some

cases other resources (e.g. explanation, class diagrams, outputs, etc.). In other words, 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 available in the appendix and the reader is invited to review the external sources for completeness and for the algorithms not explained in this document either because of their complexity or because they were not in the scope of the project (David, 2016; Toro, 2016).

In section 2 ‘Glossary’ on the facing page I present some of the key concepts used in the document, section 3 ‘Art State’ on page 24 introduces some of the investigated works and how they could be used in the project, section 4 ‘Methodology’ on page 29 shows how was carried out the project and the taken steps, section 5 ‘Algorithms design’ on page 43 explain some of the used algorithms and their results, section 6 ‘Implementation’ on page 88 introduces the main program functionality and components (i.e. the back-end) with the application in the section 7 ‘User Interface’ on page 103 (i.e. the front-end), section 8 ‘Results’ on page 110 shows all the results with most of them using the automatic routine of the program, section 9 ‘Validation’ on page 153 summarizes and explains the qualification rates for the used images, performance and program results, the program capabilities and deficiencies are discussed in section 10 ‘Discussion’ on page 163 while section 11 ‘Conclusions’ on page 165 points out the finals results to appreciate the importance of the program, letting the user know some issues and respective recommendations in section 12 ‘Recommendations’ on page 166 and finally section 13 ‘Future Work’ on page 167 shows some progress in the things that are still being work on. Used citations can be found in section 14 ‘References’ on page 170 while other related work in section 15 ‘Further Reading’ on page 181. Additional important material is presented in A ‘RRtoolbox: Python Implementation’ on page 185.

2 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 the explanations 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:

2.1 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 (Python Software Foundation, 2015a).

2.2 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 (Microsoft, n.d.-c).

2.3 De-serialization

It is the contrary process of serialization (Microsoft, n.d.-c).

2.4 Level

Or gray level, represents the value that any pixel can take in a monochrome image i.e. there are usually 256 levels in a gray image and 2 levels in a binary image (Gonzalez, Woods, & Eddins, 2004, section 2.1).

2.5 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 (Médioni, 2005).

2.6 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 (Evening, 2007).

2.7 Intensity

Pixel-level or the pixel's value (Gonzalez et al., 2004, section 2.1).

2.8 BGR image

Refers to a RGB image with its channels reversed, thus channel 0 is for blue color conversely to Red in RGB images (OpenCV, n.d.-c).

2.9 BGRA image

It is an RGB image that contains an additional A channel to support additional information like transparency for PNG formats (OpenCV, n.d.-c).

2.10 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 (MathWorks, n.d.-c).

2.11 Segmentation

It is the partitioning of an image into simpler and meaningful segments for the simplification of its analysis (MathWorks, n.d.-b).

2.12 Registration

It is the process of overlaying two or more images of different perspectives of the same scene (Liu & Yu, 2015).

2.13 Numpy Arrays

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 (SciPy, 2015b).

2.13.1 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 (SciPy, 2015b).

2.13.2 ndarray.size. The total number of elements of the array (SciPy, 2015b).

2.13.3 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 (SciPy, 2015b).

2.13.4 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) (SciPy, 2015b).

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, 2015b).

3 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 et al., 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 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 (Chang & Tseng, 2010) 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.

In “Super-resolution restoration of MMW image based on sparse representation method” published by (Shang, Zhou, & Su, 2014) *millimetre wave* (MMW) images were restored by using a dictionary to map high- and low-resolution parts of the images. The Super-resolution (SR) image restoration can be considered as the inverse problem of recovering an original high-resolution image by fusing the low-resolution images, as it needs prior knowledge of the observation model to map the high- resolution to the low-resolution image. So they started from the sparse coding (SC) which provides algorithms for finding succinct representations of stimuli pretty much as the visual cortex model. Then they trained a dictionary using a modified *fast space coding* (FSC) algorithm to obtain the high- and low-resolution dictionary pairs or keys with the same sparse representation to correspond to the same patch in the images. This dictionary was used to restore both natural images (e.g. photos) or the MMW images using the neighbor embedding (NE) method and the bicubic method. The quality of the images were measured using *single noise ratio* (SNR) (Thong, Sim, & Phang, 2001) for natural images and relative SNR (RSNR) for MMW images due to the quantity of noise present in them.

The work done by (Kallel, Aboulaich, Habbal, & Moakher, 2014) published as “A Nash-game approach to joint image restoration and segmentation” approach differently

to restore image by using game theory with two players, "restoration" and "segmentation". Due that these two players are antagonistic with the task of solving two different minimization problems, one favouring regularization and the other enhancing contours, it leads to different results so the authors switch from the notion of an optimum solution to an equilibrium solution called "Nash equilibrium" (NA) using a relaxation algorithm to achieve this. The objective is then to solve the game by reaching NA. The proposed method solves the minimization problems which compared to the Mumford–Shah algorithm reduced considerably the iterations and time needed while producing better restoration and segmentation results.

In “A novel modified cepstral based technique for blind estimation of motion blur” published by (Deshpande & Patnaik, 2014) addresses the problem of automating deblurring by introducing a new modified-cepstral approach with a novel bit-plane slicing method to estimate the motion blur parameters in an automated faction provided that the camera motion is limited and uniform under any arbitrary direction. They claim their method to perform similar to the Bayesian approach while providing a simpler solution. But the restoration from motion blur still creates unwanted ring-like artefacts that could not be beneficial for our feature detection algorithms. Nonetheless, knowing of this, “A novel modified cepstral based technique for blind estimation of motion blur” promised the removal of these artefacts and handling of non-uniform motions in blurred images for future releases.

In “Automatic detection and correction for glossy reflections in digital photograph,” (Chang & Tseng, 2010) present an unsupervised approach for correcting glossy reflections on oily skin using a novel inpainting algorithm. They do this by converting the input image into the YCbCr color space, detecting the skin using a Gaussian mixture model and Sobel for edges among other procedures, ultimately leading to the detection of the face where colors near to white can be easily identified to apply in them the inpainting technique and correct the overexposed areas to produce visually pleasant results for the digital photograph. But though the inpainting method produce appealing results it uses near colors to fill the affected area which for our purposes it would no be convenient due that this approach can introduce false information in the retinal images where the present information has to be true for the real scene.

There are several works that created retinal images databases and resorted to ex-

pert systems to qualify their results like DRIVE and STARE databases (Staal, Abramoff, Niemeijer, Viergever, & van Ginneken, 2004; Hoover & Goldbaum, 2003). The photographs for DRIVE database were obtained from a diabetic retinopathy screening program in The Netherlands. The images were acquired using a Canon CR5 non-mydriatic 3CCD camera with a 45 degree *field of view* (FOV), using 8 bits per color plane at 768 by 584 pixels. Each images was cropped around the circular FOV of diameter of about 540 pixels and for each image mask were provided to delimit the FOV (Staal et al., 2004). In the STARE database, the full set contains around 400 raw images. They offer a list of diagnosis codes and diagnoses for each image, expert annotations of the manifestations (features) visible in each image, tabulated in text files. They did a Blood vessel segmentation work including 40 hand labeled images with their results and a demo. And additionally an Optic nerve detection work including 80 images with ground truth tests and their results (Hoover & Goldbaum, 2003).

4 Methodology

Before we discuss the algorithms, it is important to understand the context in which they were conceived. While working in the project several 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 counter parts, interpreted languages like MATLAB and Python, sacrifice execution speed to offer fast code prototyping (i.e. reducing developing) because of their simpler and readable languages and even in some cases offering means to improve speed by compiling or translating to other languages like C or using them as back-ends (Cython, n.d.; Pyzo, n.d.). To demonstrate these differences I made a rough comparison of C++, MATLAB and Python in Table 1 and a sample code in Table 2. This way I concluded on using Python as it is free reducing costs, open source to be able to code anything from previous works with a big supporting community and because it has increasing scientific research backgrounds.

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

4.1 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

Table 2

Comparison example of "for loops" in C and Python

C for loop	Python simulation of C for loop	Pythonic for loop

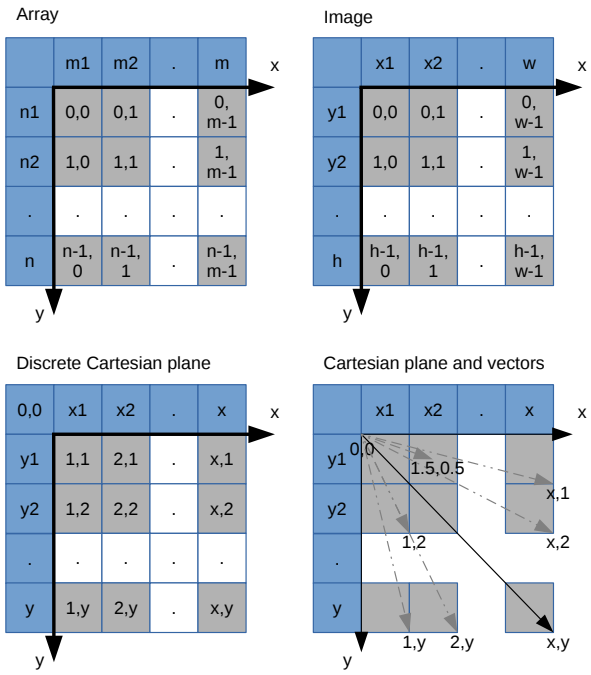
`#include <stdio.h>`
`#include <string.h>`
`int main() {`
`char s[5] = "Hello";`
`for(int i=0;i<strlen(s);i++){`
`  printf("%c", s[i]);`
`  return 0;`
`}`
	`s = "Hello"`
	`for i in range(len(s)):`
	`  print s[i]`
		`s = "Hello"`
		`for letter in s:`
		`  print letter`

for scientific development in python) (NumPy, n.d.), though OpenCV uses other representations originally for other programming languages like C++ (OpenCV, n.d.-c). 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, 2015c).

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 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 (SciPy, 2015c).

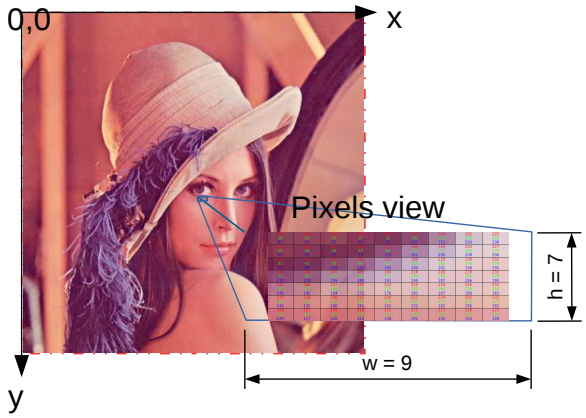
Figure 3 better illustrates the notion of arrays representing images with the height as the number of rows and the width as the number of columns. Notice that from the

Figure 2. Used representations



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



4.2 Restoration Model

Most restoration methods part from a common model which involves a degradation function and an additive noise (Shang et al., 2014; Gal, Kiryati, & Sochen, 2014) that can

be represented as in Equation 1.

$$g(x,y) = D[f(x,y)] + \eta(x,y) \tag{1}$$

where $g(x,y)$ is the degraded image, D is the degradation function usually refereed as the *point spread function* (PSF) in the spacial domain behaving as a convolution or *optical transfer function* (OTF) in the frequency domain behaving as a simple multiplication and η is the additive noise usually considered independent of the original image $f(x,y)$ (Gonzalez et al., 2004, 5.1 A Model of the Image Degradation/Restoration Process).

Due that there are several images for the same scene I opted in using image registration for the restoration. So I focused on using all the available perspectives to restore a particular image among them (i.e. base image) using the remaining images (i.e. merging images) to produce one approximating the original image $f(x,y)$ (i.e. the restored image). I investigated a lot of algorithms that could solve individual problems (see section 5 ‘Algorithms design’ on page 43) and came up with an integrated solution capable of automatic capabilities (see section 6 ‘Implementation’ on page 88) and then created a program for deployment and utilization with a server (see subsection 7.2 ‘Deployment: Executable’ on page 104). The results presented in section 8 ‘Results’ on page 110 were validated using an expert system (see section 9 ‘Validation’ on page 153) because the parties involved in the project did not provided with images taken by fundus cameras as agreed.

4.3 Optimization Methods

Some optimization techniques were contemplated in the implementation of the algorithms. These techniques were learnt while doing the research and learning to program in Python which as the finial implementation proved to be are really useful indeed.

4.3.1 Normalization. This optimization method consist in reducing the quantity of elements in data without loosing relevant information. I used it for normalizing data to small values (i.e. normalize to 1) by reducing the size of arrays (Fraz, Barman, et al., 2012, 3.1.2 Image Normalization)(Garcia-Fidalgo & Ortiz, 2015).

4.3.2 Process big data. Because arrays can be ridiculously big they can result in performance penalties and be really difficult to dealt with if not measures are taken. In some cases the algorithms can support images with small resolution (Shang et al., 2014, 5.2. Restoration of the MMW image). In other cases the size of images can produce performance penalties if it is too big to process (e.g. $10000x8000$) (Giancardo et al., 2012; Imani, Pourreza, & Banaee, 2015, 4.1 Preprocessing) or is not convenient for the algorithm like in (Fathi & Naghsh-Nilchi, 2013) or can work with any resolution (Deshpande & Patnaik, 2014). My solution is just to resize it (e.g. $500x400$) using a good interpolation (Quellec et al., 2012, 9.1. e-ophtha — examination record dataset e-ophtha) (though linear interpolation does fine for most cases)(Staal et al., 2004; Yasukawa, Okuno, Ishii, & Yagi, 2016), process the resized image and at the end resize to the original image (Fraz, Barman, et al., 2012, 3.1.2 Image Normalization). For data manipulation like points or domain transformations the process it similar (in fact resizing is a common transformation) and subsubsection 5.1.4 'Domain transformations: scaling to original results' on page 48 covers it in more detail. There must be special care that the resizing does not interfere with the developed algorithms so that they can continue to work well or make the respective algorithm adaptation (Chang & Tseng, 2010).

4.3.3 Lazy evaluation. Lazy evaluations can be described as an evaluation of a expression on demand or when needed (Haskell, n.d.-a). I use it in my case to load data and arrays from disk when needed which is advantageous to keep RAM memory free of unnecessary data and in the case of structures, I use it to compute data only when it is going to be used which reduces the processing times when classes are initialized when they are instantiated and most of the calculations are not going to be used (Toro, 2016, e.g. Rtoolbox.lib.image.ImCoors class). Python offers good solutions as the *generators* (Python Software Foundation, 2015b, see generator) or by assigning values by pass-by-value semantics (SciPy, 2015a) 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 demand instead of processing everything at the class instantiation (Python Software Foundation, 2015b, see descriptor and decorator).

4.3.4 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 (Hearsay Social Engineering, n.d.). In the case of python there exist a garbage collector (it can be imported as *gc*) to manage memory in the programs (Python Software Foundation, n.d.-i) but currently objects with circular references should not be collected and some of them are kept alive until the whole program ends. To prevent it I used the *weakref* module to provide weak references to an object which do not prevents it from being garbage collected or deleted so that memory can be freed as soon as an object is not used any mode(Python Software Foundation, n.d.-g). This resulted especially useful when using images, I did an interface to load images, keep them in memory while in use and delete them to free memory if not needed or if certain limit was exceeded. If the image is not in memory and I request it the structure just seamlessly loads it without the user noticing that it was not in memory. The implementation is called *ResourceManager* found in the *RRtoolbox* package and can be imported in python as `from RRtoolbox.lib.cache import ResourceManager`.

4.3.5 Multitasking. I used multitasking techniques for heavy tasks, multiprocessing to take advantage of multi-core computers (Python Software Foundation, n.d.-c) and multithreading for lighter tasks or for task incompatible with the multiprocessing API in Python (Python Software Foundation, n.d.-b).

4.3.6 Memory mapped files. It is another useful technique that I employed to keep memory free at the expense of more processing time, this caused by the object being accessed not from RAM but from the local disk which is slower to read and write from (Microsoft, n.d.-b). This also has proven to be useful when computing big data which cannot be loaded to RAM memory (i.e. RAM could be overflown) 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) (MathWorks, n.d.-d; Microsoft, n.d.-a). Python offers the `mmap` function (Python Software Foundation, n.d.-d) and Numpy the `numpy.memmap` function (SciPy, n.d.-a) to support memory mapped files.

4.3.7 Memoization and caching. I decided to save data with expensive function calls or difficulty to process them for later use in structures. In this way, results are cached to be used in the same session (i.e. keep in RAM) or persisted to disk to keep these computations between sessions (Dominus, 2005; Haskell, n.d.-b). This persistence of data use a serialization and de-serialization process which involves saving and loading

information from disk (Python Software Foundation, 2015a). In Python, this is achieved using the package 'pickle' (Python Software Foundation, n.d.-a) and the caching using hashes to generate key-like values usually in dictionary structures (other structures can be used too) (Python Software Foundation, n.d.-f, 5.5. Dictionaries).

4.4 Gathering information

As any big project I began by evaluating the case problems and reviewing accordingly the art state related to restoration processes, diabetic retinopathy, de-blurring techniques, de-noising, segmenting and other topic-related works (see the section 3 'Art State' on page 24 for more details).

Figure 4. Handheld with PanOptic to use with mobile devices

(a) Handeld Parts (b) Assembled handled

Source: (Lozano Forero, 2015)

According to the project's requirements images had to be taken by mobile devices and processed in a server which were dependent to other previous works. These methods were developed by (Velandia Calderón & Cifuentes Cifuentes, 2015) to provide a platform for web services and the means to acquire de photos by (Lozano Forero, 2015) which included making a handheld and an application to support several types of mobile devices. The handheld was designed and provided as shown in Figure 4 with the assemble parts to be attached in the PanOptic (Figure 4a) and the assembled pieces to be used with mobile devises (Figure 4b).

The mobile application is called RetinoTest which is shown in Figure 5 where users have to register as shown in Figure 5a to interact with the application. Figure 5b shows

Figure 5. RetinoTest mobile application

(a) RetinoTest main window

(b) RetinoTest without photos

(c) RetinoTest with adquired photos

Source: (Lozano Forero, 2015)

the fields that can be used in the RetinoTest app and the corresponding acquisitions shown in Figure 5c. This information is then later sent to a server so that the restoration program can process the distorted images and select the best images (Lozano Forero, 2015).

The web services developed by (Velandia Calderón & Cifuentes Cifuentes, 2015) are appreciated in Figure 6 with the main page shown in Figure 6a and the registration of new patients in Figure 6b. Images acquired by the RetinoTest program could be then visualized from the web as shown in Figure 7 for a registered patient with important collected data (Figure 7) and make consults (Figure 7b).

4.4.1 Taking samples. Images acquired using the methods from (Lozano Forero, 2015) are evidenced in Figure 8 which demonstrates how the images presented in this document were taken. Nonetheless, the reader is encourage to read the previous projects in (Lozano Forero, 2015; Velandia Calderón & Cifuentes Cifuentes, 2015).

4.5 Tests and prototyping

I tried out many prototypes from the researched algorithms, this involves all the algorithm presented in section 5 ‘Algorithms design’ on page 43 and the ones not pre-

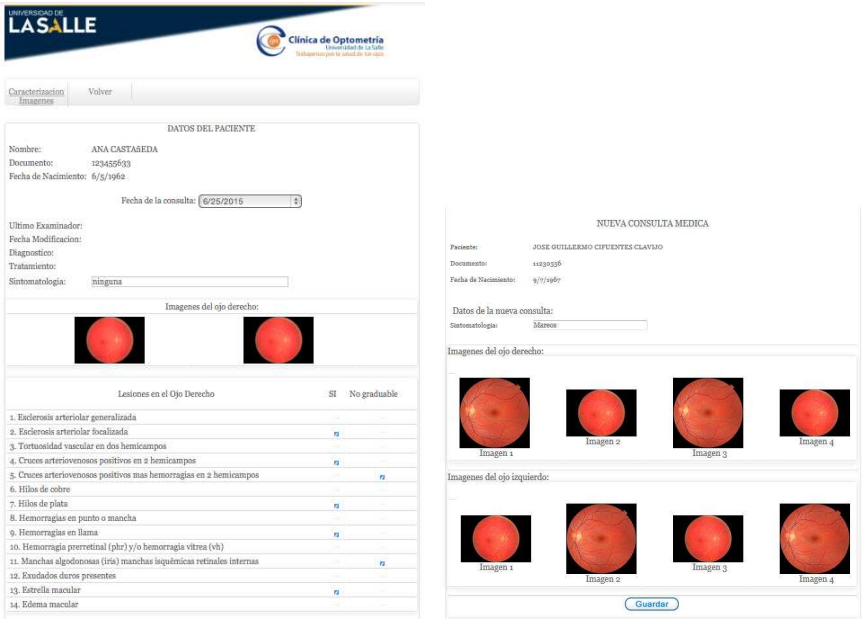
Figure 6. Platform web page

(a) Web page index (b) New pacient page

Source: (Velandia Calderón & Cifuentes Cifuentes, 2015)

sented which can be found in (Toro, 2016) under the *tests* folder with currently a total of 95 script tests without counting other files and subfolders. Algorithms that where proven useful or mature where incorporated into the *RRtoolbox* package. For example there are several testing frameworks that I used and developed to visualize results and concepts. These can be imported as `from RRtoolbox.lib.plotter import fastplt, graph_filter, plotPointsContour, Plotim, MatchExplorer, Edger, Imtester`. The `fastplt` function quickly plots any array to show its contents using matplotlib, `graph_filter` graphs the developed filters explained in subsubsection 5.3.8 'Custom filters using $normSigmoid$' on page 63 and `plotPointsContour` is used to visualize contours from image's segmentations. `Plotim` which is testing base class framework is used to visualize arrays with integrated tools called by commands that can be used on the fly in the image, the class design let to the creation of derived testing frames by sublassing like `MatchExplorer` which lets the user comfortably visualize key-points and matches between two images (see subsubsection 5.5.4 'Feature detection and matching' on page 78) with several options as to see good or bad matches and the projection of one image onto the other. Others `Plotim` subclasses include `Edger` to visualize edge methods in images while changing their parameters and `Imtester` used to visualize the implication of the

Figure 7. Managing Patient consults



(a) Patient caracterization (b) Patient consults

Source: (Velandia Calderón & Cifuentes Cifuentes, 2015)

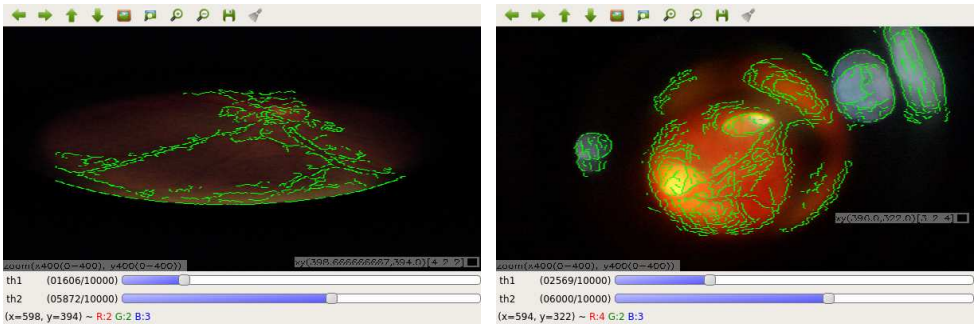
Figure 8. Adquisition of retinal images using mobile devices

Source: (Lozano Forero, 2015) and Author

segmentation methods and filters applied to images.

Using the aforementioned prototyping frameworks led to easily visualize results from the developing algorithms so that I could choose wisely with ground prof tests. Obviously, they where supported using the theory (see section 5 ‘Algorithms design’ on page 43), relying in the art state works (see section 3 ‘Art State’ on page 24), the OpenCV package and other tests like ’unittest’ to ensure integrity of codes (Python Software Foundation,

Figure 9. Experiments using edges



(a) Sample from test_edge_params.py (b) Sample from experimental_edge.py

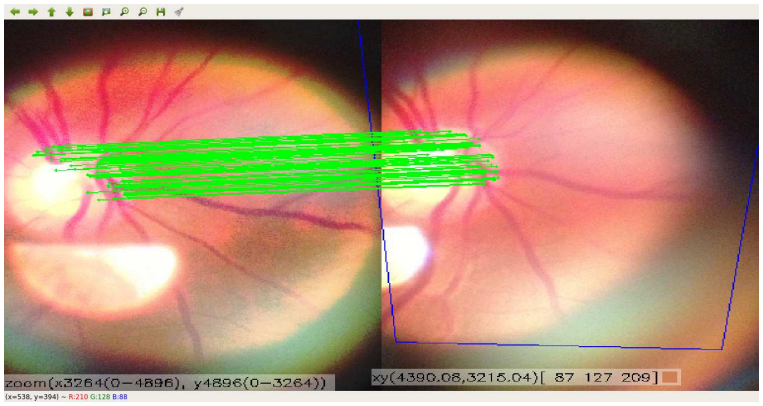
n.d.-e).

Some examples using the `Edger` class can be found in Figure 9 which are extracted from *test_edge_params.py* or *experimental_edge.py* scripts under the *tests* folder. As it is noticed, there are to bars that can be moved to achieve the ideal edge segmentation in the lower part of each image. The difference here is that *test_edge_params.py* tests the `Edger` class functionality and *experimental_edge.py* uses an expert system in a set of images to collect patter-like data (a different automatic expert system is used for the main program in subsection 6.1 ‘Expert System’ on page 101).

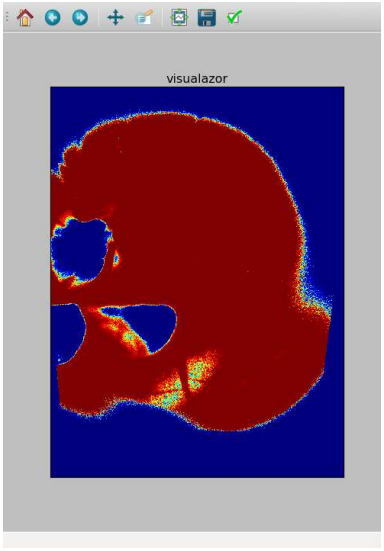
The *experimental_restoration.py* script contains experiments made while developing the structure to implement the feature descriptors algorithms provided by OpenCV (see subsubsection 5.5.4 ‘Feature detection and matching’ on page 78) extended to support the ASIFT method from (Guoshen Yu & Morel, 2011) and based from (Alexander Alekhin, n.d.). The function `asift_demo1` from *experimental_restoration.py* merges two color images as shown in Figure 10 which is already and improved version from (Alexander Alekhin, n.d.). The matched good key-points of green lines and the homography (blue trapezoid) are shown in Figure 10a with a experimental alpha mask in Figure 10b which is further explained in subsubsection 5.4.3 ‘Alpha masks’ on page 73 with the final proposed methods. Finally an experimental merging is shown in Figure 10c which is not an appealing result to say the least but that is improved using the proposed alpha method.

The function `asift_demo2` takes a step further by allowing multiple merging images to be efficiently merged to a base image while discriminating those that do not correspond

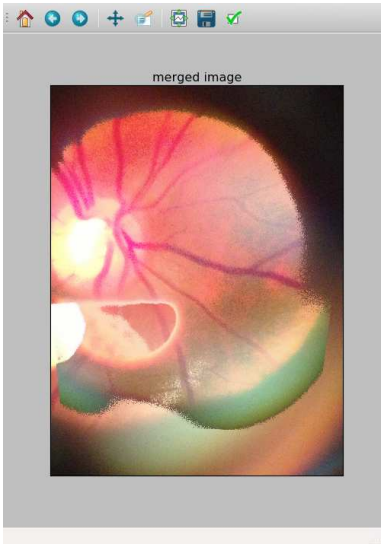
Figure 10. ASIFT demo and tests



(a) Matched images



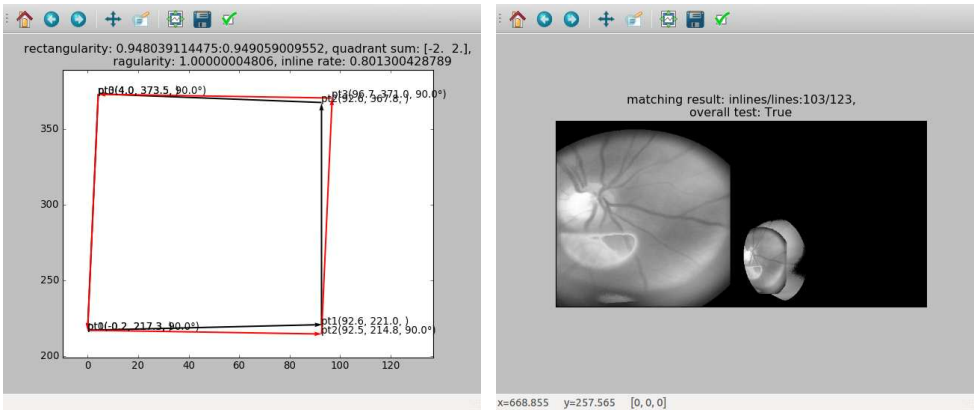
(b) Experimental alpha mask



(c) Experimental merging

to the same scenery. To achieve this several rating algorithms were developed and tested to work with the matching results so that any false image could be discarded. This was tested using the `testRates` in *experimental_restoration.py* which produces some qualification questions for the expert user so that the probability tests could be tuned and reformatted to improve on (a different automatic expert system is used for the main program in subsection 6.1 ‘Expert System’ on page 101). Evidence of this is shown in Figure 11 were the expert sees the rates (Figure 11a) and the corresponding overlay result (Figure 11b) to qualify it and be collected by an statistical system to yield the results of each developed rating method. The successful methods are discussed in subsection 5.6 ‘Using rates and

Figure 11. Experimental probability tests

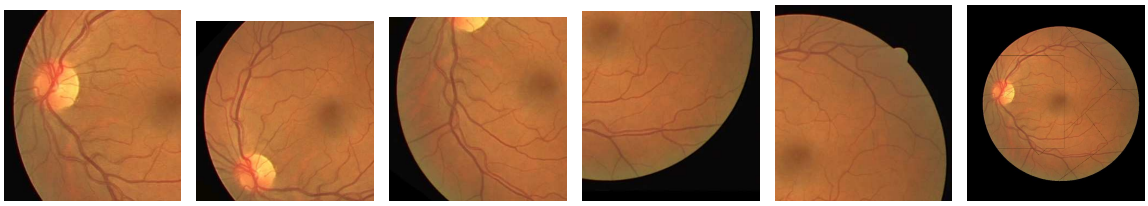


(a) Rectangularity and other rates (b) Experimental overlay

probabilities' on page 80.

Other experiments included in the 95 script tests could be shown, even failed ones, like the attempt to collect statistical data based in the histograms of many retinal images implemented in *experimental_data.py* to produce a *experimental_data_figs.xlsx* and perform manual statistical manipulation to find patters without yielding relevant important results. Nevertheless, most tests were successful and they are left out for the user to ascertain in (Toro, 2016) under the *tests* folder or by using the *RRtoolbox* package and the other implemented programs. Be aware that most of these tests in the repository are pointing to a location of the computer where the project was developed and must be changed to the desired paths. This is due to the heinous collected data that cannot be allocated in the repository (Toro, 2016) as additional information occupies more than 1.2 GiB of only images and more than 5 GiB in other type of files.

Figure 12. Result for test 1 using command '`imrestore ../results/test1/*.* --grow_scene --feature a-surf-flann --lens --overwrite --cachePath {temp}`'



(a) test1_5 (b) test1_1 (c) test1_2 (d) test1_4 (e) test1_3 (f) Restored image

4.6 Selecting algorithms and final application

The refined and selected algorithms after the prototyping discussed in subsection 4.5 ‘Tests and prototyping’ on page 36 were collected to create the so called *RRtoolbox* package where some of its algorithms are explained and demonstrated in section 5 ‘Algorithms design’ on the next page with the final implementation in section 6 ‘Implementation’ on page 88 found as the *imrestore* program. Of course, an extensive testing of the application was carried out. These tests where ideally applied using images in DRIVE database in (Staal et al., 2004) to tune the algorithm with ideal images. These tests can be obtained in the *results* folder of the repository (Toro, 2016) in sets that start with ‘test<number>’ following with a number. Figure 12 shows the results of test1 executing the command ‘`imrestore ../results/test1/*.* --grow_scene --feature a-surf-flann --lens --overwrite --cachePath {temp}`’. The folder ‘test1’ contains cropped subimages obtained from an image in the DRIVE database which can be seen from the *base image* in Figure 12a to just before the restored image in Figure 12f. This test result demonstrates the capabilities of the restoration program *imrestore*.

Real-world images are in the sets that start with ‘set<number>’ also followed by a number. These are presented in section 8 ‘Results’ on page 110 to review them in more detail with the respective validation in section 9 ‘Validation’ on page 153 using ratios generated from the expert system in subsection 6.1 ‘Expert System’ on page 101 and the statistical data generated when running the *imrestore* program.

5 Algorithms design

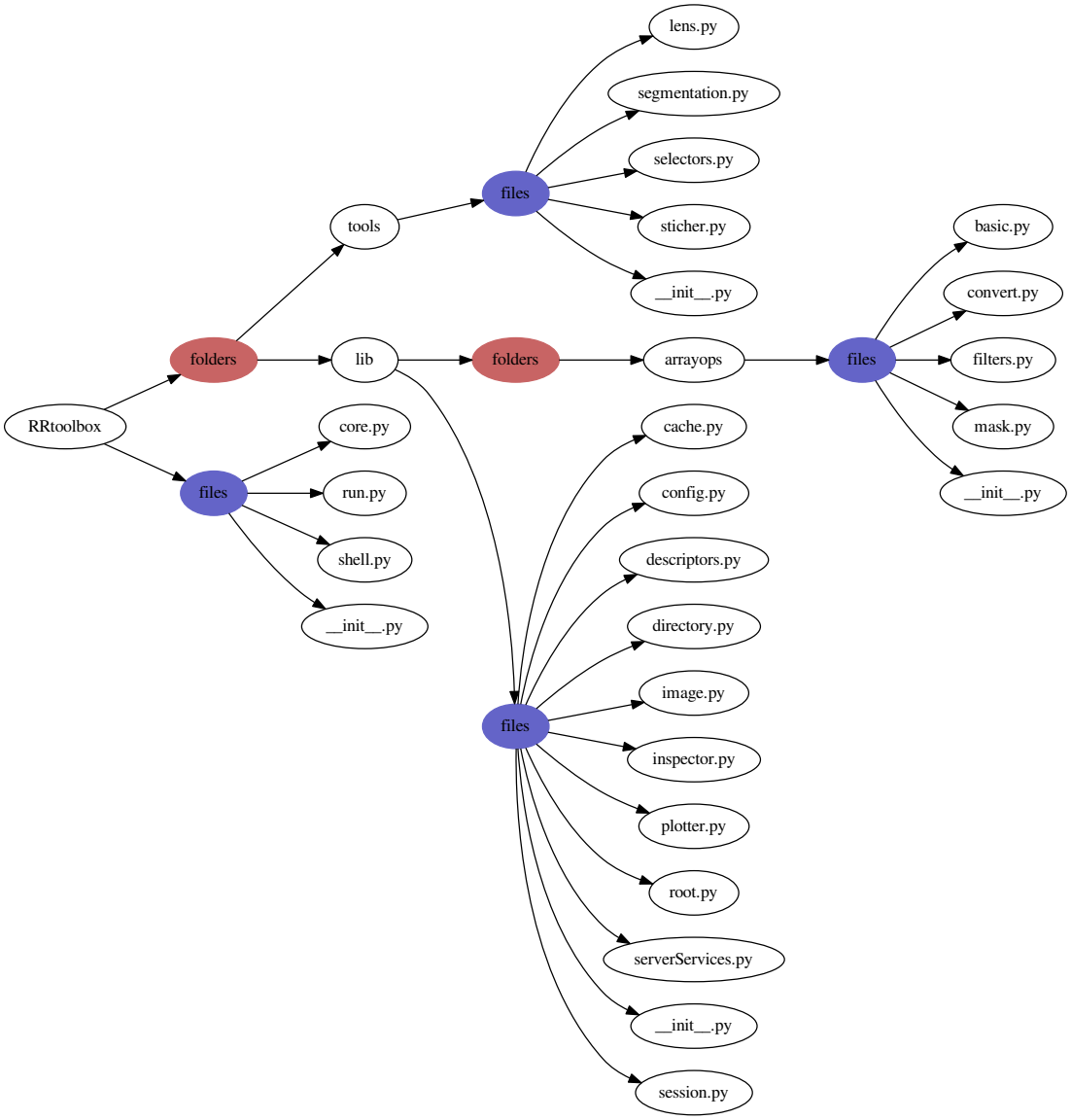
This section explains some of the algorithms used in the main implementation and is based from experiments made by me when testing the researched sources. There always emerge important questions in the development of every project like how should it be made? perhaps this peace of code is not robust enough 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 (Toro, 2016). 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 13 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 (David, 2016). 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.

5.1 General algorithms

There are many algorithms that proved 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 created to support higher level of complexity in the main algorithms.

Figure 13. File estructure of *RRtoolbox* package



5.1.1 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:

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

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 2:

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

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 14):

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

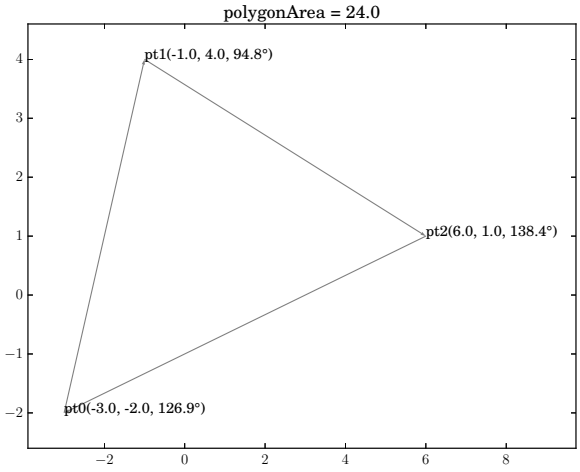
Then we solve it as in Equation 2 to obtain the individual summations.

$$\begin{aligned} ysum &= x_0 y_2 + x_1 y_0 + x_2 y_1 \\ &= (-3) \cdot 1 + (-1) \cdot (-2) + 6 \cdot 4 \\ &= 23 \\ xsum &= x_0 y_1 + x_1 y_2 + x_2 y_0 \\ &= (-3) \cdot 4 + (-1) \cdot 1 + 6 \cdot (-2) \\ &= -25 \end{aligned}$$

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

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

Figure 14. Find the area of a polygon using Equation 2



The *polygonArea* method in Equation 2 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.

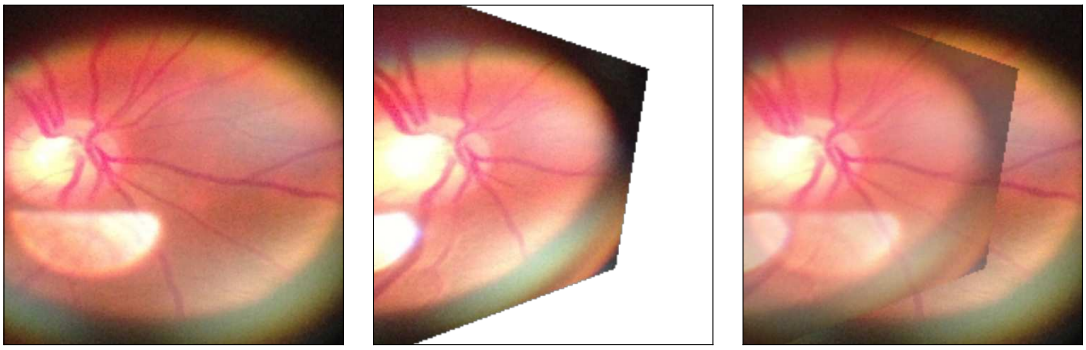
5.1.2 Overlay. This method as its name implies is used to overlay a foreground ($Fore$) over background ($Back$) array with an $alpha$ transparency which can be either and array or a number (Médioni, 2005; Szeliski, 2010). The following equation is used to with an $alpha$ array:

$$Overlay_{i,j} = Fore_{i,j} \cdot alpha_{i,j} + Back_{i,j} \cdot (1 - alpha_{i,j}) \qquad \forall \quad ||alpha_{i,j} \leq 1|| \tag{3}$$

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 3 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 15. Overlay example using Equation 3



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

Figure 15 shows an overlay example of the implemented Equation 3 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 3.

5.1.3 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 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:

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

To convert from I_r to I coordinates:

$$x(x_r),\ y(y_r) = x_1 + x_r \cdot \frac{(x_2 - x_1)}{W_r},\ y_1 + y_r \cdot \frac{(y_2 - y_1)}{H_r} \tag{5}$$

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 4 and Equation 5 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.

5.1.4 Domain transformations: scaling to original results. As explained in subsubsection 5.1.3 'Real image – Rendered image' on the preceding page, 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 subsubsection 5.1.3 'Real image – Rendered image' on the previous page) 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 (Yang, Liu, You, Li, & Zhang, 2014; Zdešar, Škrjanc, & Klančar, 2014; Y. Zhang, Zhou, Shang, Zhang, & Yu, 2016).

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

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:

$$
M_o = \begin{bmatrix} \left(\frac{Aw_s}{Aw_o}\right)\left(\frac{Bw_o}{Bw_s}\right) M_{s11} & \left(\frac{Ah_s}{Ah_o}\right)\left(\frac{Bw_o}{Bw_s}\right) M_{s12} & \left(\frac{Bw_o}{Bw_s}\right) M_{s13} \\ \left(\frac{Aw_s}{Aw_o}\right)\left(\frac{Bh_o}{Bh_s}\right) M_{s21} & \left(\frac{Ah_s}{Ah_o}\right)\left(\frac{Bh_o}{Bh_s}\right) M_{s22} & \left(\frac{Bh_o}{Bh_s}\right) M_{s23} \\ \left(\frac{Aw_s}{Aw_o}\right) M_{s31} & \left(\frac{Ah_s}{Ah_o}\right) M_{s32} & M_{s33} \end{bmatrix} \tag{6}
$$

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 6 can be imported in Python as `from RRtoolbox.lib.arrayops.convert import sh2oh`.

5.1.5 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:

$$
N(A) = \frac{A - min(A)}{max(A)} \tag{7}
$$

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.

5.2 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`.

5.2.1 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.

5.2.2 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.

5.2.3 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.

5.2.4 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).

5.3 Pre-processing, Filters and Enhancing Methods

5.3.1 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 et al., 2004, section 3.3.2).

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

Code 1: Histogram equalization MWE

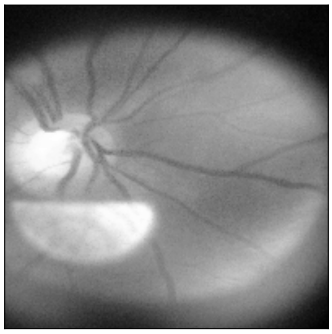
```
import cv2
import numpy as np
img = cv2.imread('image.jpg',0) # load the image
hist,levels = np.histogram(img.flatten(),256,[0,256]) # get histogram
cdf = hist.cumsum() # get CDF by accumulating histogram
cdf_m = np.ma.masked_equal(cdf,0) # mask to convert to range [0,1]
cdf_m = (cdf_m - cdf_m.min())*255/(cdf_m.max()-cdf_m.min()) # 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
```

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).

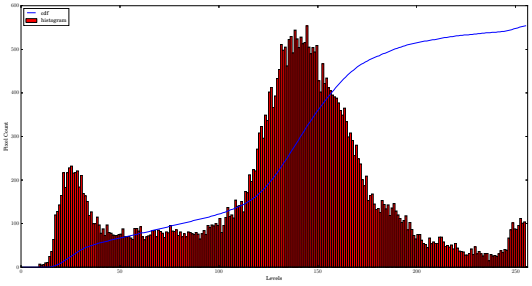
In general *HE* techniques are used in pre-processing operations to usually normalize the image patterns and lighting conditions (Médioni, 2005) but it has been observed that it intensifies noise (Antal & Hajdu, 2012) and for that noise attenuation is recommended before using *HE* (Sopharak, Uyyanonvara, & Barman, 2013). There are other works that use CLAHE as an enhancing method (Ramlugun, Nagarajan, & Chakraborty, 2012; Sopharak et al., 2013) or algorithms derived from it (Kumar, Deepak, Sathar, Sahasranamam, & Kumar, 2016) as it is very effective emphasizing locally salient values at the cost

of small noise (Antal & Hajdu, 2012). For further optimizations using C++ and CUDA to process in the computer's GPU you can follow the advices in (Fierval, 2015).

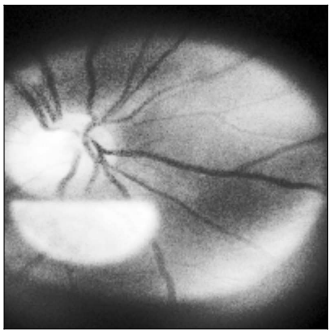
Figure 16. Histogram equalization example



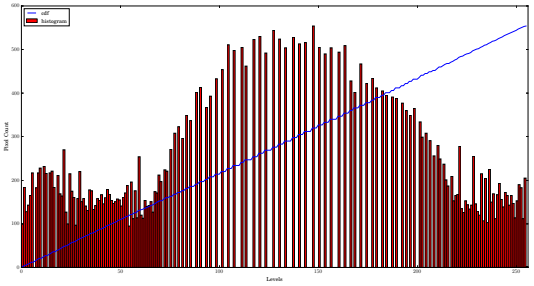
(a) Input



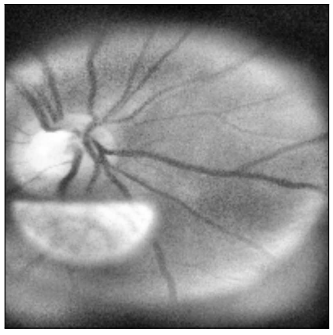
(b) Histogram of input



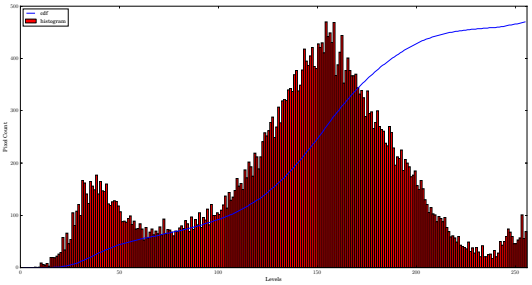
(c) global Equalization



(d) Histogram of global Equalization



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

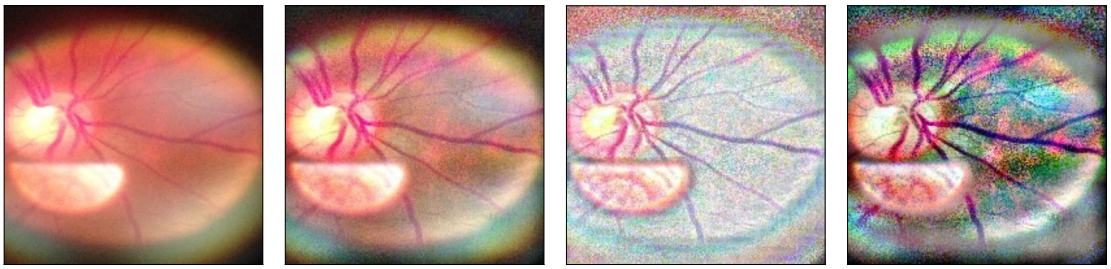


(f) Histogram of CLAHE with clipLimit=2.0 and tileGridSize=(8,8)

Figure 16 shows a *HE* and *CLAHE* comparison with their respective histogram of colors. Notice the input image (Figure 16a) and how the *HE* equalizes the input (Figure 16c) by converting its non-linear CDF (Figure 16b) in an uniform CDF (Figure 16d). This

leads to some uneven parts in the image but *CLAHE* solves it as explained before (Figure 16e).

Figure 17. 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

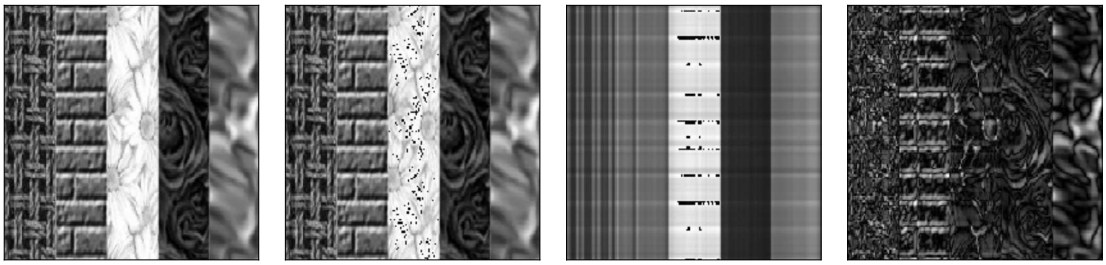
Figure 17 shows additional experiments with the $CLAHE$ method using color images. The default application with parameters $tileGridSize = (8, 8)$ and $clipLimit = 2.0$ is found in Figure 17b. In Figure 17c the parameters $tileGridSize = (30, 30)$ and $clipLimit = 2.0$ are used and in Figure 17d changed 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.

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

Figure 18 shows the two resulting decomposition tests for a gray image with different patterns and intensities (Figure 18a). 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 18b) while with slice s[1:]=0 produces an background-like image without patterns (Figure 18c) and with slice s[:1]=0 only the patterns or edges of the original image are preserved (Figure 18d).

Figure 18. 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

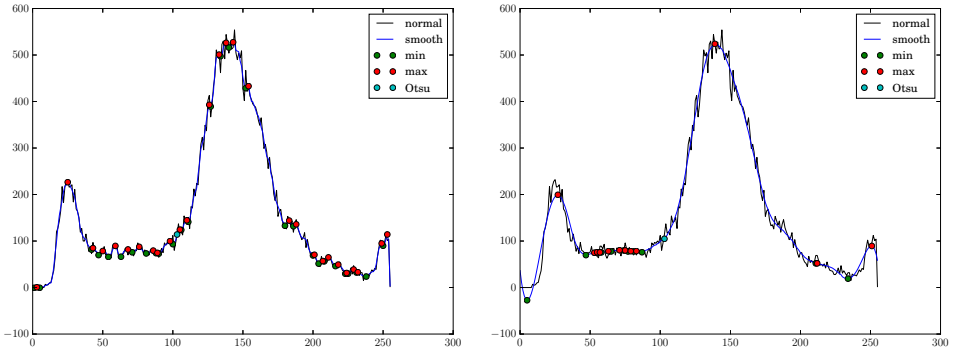
There are others works that use different decompositions like wavelet decomposition (Kumar et al., 2016; Giancardo et al., 2012; Garcia-Fidalgo & Ortiz, 2015), Kronecker decomposition (Bouhamidi, Enkhbat, & Jbilou, 2014) and other variants derived from its concept (Faust et al., 2012; Collet, Berenson, Srinivasa, & Ferguson, 2009; Akram, Khalid, & Khan, 2013; Muthu Rama Krishnan Mookiah et al., 2013; Imani et al., 2015; Fraz, Remagnino, et al., 2012) included the *Singular Value Decomposition* (SVD) (Bouhamidi et al., 2014; Yang et al., 2014; Médioni, 2005). There is a sparse matrix decomposition technique used to decompose features of images into a low-rank matrix and a noise matrix. By doing this images with arbitrary large magnitudes of noise can be separated from the original to encode the histogram information of the images (C. Zhang et al., 2014).

5.3.3 Smoothing with 1D-filters. There is an filter algorithm in (SciPy, n.d.-b) 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 19 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 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 19b). 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 19. Savgol filter



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

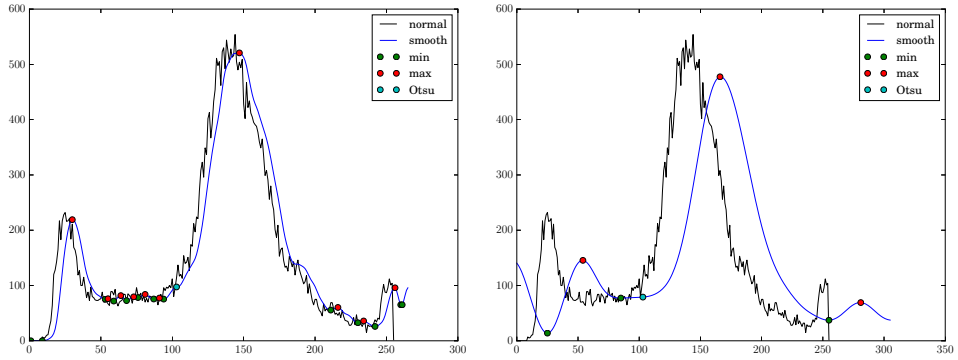
(b) Savgol with $window_{len} = 51$ and polyorder = 5

Figure 20 shows the filter as implemented in (SciPy, n.d.-b) 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 21 shows the improved filter using the Hanning window with shifted convolution to correct filtered data offset presented in Figure 20. 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`.

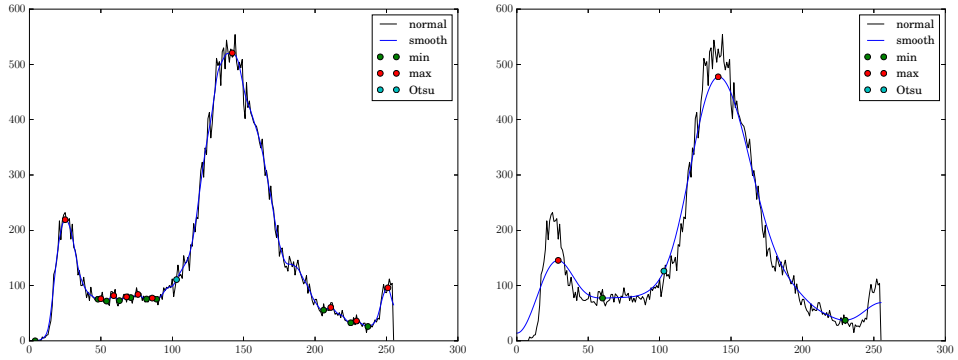
5.3.4 Gaussian Filter. It is one of the most widely used filters to reduce Gaussian noise in images (Yuan et al., 2014; Fraz, Barman, et al., 2012; Yasukawa et al., 2016; Zhu, Bichot, & Chen, 2013; Jiménez et al., 2011) but it produce a blur-like effect

Figure 20. Filter with Hanning window



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

Figure 21. Filter with Hanning window and shifted convolution



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

that eliminates details along with its application (Paris, Kornprobst, Tumblin, & Durand, 2008). 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; Sinha, n.d.). There are even many derivations and applications from it (Akram et al., 2013; Li, Liu, Li, Huang, & Li, 2014; G. Wang, Wang, Chen, & Zhao, 2015; Usman Akram, Khalid, Tariq, Khan, & Azam, 2014; Chang & Tseng, 2010).

5.3.5 Bilateral Filter. The bilateral filter can reduce unwanted noise very well while keeping edges fairly sharp (Dai, Han, Wu, & Gong, 2007). 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, this was confirmed in the experiments. In the other hand, this filter does very well in

applications where the 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 et al., 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 $sigmaSpace$.
- $sigmaColor$– Filter sigma in the color space. A larger value of the parameter means that farther colors within the pixel neighbourhood (see $sigmaSpace$) will be mixed together, resulting in larger areas of semi-equal color.
- $sigmaSpace$ – 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 $sigmaSpace$. Otherwise, d is proportional to $sigmaSpace$.

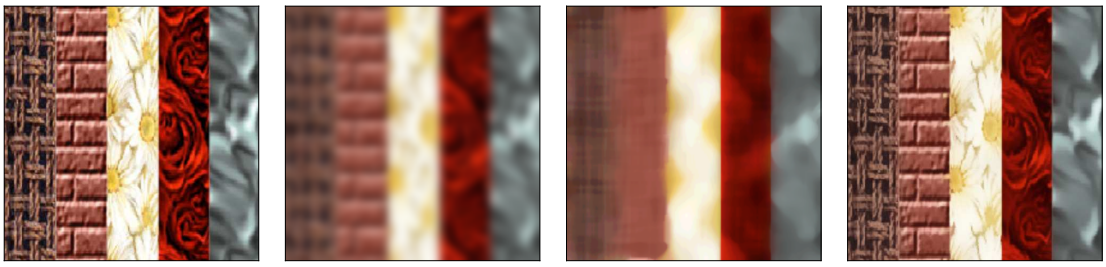
Filter size. Large filters (d>5) are very slow, so it is recommended to use d=5 for real-time applications, and perhaps d=9 for off-line applications that need heavy noise filtering.

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 22 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 22b) produces a blurry-like result from the original (Figure 22a) and the Mean filter eliminates most of the edges from it (Figure 22c) while the bilateral filter preserves the edges and retains most of the features (Figure 22d)

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

Figure 22. Spacial filters comparison



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

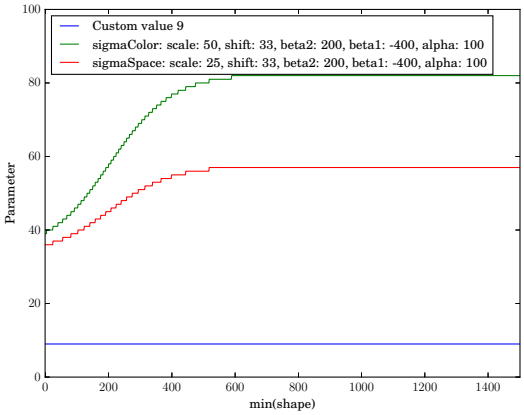
some tests were made using the filter and getting good parameters to make a selector function for the correct bilateral filter parameters based in the image shape and the 'mild', 'normal' and 'heavy' modes of noise in the image. Figure 23 shows the selection for the parameters in the Bilateral filter 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 24 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 23 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.

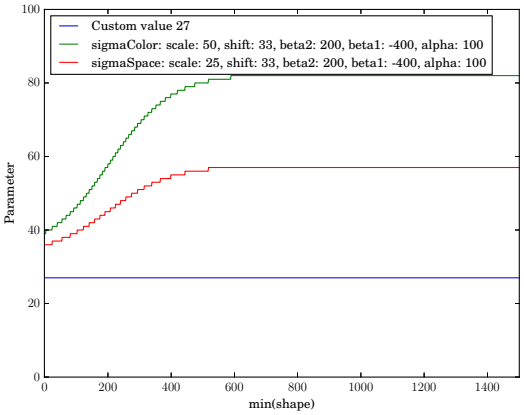
5.3.6 SigmoidImageFilter. The sigmoid filter is based in the sigmoid function which is characterized by having an "S" shape with a dampening behavior when tending to $-\infty$ and an approximation to the value "1" when tending to ∞ (i.e. an adequate offset can be used to determine the threshold to damper or let through certain values). In particular this filter is based 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:

$$Sigmoid(x_i, \alpha, \beta, Max, Min) = (Max - Min)\left(\frac{1}{1 + e^{-(x_i - \beta)/\alpha}}\right) + Min \tag{8}$$

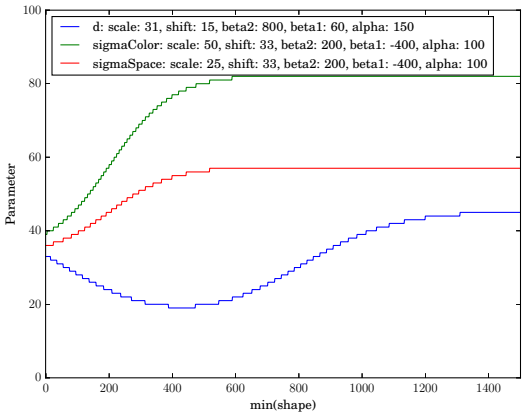
Figure 23. Selection of bilateral parameters according to image shape



(a) Bilateral parameters in 'mild' mode



(b) Bilateral parameters in 'normal' mode



(c) Bilateral parameters in 'heavy' mode

where Max is the maximum value at the output, Min is the minimum value at the output, β the offset and α a scaling factor. The β 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 β value that is around that value and the α parameter can be thought of as the scaling or variance of the sigmoid (CMISS, n.d.; Akram et al., 2013; Usman Akram et al., 2014).

5.3.7 Normalized SigmoidImageFilter. I normalized the sigmoid filter extracting it as a component of Equation 8 that computes any x value (for discrete it is x_i) with an offset β and a scaling factor α that its result is inside the range of $[0, 1]$. This is:

Figure 24. Bilateral filter exposed to different shapes

(a) Input shape = (50, 50)

(b) Filtered with d = 31, sigmaC = 42, sigmaS = 37 and time = 0.063 secs

(c) Input shape = (100, 100)

(d) Filtered with d = 28, sigmaC = 46, sigmaS = 39 and time = 0.091 secs

(e) Input shape = (600, 600)

(f) Filtered with d = 22, sigmaC = 82, sigmaS = 57 and time = 0.637 secs

(g) Input shape = (1500, 1500)

(h) Filtered with d = 45, sigmaC = 82, sigmaS = 57 and time = 13.080 secs

$$nS_i = normSigmoid(x_i, \alpha, \beta) = \frac{1}{1 + e^{(\beta - x_i)/\alpha}} \qquad \forall \quad x_i, \alpha, \beta \in \mathbb{R} \quad \wedge \quad 0 \leqslant nS_i \leqslant 1 \tag{9}$$

which stabilizes at the limits:

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

This facilitates the calculation of α and β. For example, solving for α yields:

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

And solving for β yields:

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

Also, Equation 8 can be expressed in therm of the Equation 9 as follows:

$$Sigmoid\,(x, \alpha, \beta, Max, Min) = (Max - Min) * normSigmoid\,(x, \alpha, \beta) + Min \tag{10}$$

which stabilizes at the limits:

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

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

5.3.8 Custom filters using *normSigmoid*. Observing that the $normSigmoid$ function is similar to a Butterworth filter I decided to use it 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 7).

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

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

$$Highpass(x, \alpha, \beta) = normSigmoid(x, \alpha, \beta) \qquad \forall \quad \alpha > 0 \tag{12}$$

$$Bandstop(x,\alpha,\beta) = Lowpass(x,\alpha,\beta_1) - Lowpass(x,\alpha,\beta_2) + 1 \qquad \forall \quad \beta_1 > \beta_2 \qquad (13)$$

$$Bandpass(x,\alpha,\beta) = Highpass(x,\alpha,\beta_1) - Highpass(x,\alpha,\beta_2) \qquad \forall \quad \beta_1 > \beta_2 \qquad (14)$$

$$Bandstop_{Inverted}(x,\alpha,\beta) = Lowpass(x,\alpha,\beta_2) - Lowpass(x,\alpha,\beta_1) - 1 \qquad \forall \quad \beta_1 > \beta_2 \qquad (15)$$

$$Bandpass_{Inverted}(x,\alpha,\beta) = Highpass(x,\alpha,\beta_2) - Highpass(x,\alpha,\beta_1) \qquad \forall \quad \beta_1 > \beta_2 \qquad (16)$$

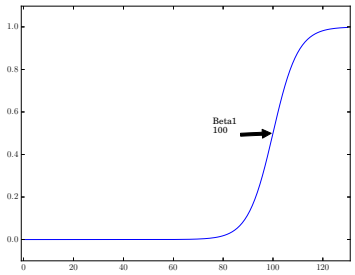
Figure 25 shows the response of each normalized filter and Figure 26 the comparison among them. Notice how all the filters converge if they share the same α and β parameters. These filters are imported using `from RRtoolbox.lib.arrayops import FilterBase, Lowpass, Highpass, Bandstop, Bandpass, InvertedBandstop, InvertedBandpass, filterFactory`. The additional import, `ffilterFactory`, is a function used to create the necessary filter from the provided arguments.

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 27. 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, deleter, getter and setter properties. As the normalized filters suggest the color levels in the image I are saturated when passed through them:

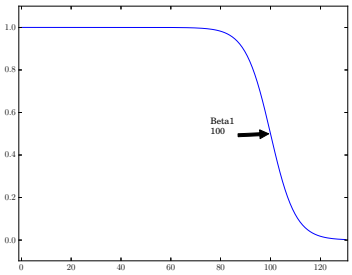
$$filteredI_{saturated} = filter_{normalized}(I)$$

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:

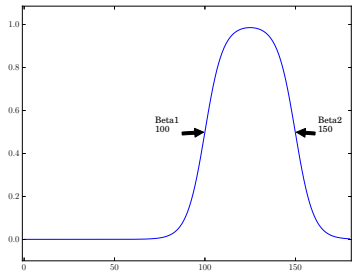
Figure 25. Common filters response using $normSigmoid$



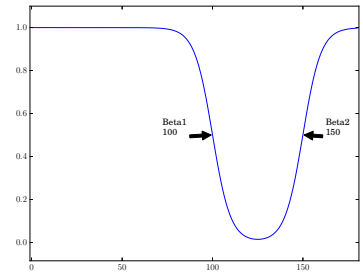
(a) Highpass with alpha=5, beta1 = 100and beta2=None



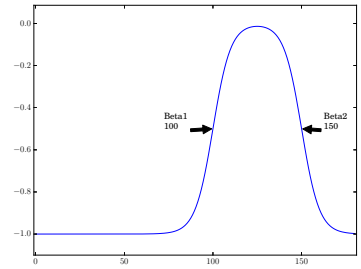
(b) Lowpass with alpha=5, beta1 = 100and beta2=None



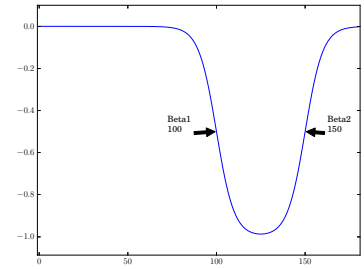
(c) Bandpass with alpha=5, beta1 = 100and beta2=150



(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

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

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

Figure 28 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.

5.3.9 Sigmoid filtering and saturation.

Figure 26. Common filters comparison

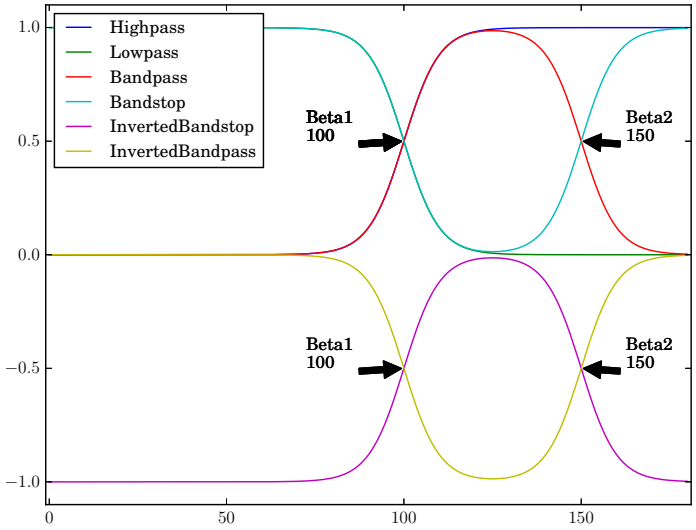
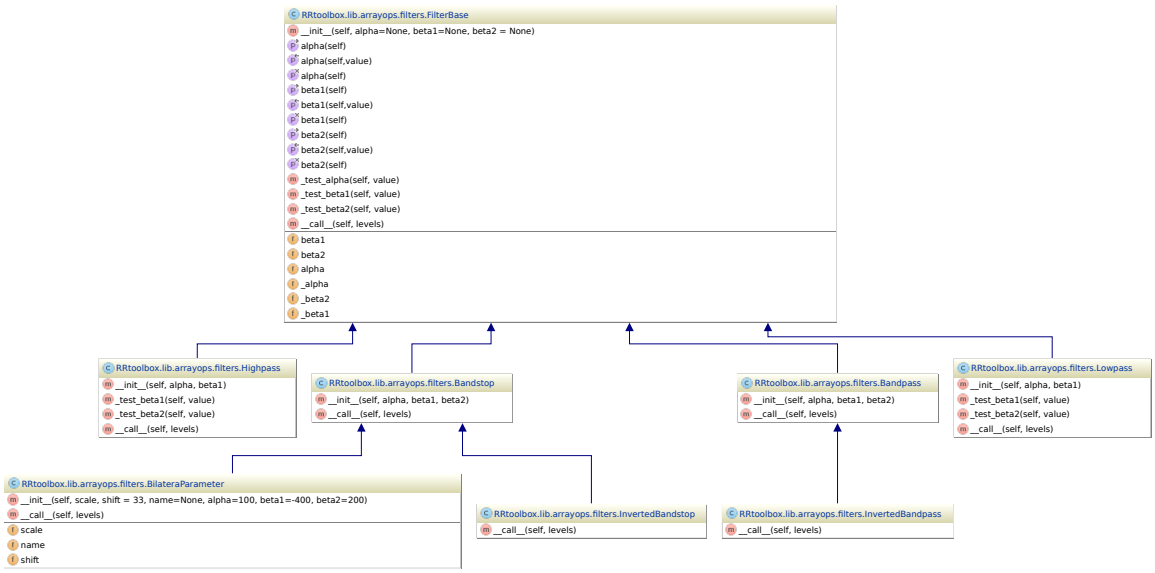


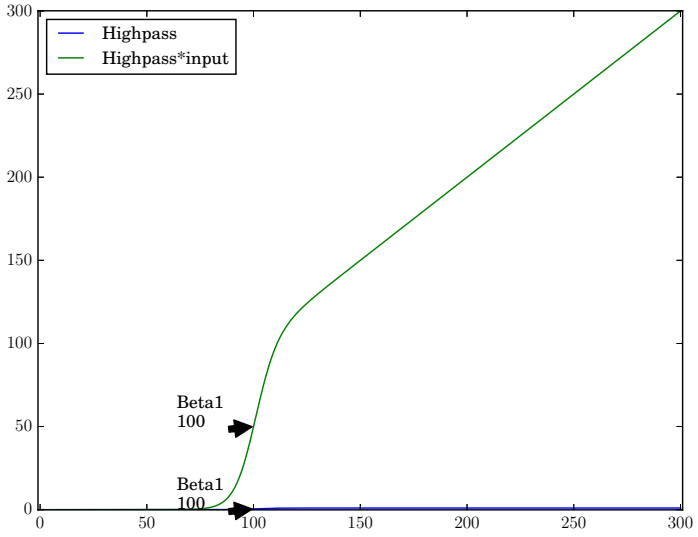
Figure 27. Filters class diagrams



5.4 Segmentations

The segmentation is a fundamental procedure taken when processing images and it has become in a common step in this field used in many works (Kallel et al., 2014; Moghimirad, Hamid Rezatofighi, & Soltanian-Zadeh, 2012; Staal et al., 2004; Nguyen, Bhuiyan, Park, & Ramamohanarao, 2013; Fathi & Naghsh-Nilchi, 2013; Jan, Odstrcilik, Gazarek, & Kolar, 2012; Fraz, Barman, et al., 2012). Several segmenting methods

Figure 28. Custom filtering without saturation



were taken into account (Reis et al., 2015; Szeliski, 2010; Gonzalez et al., 2004) 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 of my developed 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 for an specific problem that wants to be solved.

5.4.1 Convex hull with line cuts. This is a developed method which 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. This can be compared to (Szeliski, 2010, 5.4 Normalized cuts) if the area of the objects are seen as clusters instead of a whole and of course it is a region splitting method.

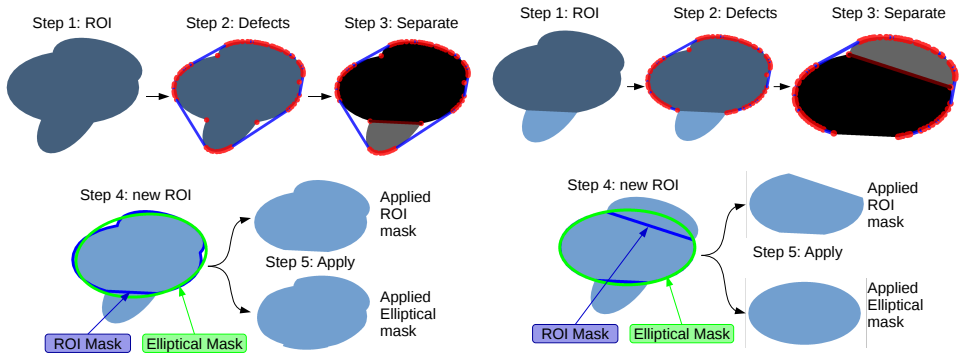
The iterative splitting of the object can be stopped using the convexity ratio $r_{convexity}$ explained in the subsubsection 5.6.1 ‘Convexity ratio’ on page 80.

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. The pseudo code is as shown in Algorithm 1.

Algorithm 1 Split binary mask using defects

procedure SPLIT_BY_DEFECTS($bImage$) ▷ Input binary mask to split by defects
  $contours \leftarrow$ cv2.findContours($bImage$) ▷ list of contours for each object
  $cnt \leftarrow$ biggestCnt($contours$) ▷ only contours of the biggest object
  $cnt_{area} \leftarrow$ cv2.contourArea(cnt)
  while True **do**
    $hull \leftarrow$ cv2.convexHull(cnt) ▷ get contours of the convex hull
    $C_r \leftarrow cnt_{area}$ / cv2.contourArea($hull$) ▷ calculate the convexity ratio
    if $C_r > C_{rdesired}$ **then** ▷ when desired ratio is reached
      break
    end if
    $defects \leftarrow$ convexityDefects(cnt, $hull$) ▷ contours difference of cnt and $hull$
    ▷ To split cnt by the two most biggest distances
    $distances \leftarrow$ distances from $defects$
    $twoMax \leftarrow$ get the two maximum values from $distances$
    $twoIndexes \leftarrow$ use $twoMax$ and $defects$ to get the equivalent indexes in cnt
    ▷ Side A: slice of cnt
    for i $\leftarrow$ min($twoIndexes$) **to** $index_{right} - 1$ **do**
      $sideA_i \leftarrow cnt_i$
    end for
    ▷ Side B: the other slice of cnt
    for i $\leftarrow$ 0 **to** $index_{left} - 1$ **do**
      $sideB_i \leftarrow cnt_i$
    end for
    for i $\leftarrow$ max($twoIndexes$) **to** len(cnt)-1 **do**
      $sideB_i \leftarrow cnt_i$
    end for
    ▷ Get biggest side and discard little side
    $areaA \leftarrow$ cv2.contourArea($sideA$)
    $areaB \leftarrow$ cv2.contourArea($sideB$)
    if $areaA > areaB$ **then**
      $cnt \leftarrow sideA$
      $cnt_{area} \leftarrow areaA$
    else
      $cnt \leftarrow sideB$
      $cnt_{area} \leftarrow areaB$
    end if
  end while
  $bImage_{new} \leftarrow$ from cnt contours make a binary image
  return $bImage_{new}$ ▷ The biggest split binary image
end procedure

Figure 29. Ideal threshold with defect lines

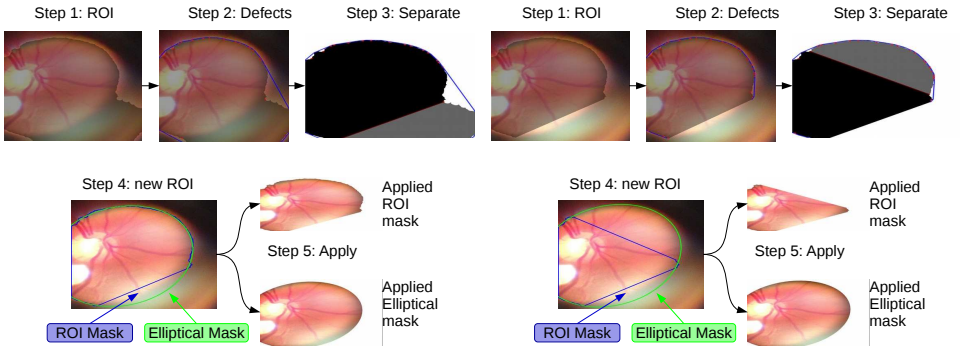


(a) Iteration 0: $C_r = 0.945903192491$ (b) Iteration 1: $C_r = 0.987807853408$

This concept in *tests / hypothesis2_5_masks_defect_lines2.py* is demonstrated using a model object as in Figure 29. 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 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 30. 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 30 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).

5.4.2 Binary masks. I tested common methods to threshold images to produce binary mask and use them 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 31. Histogram analysis to find threshold values to use in watershed method

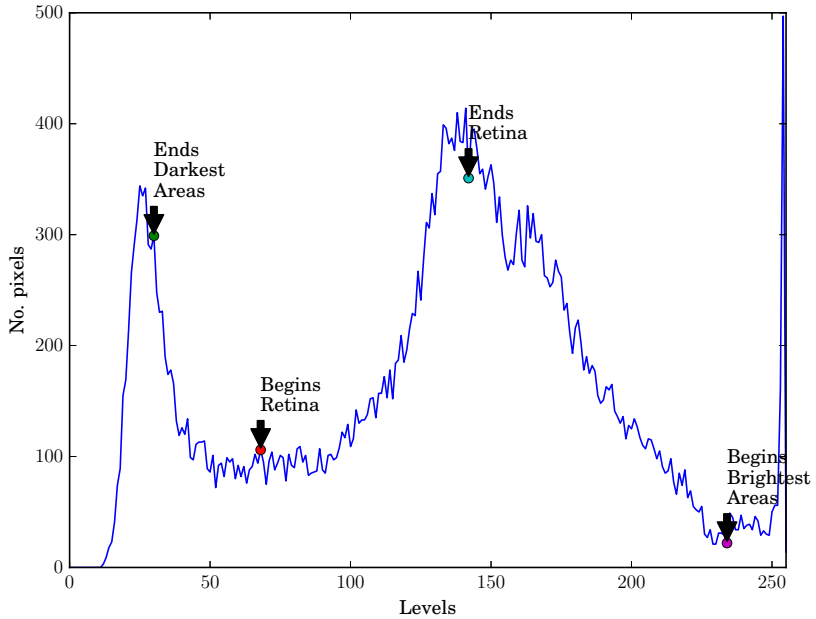
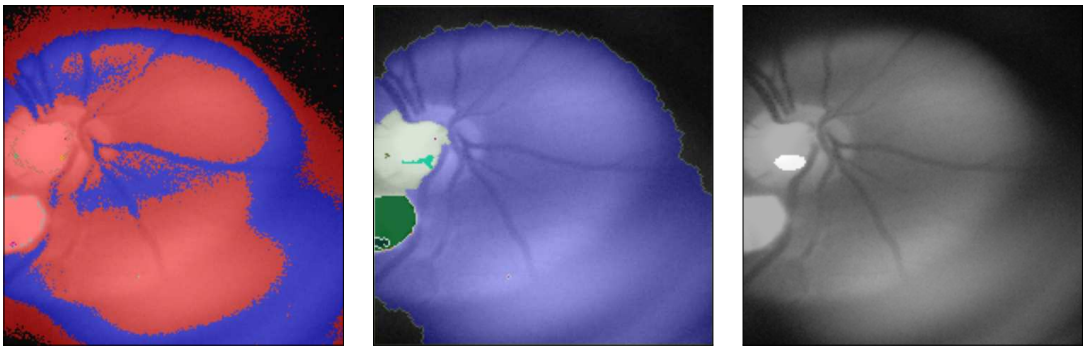


Figure 31 shows a case with the result of the automatic histogram analysis applied to provide the necessary information in the watershed method. The result of feeding these parameters in the watershed method are shown in Figure 32. In Figure 32a the seeds are placed in the sample image which produce the watershed in Figure 32b. 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 32c).

Figure 32. Brightest areas in image using watershed method



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

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.

Proposed general 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 33 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.

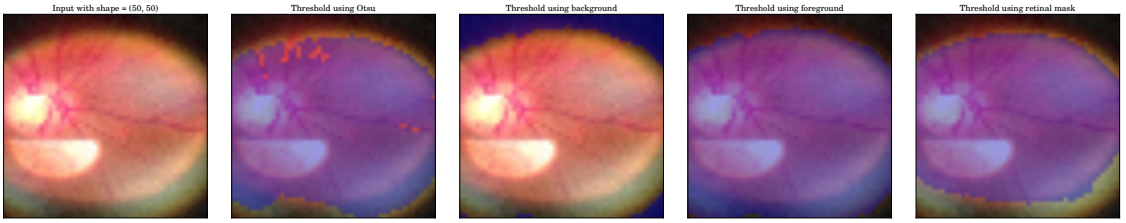
Developed method to segment retinal area. There is another developed algorithm that can find 'solid' objects in an image without getting confused by blurry areas. This algorithm is called `layeredfloods` because it uses a weighted sum with the flooding algorithm but it has the condition of using color 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 subsubsection 5.4.3 'Alpha masks' on the current page. 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 33 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.

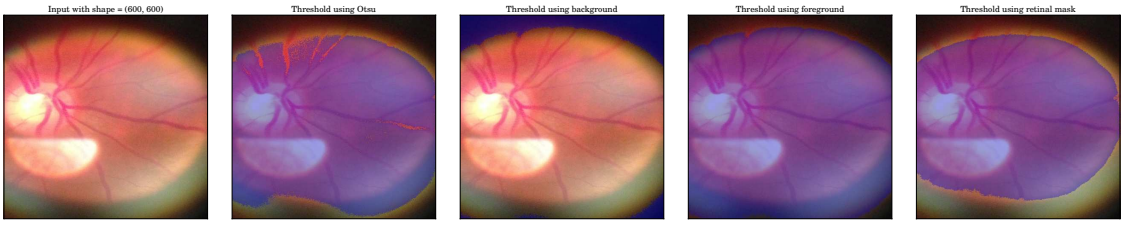
5.4.3 Alpha masks.

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. I decided to use a better way which can be offered using Alpha masks that present 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.

Figure 33. Threshold comparisons



(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

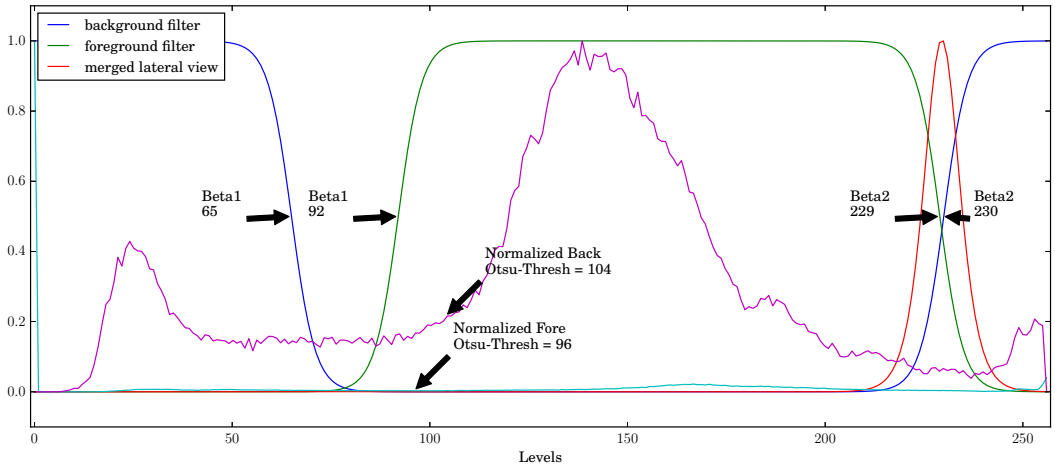
Alpha masks add a layer of information to images by appending an additional dimension to its array. Then BGR images must be transformed to BGRA or RGB to RGBA corresponding to the Red, Green, Blue and Alpha channels used in the image. This is usually used in the PNG image format allowing its images to have transparency when overlying with others to create composites. A similar example was shown in Figure 15 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.

Proposed alpha mask using filters. The subsubsection 5.3.8 'Custom filters using $normSigmoid$' on page 63 explained the use of the $normSigmoid$ in Equation 9 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 7 from the subsubsection 5.1.5 ‘Normalization’ on page 50 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 34 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.

Figure 34. Alpha mask histogram analysis and filters response

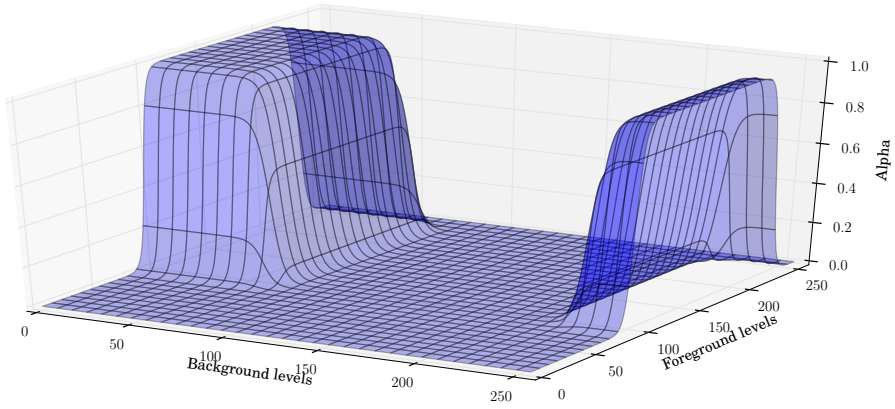


A clearer representation of what the Code 3 does is shown in Figure 35 with the 3D filter response of the simulated 0-255 grey levels of the background and foreground images respectively instead of the actual real images. This demonstrates that the algorithm in Code 3 acts in the color channels and not 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 34.

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 37c.

Figure 35. Alpha mask obtained using filters



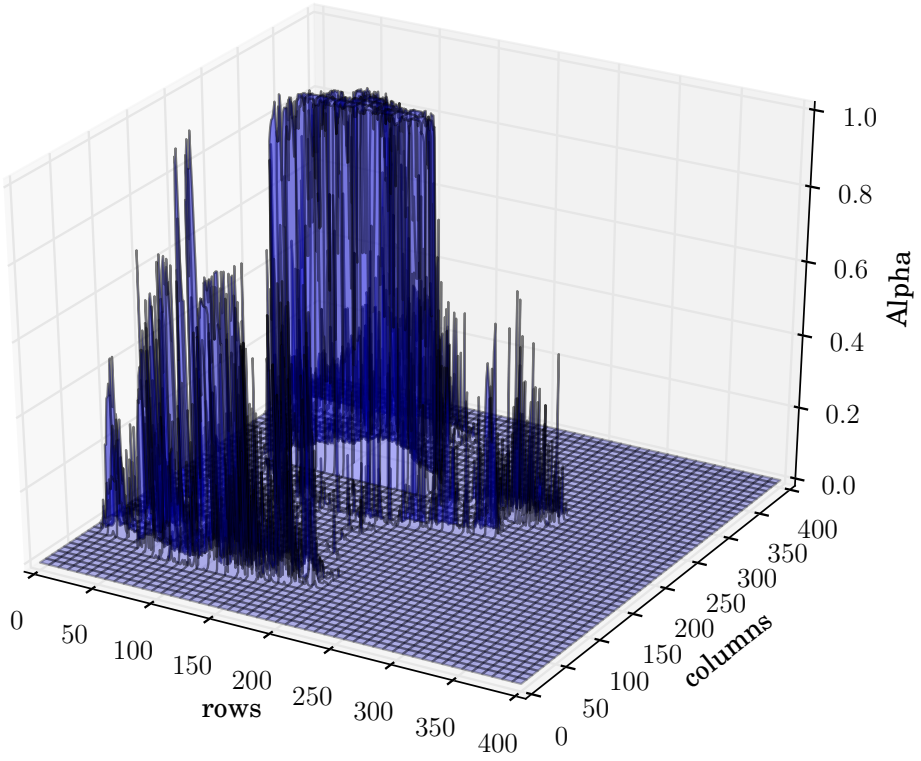
Proposed alpha mask for the space domain. There is an sturdier method using the `layeredfloods` algorithm and it is implemented in the *RRtoolbox* package. This algorithm can be imported using `from RRtoolbox.tools.segmentation import get_layered_alpha`. Figure 36 shows the 3D representation of the alpha 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 37d.

5.5 Object Recognition and Matching algorithms

In this section I explain some of the algorithms used to match patterns, colors, images and features that are included in the*RRtoolbox* package to develop the *imrestore* program.

5.5.1 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

Figure 36. Alpha mask obtained using layered masks

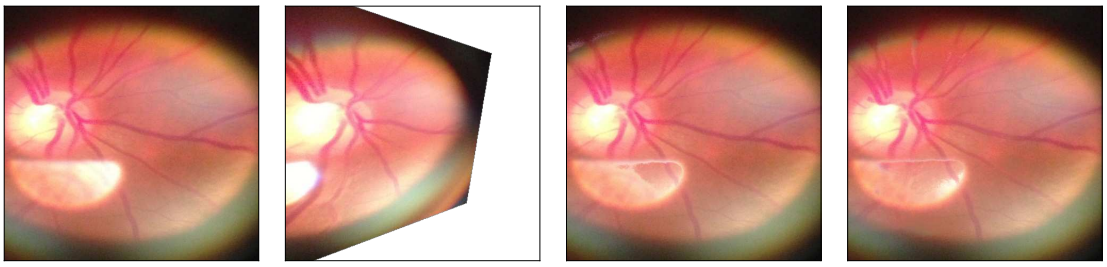


imported from the *RRtoolbox* package using `from RRtoolbox.tools.selectors import entropy` and Figure 38 shows an example sorting images using this function to classify a set of unfocused images of the same object.

5.5.2 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 (Toro, 2016; David, 2016). Import it for use in Python with `from RRtoolbox.tools.selectors import hist_comp`. Figure 39 shows an application sorting images using the histogram comparison algorithm.

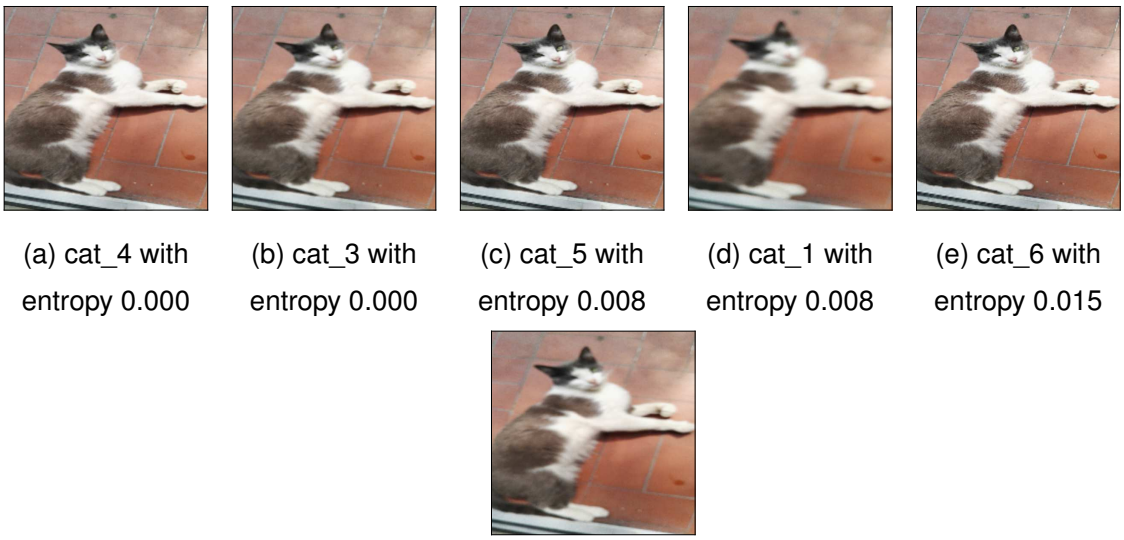
5.5.3 Histogram matching. The purpose of this algorithm is as its name indicates match the histogram of two image to convert the color of one images to the other. This algorithm was conceived trying to recreate imhistmatch function from MATLAB (MathWorks, n.d.-a) and based from an stackoverflow answer in (Ali_m, n.d.). Import the created algorithm with `from RRtoolbox.lib.image import hist_match` or see the source

Figure 37. Overlay example using alpha masks



(a) background (b) foreground (c) Overlay with alpha mask in Figure 35 (d) Overlay with alpha mask in Figure 36

Figure 38. Ordered images using entropy



(a) cat_4 with entropy 0.000 (b) cat_3 with entropy 0.000 (c) cat_5 with entropy 0.008 (d) cat_1 with entropy 0.008 (e) cat_6 with entropy 0.015

(f) cat_2 with entropy 0.016

code and documentation in the repository (Toro, 2016; David, 2016). Additional explanations found in (Gonzalez et al., 2004, section 3.3.3). Figure 40 shows an example using the matching algorithm.

5.5.4 Feature detection and matching. Feature detection and matching are considered to be essential in many computer vision applications which are under the realm of object recognition. In this case I use it to align two or more images so that they can be stitched into a composite mosaic or can be merged as in image registra-

tion (Szeliski, 2010, Chapter 4). 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 (M. 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.

For the matching of the images I discarded the regular least squares method and chose the more robust version *RANdom SAmple Consensus* (RANSAC) required for the presence of outliers generated among the matched images (Szeliski, 2010, 6.1.4 Robust least squares and RANSAC)(Liu & Yu, 2015). It can be looked into other improved matcher techniques like PROSAC, GroupSAC and iRANSAC but they are outside our scope (Z. Wang, Kieu, Nguyen, & Le, 2015).

Figure 41 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.

5.5.5 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 (Guoshen 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`.

5.6 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 used in the developed program. 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 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`.

5.6.1 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 (James R. Bozeman, Matthew Pilling, Bozeman, & Pilling, 2013, Definition 5). This is what the convexity ratio $r_{convexity}$ does:

$$r_{convexity} = \frac{Area_{object}}{Area_{hull}} \tag{17}$$

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 2 found in subsubsection 5.1.1 ‘The area under a polygon ($poligonArea$)’ on page 44. 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 42 shows an example using the convexity ratio equation. In this example the object points are:

$$object_{points} = \begin{bmatrix} 6.0 & -3.0 \\ 4.0 & 1.0 \\ 4.0 & 4.0 \\ 0.0 & 3.0 \\ -5.0 & 5.0 \\ -6.0 & 3.0 \\ -7.0 & -4.0 \\ -3.0 & -4.0 \\ 0.0 & -6.0 \\ 1.0 & -3.0 \end{bmatrix}$$

and the points from its Convex hull:

$$hull_{points} = \begin{bmatrix} 6.0 & -3.0 \\ 4.0 & 4.0 \\ -5.0 & 5.0 \\ -6.0 & 3.0 \\ -7.0 & -4.0 \\ 0.0 & -6.0 \end{bmatrix}$$

then using Equation 17 and Equation 2 yields:

$$\begin{aligned} r_{convexity} &= \frac{Area_{object_{points}}}{Area_{hull_{points}}} \\ &= \frac{86.0}{107.0} \\ &= 0.803738317757 \end{aligned}$$

5.6.2 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.

$$rectangularity = \frac{Area_{object}}{Area_{rectangular}} \tag{18}$$

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 43.

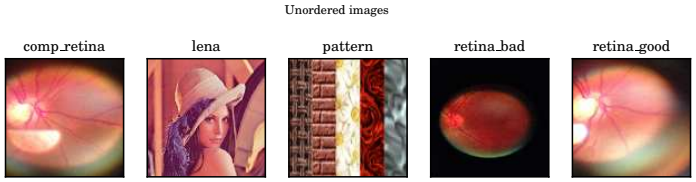
Code 2: Decomposition MWE

```
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
```

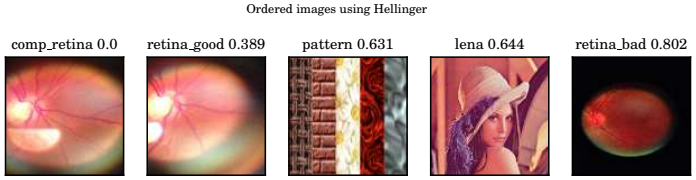
Code 3: MWE to create alpha mask with filters

```
# 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)
```

Figure 39. Ordered images using histogram comparison

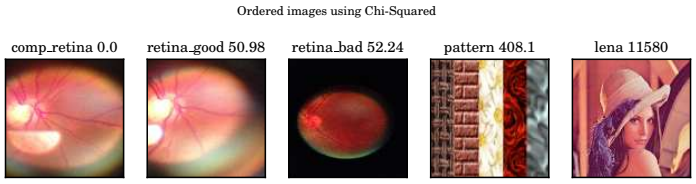


(a) Raw ordered images to sort using histogram comparison

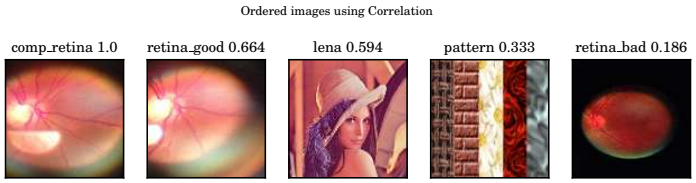


(b) Ordered images using Hellinger

(c) Ordered images using Intersection



(d) Ordered images using Chi-Squared



(e) Ordered images using Correlation

Figure 40. Histogram matching example

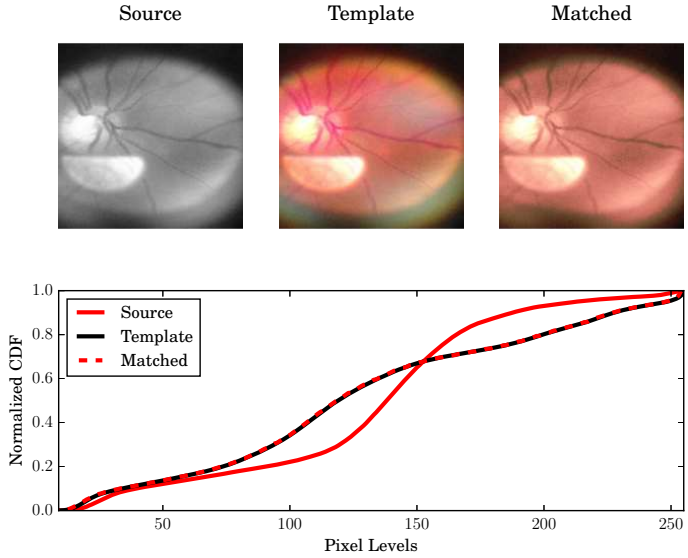


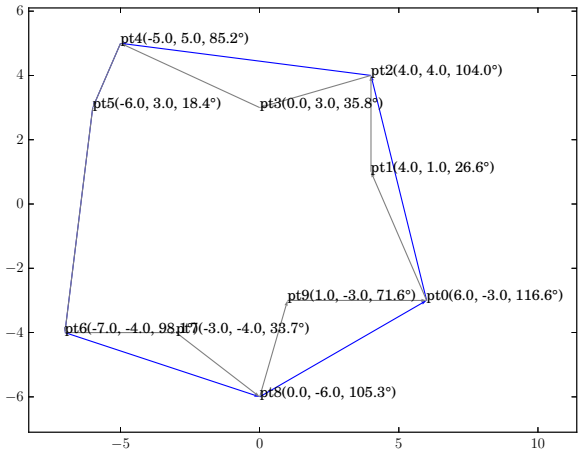
Figure 41. Features using SIFT

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

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

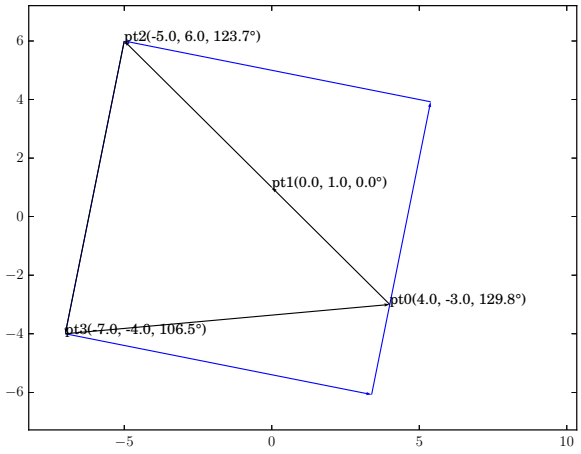
(c) SIFT in image with shape = (300,300) and rotated 30 degrees

Figure 42. Convexity ratio of an object



(a) ConvexityRatio = 0.803738317757

Figure 43. Rectangularity example



(a) Rectangularity: 0.500000059605

6 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.

I abandoned the application of de-blurring techniques in the developed program not only because there are almost non-existent in the problematic images but because most of these techniques tent to produce unsatisfactory results for our purposes (Gal et al., 2014) by reducing the success rate of restored images due to the computed local features in images being badly obtained as false key-points are increased for the introduction of edge-like artefacts.

Figure 44 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. From now on *imrestore* will refer to its script in general, where either *ImRestore* or the specialized *RetinalRestore* classes are used.

Now, lets review *imrestore* automatic execution which is typically used when restoring retinal images. Figure 45 summarizes with a work-flow the typical steps taken by *imrestore*, in this case the procedures taken using *RetinalRestore* class. The following paragraphs explain each block of Figure 45 in more detail and reference to their respective sources and pseudo codes.

Initialize RetinalRestore. The procedures in Figure 45 start by initializing the class either by using an *Command Line Interface* (CLI), a *Graphical User Interface* (GUI) or using directly the class instantiation (Figure 45,Block 1). For the server, I decided to implement the *CLI* which is a `shell` function in the *imrestore.py* script (Toro, 2016). The initialization is performed by the `__init__` methods in *RetinalRestore* and *ImRestore* classes (see Figure 44). Once the class is initialized with the option variables, *imrestore*

classes can call the `restore` method to start the restoration process.The pseudo code showing the RESTORE method in *ImRestore* class which is inherited by *RetinalRestore* is presented in Algorithm 2. This is the main method used to restore images or set of images as its name implies.

Algorithm 2 Main restoring method in ImRestore class

```
1:  procedure ImRestore.restore                                   ▷ Method of ImRestore's class
2:      I_restored ← self.pre_selection()                         ▷ Selects the base image
3:      while True do
4:          matches_ordered ← self.matching()
5:          if No more matches then
6:              break
7:          end if
8:          for i ← 0to len(matches_ordered) do
9:              path,match_data ← matches_ordered[i]
10:             kps_coors ← get key-points coordinates from match_data
11:             if kps_coors has enough coordinates then
12:                 H ← find homography using cv2.findHomography and kps_coors
13:             else
14:                 H ← null
15:             end if
16:             if H is not null then
17:                 tests ← calculate tests to approve homography transformation
18:                 if tests is approved then
19:                     remove path from the list of failed images
20:                     I_restored ← self.merge(path, H)          ▷ Merge image in path to I_restored using H
21:                 else
22:                     register path in the list of failed images
23:                 end if
24:             else
25:                 register path in the list of failed images
26:             end if
27:         end for
28:         if All image paths are in the list of failed images then
29:             break
30:         end if
31:     end while
32:     I_restored ← self.post_process_restoration(I_restored)    ▷ Post-process I_restored to filter, apply lens simulation and others
33:     save I_restored if necessary
34:     return I_restored
35: end procedure
```

Find local features from images. When `self.restore` is called then the local features of the images can be processed (Figure 45, Block 2). This is done automatically when a feature is requested for a certain image which is one of the characteristics

provided by the lazy evaluations techniques explained in subsubsection 4.3.3 'Lazy evaluation' on page 33. To do this, it uses the `feature_dict` and `feature_list` properties (i.e. variables with getters, setters and deletters) by calling `compute_keypoints(self)` or `compute_keypoint(self, path)` methods to calculate the local features and temporally load the images using `load_image` method (see Figure 44). The dictionary `feature_dict` is used to cache data of each image. If the caching techniques are enabled for *imrestore* then the dictionary is replaced by `LazyDict` and `MemoizedDict` (this enables lazy and memoized evaluations) found in *RRtoolbox* package under 'RRtoolbox/lib/cache.py' or imported in Python with `from RRtoolbox.lib.cache import LazyDict, MemoizedDict`. The list `feature_list` is used as a structure to organize the features of each image which is not provided by `feature_dict` that behaves as a unordered look up table.

Algorithm 3 shows the COMPUTE_KEYPOINT method in ImRestore class which as its name implies it computes the key-points from a requested image. Notice that the *self.compute_keypoints* computes the key-points of all images and just calls COMPUTE_KEYPOINT under the hook to achieve this. The instance variable self.feature in line 21 is instantiated from the *Feature* class provided by the *RRtoolbox* package and can be imported as `from RRtoolbox.lib.descriptors import Feature`. Notice that *imrestore* uses by default the ASIFT method explained by (Guoshen Yu & Morel, 2011) and introduced in subsubsection 5.5.5 'ASIFT' on page 80. For the sake of completeness the ASIFT method implemented in the *Feature* class can be combined with most feature descriptors presented in subsubsection 5.5.4 'Feature detection and matching' on page 78 that are supported in *OpenCV* (it depends on which feature descriptors are added when compiling OpenCV).

***Select* base image.** As seen in line 2 of Algorithm 3 the PRE_SELECTION method is called making the features be computed for each restoring image. When the image features are processed then the *base image* is chosen (Figure 45, Block 3), in this case by the image with more key-points. This process is achieved by calling simply comparing which image processed more key-points in `pre_selection` method, then loading the *base image* as $I_{restored}$ in line 11 and registering it as an used image from the list of restoring images. The PRE_SELECTION method in *ImRestore* class is also inherited by *RetinalRestore* and is shown in Figure 44.

Algorithm 3 Method in ImRestore class in charge of computing key-points from an image

```
procedure ImRestore.compute_keypoint(path)                          ▷ Method of ImRestore's class
    I ← self.load_image(path)
    if path is in self.feature_dict then
        keypoints, descriptors, P_shape ← self.feature_dict[path]
        if P_shape ← I_shape then                                   ▷ if cached key-points do not match with image I
            ▷ get relation from P_shape to I_shape
            R_x, R_y ← RRtoolbox.lib.arrayops.convert.getSOpointRelation(P_shape, I_shape)
            for i ← 0 to len(keypoints) do                          ▷ transform key-points to match I_shape
                kp ← keypoints[i]                                   ▷ get key-point i
                x, y ← kp["pt"]                                     ▷ get key-point coordinate
                kp["pt"] ← x × R_x, y × R_y                         ▷ save transformed key-point coordinate
            end for
        end if
    else
        mask ← null
        if User provided self.maskforeground function to get mask then
            mask ← self.maskforeground(I)
        else if Enabled automatic calculation of mask then
            mask ← RRtoolbox.lib.arrayops.mask.foreground(I)
        end if
        keypoints, descriptors ← self.feature.detectAndCompute(I, mask)   ▷ compute features
        P_shape ← I_shape                                           ▷ shape of the processed image
        self.feature_dict[path] ← keypoints, descriptors, P_shape   ▷ cache features
    end if
    return keypoints, descriptors, P_shape
end procedure
```

Algorithm 4 Preselection method in ImRestore class

```
procedure ImRestore.pre_selection                                   ▷ Method of ImRestore's class
    if There is no option then                                      ▷ select first image as base image
        path ← path of the the first image in the list of images
    else if Option is a String then
        path ← path exactly matching the user demand
    else if Option of better image then                             ▷ Usually the best image is the one with most key-points
        path ← path of image with most key-points
    else
        raise Exception error
    end if
    I_restored ← loads image in path
    Register base image as the first used image
    return I_restored
end procedure
```

***Match remaining images with* base image.** The *base image* is used to stitch or merge the remaining images on it and for that a general matching is performed (Figure 45, Block 4) in line 4 of Algorithm 2 where the *base image* key-points are used as target and the key-points of the other images used to train the feature detector matcher. The matching performed by the method MATCHING is shown in Algorithm 5.

Algorithm 5 Matching method in ImRestore class

```
1:  procedure ImRestore.matching                          ▷ Method of ImRestore's class
2:      for i ← 0 to len(self.feature_list) do
3:          path ← self.feature_list[i]                   ▷ get path number i from images
4:          if path is not in list of used images for the restoration then
5:              keypoints ← key-points from path
6:          end if
7:      end for
8:      if There are no keypoints then                    ▷ all keypoints were used
9:          return null                                   ▷ do not process matching
10:     end if
11:     matches_raw ← match keypoints with the key-points from the base image
12:     matches ← filter out fake matches from matches_raw
13:     if by entropy then
14:         matches_ordered ← order matches by their image entropy
15:     else if by histogram comparison then
16:         matches_ordered ← order matches by their image histogram comparison
17:     else if by custom function then
18:         matches_ordered ← order matches by user provided custom function
19:     else                                              ▷ Option of better image
20:         matches_ordered ← order matches by most probable or best matches
21:     end if
22:     return matches_ordered
23: end procedure
```

***Select best images and test them against* base image.** In Algorithm 5 the matches are classified to not use those that are inside the restored image in line 5, then these are matched to the ones in the restored images in line 11. Once the matches are produced *imrestore* selects the best matches (Figure 45,Block 5) by filtering the ones that do not have a good Hamming distance (see line 12), then they are classified to the respective images and these groups are ordered according to a criteria. In line 14 matches are ordered by the entropy that the merging images have with respect to the *base image* (see subsubsection 5.5.1 ‘Entropy’ on page 76). In line 16, matches are organized with respect to the histogram comparison (see subsubsection 5.5.2 ‘Histogram comparison’ on page 77). In line 18, matches are ordered with respect to a user or custom function

to provide more possibilities. And in line 20 the matches are ordered to correspond to the image that shares more matches with the *base image*. The ordering of the matches determine which images are tried to be merged first in the *base image*. The pseudo code showing the MATCHING method in *ImRestore* class presented in Algorithm 5 is instantiated by the *RestinalRestore* class shown in Figure 44.

***Use homography to transform* merging image.** Once the matches are obtained in an order form in line 4 of Algorithm 2 they are processed in a for loop to begin merging. For this a homography is applied in line 12 to the matched key-points (Yang et al., 2014; Zdešar et al., 2014; Y. Zhang et al., 2016) between those of the merging image and the restored image to find the transformation matrix H to transform the merging image as the foreground over the restored image treated as the background (Figure 45, Block 6). The matrix H is found using `cv2.findHomography` function from OpenCV in line 12 which is later fed to the MERGE method in Figure 45.

After the homography is applied to find H some tests are carried out in line 17 of Algorithm 2 before transforming the merging image to determine if it can be adequately merged with the *base image*. Once the tests are cleared the merging proceeds in line 20 by the `merge` method in *ImRestore* class covering the Blocks 7,8 to 9 in Figure 45. The pseudo code of MERGE is presented in Algorithm 6.

Histogram mathing. The histogram matching in Block 7 is applied as in subsubsection 5.5.3 'Histogram matching' on page 77 to match the merging image color with the color of the restored base image. This is evidenced in line 4 of Algorithm 6.

***Calculate alpha mask of* merging image *over* base image.** In Block 8 of Figure 45 the transformed *merging image* and *base image* are used to create the alpha mask to complete missing information from *base image* using the *merging image*.

This is tried to be done numerous times. In line 7 of Algorithm 6 the alpha mask is created if there is expert data available. If there is still no alpha mask created, the in line 10 the `pre_process_fore_Mask` method is called. The pseudocode of PRE_PROCESS_FORE_MASK method is left out as it is not used returning just $null$ but is there for future development (actually it was used in early development and now is deprecated due to some encountered problems). And in line 25, if the alpha mask is not calculated beforehand, the alpha mask creation is left out to the POST_PROCESS_FORE_MASK method in

Algorithm 6 Merging and stitching method in ImRestore class

procedure IMRESTORE.MERGE($path$,H) ▷ Method of ImRestore's class
  $I_{merging} \leftarrow$ load image from $path$ ▷ this is the *merging image*
  if histogram matching is True **then** ▷ Apply histogram matching to $I_{merging}$ with respect to the *base image*
    $I_{merging} \leftarrow$ RRtoolbox.lib.image.hist_match($I_{merging}$,$I_{restored}$)
  end if
  $Mask_{alpha} \leftarrow null$
  if expert data is not *null* **then**
    $Mask_{alpha} \leftarrow$ produce alpha mask using expert data, $I_{merging}$ and $I_{restored}$
  end if
  if $Mask_{alpha}$ is *null* **then** ▷ there was no expert data or it failed to produce alpha mask
    $Mask_{alpha} \leftarrow$ self.PRE_PROCESS_FORE_MASK($I_{restored}$, $I_{merging}$, H)
  end if
  if self.grow_scene is $True$ **then** ▷ Expand restored image using mosaic method
    Calculate a bigger image I_{bigger} where $I_{restored}$ and $I_{merging}$ fit together
    $H_{back} \leftarrow$ calculate TM to convert from $I_{restored}$ to I_{bigger}
    $H_{fore} \leftarrow H_{back} \times H$ ▷ H_{fore} converts from $I_{merging}$ to I_{bigger}
    $height, width \leftarrow$ shape(I_{bigger})
    $I_{merging} \leftarrow$ cv2.warpPerspective($I_{merging}$, H_{fore}, ($width$, $height$))
  else
    $H_{back} \leftarrow IM_{3x3}$ ▷ Identity matrix
    $H_{fore} \leftarrow H$
    $height, width \leftarrow$ shape($I_{restored}$)
    $I_{merging} \leftarrow$ cv2.warpPerspective($I_{merging}$, H_{fore}, ($width$, $height$))
  end if
  if expert data is not *null* **then**
    $Mask_{alpha} \leftarrow$ self.post_process_fore_Mask($I_{restored}$, fore)
  end if
  if expert data is not *null* **then**
    $Mask_{alpha} \leftarrow$ cv2.warpPerspective(np.ones(shape($Mask_{alpha}$)), H_{fore}, ($width$, $height$))
  end if
  $I_{restored} \leftarrow$ Rtoolbox.lib.arrayops.basic.overlay($I_{restored}$, $I_{merging}$, $Mask_{alpha}$) ▷ overlay foreground on top of background image
  Transform key-points position inside $I_{merging}$ using H_{fore}
  Transform key-points position inside $I_{restored}$ using H_{back}
  Register $path$ in the list of used images
  return $I_{restored}$
end procedure

RetinalRestore class as presented in Algorithm 7. In the pseudo code it can be appreciated the use of the developed `layeredfloods` method in line 7 seen in section 5.4.2 which uses the `get_layered_alpha` function as explained in section 5.4.3. The alpha mask plays an important role in the solution of the PSF in Equation 1. The application of it shows its benefits as demonstrated in Figure 37d. The overlay method is used as it is a simple and effective approach for merging (Médioni, 2005, 3.10.1 3D surface reconstruction The).

Algorithm 7 Post processing method for the foreground alpha mask in RetinalRestore class

```
1:  procedure RetinalRestore.post_process_fore_Mask(back, fore)        ▷ Method of RetinalRestore's class
2:      P_shape ← (400, 400)                                          ▷ process shape, usually small
3:      if P_shape is not null then                                   ▷ rescale image to process mask
4:          O_shape ← back or fore original shape
5:          convert back and fore to P_shape
6:      end if
7:      alpha_mask ← RRtoolbox.tools.segmentation.get_layered_alpha(back, fore)
8:      if s thenelf.grow_scene:                                      ▷ let dark areas be treated as lens, so expand them
9:          assign highest values in alpha_mask to dark values in back
10:     end if
11:     Rescale alpha_mask to original shape using O_shape
12:     return alpha_mask
13: end procedure
```

***Merge in* base image.** The merging applied in Block 9 is performed using the calculated alpha mask obtained from the merging image over the base restored image. Then this mask is applied using the `overaly` function in line 30 of Algorithm 6 which is further explained in subsubsection 5.1.2 ‘Overlay’ on page 46.

Update key-points positions. Block 10 in Figure 45 can be appreciated in the while loop of Algorithm 2. The idea is to merge as many images as possible to the base image so the process is repeated excluding the already used images.

***Filter* restored image.** If there are not more merging images to register in the *base image* then the program leave the while loop in Algorithm 2 and post-process it in line 32 using the POST_PROCESS_RESTORATION method in *RetinalRestore* class as presented in Algorithm 8.

The restored image is then filtered in Block 11 using the bilateral filter function `cv2.bilateralFilter` in line 2 of Algorithm 8 provided by OpenCV to eliminate noise and keep edges as explained in subsubsection 5.3.5 ‘Bilateral Filter’ on page 58. The line 2 additionally uses the getBilateralParameters function to find the parameters from the noisy image which is also explained in subsubsection 5.3.5 with an example image in Figure 23.

I selected the bilateral filtering as the preferred method to apply to the restored images as it reduces noise while keeping important features like edges (Paris et al., 2008)

which is compatible with the feature extractors methods studied in subsubsection 5.5.4 ‘Feature detection and matching’ on page 78, for its availability in OpenCV (Bob Fisher, 2004) and because it solves similar problems as in other restoration cases (Kallel et al., 2014, 1.1. Image restoration).

Algorithm 8 Post processing method for the restored image in RetinalRestore class

```
1:  procedure RetinalRestore.post_process_restoration(image)      ▷ Method of RetinalRestore's class
2:      Apply de-noising to image using cv2.bilateralFilter
3:      if self.lens then                                          ▷ simulation of lens
4:          image ← RRtoolbox.tools.lens.simulateLens(image)       ▷ overlay lens on image
5:      end if
6:      if self.enclose then
7:          crop only retinal area on image
8:      end if
9:      return image
10: end procedure
```

Apply lens simulation. If the option –lens in the *imrestore* program is enabled then Block 12 is executed and lens are overlay over the retinal area using the function `simulateLens` from ‘RRtoolbox/tools/lens.py’ from the *RRtoolbox* package found in (Toro, 2016) repository. The lens simulation is evidenced in line 4 of Algorithm 8. It can be imported in python as `from RRtoolbox.tools.lens import simulateLens` and can be reviewed in the RRtoolbox documentation in (David, 2016).

Figure 46 summarizes *imrestore* execution focused on *RetinalRestore* class with some of the procedures and methods used in a typical application of the program.

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.

Everything related to the code is hosted in (Toro, 2016) 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, 2016). The current algorithm can be further improved but already complies to precise segmentation of objects under irregular cases to offer robustness to noisy areas like light 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 (see subsubsection 5.4.3 ‘Alpha masks’ on page 73 for the developed method).

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 section 7 ‘User Interface’ on page 103.

Figure 44. *imrestore.py* classes: *ImRestore* and its inheritor *RetinalRestore*

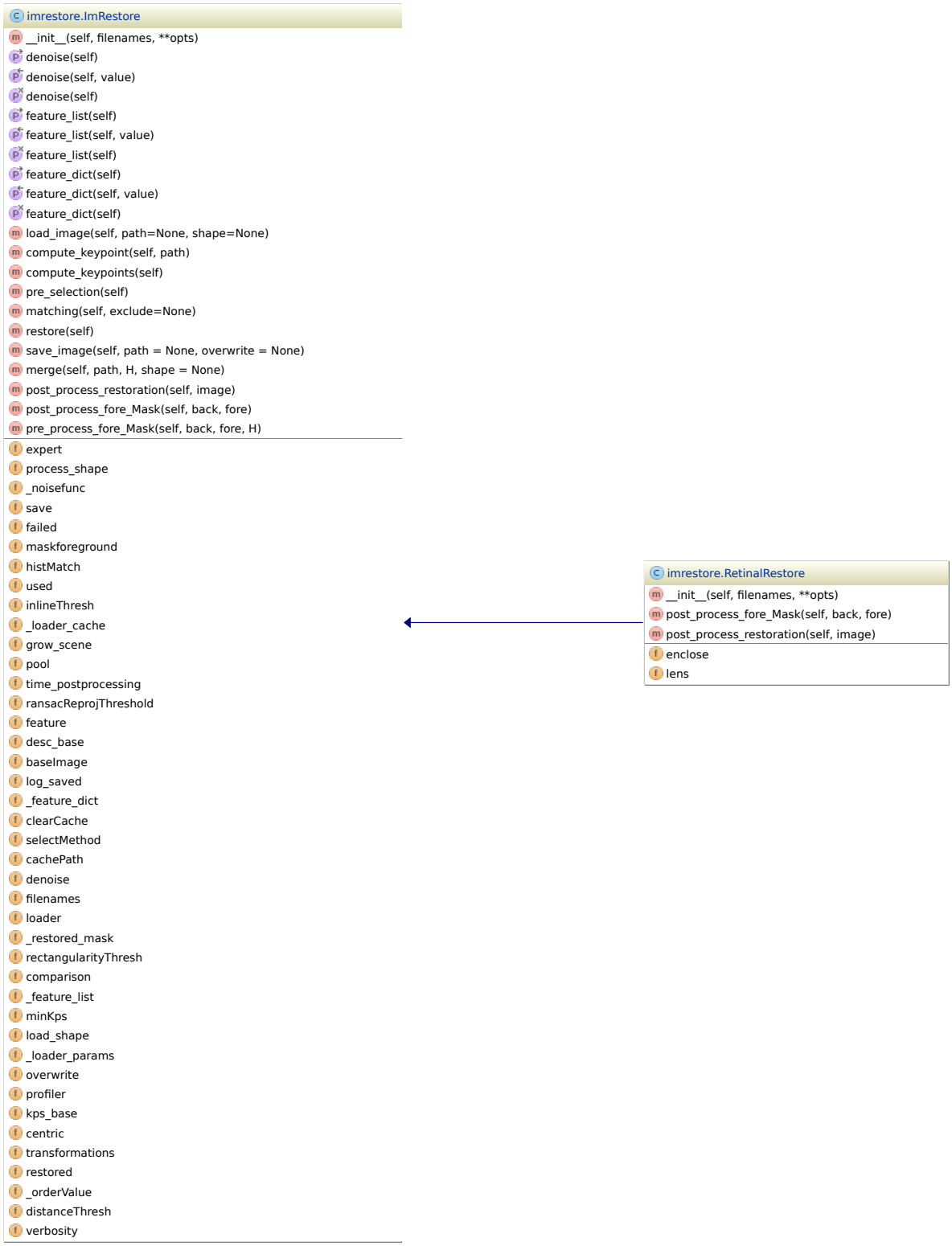


Figure 45. Diagram simplifying *imrestore* typical execution order

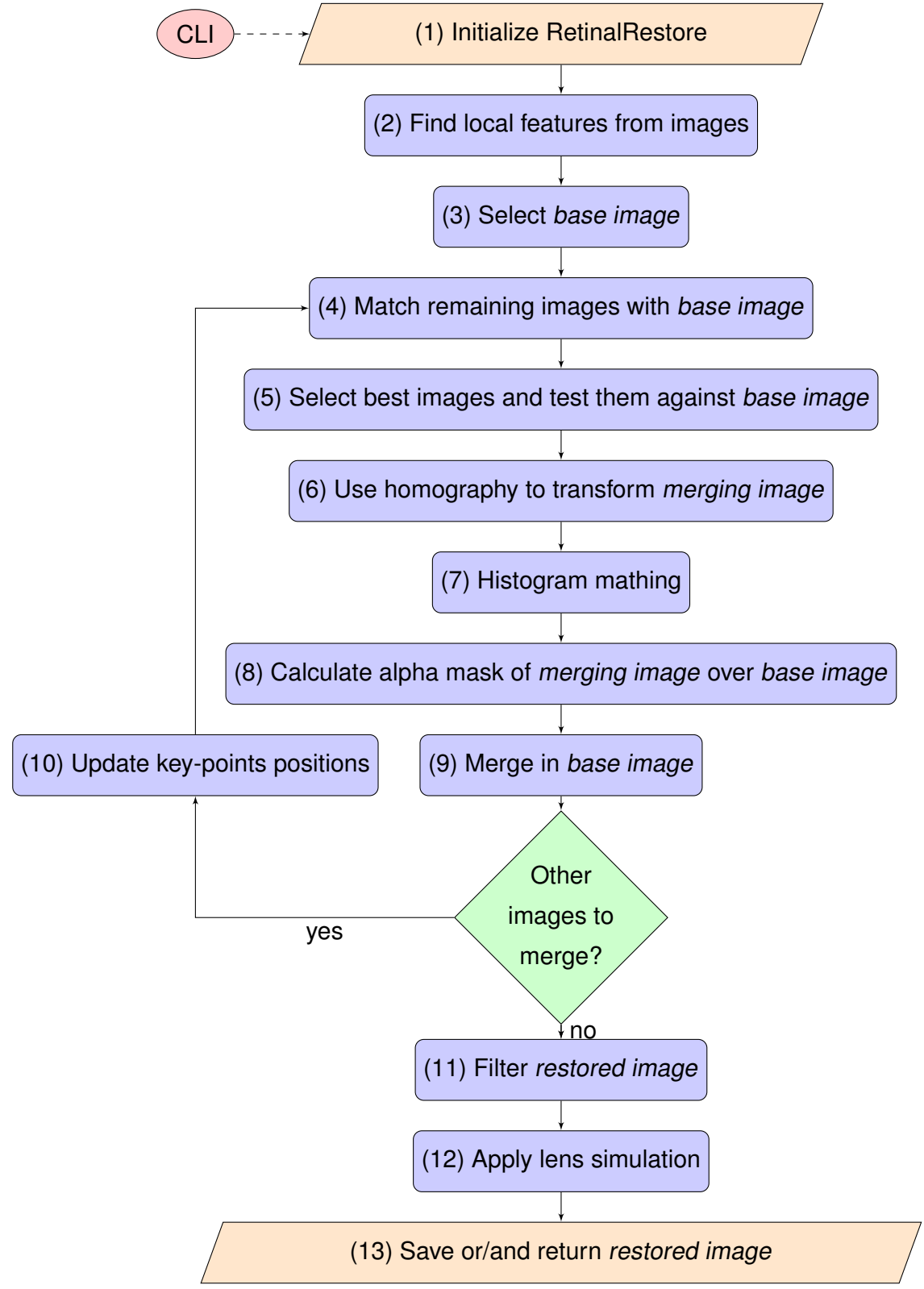
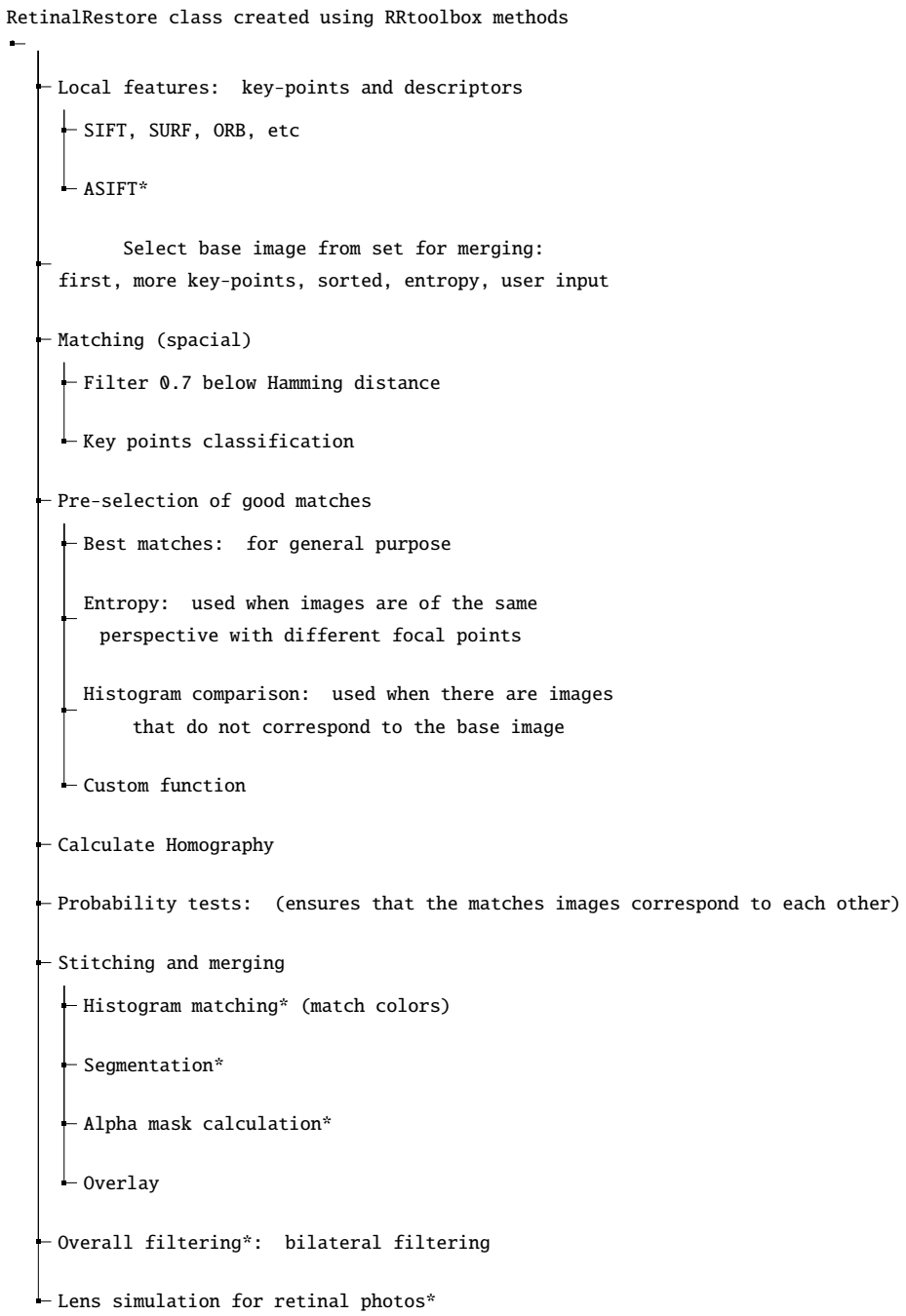


Figure 46. Diagram simplifying *imrestore* applied methods and steps



6.1 Expert System

An expert system was developed to provide information for each image so that each image can have the correct segmentation of the retinal area, optic disc, defects and blurry areas for qualification purposes for the provided sets of test images and the output of the developed program. This ensures that the qualification of the presented results are transparent by being calculated directly from the expert system and the statistical data generated from the application of the *imrestore* program. This data was also used when the concepts and algorithms were being developed so that they could be coded and tuned to yield the best results for each possible case.

In the final implementation of *imrestore* there is an experimental implementation which retrieves the expert data in the restoration process to restore the images as it was intended by the expert by using the option `--expert` or the short option `-x` pointing to the expert data of the images involved in the restoration. This utility is still experimental and not always will yield appealing results.

The expert data can be generated for any image by running the script *expert.py* in the root of the repository (Toro, 2016). All the expert data contained in the results folder can be reviewed by typing in a terminal the command presented in Code 4.

Code 4: Review expert data for test images

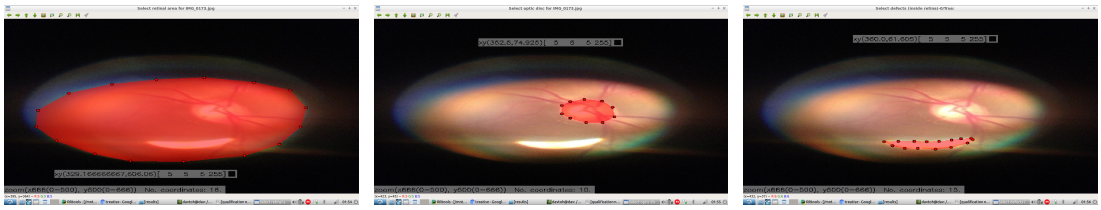
```
python expert.py ./results/ --subfolders --review
```

Which will let the user review the expert data for the test images contained in the *results* folder of the repository or inquire for new expert data if new images are added to it. Figure 47 shows a sample of the interface generated by the *expert* program to qualify the images. Notice that the program asks the user to selects some regions in the images, in this case the retinal area, optic disk and defects inside the retina. The user is invited to use the program and for that please type in the terminal `python expert.py -h` to know more about the options that the *expert* program can offer.

The data structure of the expert data lets the *expert* program add additional information to the already existent one, so if an image expert data is outdated for possible new

information requested in future it will only ask for the new information and update the old expert data with the new one. This ensures that future updates in the *expert* program and new information inquiries are compatible with the current collected expert data.

Figure 47. Example of expert system inquiries



(a) Selecting retinal area

(b) Selecting optic disk

(c) Selecting defects inside retinal area

7 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.

7.1 Command-line interface

As explained before in section 6 ‘Implementation’ on page 88, I decided to use a *CLI* because it is a program intended to be used in servers and it must provide the means for it to comply with our objectives. Because it is for a server I decided, from a technical point of view, to favour *CLI* over *GUI* due to its simplicity, efficiency, less resource hungry, the capability of automating the restoration process by programming with and on top of it letting the user have a better control to achieve complex task and because it is usually done this way in Unix based operating systems like Linux where the *GUI* is developed on top of the *CLI* (Computerhope, n.d.; About.com, n.d.). This way we can access the server directly by terminal or remotely using Secure Shell (*SSH*) and create scripts to integrate the program with other interfaces in the server. The following command format is used to run *imrestore* script:

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

`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 7.2 ‘Deployment: Executable’ on the next page just enter the command in Linux as:

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

The `./` is used to signify that the operation is from respect to the current path. In Windows the command would be:

```
"dist/imrestore" tests/im1*
```

If the terminal’s path is in *dist* folder use in Linux:

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

And in Windows it would be:

```
imrestore tests/im1*
```

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

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

Or its equivalent in Windows:

```
imrestore --help
```

Which will print out the available options that *imrestore* program has to offer. Read more of how imrestore can be used in subsection 7.3 ‘Usage’ on the facing page and the results that it produce in section 8 ‘Results’ on page 110.

7.2 Deployment: Executable

There are several options to create executables and installers from python sources (Python Software Foundation, n.d.-h). From them Pyinstaller suffices for most of 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 ‘Install packages to Python using pip’ on page 187 and further reading in (David Cortesi, Giovanni Bajo, William Caban, & Gordon 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:

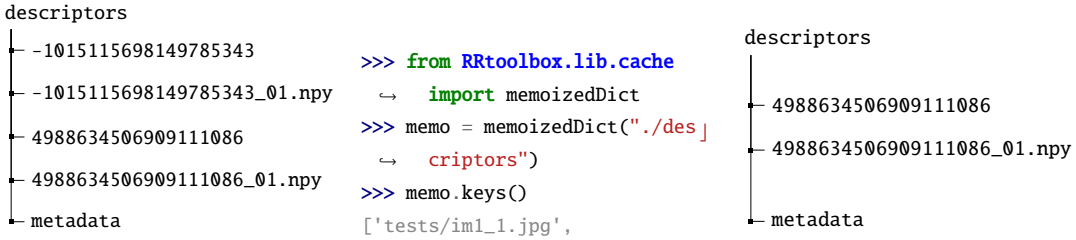
```
pyinstaller -p ./ -n imrestore -F ./imrestore.py/ --version-file=version
```

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.

7.3 Usage

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 (Toro, 2016) to use it just by placing in the console the command '`python imrestore.py tests/im1* --lens --overwrite --cachePath .`'.

Figure 48. Persistence to *descriptors* folder and its manipulation



(a) Created *descriptors* folder

(b) Manipulation of persintent data

(c) After manipulation of *descriptors* folder

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 48a. 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 reprocess it and if data is missing in the cache it simply recalculates it. For the sake of demonstration 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 48b. Figure 48c 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.

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
```

8 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 to ensure that a professional ophthalmologist can obtain better results than the ones here presented. The original results were performed in a Linux distribution Ubuntu 15.10 in a computer with Processor 4x Intel(R) Core(TM) i5-2450M CPU @ 2.50GHz, Memory of 8121MB and hard drive ATA TOSHIBA MQ01ABD1. The compatibility tests were performed in virtual machines both for Windows 7 and another Linux distribution different to the host computer with 2048 MB of RAM and two processors which are not present in this document (see subsection A.3 'Procedures to download *imrestore* program and source' on page 190 to get the programs). There is an observation worth pointing out regarding the difference when *imrestore* executable is booting under Linux and that is that it takes more time in loading the necessary files than in Window (about 5 seconds more!, disappointing for Linux users) possibly due that Linux executable is bigger in size and need to be extracted at each run, other than that once the program is running, both Linux and Windows perform equally. Nevertheless, the Linux executable can be changed to a permanent installation or the sources can be run with a python interpreter if installed which is something really easy to do in Linux, solving the booting delay of *imrestore* executable.

Now that we know *imrestore* basic use explained in subsection 7.3 'Usage' on page 105 we proceed to run the generic command in Code 5 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>`.

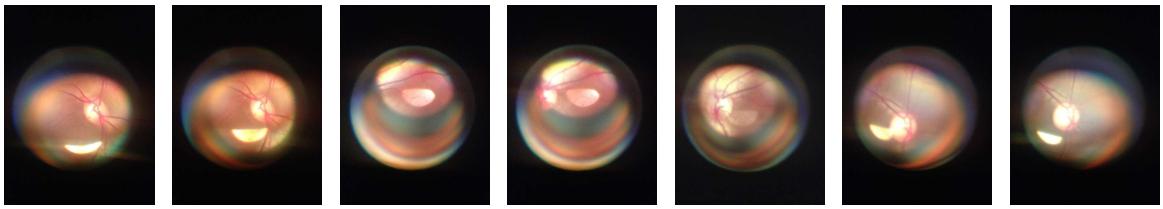
Only the images that where used are shown in the result figures, so that the first image is the *base image* (e.g. Figure 50a and Figure 51a) used for the set restoration and the subsequent merging images except the last one are used to improve it (e.g. Table 6 do not have merging images and Table 7 has Figure 51b). The last image is the restored set starting from the *base image* and it is the result that *imrestore* saves as output (e.g. Figure 50b and Figure 51c). In addition to it, a profiling which is like a report is presented for each set to let the user see the performance 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>
```

8.1 Result for set 1

Figure 49. Images in set 1



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

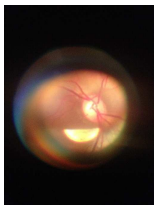
Table 6

Profiling for set 1 using command '`imrestore ../results/set1/*.* --lens --overwrite --cachePath {temp}`'

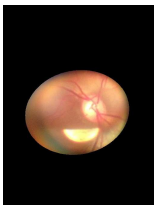
```
'ImRestore init' -> 13.4487 secs
└─ 'Computing features' -> 13.0310 secs
   ├─ (processed) 'IMG_0173' -> 2.0793 secs
   ├─ (processed) 'IMG_0176' -> 1.8577 secs
   ├─ (processed) 'IMG_0179' -> 1.7805 secs
   ├─ (processed) 'IMG_0180' -> 1.8141 secs
   ├─ (processed) 'IMG_0182' -> 1.8465 secs
   ├─ (processed) 'IMG_0184' -> 1.8079 secs
   └─ (processed) 'IMG_0185' -> 1.8103 secs
└─ 'Restoring' -> 0.2070 secs
   ├─ 'Matching' -> 0.1028 secs
   └─ 'Merging' -> 0.1040 secs
└─ 'Post-processing' -> 0.2107 secs
```

```
'ImRestore init' -> 2.7555 secs
└─ 'Computing features' -> 2.4322 secs
   ├─ (memoized) 'IMG_0173' -> 0.2522 secs
   ├─ (memoized) 'IMG_0176' -> 0.1923 secs
   ├─ (memoized) 'IMG_0179' -> 0.3316 secs
   ├─ (memoized) 'IMG_0180' -> 1.0289 secs
   ├─ (memoized) 'IMG_0182' -> 0.1775 secs
   ├─ (memoized) 'IMG_0184' -> 0.3719 secs
   └─ (memoized) 'IMG_0185' -> 0.0465 secs
└─ 'Restoring' -> 0.0853 secs
   ├─ 'Matching' -> 0.0332 secs
   └─ 'Merging' -> 0.0518 secs
└─ 'Post-processing' -> 0.2381 secs
```

Figure 50. Result for set 1 using command '`imrestore ../results/set1/*.* --lens --overwrite --cachePath {temp}`'



(a) IMG_0176



(b) Restored image

This set have 7 images as shown in Figure 49 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 13.45 to 2.76 seconds when cached, this means that 10.69 seconds were spared with almost 10.69 times the gain.

Figure 50 shows the result for set 1. In it *imrestore* selects Figure 50a 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 50a is left unchanged and only post-processed to generate the restored image shown in Figure 50b.

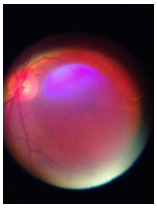
Table 7

Profiling for set 2 using command '`imrestore ../results/set2/*.* --lens --overwrite --cachePath {temp}`'

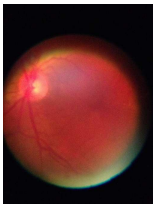
```
'ImRestore init' -> 17.5139 secs
└─ 'Computing features' -> 16.2102 secs
   ├─ (processed) 'IMG_0405' -> 2.8743 secs
   ├─ (processed) 'IMG_0406' -> 2.6034 secs
   ├─ (processed) 'IMG_0407' -> 2.5388 secs
   ├─ (processed) 'IMG_0408' -> 2.5192 secs
   ├─ (processed) 'IMG_0409' -> 2.8152 secs
   └─ (processed) 'IMG_0410' -> 2.8106 secs
└─ 'Restoring' -> 1.0051 secs
   ├─ 'Matching' -> 0.0283 secs
   └─ 'Merging' -> 0.9765 secs
      └─ 'IMG_0408' -> 0.9515 secs
└─ 'Post-processing' -> 0.2986 secs
```

```
'ImRestore init' -> 1.9135 secs
└─ 'Computing features' -> 0.3068 secs
   ├─ (memoized) 'IMG_0405' -> 0.0679 secs
   ├─ (memoized) 'IMG_0406' -> 0.0347 secs
   ├─ (memoized) 'IMG_0407' -> 0.0415 secs
   ├─ (memoized) 'IMG_0408' -> 0.0338 secs
   ├─ (memoized) 'IMG_0409' -> 0.0591 secs
   └─ (memoized) 'IMG_0410' -> 0.0383 secs
└─ 'Restoring' -> 1.3541 secs
   ├─ 'Matching' -> 0.0451 secs
   └─ 'Merging' -> 1.3087 secs
      └─ 'IMG_0408' -> 1.2898 secs
└─ 'Post-processing' -> 0.2525 secs
```

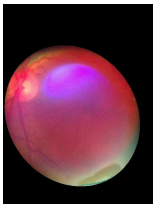
Figure 51. Result for set 2 using command '`imrestore ../results/set2/*.* --lens --overwrite --cachePath {temp}`'



(a) IMG_0409



(b) IMG_0408



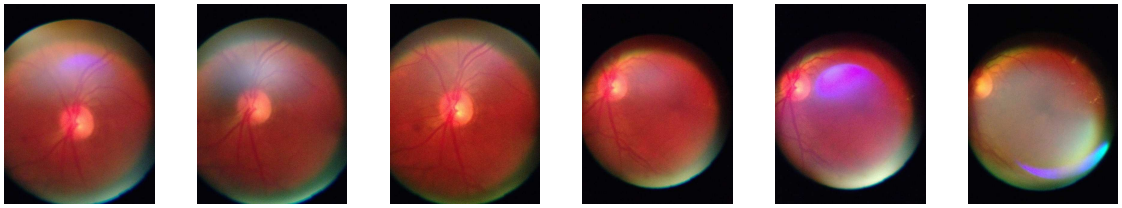
(c) Restored image

8.2 Result for set 2

As in the Result for set 1 the generic Code 5 is used for set 2 with images in Figure 52 and presenting the profiling results in Figure 51 for 6 processed images. It continues to show the advantage of using the cache option reducing the time from 17.51 to 1.91 seconds which is the 10.93% of the total with 15.60 seconds of gained time.

Images used in the restoration are shown in Figure 51 and the ones in the set are in Figure 51. This time it does finds the merging image shown in Figure 51b to try to restore the *base image* shown in Figure 51a resulting in the restored image in Figure 51b. 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 51b or the one which was not used in Figure 52c. 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.

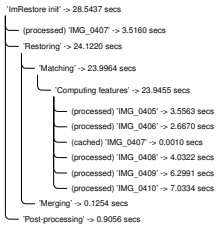
Figure 52. Images in set 2

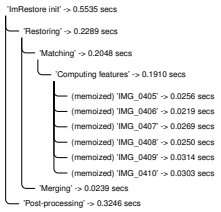


(a) IMG_0405 (b) IMG_0406 (c) IMG_0407 (d) IMG_0408 (e) IMG_0409 (f) IMG_0410

Table 8

Profiling for set 2 using command '`imrestore ../results/set2/*.* -b IMG_0407.* -y --lens --overwrite --cachePath {temp}`'





Lets try to enhance the result showed by Figure 51 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 53 shows the new result, with this *imrestore* produced one of the best results that could be obtained from set 2 and though it did not find good matches for Figure 53a it applied the post processing methods to generate the restored image in Figure 53b. 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

Figure 53. Customized result for set 2 using command '`imrestore ../results/set2/*.* -b IMG_0407.* -y --lens --overwrite --cachePath {temp}`'

(a) IMG_0407

(b) Restored image

problems whatsoever for the other images in the set to enhance, these images were not that good (Figure 52).

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 28.54 to 0.55 seconds, only the 1.94%. 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`.

8.3 Result for set 3

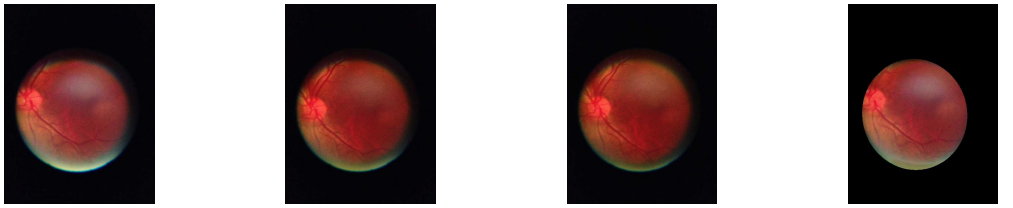
Table 9

Profiling for set 3 using command 'imrestore ../results/set3/.* --lens --overwrite --cachePath {temp}'*

```
'ImRestore init' -> 22.3677 secs
    'Computing features' -> 19.9608 secs
        (processed) 'IMG_0428' -> 3.2786 secs
        (processed) 'IMG_0429' -> 2.9612 secs
        (processed) 'IMG_0430' -> 2.7723 secs
        (processed) 'IMG_0431' -> 2.7574 secs
        (processed) 'IMG_0432' -> 2.6672 secs
        (processed) 'IMG_0433' -> 2.7130 secs
        (processed) 'IMG_0434' -> 2.7618 secs
    'Restoring' -> 2.1034 secs
        'Matching' -> 0.0930 secs
        'Merging' -> 2.0099 secs
            'IMG_0429' -> 0.9610 secs
            'IMG_0430' -> 0.9781 secs
    'Post-processing' -> 0.3034 secs
```

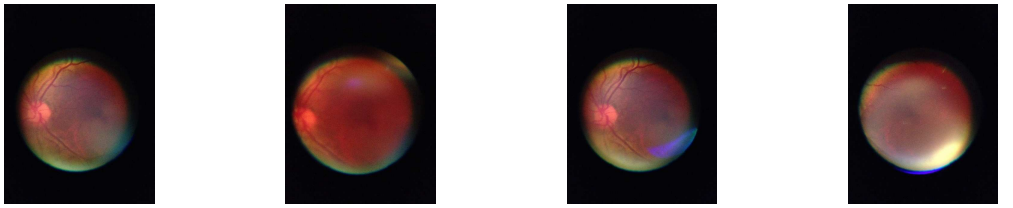
```
'ImRestore init' -> 3.4650 secs
    'Computing features' -> 1.2328 secs
        (memoized) 'IMG_0428' -> 0.1595 secs
        (memoized) 'IMG_0429' -> 0.1947 secs
        (memoized) 'IMG_0430' -> 0.1607 secs
        (memoized) 'IMG_0431' -> 0.3847 secs
        (memoized) 'IMG_0432' -> 0.1265 secs
        (memoized) 'IMG_0433' -> 0.1049 secs
        (memoized) 'IMG_0434' -> 0.0599 secs
    'Restoring' -> 1.9898 secs
        'Matching' -> 0.0678 secs
        'Merging' -> 1.9214 secs
            'IMG_0429' -> 0.9257 secs
            'IMG_0430' -> 0.9519 secs
    'Post-processing' -> 0.2424 secs
```

Figure 54. Result for set 3 using command 'imrestore ../results/set3/*.* --lens --overwrite --cachePath {temp}'



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

Figure 55. Images not used in restoration of set 3



(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 54 with the *base image* in Figure 54a, the restored in

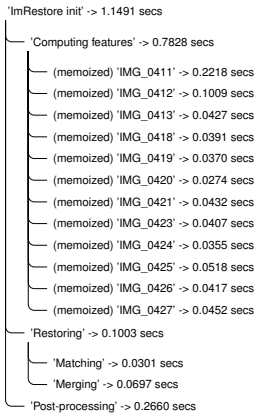
Figure 54d and the remaining merged images. In the restored image we can see that the bright part at the bottom in Figure 54a was replaced by the one in Figure 54b which is more convenient. The lens also did a good job in segmenting correctly the retinal area despite the blurry-like aura enclosing the 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 84.51% from 22.37 to 3.47 seconds. The rate of used images also remains low in 0.43 with 3 used of 7 in the set (all images in Figure 54 except the restored in Figure 54d plus the 4 images not used).

8.4 Result for set 4

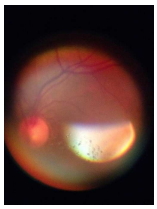
Table 10

Profiling for set 4 using command '`imrestore ../results/set4/*.* --lens --overwrite --cachePath {temp}`'

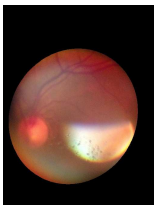


This set presents similar results as the previous sets with 12 computed images and all of them with different affected areas as shown in Figure 56. Despite all those images only the image in Figure 57a got to be used without being merged with the others and only applying the post-processing step with lens simulation as shown in Figure 56b.

Figure 56. Result for set 4 using command '`imrestore ../results/set4/*.* --lens --overwrite --cachePath {temp}`'

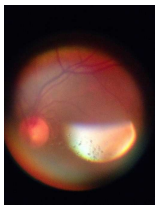


(a) IMG_0411

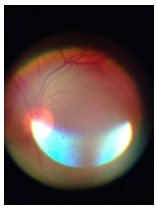


(b) Restored image

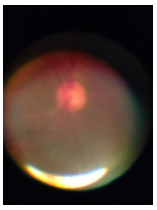
Figure 57. Images in set 4



(a) IMG_0411

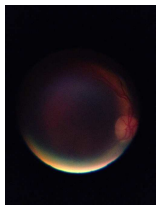


(b) IMG_0412

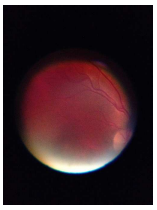


(c) IMG_0413

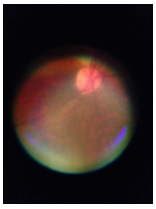
(d) IMG_0418



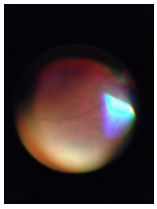
(e) IMG_0419



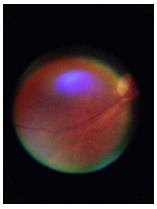
(f) IMG_0420



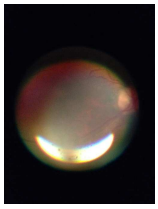
(g) IMG_0421



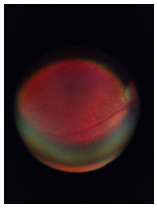
(h) IMG_0423



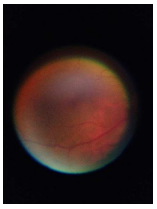
(i) IMG_0424



(j) IMG_0425



(k) IMG_0426



(l) IMG_0427

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.

Figure 56b shows again a reduction in time of 33.88 seconds from 35.03 to 1.15 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 57).

8.5 Result for set 5

In this 2 images were used from 7 images leading to a rate of 0.29 which is evidenced in Figure 58. The *base image* is in Figure 58a an the other in Figure 58b. Both of them presented the same flare in the same position and though they could be perfectly

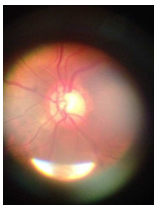
Table 11

Profiling for set 5 using command '`imrestore ../results/set5/*.* --lens --overwrite --cachePath {temp}`'

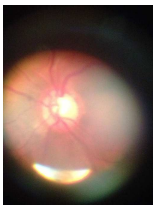
```
'ImRestore init' -> 22.4091 secs
'Computing features' -> 21.0103 secs
    (processed) 'IMG_0249' -> 3.2052 secs
    (processed) 'IMG_0250' -> 3.1645 secs
    (processed) 'IMG_0251' -> 2.9983 secs
    (processed) 'IMG_0252' -> 2.7007 secs
    (processed) 'IMG_0253' -> 2.8021 secs
    (processed) 'IMG_0254' -> 3.0260 secs
    (processed) 'IMG_0256' -> 3.0518 secs
'Restoring' -> 1.0859 secs
    'Matching' -> 0.0644 secs
    'Merging' -> 1.0211 secs
        'IMG_0254' -> 0.9824 secs
'Post-processing' -> 0.3129 secs
```

```
'ImRestore init' -> 3.0115 secs
'Computing features' -> 1.7968 secs
    (memoized) 'IMG_0249' -> 0.1426 secs
    (memoized) 'IMG_0250' -> 0.2903 secs
    (memoized) 'IMG_0251' -> 0.3198 secs
    (memoized) 'IMG_0252' -> 0.3195 secs
    (memoized) 'IMG_0253' -> 0.2270 secs
    (memoized) 'IMG_0254' -> 0.2367 secs
    (memoized) 'IMG_0256' -> 0.2170 secs
'Restoring' -> 0.9935 secs
    'Matching' -> 0.0329 secs
    'Merging' -> 0.9602 secs
        'IMG_0254' -> 0.9358 secs
'Post-processing' -> 0.2211 secs
```

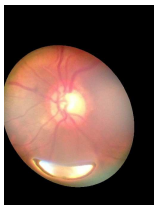
Figure 58. Result for set 5 using command '`imrestore ../results/set5/*.* --lens --overwrite --cachePath {temp}`'



(a) IMG_0253



(b) IMG_0254

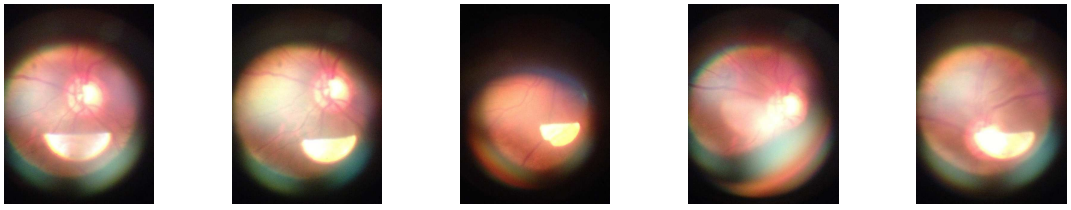


(c) Restored image

matched they produced a similar flare which can be seen in Figure 58c. 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 59).

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 22.41 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 section 13 'Future Work' on page 167). Even better the uncached time is bested when caching techniques are applied saving 19.40 seconds

Figure 59. Images not used in restoration of set 5



(a) IMG_0250 (b) IMG_0252 (c) IMG_0256 (d) IMG_0249 (e) IMG_0251

with its astonishing processing time of 3.01 seconds. The profiling puts in evidence that computing the feature is what takes most of the time in the application.

8.6 Result for set 6

Table 12

Profiling for set 6 using command 'imrestore ../results/set6/.* --lens --overwrite --cachePath {temp}'*

```
'ImRestore init' -> 21.1854 secs
└─ 'Computing features' -> 20.8283 secs
   ├─ (processed) 'IMG_0257' -> 2.9596 secs
   ├─ (processed) 'IMG_0258' -> 2.9003 secs
   ├─ (processed) 'IMG_0259' -> 3.0361 secs
   ├─ (processed) 'IMG_0260' -> 2.8409 secs
   ├─ (processed) 'IMG_0262' -> 3.0032 secs
   ├─ (processed) 'IMG_0263' -> 3.0359 secs
   └─ (processed) 'IMG_0264' -> 2.9956 secs
└─ 'Restoring' -> 0.0501 secs
   ├─ 'Matching' -> 0.0217 secs
   └─ 'Merging' -> 0.0282 secs
└─ 'Post-processing' -> 0.3070 secs
```

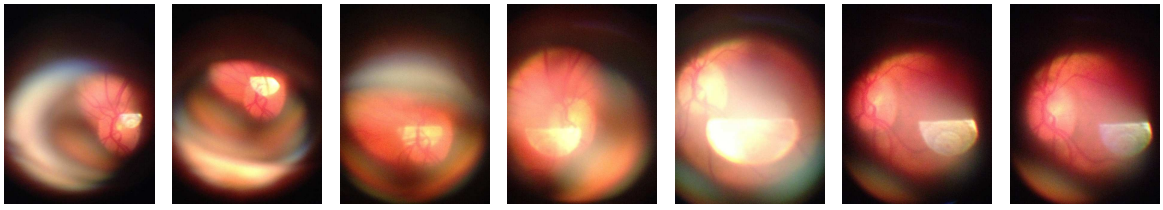
```
'ImRestore init' -> 0.7276 secs
└─ 'Computing features' -> 0.4117 secs
   ├─ (memoized) 'IMG_0257' -> 0.1135 secs
   ├─ (memoized) 'IMG_0258' -> 0.0637 secs
   ├─ (memoized) 'IMG_0259' -> 0.0364 secs
   ├─ (memoized) 'IMG_0260' -> 0.0338 secs
   ├─ (memoized) 'IMG_0262' -> 0.0397 secs
   ├─ (memoized) 'IMG_0263' -> 0.0523 secs
   └─ (memoized) 'IMG_0264' -> 0.0378 secs
└─ 'Restoring' -> 0.0381 secs
   ├─ 'Matching' -> 0.0123 secs
   └─ 'Merging' -> 0.0256 secs
└─ 'Post-processing' -> 0.2778 secs
```

Figure 60. Result for set 6 using command 'imrestore ../results/set6/*.* --lens --overwrite --cachePath {temp}'

(a) IMG_0257 (b) Restored image

In Figure 60 there is an interesting case, from the 7 images in the set the one in Figure 60a got selected as the *base image* but it has some parts of the *Sclera* which is

Figure 61. Images in set 6



(a) IMG_0257 (b) IMG_0258 (c) IMG_0259 (d) IMG_0260 (e) IMG_0262 (f) IMG_0263 (g) IMG_0264

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 set 5 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 set 6 shown in Figure 61. 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 set 6. It is evidenced that the time was reduced by 29.12 times of the uncached session. In this way time passed from 21.19 to 0.73 seconds saving in total 20.46 seconds. This express a gain of the 96.57% with cached time only being 3.43%, this means that an excess in time of the 2911.60% 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.

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.

8.7 Result for set 7

Using again Code 5 in this set *imrestore* processed 2 of the 5 images in it. The results can be appreciated in Figure 62 and the images not used in the restoration in Figure 63. There was a total of 1 merged images with the *base image* in Figure 62a and 3 unused images for the set 7. The restored image in Figure 62c does not evidence any change from the initial image but the lens simulation did a good job. There was nothing

Table 13

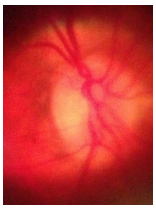
Profiling for set 7 using command '`imrestore ../results/set7/*.* --lens --overwrite --cachePath {temp}`'

```
'ImRestore init' -> 17.1325 secs
└─ 'Computing features' -> 15.7498 secs
   ├─ (processed) 'IMG_0267' -> 2.8843 secs
   ├─ (processed) 'IMG_0269' -> 3.2791 secs
   ├─ (processed) 'IMG_0270' -> 3.3335 secs
   ├─ (processed) 'IMG_0271' -> 2.9818 secs
   └─ (processed) 'IMG_0273' -> 3.2186 secs
└─ 'Restoring' -> 1.0576 secs
   ├─ 'Matching' -> 0.0854 secs
   └─ 'Merging' -> 0.9717 secs
      └─ 'IMG_0271' -> 0.9530 secs
└─ 'Post-processing' -> 0.3252 secs
```

```
'ImRestore init' -> 2.1764 secs
└─ 'Computing features' -> 1.1410 secs
   ├─ (memoized) 'IMG_0267' -> 0.3219 secs
   ├─ (memoized) 'IMG_0269' -> 0.1862 secs
   ├─ (memoized) 'IMG_0270' -> 0.2433 secs
   ├─ (memoized) 'IMG_0271' -> 0.2126 secs
   └─ (memoized) 'IMG_0273' -> 0.1477 secs
└─ 'Restoring' -> 0.8380 secs
   ├─ 'Matching' -> 0.0392 secs
   └─ 'Merging' -> 0.7984 secs
      └─ 'IMG_0271' -> 0.7822 secs
└─ 'Post-processing' -> 0.1974 secs
```

Figure 62. Result for set 7 using command '`imrestore ../results/set7/*.* --lens --overwrite --cachePath {temp}`'

(a) IMG_0270



(b) IMG_0271

(c) Restored image

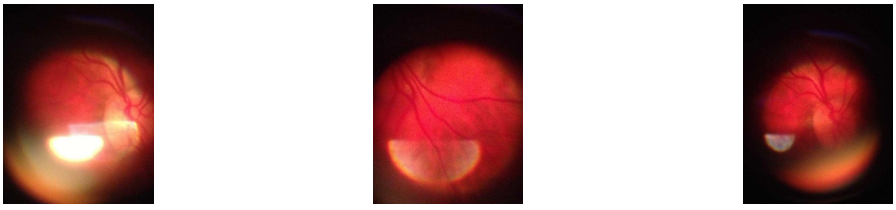
that the merged image in Figure 62b could provide to it an so nothing was introduced. The other images where not selected obviously for their lack of information and affected areas that could worsen the restoring process of the set.

Table 13 presents the profiling produced for set 7. It is evidenced that the time was reduced by 7.87 times of the uncached session. In this way time passed from 17.13 to 2.18 seconds saving in total 14.96 seconds. This is translated in a gain of the 87.30% with cache time being 12.70% and this means that an excess in time of the 787.19% 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.

8.8 Result for set 8

Images from this set were quickly computed and Figure 64 presents its results processing only 1 of the 7 images in it where the image in Figure 64a was selected as the *base image* leaving 6 images not used in the restoration of the set, this is shown in Figure 65. The restored image in Figure 64b shows a huge improvement over the input

Figure 63. Images not used in restoration of set 7



(a) IMG_0267 (b) IMG_0273 (c) IMG_0269

Table 14

Profiling for set 8 using command '`imrestore ../results/set8/*.* --lens --overwrite --cachePath {temp}`'

```
'ImRestore init' -> 21.1218 secs
'Computing features' -> 20.7679 secs
    (processed) 'IMG_0306' -> 3.2501 secs
    (processed) 'IMG_0307' -> 3.1120 secs
    (processed) 'IMG_0308' -> 3.0214 secs
    (processed) 'IMG_0309' -> 2.9660 secs
    (processed) 'IMG_0310' -> 2.9401 secs
    (processed) 'IMG_0311' -> 2.8666 secs
    (processed) 'IMG_0357' -> 2.5574 secs
'Restoring' -> 0.0548 secs
    'Matching' -> 0.0202 secs
    'Merging' -> 0.0345 secs
'Post-processing' -> 0.2990 secs
```

```
'ImRestore init' -> 1.3278 secs
'Computing features' -> 1.0215 secs
    (memoized) 'IMG_0306' -> 0.1880 secs
    (memoized) 'IMG_0307' -> 0.1139 secs
    (memoized) 'IMG_0308' -> 0.0432 secs
    (memoized) 'IMG_0309' -> 0.1384 secs
    (memoized) 'IMG_0310' -> 0.2814 secs
    (memoized) 'IMG_0311' -> 0.1659 secs
    (memoized) 'IMG_0357' -> 0.0515 secs
'Restoring' -> 0.0820 secs
    'Matching' -> 0.0147 secs
    'Merging' -> 0.0670 secs
'Post-processing' -> 0.2243 secs
```

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 set 8 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 21.12 seconds to 1.33 seconds in the cached session saving in total 19.79 seconds that is 15.91 times faster than the uncached session and cache time only being the 6.29% 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 Figure 65 present many problems and lack of information that cannot be used for the restoration of the *base image* in Figure 64a. 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

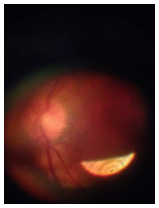
Figure 64. Result for set 8 using command '`imrestore ../results/set8/*.* --lens --overwrite --cachePath {temp}`'

(a) IMG_0306

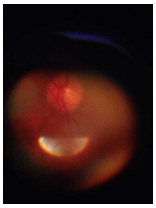
(b) Restored image

Figure 65. Images not used in restoration of set 8

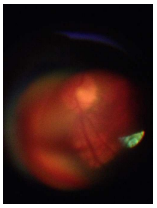
(a) IMG_0307



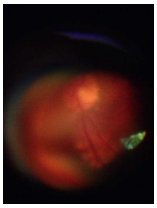
(b) IMG_0308



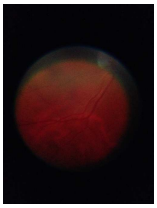
(c) IMG_0311



(d) IMG_0310



(e) IMG_0309



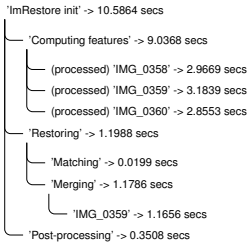
(f) IMG_0357

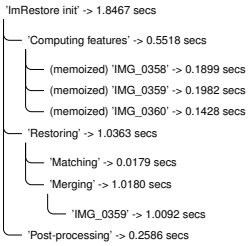
low then images were not adequately taken but if the used rate reaches 1 or unused rate 0 it means that images present high quality from good perspectives of the same retinal image.

8.9 Result for set 9

Table 15

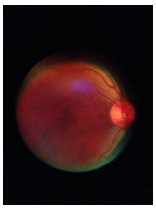
Profiling for set 9 using command '`imrestore ../results/set9/*.* --lens --overwrite --cachePath {temp}`'





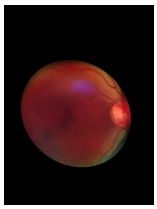
This set is only conformed by 3 images that are shown in Figure 67. Here no image presents good quality and there is some kind of blue light in two of them (Figure 67a and Figure 67b), the other is too bright to be selected appreciated in Figure 67a. In spite of that 2 were used for the restoration with 1 used in Figure 66b to merge in the *base image*

Figure 66. Result for set 9 using command '`imrestore ../results/set9/*.* --lens --overwrite --cachePath {temp}`'



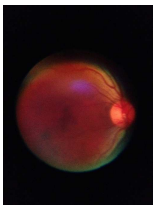
(a) IMG_0358

(b) IMG_0359



(c) Restored image

Figure 67. Images in set 9



(a) IMG_0358

(b) IMG_0359

(c) IMG_0360

in Figure 66a and the other in Figure 67c was wisely left out from the restoration for its brightness. Figure 66 shows the merging process with the restored image in Figure 66c. 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 66b preventing the *base image* to be ruined.

The profiling produced for set 9 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 5.73 times with the cached session because the process took 10.59 seconds in the uncached session (lets assign it the 100%) and then 1.85 seconds in the cached session (the 17.44%), taking 8.74 seconds of difference (82.56%). If the cached session consumed the 100% of the processed time then the uncached time would be 573.27%.

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.

8.10 Result for set 10

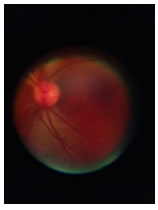
Table 16

Profiling for set 10 using command 'imrestore ../results/set10/.* --lens --overwrite --cachePath {temp}'*

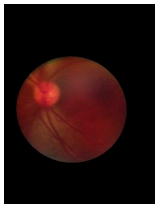
```
'ImRestore init' -> 19.6042 secs
└─ 'Computing features' -> 19.2559 secs
   ├─ (processed) 'IMG_0361' -> 1.8585 secs
   ├─ (processed) 'IMG_0362' -> 1.8819 secs
   ├─ (processed) 'IMG_0363' -> 1.8496 secs
   ├─ (processed) 'IMG_0364' -> 1.7994 secs
   ├─ (processed) 'IMG_0365' -> 1.8796 secs
   ├─ (processed) 'IMG_0366' -> 2.0074 secs
   ├─ (processed) 'IMG_0367' -> 1.9716 secs
   ├─ (processed) 'IMG_0368' -> 1.9464 secs
   ├─ (processed) 'IMG_0369' -> 1.9788 secs
   └─ (processed) 'IMG_0370' -> 2.0322 secs
└─ 'Restoring' -> 0.1014 secs
   ├─ 'Matching' -> 0.0222 secs
   └─ 'Merging' -> 0.0790 secs
└─ 'Post-processing' -> 0.2468 secs
```

```
'ImRestore init' -> 3.0809 secs
└─ 'Computing features' -> 2.7384 secs
   ├─ (memoized) 'IMG_0361' -> 0.1431 secs
   ├─ (memoized) 'IMG_0362' -> 0.1454 secs
   ├─ (memoized) 'IMG_0363' -> 0.2504 secs
   ├─ (memoized) 'IMG_0364' -> 0.1630 secs
   ├─ (memoized) 'IMG_0365' -> 0.1739 secs
   ├─ (memoized) 'IMG_0366' -> 0.1289 secs
   ├─ (memoized) 'IMG_0367' -> 1.0897 secs
   ├─ (memoized) 'IMG_0368' -> 0.3084 secs
   ├─ (memoized) 'IMG_0369' -> 0.0887 secs
   └─ (memoized) 'IMG_0370' -> 0.1997 secs
└─ 'Restoring' -> 0.0968 secs
   ├─ 'Matching' -> 0.0208 secs
   └─ 'Merging' -> 0.0758 secs
└─ 'Post-processing' -> 0.2457 secs
```

Figure 68. Result for set 10 using command 'imrestore ../results/set10/*.* --lens --overwrite --cachePath {temp}'

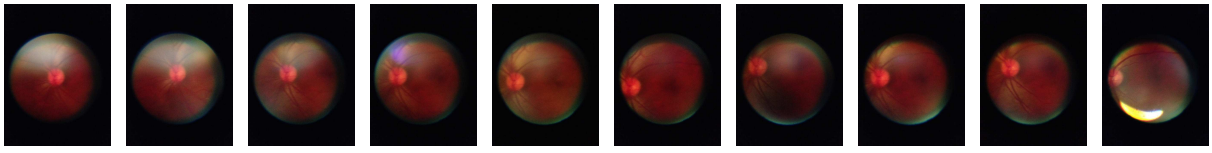


(a) IMG_0369



(b) Restored image

Figure 69. Images in set 10



(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

This set is presented in set 10 which exhibit similar to the cases of 2 and 3 as it is from the same eye but with different perspectives and problems. This time the set presents images with bright and dark areas, blurriness, flares and violet light distributed among the images and *imretore* must select and restorer the best images from it.

As always using the generic Code 5 to run *imrestore* in Figure 69 then just 1 image is processed of the 10 images in set 10 which is shown in Figure 68. This means that at least a selection has been carried out from the set to obtain the best image shown in Figure 68a and then try to mere more images to it but because no more images could be matched to the *base image* it was post-processed with lens simulation producing what is observed in Figure 68b.

The profiling produced for running Code 5 in the set 10 is shown in Table 16 . The analysis clearly can be extracted from this table which shows how time passed from 19.60 (left) to 3.08 seconds (right) meaning that time was reduced by 6.36 times from the uncached session saving in total 16.52 seconds, with a time difference from uncahed to cached of 84.28% and cache time being the 15.72%. The rate of used images is maintained low in 0.10 due to low image usage and consecutively the rate of failed images high in 0.90.

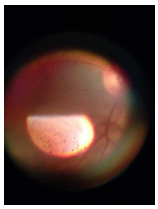
8.11 Result for set 11

Table 17

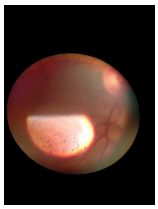
Profiling for set 11 using command '`imrestore ../results/set11/*.* --lens --overwrite --cachePath {temp}`'

This set is shown in Figure 71. By now, there is a common problem regarding how images were acquired for the set and that is that images are not taken in an organized manner without any strategy and lacking enough information for their restoration, This set is not exception to this, which presents some images that are from a central perspective (images in Figure 71a to Figure 71d) and other of a lower perspective in the retinal image (images in Figure 71e and Figure 71f).

Figure 70. Result for set 11 using command '`imrestore ../results/set11/*.* --lens --overwrite --cachePath {temp}`'



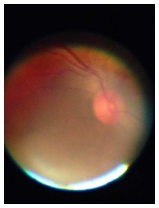
(a) IMG_0379



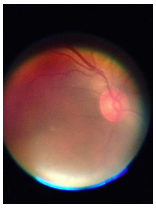
(b) Restored image

Figure 71. Images in set 11

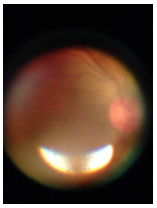
(a) IMG_0373



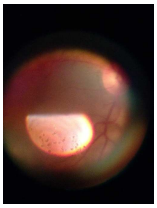
(b) IMG_0374



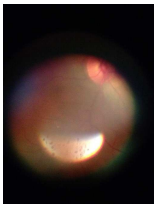
(c) IMG_0375



(d) IMG_0377



(e) IMG_0379



(f) IMG_0380

As before, using Code 5 in set 11, which was not adequately taken, process 1 of the 6 images in it (Figure 70), where *imrestore* tries to restore as better as possible with the selected configuration all the images that it can get from the set. This time it selected Figure 70a but as we can appreciate it contains a big flare in the lower half of it that do not facilitate the restoring of this image. Despite it, there is another image of the same perspective, in Figure 71f, but it has a similar problem as the *base image* and was not merged by *imrestore* possibly due to its blurriness leading to a result without much changes as is seen in Figure 70b.

Timings can be appreciated in the profiling table presented in Table 17 produced for set 11. It shows all the processing times of each step of the restoration, the time in the uncached session was of 13.06 seconds and 1.22 in the the cached session. Meaning that time was reduced by 10.72 times saving in total 11.84 seconds which is the 90.67% of the total and cache time percentage only 9.33%.

As it was stated before then one image was selected as the *base image* and 0 merged in the it, making a total of 1 used images and 5 images not used in set 11. This expressed in rates is 0.17 for used images and 0.83 for images not used from the set.

8.12 Result for set 12

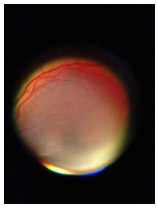
Table 18

Profiling for set 12 using command 'imrestore ../results/set12/.* --lens --overwrite --cachePath {temp}'*

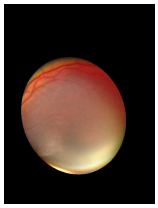
```
'ImRestore init' -> 9.1315 secs
  'Computing features' -> 8.5797 secs
    (processed) 'IMG_0381' -> 2.2058 secs
    (processed) 'IMG_0382' -> 2.1365 secs
    (processed) 'IMG_0383' -> 2.0798 secs
    (processed) 'IMG_0384' -> 2.1274 secs
  'Restoring' -> 0.0690 secs
    'Matching' -> 0.0132 secs
    'Merging' -> 0.0556 secs
  'Post-processing' -> 0.4828 secs
```

```
'ImRestore init' -> 0.5691 secs
  'Computing features' -> 0.2253 secs
    (memoized) 'IMG_0381' -> 0.0467 secs
    (memoized) 'IMG_0382' -> 0.0536 secs
    (memoized) 'IMG_0383' -> 0.0402 secs
    (memoized) 'IMG_0384' -> 0.0620 secs
  'Restoring' -> 0.0495 secs
    'Matching' -> 0.0092 secs
    'Merging' -> 0.0401 secs
  'Post-processing' -> 0.2944 secs
```

Figure 72. Result for set 12 using command 'imrestore ../results/set12/*.* --lens --overwrite --cachePath {temp}'

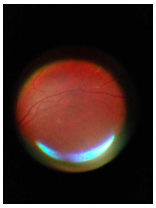


(a) IMG_0382

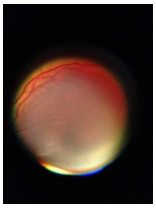


(b) Restored image

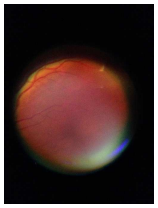
Figure 73. Images in set 12



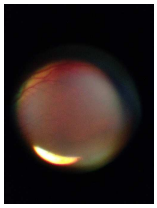
(a) IMG_0381



(b) IMG_0382



(c) IMG_0383



(d) IMG_0384

This set contains plain images with not optic disks and few arteries appreciated in Figure 73. *imrestore* selected one of the 4 images in set 12. Leading to 0 images merged in the *base image* and 3 images not used for the set 12. Figure 72 presents the results for set 12. Figure 72a shows the *base image* and Figure 72b shows the restored image.

Table 18 presents the profiling produced for set 12. Time passed from 9.13 to 0.57 seconds subtracting 8.56 seconds to the uncached session. This is translated in a gain

of the 93.77% with cache time being 6.23%. The rate of used images is 0.25 and the rate of failed images is 0.75.

8.13 Result for set 13

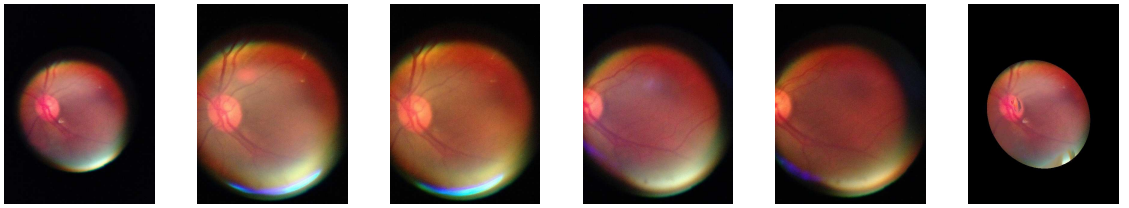
Table 19

Profiling for set 13 using command 'imrestore ../results/set13/.* --lens --overwrite --cachePath {temp}'*

```
'ImRestore init' -> 17.8982 secs
└─ 'Computing features' -> 14.2303 secs
   ├─ (processed) 'IMG_0386' -> 2.5270 secs
   ├─ (processed) 'IMG_0387' -> 3.4327 secs
   ├─ (processed) 'IMG_0388' -> 3.8942 secs
   ├─ (processed) 'IMG_0389' -> 2.2344 secs
   └─ (processed) 'IMG_0390' -> 2.1062 secs
└─ 'Restoring' -> 3.4140 secs
   ├─ 'Matching' -> 0.0399 secs
   └─ 'Merging' -> 3.3736 secs
      ├─ 'IMG_0387' -> 0.8304 secs
      ├─ 'IMG_0388' -> 0.8383 secs
      ├─ 'IMG_0390' -> 0.8454 secs
      └─ 'IMG_0389' -> 0.8595 secs
└─ 'Post-processing' -> 0.2539 secs
```

```
'ImRestore init' -> 4.9401 secs
└─ 'Computing features' -> 0.9341 secs
   ├─ (memoized) 'IMG_0386' -> 0.1664 secs
   ├─ (memoized) 'IMG_0387' -> 0.2511 secs
   ├─ (memoized) 'IMG_0388' -> 0.1996 secs
   ├─ (memoized) 'IMG_0389' -> 0.1565 secs
   └─ (memoized) 'IMG_0390' -> 0.1310 secs
└─ 'Restoring' -> 3.7689 secs
   ├─ 'Matching' -> 0.0358 secs
   └─ 'Merging' -> 3.7326 secs
      ├─ 'IMG_0387' -> 0.9965 secs
      ├─ 'IMG_0388' -> 0.9516 secs
      ├─ 'IMG_0390' -> 0.8881 secs
      └─ 'IMG_0389' -> 0.8962 secs
└─ 'Post-processing' -> 0.2371 secs
```

Figure 74. Result for set 13 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

Until now not many images had been merged to a single *base image* but in this set all the images where used in the restoration, that is 5 images used of the 5 images in set 13, 4 images merged in the *base image* and 0 images not used for the set 13. Figure 74 presents the results for set 13 with the *base image* in Figure 74a, the restored image in Figure 74f and the remaining merged images.

Table 19 shows the profiling produced for set 13. Uncached time was reduced 3.62 times from 17.90 to 4.94 seconds sparing 12.96 seconds of difference with the 72.40%

and cache time the 27.60%. Finally this set shows a used rate of 1.00 and images not used of 0.00.

8.14 Result for set 14

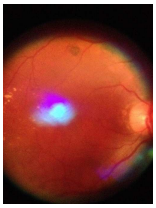
Table 20

Profiling for set 14 using command '`imrestore ../results/set14/*.* --lens --overwrite --cachePath {temp}`'

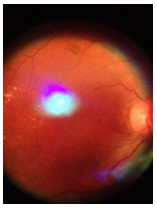
```
'ImRestore init' -> 11.5620 secs
  'Computing features' -> 9.8067 secs
    (processed) 'IMG_0401' -> 2.4863 secs
    (processed) 'IMG_0402' -> 2.4807 secs
    (processed) 'IMG_0403' -> 2.4570 secs
    (processed) 'IMG_0404' -> 2.3533 secs
  'Restoring' -> 1.5220 secs
    'Matching' -> 0.0431 secs
    'Merging' -> 1.4785 secs
      'IMG_0403' -> 0.7409 secs
      'IMG_0404' -> 0.7289 secs
  'Post-processing' -> 0.2333 secs
```

```
'ImRestore init' -> 3.2764 secs
  'Computing features' -> 1.1908 secs
    (memoized) 'IMG_0401' -> 0.2324 secs
    (memoized) 'IMG_0402' -> 0.5368 secs
    (memoized) 'IMG_0403' -> 0.2392 secs
    (memoized) 'IMG_0404' -> 0.1547 secs
  'Restoring' -> 1.8744 secs
    'Matching' -> 0.0382 secs
    'Merging' -> 1.8356 secs
      'IMG_0403' -> 0.8945 secs
      'IMG_0404' -> 0.9334 secs
  'Post-processing' -> 0.2113 secs
```

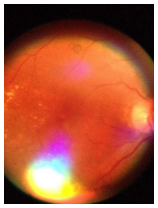
Figure 75. Result for set 14 using command '`imrestore ../results/set14/*.* --lens --overwrite --cachePath {temp}`'



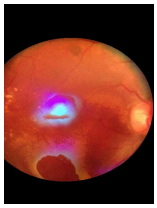
(a) IMG_0402



(b) IMG_0403

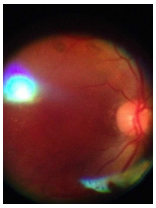


(c) IMG_0404

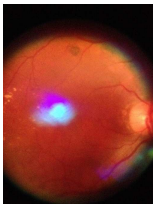


(d) Restored image

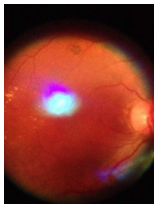
Figure 76. Images in set 14



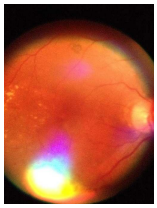
(a) IMG_0401



(b) IMG_0402



(c) IMG_0403



(d) IMG_0404

Using Code 5 in set 14 shown in Figure 76 *imrestore* processed 3 of the 4 images in it, this is shown in Figure 75. 2 images were merged into the *base image* in Figure 75a

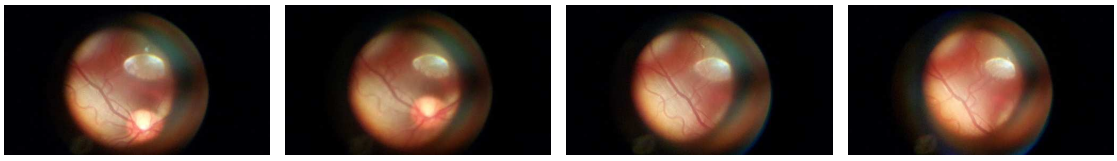
and 1 were left out from the set. As it can be seen the *base image* has a whitish flare in the center and the merging images are used to try to solve this but they also present similar problems. At the end. the restored image in Figure 75d is introduced with some of the noises in the merging images, as they present the same and more problems than the *base image*. But this set also demonstrates something important about the algorithm of *imrestore*, which is shown in the inferior part of the restored image where the merging image in Figure 75c had a big flare in that position but *imrestore* tried to no introduce more noise than it already had and this flare was filter out, though this process was not perfect.

Table 20 presents the profiling produced for set 14 where 11.56 seconds elapsed when the `--cachePath` option was not specified and 3.28 seconds when it was used. This way time was reduced by 3.53 times saving in total 8.29 seconds. This is translated in a reduction of the 71.66% with cache time being only 28.34% from the normal time when it is not cached.

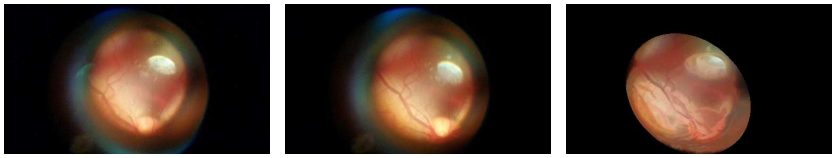
As stated before 3 images were used in this set leading to a rate of used images in 0.75 and a rate of not used images in 0.25.

8.15 Result for set 15

Figure 77. Result for set 15 using command '`imrestore ../results/set15/*.* --lens --overwrite --cachePath {temp}`'



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

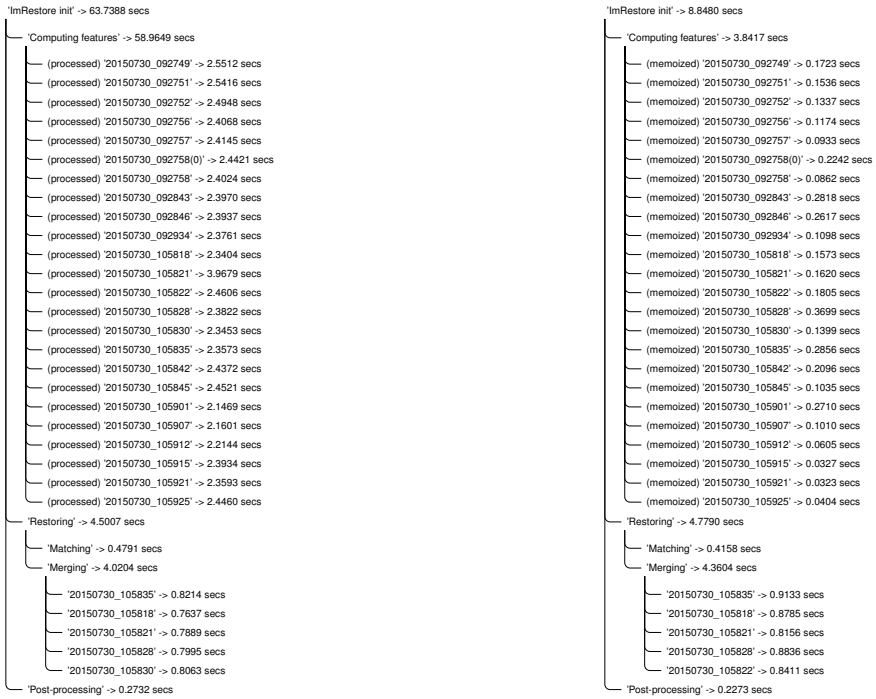


(e) 20150730_105828 (f) 20150730_105822 (g) Restored image

Here it is presented one of the sets with most images in it with an astonishing number of 24 images. Running Code 5 in set 15 produced the result in Figure 77. As expected

Table 21

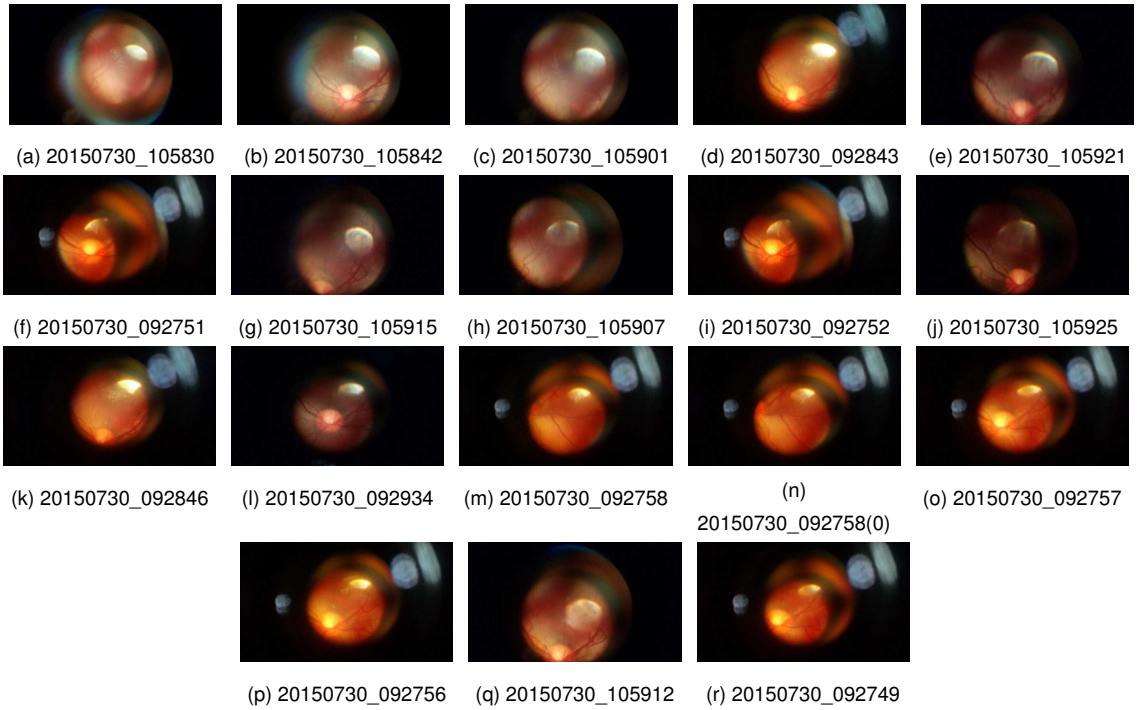
Profiling for set 15 using command 'imrestore ../results/set15/.* --lens --overwrite --cachePath {temp}'*



more images were used than the sets with few images, this time using 6 images. Figure 78 shows the images not used in set 15, where 18 images were left out from the restoration. These images could not be used to be merged in the *base image* presented in Figure 77a because they don't presented similar features than it, opposed to the 5 images used to be merged in Figure 77a which are very similar to it.

The restored image is shown in Figure 77g but it does not look so good as the optic disk has disappeared and the merged images look like patches in it. This is due to three main things: first, there are two images in Figure 77c and Figure 77d which are located in the upper part of the optic disk from the others which where not adequately matched causing the placement of parts that did not correspond to the optic disk; two, there was not applied the histogram matching method to provide similar colors between merges, this can be activated using `--hist_match` in the command; three, the implementation of the optic disk localization and segmentation was not used in this test as it is still in development (see Experimental watershed method to find optic disk area). Despite all this, it did a good job selecting the best images in the set but more tweaking is needed.

Figure 78. Images not used in restoration of set 15



Because there was a lot of images to choose there should be a lot of time involved in the 'Matching' step which is where images are related to the *base image* to be passed to the other classification steps and eventually for merging if they pass all tests. Nonetheless, it took less than 1 second for both the cached and uncached sessions. What took most of the time were the 'Merging' in both sessions and 'Computing features' in the uncached session.

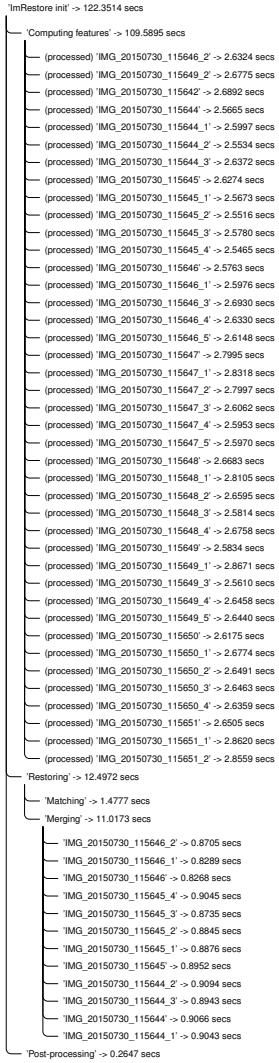
In Table 21 the profiling is presented which was produced for set 15. It is observed that the unchached time was of 63.74 seconds, cached time 8.85 seconds and the difference 54.89 seconds of saved time. The reduced time was 86.12% and the cache time 13.88%. If the cache time is selected as the 100% then the uncached time would have an excess of 720.38%.

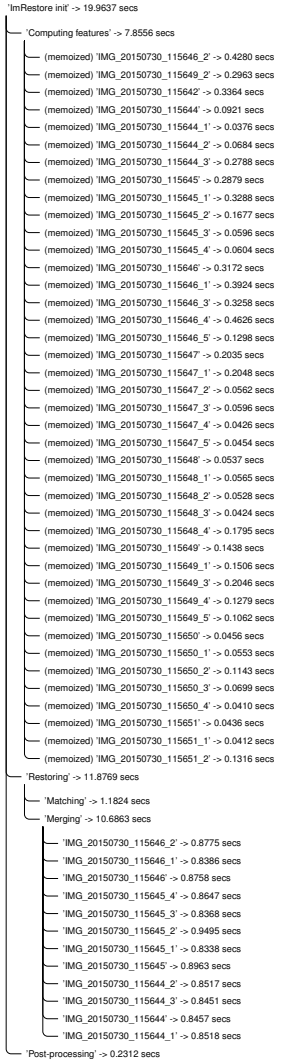
The rate of used images is 0.25, a low value because there were a lot of images but less than half of them were used. Consequently the rate of images not used in set 15 is 0.75.

8.16 Result for set 16

Table 22

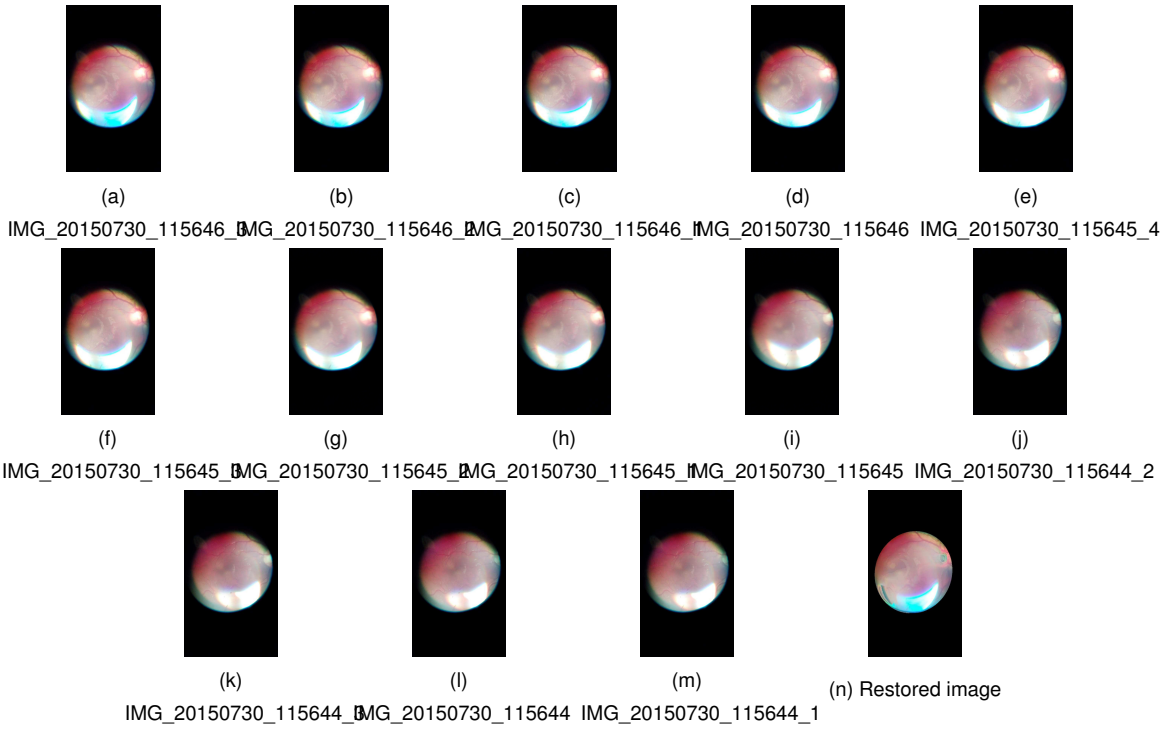
Profiling for set 16 using command '`imrestore ../results/set16/*.* --lens --overwrite --cachePath {temp}`'





This is a big set with 41 images thus the image for it will not be presented in here but it can be found as any other result in the repository under 'results/set16' (Toro, 2016). Once Code 5 is run in set 16 *imrestore* produces the result as shown in Figure 79. There is not even need to see images in the set as they all are similar to the ones in the restoration and because they present the same same problem without optic disk and not much in it the restored image in Figure 79n ends up with not much to show but a similar image

Figure 79. Result for set 16 using command '`imrestore ../results/set16/*.* --lens --overwrite --cachePath {temp}`'



to the others. Even though the flare was reduced and the lens simulation was applied nothing else is changed.

In the results it is clear that 13 of the 41 images in set 16 were used and of them of course 12 correspond to the merged images used for the *base image* in Figure 79a. Thus leaving a total of 28 images which were not used. The rates then are 0.32 for used images, 0.29 for merged images and 0.68 for images not used in the set.

Table 22 presents the profiling produced for set 16. In it we can see that the time was reduced by 6.13 times because it passed from 122.35 to 19.96 seconds from uncached to cached respectively. As always the cached session reduced the time sparing 102.39 seconds in total, the 83.68% of the uncached time.

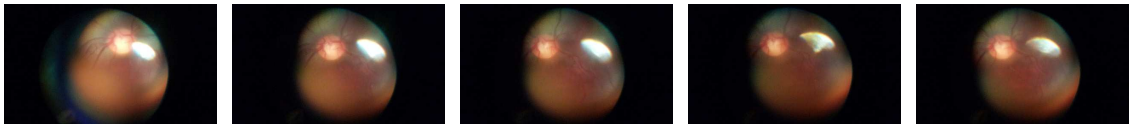
Table 23

Profiling for set 17 using command '`imrestore ../results/set17/*.* --lens --overwrite --cachePath {temp}`*'*

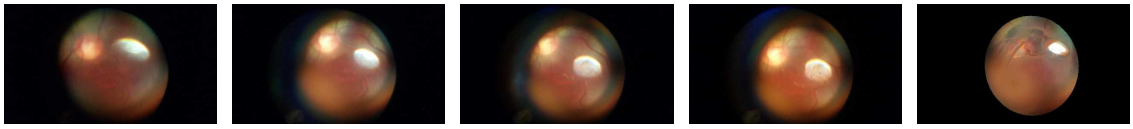
```
'ImRestore init' -> 47.7613 secs
'Computing features' -> 42.2944 secs
    (processed) '20150730_110405' -> 2.5507 secs
    (processed) '20150730_110530' -> 2.4547 secs
    (processed) '20150730_110539' -> 2.4319 secs
    (processed) '20150730_110545' -> 2.4963 secs
    (processed) '20150730_110548' -> 2.6405 secs
    (processed) '20150730_110552' -> 2.3985 secs
    (processed) '20150730_110645' -> 2.3480 secs
    (processed) '20150730_110648' -> 2.3984 secs
    (processed) '20150730_110650' -> 2.4408 secs
    (processed) '20150730_110656' -> 2.3653 secs
    (processed) '20150730_110657' -> 2.4270 secs
    (processed) '20150730_110659' -> 2.4052 secs
    (processed) '20150730_110705' -> 2.4033 secs
    (processed) '20150730_110706' -> 2.6550 secs
    (processed) '20150730_110711' -> 2.6356 secs
    (processed) '20150730_110714' -> 2.5944 secs
    (processed) '20150730_110717' -> 2.5888 secs
'Restoring' -> 5.1680 secs
    'Matching' -> 0.2958 secs
    'Merging' -> 4.8711 secs
        '20150730_110659' -> 0.9549 secs
        '20150730_110706' -> 0.9537 secs
        '20150730_110714' -> 0.9653 secs
        '20150730_110705' -> 0.9496 secs
        '20150730_110711' -> 0.9950 secs
'Post-processing' -> 0.2989 secs
```

```
'ImRestore init' -> 10.6162 secs
'Computing features' -> 2.6276 secs
    (memoized) '20150730_110405' -> 0.1682 secs
    (memoized) '20150730_110530' -> 0.1983 secs
    (memoized) '20150730_110539' -> 0.1323 secs
    (memoized) '20150730_110545' -> 0.1078 secs
    (memoized) '20150730_110548' -> 0.1146 secs
    (memoized) '20150730_110552' -> 0.2861 secs
    (memoized) '20150730_110645' -> 0.1379 secs
    (memoized) '20150730_110648' -> 0.0720 secs
    (memoized) '20150730_110650' -> 0.0594 secs
    (memoized) '20150730_110656' -> 0.1289 secs
    (memoized) '20150730_110657' -> 0.1101 secs
    (memoized) '20150730_110659' -> 0.1245 secs
    (memoized) '20150730_110705' -> 0.1449 secs
    (memoized) '20150730_110706' -> 0.2813 secs
    (memoized) '20150730_110711' -> 0.1762 secs
    (memoized) '20150730_110714' -> 0.1405 secs
    (memoized) '20150730_110717' -> 0.2112 secs
'Restoring' -> 7.7371 secs
    'Matching' -> 0.3296 secs
    'Merging' -> 7.4050 secs
        '20150730_110659' -> 0.8815 secs
        '20150730_110706' -> 0.9012 secs
        '20150730_110714' -> 0.8845 secs
        '20150730_110705' -> 0.8919 secs
        '20150730_110711' -> 0.8832 secs
        '20150730_110545' -> 0.9031 secs
        '20150730_110530' -> 0.9007 secs
        '20150730_110539' -> 1.1304 secs
'Post-processing' -> 0.2515 secs
```

Figure 80. Result for set 17 using command '`imrestore ../results/set17/*.* --lens --overwrite --cachePath {temp}`'



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

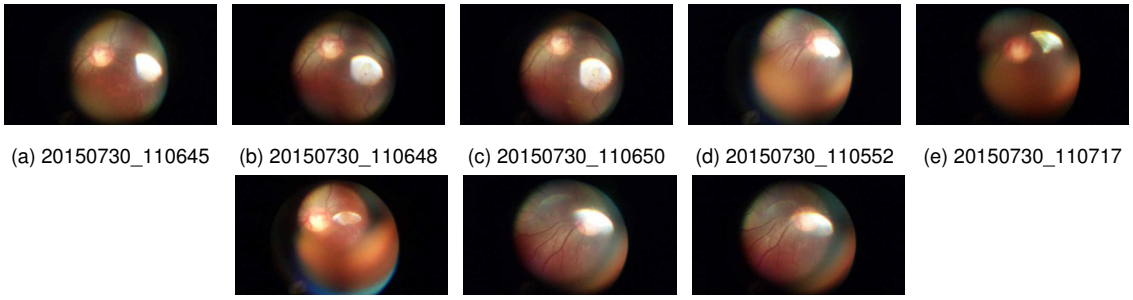


(f) 20150730_110711 (g) 20150730_110545 (h) 20150730_110530 (i) 20150730_110539 (j) Restored image

8.17 Result for set 17

This set has almost all images of the same perspective, when Code 5 is applied in set 17 *imrestore* process 9 of the 17 images in it with 8 merged images into the *base image* which is in Figure 80a. Figure 81 shows the images not used in set 17 with a number of 8 not used for this set. The restored image is shown in Figure 80j and it does not look good due to the replacement of the optic disk and the brush-like strokes of the

Figure 81. Images not used in restoration of set 17



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

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

merging produced in it.

Once more the profiling is presented in Table 23 produced for set 17. Time was reduced by 4.50 times in the cached session compared to the uncached session. Time passed from 47.76 seconds to 10.62 seconds saving in total 37.15 seconds. This in percentage is the 77.77% with cache time being 22.23%.

The rate of used images is 0.53 and the rate of failed images is 0.47.

8.18 Result for set 18

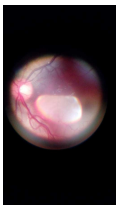
Table 24

Profiling for set 18 using command '`imrestore ../results/set18/*.* --lens --overwrite --cachePath {temp}`'

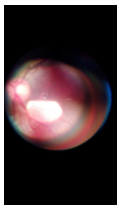
'ImRestore init' -> 45.8762 secs
- 'Computing features' -> 45.3277 secs
  - (processed) 'IMG_20150730_105626' -> 2.8179 secs
  - (processed) 'IMG_20150730_105632' -> 2.7198 secs
  - (processed) 'IMG_20150730_105655' -> 2.7248 secs
  - (processed) 'IMG_20150730_105748' -> 2.6410 secs
  - (processed) 'IMG_20150730_105753' -> 2.8209 secs
  - (processed) 'IMG_20150730_105800' -> 2.7615 secs
  - (processed) 'IMG_20150730_115251' -> 2.6234 secs
  - (processed) 'IMG_20150730_115254' -> 2.6153 secs
  - (processed) 'IMG_20150730_115301' -> 2.6390 secs
  - (processed) 'IMG_20150730_131444' -> 2.5613 secs
  - (processed) 'IMG_20150730_131454' -> 2.5061 secs
  - (processed) 'IMG_20150730_134219' -> 2.6796 secs
  - (processed) 'IMG_20150730_134225' -> 2.6494 secs
  - (processed) 'IMG_20150730_134231' -> 2.6358 secs
  - (processed) 'IMG_20150730_134235' -> 2.6547 secs
  - (processed) 'IMG_20150730_134254' -> 2.5857 secs
  - (processed) 'IMG_20150730_134300' -> 2.5494 secs
- 'Restoring' -> 0.2546 secs
  - 'Matching' -> 0.0799 secs
  - 'Merging' -> 0.1744 secs
- 'Post-processing' -> 0.2939 secs

'ImRestore init' -> 4.3498 secs
- 'Computing features' -> 3.0191 secs
  - (memoized) 'IMG_20150730_105626' -> 0.2472 secs
  - (memoized) 'IMG_20150730_105632' -> 0.1855 secs
  - (memoized) 'IMG_20150730_105655' -> 0.1630 secs
  - (memoized) 'IMG_20150730_105748' -> 0.1199 secs
  - (memoized) 'IMG_20150730_105753' -> 0.0982 secs
  - (memoized) 'IMG_20150730_105800' -> 0.0760 secs
  - (memoized) 'IMG_20150730_115251' -> 0.3270 secs
  - (memoized) 'IMG_20150730_115254' -> 0.1426 secs
  - (memoized) 'IMG_20150730_115301' -> 0.1754 secs
  - (memoized) 'IMG_20150730_131444' -> 0.1642 secs
  - (memoized) 'IMG_20150730_131454' -> 0.1621 secs
  - (memoized) 'IMG_20150730_134219' -> 0.1458 secs
  - (memoized) 'IMG_20150730_134225' -> 0.3793 secs
  - (memoized) 'IMG_20150730_134231' -> 0.1841 secs
  - (memoized) 'IMG_20150730_134235' -> 0.1107 secs
  - (memoized) 'IMG_20150730_134254' -> 0.1548 secs
  - (memoized) 'IMG_20150730_134300' -> 0.1043 secs
- 'Restoring' -> 1.0845 secs
  - 'Matching' -> 0.1135 secs
  - 'Merging' -> 0.9702 secs
    - 'IMG_20150730_105748' -> 0.8686 secs
- 'Post-processing' -> 0.2462 secs

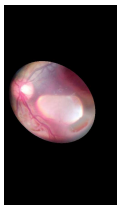
Figure 82. Result for set 18 using command '`imrestore ../results/set18/*.* --lens --overwrite --cachePath {temp}`'



(a) IMG_20150730_105626



(b) IMG_20150730_105748

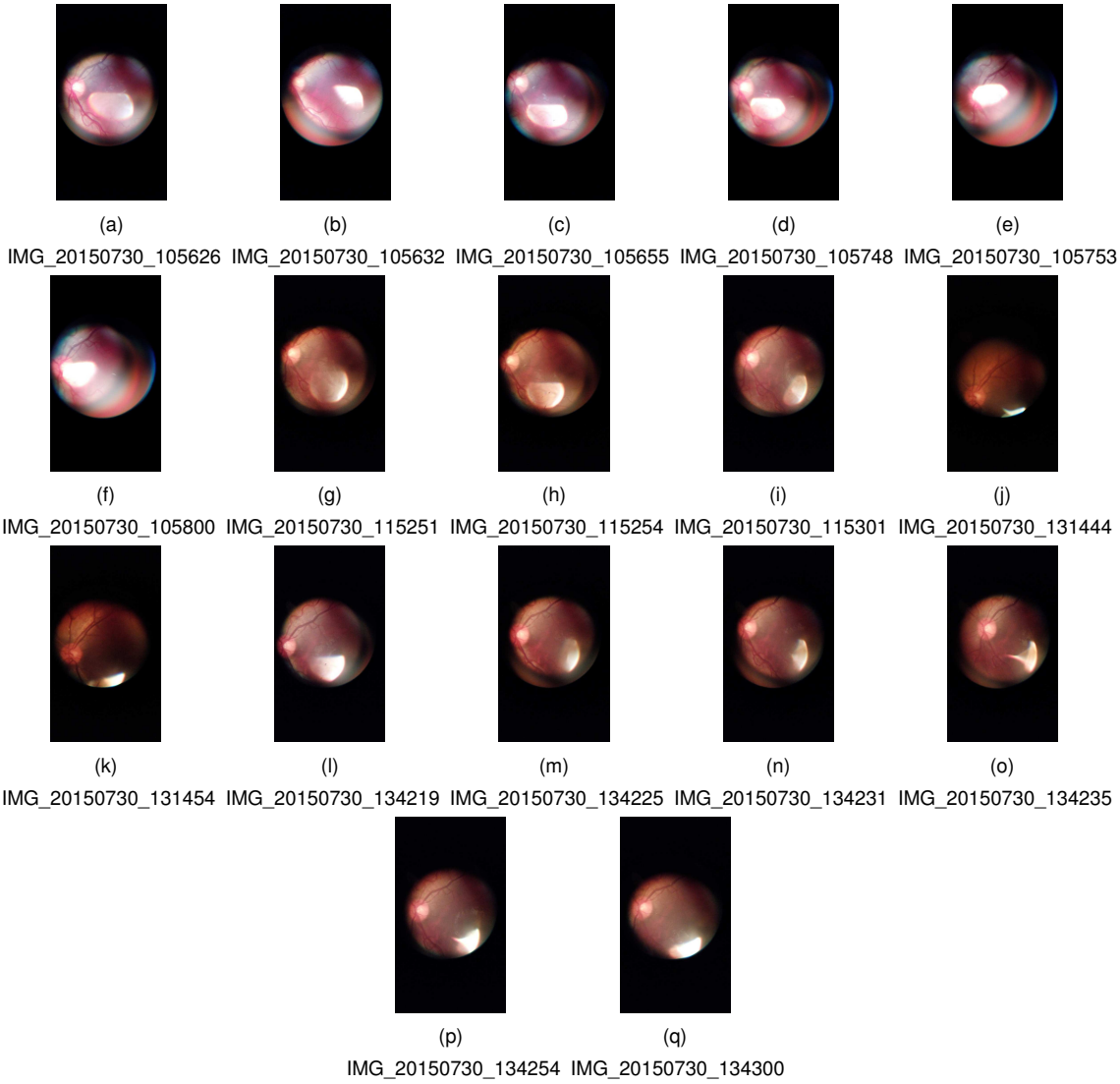


(c) Restored image

This set is shown in Figure 83. Using Code 5 *imrestore* processed 2 of the 17 images in set 18. Figure 82 presents the results for set 18. One image was selected as the *base image*, 1 image was merged in it and 15 images were not used. The *base image* can be seen in Figure 82a and the restored image in Figure 82c. The rate of used images is 0.12 and the rate of unused images is 0.88.

Table 24 presents the profiling produced for set 18. No much change is observed when processing the restoration with respect to the other sets. Time is always reduced when it is switched from uncached to cached mode and this time 10.55 times from the uncached session. 45.88 seconds for the uncached time and 4.35 seconds for the cached time, saving in total 41.53 seconds, this is the 90.52% with cache time being 9.48%.

Figure 83. Images in set 18



8.19 Result for set 19

This set demonstrates one of the most important features of *imrestore* and it is its robustness with respect to brightness. The results of applying Code 5 in set 19 are shown in Figure 84 and the images that were not used from the set are in Figure 85.

Figure 84a shows the *base image*, Figure 84e shows the restored image and the remaining merged images in Figure 84. As we can see the selected *base image* is one of the most neutral images in the set without too much brightness or too much darkness

Table 25

Profiling for set 19 using command '`imrestore ../results/set19/*.* --lens --overwrite --cachePath {temp}`'

Figure 84. Result for set 19 using command '`imrestore ../results/set19/*.* --lens --overwrite --cachePath {temp}`'

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

that could produce unappealing results. Also, it is noticed that though the *base image* have no significant problems in it *imrestore* just kept on selecting matching images for merging but at the end the restored image just kept the same characteristics of the *base image*. This is due to the way *imrestore* is coded, because it does not checks if the image presents problems, it just tries to merge the best images from a set and complete lacking information based in some parameters. Prove of this is also noticed in the images that were not used, Figure 85a has too much brightness in it to be selectable or Figure 85b and Figure 85d are too dark to be included in the merging images.

imrestore processed 4 of the 8 images in set 19 where 3 images were merged in the *base image* and 4 images were not used for set 19, so the rate of used images is 0.50, the rate of merged images is 0.38 and the rate of unused images is 0.50.

Table 25 presents the profiling produced for set 19 where it is evidenced that time was reduced by 5.82 times of the uncached session when time passed from 24.55 to 4.22 seconds saving in total 20.33 seconds. This saved time in percentage is the 82.81%

Figure 85. Images not used in restoration of set 19

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

with cache time being 17.19%, meaning that an excess of the 581.65% would happen if passed from cached to a noncached session.

8.20 Result for set 20

Table 26

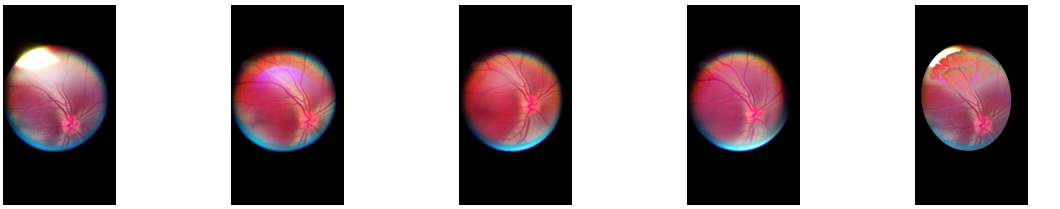
Profiling for set 20 using command '`imrestore ../results/set20/*.* --lens --overwrite --cachePath {temp}`'

```
'ImRestore init' -> 71.8632 secs
'Computing features' -> 67.3008 secs
(processed) '20150730_132852' -> 2.7460 secs
(processed) '20150730_132857' -> 2.8315 secs
(processed) '20150730_132900' -> 2.7682 secs
(processed) '20150730_132904' -> 2.5345 secs
(processed) '20150730_132917' -> 2.6611 secs
(processed) '20150730_132924' -> 2.6639 secs
(processed) '20150730_132929' -> 2.6660 secs
(processed) '20150730_132932' -> 2.6595 secs
(processed) '20150730_132936' -> 2.7086 secs
(processed) 'IMG_20150730_122502' -> 2.6189 secs
(processed) 'IMG_20150730_122504' -> 2.8477 secs
(processed) 'IMG_20150730_122506' -> 2.5372 secs
(processed) 'IMG_20150730_122520' -> 2.7516 secs
(processed) 'IMG_20150730_130654' -> 3.2706 secs
(processed) 'IMG_20150730_130659' -> 3.2889 secs
(processed) 'IMG_20150730_130706' -> 3.1995 secs
(processed) 'IMG_20150730_130713' -> 2.9236 secs
(processed) 'IMG_20150730_130719' -> 2.9012 secs
(processed) 'IMG_20150730_130723' -> 3.0035 secs
(processed) 'IMG_20150730_130729' -> 2.5721 secs
(processed) 'IMG_20150730_130756' -> 2.8247 secs
(processed) 'IMG_20150730_130759' -> 2.8306 secs
(processed) 'IMG_20150730_130803' -> 2.6994 secs
(processed) 'IMG_20150730_130809' -> 2.6433 secs
'Restoring' -> 4.2688 secs
'Matching' -> 0.4537 secs
'Merging' -> 3.8141 secs
'IMG_20150730_130659' -> 0.9110 secs
'IMG_20150730_130723' -> 0.8977 secs
'IMG_20150730_130713' -> 0.9377 secs
'IMG_20150730_130719' -> 0.9506 secs
'Post-processing' -> 0.2936 secs
```

```
'ImRestore init' -> 6.3952 secs
'Computing features' -> 3.0495 secs
(memoized) '20150730_132852' -> 0.0558 secs
(memoized) '20150730_132857' -> 0.0583 secs
(memoized) '20150730_132900' -> 0.0869 secs
(memoized) '20150730_132904' -> 0.1420 secs
(memoized) '20150730_132917' -> 0.1290 secs
(memoized) '20150730_132924' -> 0.1558 secs
(memoized) '20150730_132929' -> 0.1617 secs
(memoized) '20150730_132932' -> 0.1090 secs
(memoized) '20150730_132936' -> 0.2885 secs
(memoized) 'IMG_20150730_122502' -> 0.1432 secs
(memoized) 'IMG_20150730_122504' -> 0.1898 secs
(memoized) 'IMG_20150730_122506' -> 0.1465 secs
(memoized) 'IMG_20150730_122520' -> 0.1208 secs
(memoized) 'IMG_20150730_130654' -> 0.2447 secs
(memoized) 'IMG_20150730_130659' -> 0.4031 secs
(memoized) 'IMG_20150730_130706' -> 0.1385 secs
(memoized) 'IMG_20150730_130713' -> 0.0647 secs
(memoized) 'IMG_20150730_130719' -> 0.0537 secs
(memoized) 'IMG_20150730_130723' -> 0.0658 secs
(memoized) 'IMG_20150730_130729' -> 0.0327 secs
(memoized) 'IMG_20150730_130756' -> 0.0446 secs
(memoized) 'IMG_20150730_130759' -> 0.0275 secs
(memoized) 'IMG_20150730_130803' -> 0.0467 secs
(memoized) 'IMG_20150730_130809' -> 0.0457 secs
'Restoring' -> 3.0853 secs
'Matching' -> 0.3682 secs
'Merging' -> 2.7148 secs
'IMG_20150730_130659' -> 0.8775 secs
'IMG_20150730_130723' -> 0.8506 secs
'IMG_20150730_130713' -> 0.8866 secs
'Post-processing' -> 0.2604 secs
```

This set is shown in Figure 87. Using Code 5 in set 20 *imrestore* used in the restoration 4 of the 24 images in set 20, this result is shown in Figure 86 where 3 were merged into the selected *base image* in Figure 86a leaving which exhibit an unsatisfactory result. Some times *imrestore* does not select the images that we would expect from a set, to my

Figure 86. Result for set 20 using command '`imrestore ../results/set20/*.* --lens --overwrite --cachePath {temp}`'



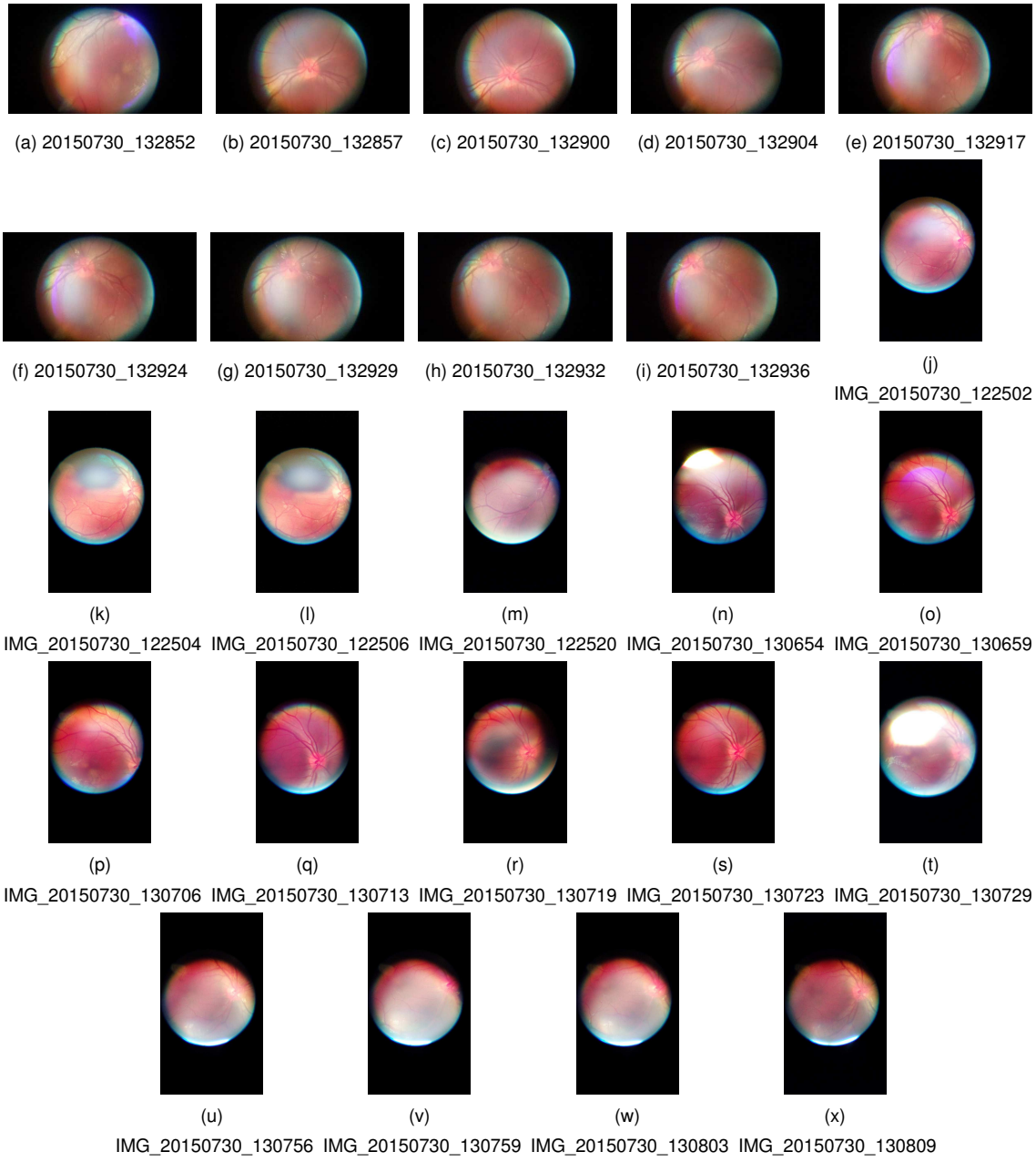
(a) IMG_20150730_130654 (b) IMG_20150730_130659 (c) IMG_20150730_130723 (d) IMG_20150730_130713 (e) Restored image

judgement the image in Figure 87s or the same in Figure 86c is the best image from set 20. Nonetheless it was selected as one of the best images from the set and the selected *base image* was tried to be restored as best as it could resulting in brush-like strokes from the merging images.

Table 26 presents the profiling produced for set 20. Time was reduced 11.24 times from 71.86 to 6.40 seconds saving in total 65.47 seconds. Saved time is 91.10% and cache time 8.90% of the uncached time.

Some images were selected from this big set but the rate of used images resulted in 0.17 and the rate of unused images in 0.83.

Figure 87. Images in set 20



8.21 Result for set 21

Images in set 21 are shown in Figure 89. These images were not adequately taken but either way Code 5 was applied to produce the results shown in Figure 88. The selected *base image* in Figure 88a is the best image in set 21 and no more images could be

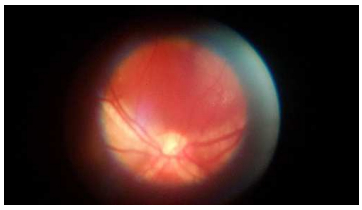
Table 27

Profiling for set 21 using command 'imrestore ../results/set21/.* --lens --overwrite --cachePath {temp}'*

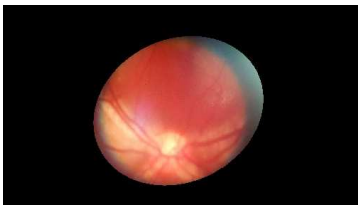
```
'ImRestore init' -> 27.0486 secs
└─ 'Computing features' -> 26.7146 secs
   ├─ (processed) '20150730_131816' -> 3.0230 secs
   ├─ (processed) '20150730_131823' -> 2.6371 secs
   ├─ (processed) '20150730_131827' -> 2.5712 secs
   ├─ (processed) '20150730_131834' -> 2.5258 secs
   ├─ (processed) '20150730_131836' -> 2.5097 secs
   ├─ (processed) '20150730_131839' -> 2.6379 secs
   ├─ (processed) '20150730_131844' -> 2.5678 secs
   ├─ (processed) '20150730_131913' -> 2.8689 secs
   ├─ (processed) '20150730_131915' -> 2.7617 secs
   └─ (processed) '20150730_131916' -> 2.5731 secs
└─ 'Restoring' -> 0.0428 secs
   ├─ 'Matching' -> 0.0156 secs
   └─ 'Merging' -> 0.0270 secs
└─ 'Post-processing' -> 0.2911 secs
```

```
'ImRestore init' -> 1.5615 secs
└─ 'Computing features' -> 1.2904 secs
   ├─ (memoized) '20150730_131816' -> 0.0428 secs
   ├─ (memoized) '20150730_131823' -> 0.1512 secs
   ├─ (memoized) '20150730_131827' -> 0.1033 secs
   ├─ (memoized) '20150730_131834' -> 0.1495 secs
   ├─ (memoized) '20150730_131836' -> 0.0887 secs
   ├─ (memoized) '20150730_131839' -> 0.1458 secs
   ├─ (memoized) '20150730_131844' -> 0.1554 secs
   ├─ (memoized) '20150730_131913' -> 0.1406 secs
   ├─ (memoized) '20150730_131915' -> 0.1576 secs
   └─ (memoized) '20150730_131916' -> 0.1332 secs
└─ 'Restoring' -> 0.0311 secs
   ├─ 'Matching' -> 0.0105 secs
   └─ 'Merging' -> 0.0204 secs
└─ 'Post-processing' -> 0.2400 secs
```

Figure 88. Result for set 21 using command 'imrestore ../results/set21/*.* --lens --overwrite --cachePath {temp}'



(a) 20150730_131816



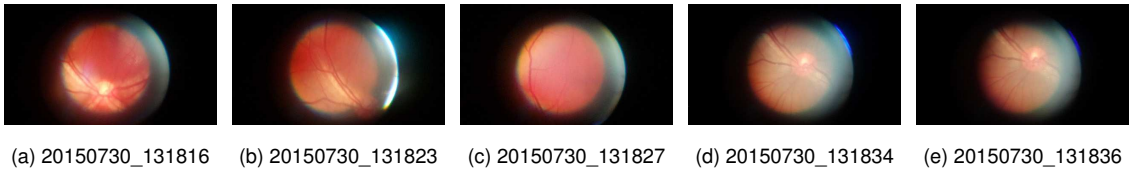
(b) Restored image

selected for merging because of the lack of more images with the same or better features than it resulting in the restored image shown in Figure 88b. The restored image was only obtained by applying the lens simulation which did not segment only the retinal area but the blurry bright part at the north of it as well. This behaviour is due to the segmenting algorithm which does not identifies the retinal area exactly but any body in the image with dense appearance, thus being confused by the dense looking fog also include in the lens simulation.

Table 27 presents the profiling produced for set 21. 27.05 seconds passed in the uncached session and 1.56 seconds in the cached session saving in total 25.49 seconds and reducing 17.32 times the processing when using cache. Saved time is the 94.23% and cache time 5.77% of the uncached time.

Imrestore processed 1 of the 10 images in set 21, thus with a rate of used images of 0.10. 0 images were merged to the *base image* thus the rate of merged images is 0.00. 9 images were not used in the restoration then the rate of unused images is 0.90.

Figure 89. Images in set 21



(a) 20150730_131816 (b) 20150730_131823 (c) 20150730_131827 (d) 20150730_131834 (e) 20150730_131836

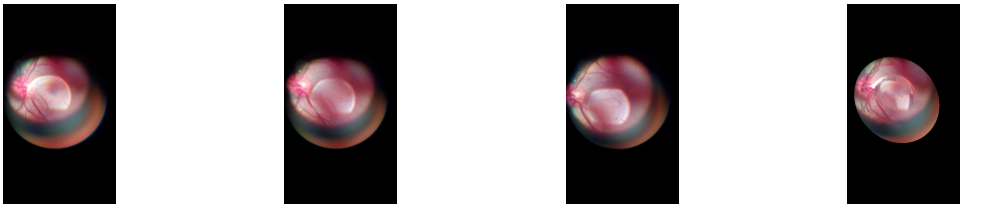
(f) 20150730_131839 (g) 20150730_131844 (h) 20150730_131913 (i) 20150730_131915 (j) 20150730_131916

8.22 Result for set 22

Table 28

Profiling for set 22 using command '`imrestore ../results/set22/*.* --lens --overwrite --cachePath {temp}`'

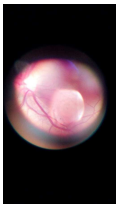
Figure 90. Result for set 22 using command '`imrestore ../results/set22/*.* --lens --overwrite --cachePath {temp}`'



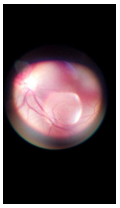
(a) IMG_20150730_121756 (b) IMG_20150730_121801 (c) IMG_20150730_121750 (d) Restored image

Once Code 5 is used in set 22 *imrestore* processed 3 of its 6 images producing the results of Figure 90. Figure 90a shows the *base image* where 2 images where merged to it, leaving 3mages not used for set 22 which is shown in Figure 91. This indicates that this set has a rate of 0.50 used images and 0.50 of unused images.

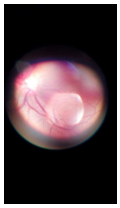
Figure 91. Images not used in restoration of set 22



(a) IMG_20150730_121804_2



(b) IMG_20150730_121804



(c) IMG_20150730_121804_1

Figure 90d shows the restored image where the remaining merged images before it were used to try and cover the whitish blob in the retinal area of the *base image*. This was not enough as more images or at least images that not contained the same problem were needed to be present in the set.

Table 28 presents the profiling produced for set 22 showing the cached and uncached cases. It can be observed in it that time was reduced from 22.07 to 3.60 seconds reducing by 6.13 times the uncached session and saving 18.47 seconds going from uncached to the cached session. Spared time is 83.70% and cache time 16.30% of the hole uncached time.

8.23 Result for set 23

Table 29

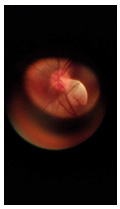
Profiling for set 23 using command '`imrestore ../results/set23/*.* --lens --overwrite --cachePath {temp}`'

```
'ImRestore init' -> 25.8744 secs
└─ 'Computing features' -> 25.3887 secs
   ├─ (processed) 'IMG_20150730_122140' -> 3.1005 secs
   ├─ (processed) 'IMG_20150730_122154' -> 3.0308 secs
   ├─ (processed) 'IMG_20150730_122200' -> 2.8967 secs
   ├─ (processed) 'IMG_20150730_122216' -> 2.8905 secs
   ├─ (processed) 'IMG_20150730_122225' -> 2.7309 secs
   ├─ (processed) 'IMG_20150730_122227' -> 2.5832 secs
   ├─ (processed) 'IMG_20150730_122229' -> 2.7980 secs
   ├─ (processed) 'IMG_20150730_122235' -> 2.5544 secs
   └─ (processed) 'IMG_20150730_122237' -> 2.7269 secs
└─ 'Restoring' -> 0.1394 secs
   ├─ 'Matching' -> 0.0666 secs
   └─ 'Merging' -> 0.0724 secs
└─ 'Post-processing' -> 0.3463 secs
```

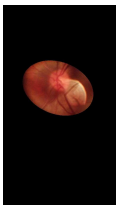
```
'ImRestore init' -> 1.7063 secs
└─ 'Computing features' -> 1.3495 secs
   ├─ (memoized) 'IMG_20150730_122140' -> 0.1262 secs
   ├─ (memoized) 'IMG_20150730_122154' -> 0.0581 secs
   ├─ (memoized) 'IMG_20150730_122200' -> 0.1698 secs
   ├─ (memoized) 'IMG_20150730_122216' -> 0.1659 secs
   ├─ (memoized) 'IMG_20150730_122225' -> 0.0478 secs
   ├─ (memoized) 'IMG_20150730_122227' -> 0.0723 secs
   ├─ (memoized) 'IMG_20150730_122229' -> 0.3161 secs
   ├─ (memoized) 'IMG_20150730_122235' -> 0.1639 secs
   └─ (memoized) 'IMG_20150730_122237' -> 0.1821 secs
└─ 'Restoring' -> 0.0961 secs
   ├─ 'Matching' -> 0.0393 secs
   └─ 'Merging' -> 0.0565 secs
└─ 'Post-processing' -> 0.2607 secs
```

There are no images in this set (see Figure 93) that could be used for a good restoration but either way to test what *imrestore* could do Code 5 was used. This resulted in *imrestore* choosing what could be considered the best image in set 23 as the *base image*

Figure 92. Result for set 23 using command '`imrestore ../results/set23/*.* --lens --overwrite --cachePath {temp}`'

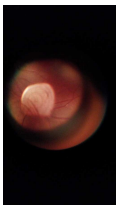


(a) IMG_20150730_122200

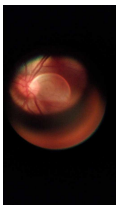


(b) Restored image

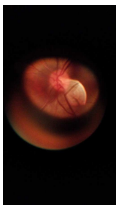
Figure 93. Images in set 23



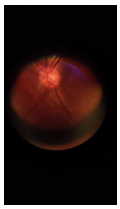
(a) IMG_20150730_122140



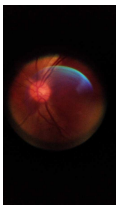
(b) IMG_20150730_122154



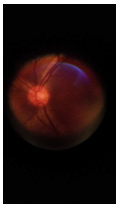
(c) IMG_20150730_122200



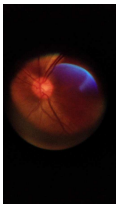
(d) IMG_20150730_122216



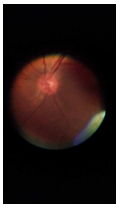
(e) IMG_20150730_122225



(f) IMG_20150730_122227



(g) IMG_20150730_122229



(h) IMG_20150730_122235

(i) IMG_20150730_122237

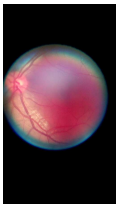
(see Figure 92a) and discarding the others as they could not be used to enhance the already selected one. The complete results can be seen in Figure 92. So *imrestore* used 1 of the 9 images in set 23, that means a rate of used images of 0.11 and had to discard 8 images, that is a rate of 0.89, to produce the restored image in Figure 92b.

The profiling produced for set 23 is presented in Table 29 where a reduction in time can be appreciated by 15.16 times from the uncached session sparing in total 24.17 seconds when time passed from 25.87 to 1.71 seconds. This means that the cached time is the 6.59% and the saved time the 93.41% of the uncached time.

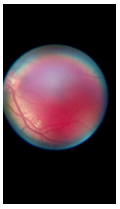
Table 30

Profiling for set 24 using command 'imrestore ../results/set24/.* --lens --overwrite --cachePath {temp}'*

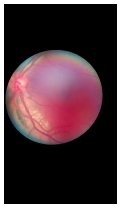
Figure 94. Result for set 24 using command 'imrestore ../results/set24/*.* --lens --overwrite --cachePath {temp}'



(a) IMG_20150730_125216



(b) IMG_20150730_125212



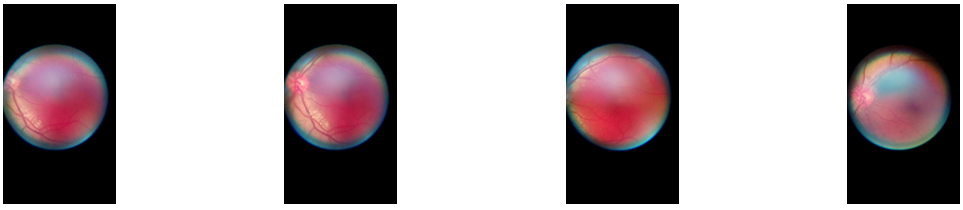
(c) Restored image

8.24 Result for set 24

This set is presented in Figure 95. This is another set that was not well acquired but lets present it here to see what *imrestore* can do. As soon as Code 5 was used for set 24 *imrestore* processed 2 of the 4 images in it, selecting Figure 94a as the *base image* and merging Figure 94b to produce the restored result in Figure 94c. The rate of used images is 0.50 because there were 2 used images of 4 images in the set and in consequence the rate of unused images is 0.50 because 2 images were not used for the restoration.

Table 30 presents the profiling produced for set 24. It is evidenced that the time was reduced by 6.61 times of the uncached session. In this way time passed from 13.89 to 2.10 seconds saving in total 11.79 seconds. This is translated in a gain of the 84.87% with cache time being 15.13%.

Figure 95. Images in set 24



(a) IMG_20150730_125212 (b) IMG_20150730_125216 (c) IMG_20150730_125222 (d) IMG_20150730_125228

8.25 Result for set 25

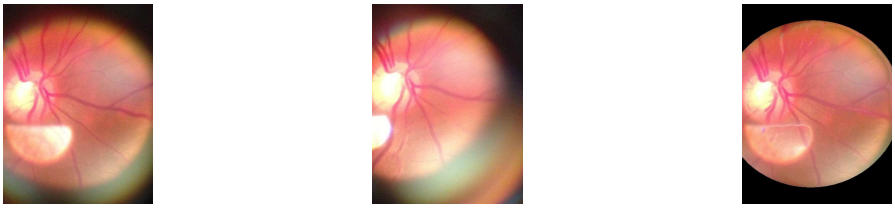
Table 31

Profiling for set 25 using command 'imrestore ../results/set25/.* --lens --overwrite --cachePath {temp}'*

```
'ImRestore init' -> 18.9791 secs
└─ 'Computing features' -> 17.8059 secs
   ├─ (processed) 'im1_1' -> 3.3500 secs
   ├─ (processed) 'im1_2' -> 3.1411 secs
   ├─ (processed) 'IMG_0243' -> 2.9382 secs
   ├─ (processed) 'IMG_0244' -> 2.9229 secs
   ├─ (processed) 'IMG_0247' -> 2.7132 secs
   └─ (processed) 'IMG_0248' -> 2.6944 secs
└─ 'Restoring' -> 0.9091 secs
   ├─ 'Matching' -> 0.0610 secs
   └─ 'Merging' -> 0.8478 secs
      └─ 'im1_2' -> 0.8162 secs
└─ 'Post-processing' -> 0.2640 secs
```

```
'ImRestore init' -> 1.5885 secs
└─ 'Computing features' -> 0.5419 secs
   ├─ (memoized) 'im1_1' -> 0.1572 secs
   ├─ (memoized) 'im1_2' -> 0.0628 secs
   ├─ (memoized) 'IMG_0243' -> 0.0358 secs
   ├─ (memoized) 'IMG_0244' -> 0.0657 secs
   ├─ (memoized) 'IMG_0247' -> 0.0613 secs
   └─ (memoized) 'IMG_0248' -> 0.1290 secs
└─ 'Restoring' -> 0.8305 secs
   ├─ 'Matching' -> 0.0498 secs
   └─ 'Merging' -> 0.7802 secs
      └─ 'im1_2' -> 0.7534 secs
└─ 'Post-processing' -> 0.2161 secs
```

Figure 96. Result for set 25 using command 'imrestore ../results/set25/*.* --lens --overwrite --cachePath {temp}'

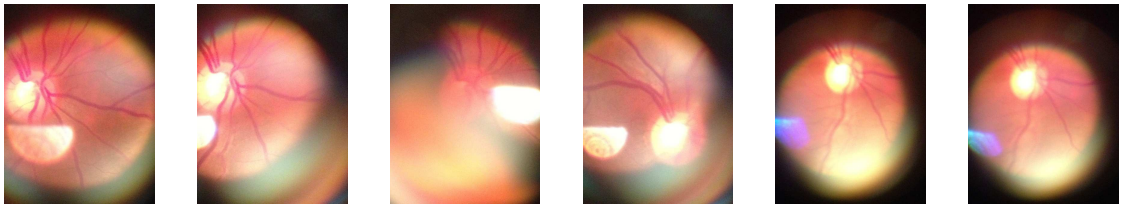


(a) im1_1 (b) im1_2 (c) Restored image

Finally we are reaching the end of the results and this promises to show for what *imrestore* was made.

Using Code 5 in set 25 *imrestore* processed 2 of the 6 images in set 25, this is shown in Figure 97. 1 is the number of images merged in the *base image*. 4 is the number of images not used for the set 25. Figure 96 presents the results for set 25. Figure 96a

Figure 97. Images in set 25



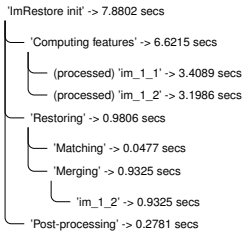
(a) im1_1 (b) im1_2 (c) IMG_0243 (d) IMG_0244 (e) IMG_0247 (f) IMG_0248

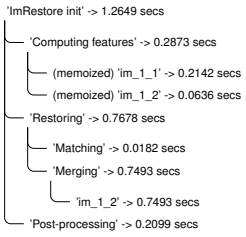
shows the *base image*, Figure 96c shows the restored image and the remaining merged images. Figure 97a shows the first image in set. Table 31 presents the profiling produced for set 25. It is evidenced that the time was reduced by 11.95 times of the uncached session. In this way time passed from 18.98 to 1.59 seconds saving in total 17.39 seconds. This is translated in a gain of the 91.63% with cache time being 8.37% and this means that an excess in time of the 1194.80% 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.

8.26 Result for set 26

Table 32

Profiling for set 26 using command 'imrestore ../results/set26/.* --lens --overwrite --cachePath {temp}'*



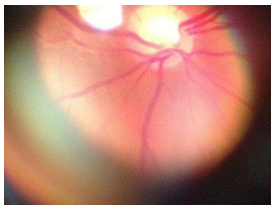


Using Code 5 in set 26 *imrestore* processed 2 of the 2 images in set 26 which means that 1 of those image got merged in the *base image*. So in this case 0 images were not used for the set 26. This is presented in Figure 98 for set 26. Figure 98a shows the *base image*, Figure 98c shows the restored image and the merged image in Figure 98b.

Table 32 presents the profiling produced for set 26. It is evidenced that the time was reduced by 6.23 times of the uncached session. In this way time passed from 7.88 to

Figure 98. Result for set 26 using command ‘`imrestore ../results/set26/*.* --lens --overwrite --cachePath {temp}`’

(a) im_1_1



(b) im_1_2

(c) Restored image

1.26 seconds saving in total 6.62 seconds. This is translated in a gain of the 83.95% with cache time being 16.05% and this means that an excess in time of the 622.99% would happen if passed from cached to a not cached session. The rate of used images is 1.00 and consequently the rate of merged images is 0.50 with the rate of failed images 0.00.

9 Validation

The expert system was applied to the 263 test images contained in the *results* folder and used for testing as explained in subsection 6.1 ‘Expert System’ on page 101. This evaluation was applied to all images excepting the ones resulting from the restoration process identified by file names preceded by an underscore ‘_’, letting the *expert* program effectively ignore them. This then was combined with the statistical data generated in the *imrestore* program to produce a set of ratios to qualify how well the sets were and how good the restoration process performed from these.

There is something worth pointing out and that is that it doesn’t matter if more images outside the ones used in the restoration were evaluated in the *expert* program, the statistical and expert data processing was safely and automatically produced for each set at the moment of the *imrestore* application producing veridical qualification data. These evaluations are briefly explained here and its actual implementation for generating them can be found in the repository.

9.1 Qualifications for sets

Table 33 resumes all the qualification ratios generated by the expert system for each set contained in the *results* folder in the repository (Toro, 2016). As it is appreciated in it the set did not obtained a good score in an overall basis and almost all of the fields are low. This means that images where no taken in an adequate form, most of them presented problems like flares and blurriness and even for the developed program these images where not ideal. Nevertheless, these images are what was provided and consequently the ones that were used in the testing of the *imrestore* program.

In a more analytic way we can extract some useful information from Table 33. The ‘Good images’ field is defined as the number of good images $N_{\text{good images}}$ over the number of images in the set N_{images}, that is the ratio $r_{\text{good images}}$ which is presented in Equation 19.

$$r_{\text{good images}} = \frac{N_{\text{good images}}}{N_{images}} \tag{19}$$

Where a good image is determined if it does not have any defects.

It is evident that in average there are 0.11 good images in overall for all sets and in the way it was calculated it also means that only 10.8% of the 263 provided images were adequate images that did not present defects or excess of blurriness. The 'Good images' column has really low rates with most of them reaching 0.0 meaning that for that particular set no images were adequate for restoration and this represents a real challenge for any restoration tool.

The field 'Good perspectives' in Table 33 determines how many of the total matches with the base image in the set are considered to conform good perspectives. The operation is performed by using a threshold Th_1 in the matches $Matches[n]$ of the merging images with respect to key-points of the *base image* (from n to N_{merged}). Notice that the matches of the *base image* which are the best possible matches are the same as $Matches$[base image], that is the matches of the *base image* with respect to itself which is denoted as $Matches_{base}$. Then the field 'Good perspectives' is the ratio $r_{\text{good perspectives}}$ in Equation 20.

$$r_{\text{good perspectives}} = \frac{\sum_{n=1}^{N_{merged}} \frac{Matches[n]}{Matches_{base} \times Th_1}}{N_{merged}} \tag{20}$$

Where Th_1 was chosen a a neutral value with half the number of matches in the base image which is $Th_1 = 0.5$. The brackets '[]' in $Matches[n]$ denotes that Matches is a sequence containing the matches of the corresponding images. The average of 'Good perspectives' compensates a little the last explained 'Good images' ratio as it represents how images are different to the others in the set so that they can complement each other to conform a complete restored image from them. The overall ratio is 0.22 which is better than before but not enough to conform scenes by attaching its images.

The field 'Set is restorable' shows a flag for each set indicating if it is restorable or not according to 'Good perspectives' and 'Good images' fields. Again, using a threshold Th_2 to average the aforementioned fields which can be represented with $flag_{\text{restorable}}$ in Equation 21.

$$flag_{\text{restorable}} = \frac{r_{\text{good perspectives}} + r_{\text{good images}}}{2} > Th_2 \tag{21}$$

Where a neutral value was chosen for Th_2 as $Th_2 = 0.5$. The field 'Set is restorable' tells if the set can be restored based in the information that it contains. From this field we

can tell that most sets where not qualified to be restored and only set 19 was completely restorable. The average of restorable sets in column 'Set is restorable' is really low with 0.04 from the 27 processed sets.

Finally, 'Set qualification' fields in Table 33 give the overall qualification for each set which is presented as the ratio $r_{\text{set qualification}}$ in Equation 22.

$$r_{\text{set qualification}} = \frac{r_{\text{good perspectives}} + r_{\text{good images}} + int(flag_{\text{restorable}})}{3} \tag{22}$$

Where the $int()$ function converts the flags from True to 1 and from False to 0 further segmenting the qualification for the set so that they are more distinct. As it is evidenced all sets present many problems and consequently most of them got low ratings. We can see though that set 19 got a rating of 0.74, the highest of them all but this is not rare as this set is also completely restorable and images did not have particular defects (see subsection 8.19 'Result for set 19' on page 140). Following is set 13 with 0.30 (see subsection 8.13 'Result for set 13' on page 130) which has less qualification than set 19 and it is below 0.5 marking the end of the good sets and labeling the others with lower ratings inadequate for restoration. At the end the average rating of all these sets is 0.12 meaning that most of then should not fare well for the restoration program.

9.2 Overall Profiling

This section address how efficient the restoration process is and how the memoization, caching, lazy evaluations and all the other optimization techniques used in the program helped on improving even more in that.

Table 34 shows all the profiling times of the results condensed in a single place for all the processed set of images presented in section 8 'Results' on page 110, this with the intention of providing a better representation of the efficiency when applying *imrestore* so that the reader can appreciate how much time is spend by using the automatic routines of the program. The header 'Images in set' represents the number of images in the set N_{images}, 'Used images' the number of images used from the set which is $N_{used} = N_{merged} + 1$, 'Uncached time' the amount of time in seconds taken by *imrestore* when restoring images in the set and there is not available cached data addressed as $T_{uncached}$ and conversely 'Cached time' the amount of time taken in restoring images when memoized data was

available known as T_{cached}. And the last column 'Reduction', the amount of time reduced from 'Uncached time' to 'Cached time' in percentage which is $P_{reduction}$ presented in Equation 23.

$$P_{reduction} = 100 \times \left(1 - \frac{T_{cached}}{T_{uncached}}\right) \tag{23}$$

Obtaining the mean of all the profiles in Table 34 tell us that on average from a set of almost 10 images (with set qualification of 0.12) *imrestore* uses 3 of them with the general processing time taking 28.61 seconds and reducing it in successive applications to just 5.50 seconds if caching is enabled and memoized data is available. The 'reduction' tab analysis demonstrates that the memoization really helps in reducing processing times, in this case 81.68%! and if you look at the different sets you will notice that all of them present different cases with different number of images and used images but all of their time were reduced around that percent with no more than a standard deviation of 10.21, this is around 23.12 seconds of saved time for any set with no more than 41 images in it and if *imrestore* does not use more than 13 images (see biggest tested numbers of fields 'Images in set' and 'Used images' respectively). In fact, besting the previous estimate, *imrestore* could reduce 85.08 seconds in set 16 going from an uncached time of 122.35 to a cached time of 37.27 seconds.

9.3 Performance and qualification for *imrestore* program

Table 35 shows the summary of all the qualification ratios generated from the expert system in combination with the statistics obtained when the *imrestore* program was run in all the sets of the *results* folder in the repository (Toro, 2016).

The column 'Used images' in Table 35 shows the ratio of used images in each set, that is r_{used} presented in Equation 24.

$$r_{used} = \frac{N_{used}}{N_{images}} \tag{24}$$

Where the number of images N_{images} and used images N_{used} in each set can be found in the fields 'Images in set' and 'Used images' respectively of Table 34. In general, 'Used images' from Table 35 has an average of 0.36, this is low but cannot be helped since the

program does not choose inadequate images that can further reduce the quality of the restored image as most of the images from the sets are worse than the selected *base image*.

From the selected images there is a ratio parameter used to determine how many good images were chosen from the total good images $N_{\text{good images}}$ of the set which is identified in the table as 'Good selected images' and as the ratio $r_{\text{good selected images}}$ in Equation 25.

$$r_{\text{good selected images}} = \frac{\sum_{n=1}^{N_{images}} (1 \text{ if } Image[n] \in r_{\text{good images}})}{N_{\text{good images}}} \tag{25}$$

Because there were few good images in each set most of the values where 0.0 or N/A, 0.0 for the sets with available good images bot non selected and N/A for sets that did not have good images to begin with, hence the status not applicable. In general N/A values can be compared for each set in Table 33 at field 'Good images' where ratios are 0.0, meaning once again that these sets did not have good images. The average resulted in 0.38 which is not as low as it would be expected as the N/A values were not taken in consideration because after all they do not apply.

The 'Base image' field centers its attention in the *base image* I_{base} selected by the restoration program relative to the other images in the set which is calculated by a simple qualification of the images using $r_{\text{image qualification}}$. Before the explanation of how to qualify an image with $r_{\text{image qualification}}$ lets define some equations used to grade the quality of them. The noise ratio in the image is defined as r_{noise} in Equation 26 taking the areas with noise over the hole area of the image.

$$r_{noise} = \frac{noise_{area}}{image_{area}} \tag{26}$$

The ratio of parts in the retina that have not defects can be defined as $r_{\text{non defects}}$ in Equation 27 using difference of 1 and the area of defects in the retina over the retinal area itself.

$$r_{\text{non defects}} = 1 - \frac{defects_{area}}{retina_{area}} \tag{27}$$

Notice that the areas $defects_{area}$, $retina_{area}$, $noise_{area}$ and $image_{area}$ are calculated in this case from the expert data provided for each image.

Now, to define the qualification of an image lets define the ratio $r_{\text{image qualification}}$ shown in Equation 28 using and unbalanced average between r_{noise} and $r_{\text{non defects}}$.

$$r_{\text{image qualification}} = \frac{r_{noise} \times 0.3 + 0.7 + r_{\text{non defects}}}{2} \tag{28}$$

Here r_{noise} has not much weight in the equation as it can be of the highest value 1 in most images because noise covers the whole image area even though it does not signify a big disturbance in the image. Then Equation 28 can be used to grade the *base image* as $r_{\text{base qualification}} = r_{\text{image qualification}}(I_{base})$. Here we can see from 'Base image' field that most *base images* where above 0.5 with the mean in 0.88 signifying that the program has a good selection algorithm.

The 'Merged images' column specifies the qualification for the selected merged images N_{merged}. The equation is expressed as the ratio $r_{\text{merged images}}$ in Equation 29.

$$r_{\text{merged images}} = \frac{\sum_{n=1}^{N_{merged}} r_{\text{image qualification}}[n]}{N_{merged}} \tag{29}$$

Notice however that $r_{\text{image qualification}}$ which is the qualification of an image from Equation 28 this time does not uses the parentheses '()' for function notation but instead brackets are used '[]' to denote that evaluations are obtained from the sequence of the merged images.

In 'Merged images' column There are N/A values, meaning that these values do no apply because there were no merged images. This corresponds to the 'Used images' fields in Table 34 where used images are of value 1 because there was only room for the selection of the *base image* and no more. The average then is 0.90 which means that most merged images did have useful information and where best suited to be merged compared to the other images that were left out.

Rates under the 'Lens simulation' only qualifies the lens simulation's algorithm regardless of the image quality based on the retinal area recorded in the expert data. So the calculated lens simulation $L_{calculated}$ is compared for changes to the lens simulation generated from expert data L_{expert}. The ratio is defined as $r_{\text{lens simulation}}$ in Equation 30.

$$r_{\text{lens simulation}} = \frac{\sum_{n=1}^{I_{size}} 1 \text{ if } (L_{expert}[n] = L_{calculated})}{I_{size}} \tag{30}$$

Where I_{size} is the total number of pixels in the specific image where the simulations was applied. Here we can see that *imrestore* excels in applying the lens simulation by delimiting exactly where the retinal area is an placing the lens similar to when photos are taken. There is a particular negative value in set 7, meaning that the lens where way inside of the retinal area an it should have covered more area according to the expert data. Other than that, the average was a impressive ratio of 0.92 ensuring that most lens simulations performed well and did a good job of not ruining the restoration.

Finally, there is a more solid overall qualification of the restoration process determined in the 'Restoration qualification' column. This value takes into consideration other qualification fields from Table 33, Table 34 and obviously from Table 35 itself. The equation is then

$$r_{\text{restoration qualification}} = \frac{r_{\text{lens simulation}} + r_{\text{base qualification}} + r_{\text{merged images}} + r_{\text{good selected images}}}{4} \tag{31}$$

This ratio takes into account if the restoration program performed well despite the bad provided sets and yields the values accordingly in combination with the other tests (lens simulations, blurriness, overlapping of defects between merges, attempt of restoring defects, etc). The final average ratio is 0.83 which can be seen as the overall qualification of everything. This means that the performance of the program in combination of how the provided sets were handled yields a qualification of 83 from 100.

Table 33

Qualifications of the provided image sets for the restoration

set	Good images (%*100)	Good perspectives (%*100)	Set is restorable	Set qualification
set 1	0.14	0.12	False	0.09
set 2	0.33	0.36	False	0.23
set 3	0.29	0.32	False	0.20
set 4	0.17	0.08	False	0.08
set 5	0.00	0.28	False	0.09
set 6	0.00	0.05	False	0.02
set 7	0.20	0.06	False	0.09
set 8	0.14	0.05	False	0.06
set 9	0.00	0.46	False	0.15
set 10	0.40	0.12	False	0.17
set 11	0.00	0.07	False	0.02
set 12	0.00	0.17	False	0.06
set 13	0.00	0.91	False	0.30
set 14	0.00	0.20	False	0.07
set 15	0.00	0.13	False	0.04
set 16	0.00	0.47	False	0.16
set 17	0.00	0.26	False	0.09
set 18	0.00	0.18	False	0.06
set 19	0.88	0.35	True	0.74
set 20	0.04	0.08	False	0.04
set 21	0.10	0.07	False	0.06
set 22	0.00	0.12	False	0.04
set 23	0.11	0.05	False	0.05
set 24	0.00	0.25	False	0.08
set 25	0.00	0.09	False	0.03
set 26	0.00	0.54	False	0.18
Mean	0.11	0.22	0.04	0.12

Generated from statistics and expert system

Table 34

Resume of profiling for imrestore program

set	Images in set	Used images	Uncached time (s)	Cached time (s)	Reduction (%)
set 1	7	1	13.45	1.67	87.61
set 2	6	2	17.51	1.91	89.07
set 3	7	3	22.37	8.07	63.90
set 4	12	1	35.03	1.15	96.72
set 5	7	2	22.41	4.76	78.78
set 6	7	1	21.19	0.73	96.57
set 7	5	2	17.13	3.98	76.79
set 8	7	1	21.12	1.33	93.71
set 9	3	2	10.59	3.97	62.54
set 10	10	1	19.60	2.11	89.26
set 11	6	1	13.06	1.68	87.12
set 12	4	1	9.13	1.02	88.84
set 13	5	5	17.90	4.94	72.40
set 14	4	3	11.56	3.28	71.66
set 15	24	6	63.74	8.85	86.12
set 16	41	13	122.35	37.27	69.54
set 17	17	9	47.76	10.62	77.77
set 18	17	2	45.88	7.25	84.20
set 19	8	4	24.55	9.01	63.31
set 20	24	4	71.86	11.74	83.66
set 21	10	1	27.05	1.31	95.15
set 22	6	3	22.07	6.85	68.94
set 23	9	1	25.87	2.19	91.55
set 24	4	2	13.89	2.10	84.87
set 25	6	2	18.98	3.89	79.52
set 26	2	2	7.88	1.26	83.95
Mean	9.92	2.88	28.61	5.50	81.68

Generated from profilings

Table 35

Resume of results using imrestore

set	Used images (%*100)	Good selected images (%*100)	Base image (%*100)	Merged images (%*100)	Lens simulation (%*100)	Restoration qualification (%*100)
set 1	0.14	0.00	0.88	N/A	0.97	0.62
set 2	0.33	0.50	0.81	0.95	0.93	0.80
set 3	0.43	1.00	0.90	1.00	0.96	0.97
set 4	0.08	0.00	0.86	N/A	0.92	0.59
set 5	0.29	N/A	0.96	0.98	0.92	0.96
set 6	0.14	N/A	0.96	N/A	0.69	0.82
set 7	0.40	1.00	0.97	1.00	0.88	0.96
set 8	0.14	0.00	0.93	N/A	0.85	0.59
set 9	0.67	N/A	0.90	0.83	0.95	0.89
set 10	0.10	0.25	0.86	N/A	0.97	0.69
set 11	0.17	N/A	0.85	N/A	0.92	0.89
set 12	0.25	N/A	0.93	N/A	0.94	0.94
set 13	1.00	N/A	0.86	0.91	0.96	0.91
set 14	0.75	N/A	0.92	0.90	0.88	0.90
set 15	0.25	N/A	0.88	0.89	0.93	0.90
set 16	0.32	N/A	0.81	0.83	0.95	0.86
set 17	0.53	N/A	0.89	0.88	0.90	0.89
set 18	0.12	N/A	0.82	0.84	0.92	0.86
set 19	0.50	0.43	0.91	0.90	0.94	0.79
set 20	0.17	0.00	0.77	0.78	0.93	0.62
set 21	0.10	1.00	0.92	N/A	0.97	0.96
set 22	0.50	N/A	0.79	0.81	0.92	0.84
set 23	0.11	0.00	0.81	N/A	0.97	0.59
set 24	0.50	N/A	0.78	0.79	0.92	0.83
set 25	0.33	N/A	0.95	0.98	0.94	0.96
set 26	1.00	N/A	0.95	0.98	0.94	0.96
Mean	0.36	0.38	0.88	0.90	0.92	0.83

Generated from statistics and expert system

10 Discussion

Illumination is not a big issue whenever using descriptor methods like SIFT, SURF or ORB (see subsection 8.19 ‘Result for set 19’ on page 140) 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 77g and Figure 82c). 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 the case, it just ranks them according to a evaluation criteria. Said cases like in subsection 8.6 ‘Result for set 6’ on page 120 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. 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 section 8 ‘Results’ on page 110). 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).

11 Conclusions

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. These sources and tests are hosted in *GitHub's* repository in (Toro, 2016) with *RRtoolbox's* manual specifically in (David, 2016). The pre-release is hosted in GitHub as well, following the instructions in subsection A.3 'Procedures to download *imrestore* program and source' on page 190 of the appendix. The *RRtoolbox* package is free to use which have a high potential as demonstrated with the implementation of *imrestore* program. The development of *RRtoolFC* GUI explained in the section 13 'Future Work' on page 167 intends to facilitate *RRtoolbox* usage by providing a prototyping platform.

The qualification generated automatically by the expert system and the statistical data demonstrated improvement in the images processed by the restoration program with a qualification of 0.83 or 83 from 100 points (details in Table 35 in subsection 9.3 'Performance and qualification for *imrestore* program' on page 156). These results can be appreciated in section 8 'Results' on page 110 where 263 noisy images were used devided in 27 sets (i.e. groups of images of the same scene) with an average of 10 images per set and an average qualification of 0.12, meaning that most of them were no suited even for restoration purposes (further explanations shown in Table 33 in subsection 9.1 'Qualifications for sets' on page 153) but still were adequately restored.

It was demonstrated that caching techniques improve considerably processing times by reducing it in 81.68%. This is usually from 28.61 to 5.50 seconds in a set of 10 images (see Table 34 'Resume of profiling for imrestore program' on page 161 and explanation in subsection 9.2 'Overall Profiling' on page 155).

12 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).

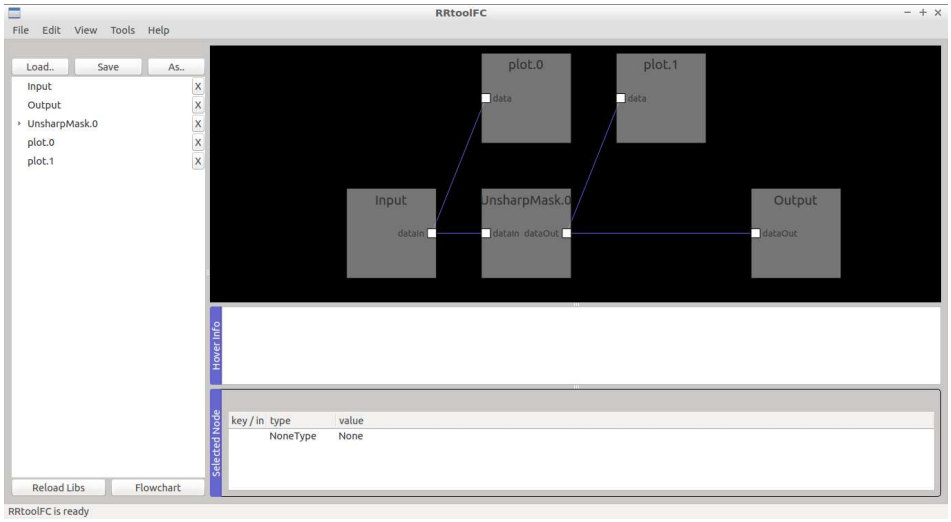
13 Future Work

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 (Toro, 2016). 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 99. 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.-e) 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 99) 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.

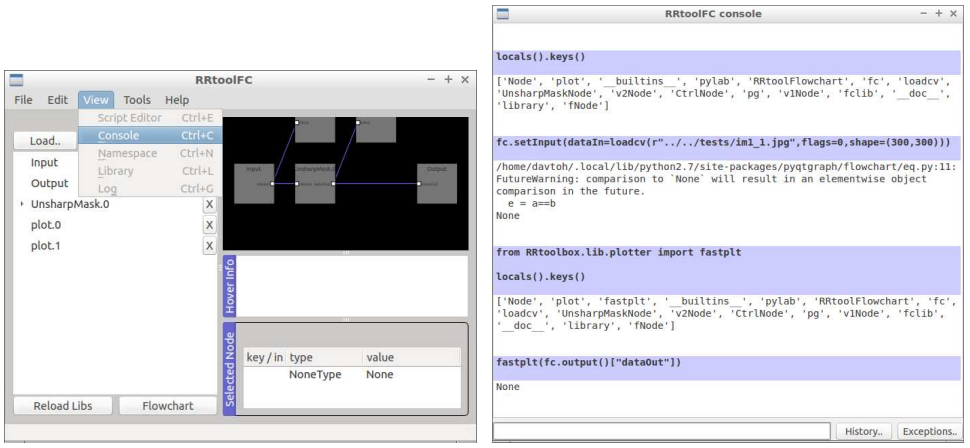
There is a console which opens the environment where the application runs. This console can be opened in 'View -> Console' (Figure 100a). In here we can make use of the *node* instances created in the graphical interface and even control internal variables not provided by it so that the advanced user can have more control over them or debug their graphical implementation should a difficult error appear to solve them. For example Figure 100b shows how to set and input and visualize an output

Figure 99. *RRtoolFC* Main developed Graphical Interface



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 101a) and "plot.1" (Figure 101b) 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 101c).

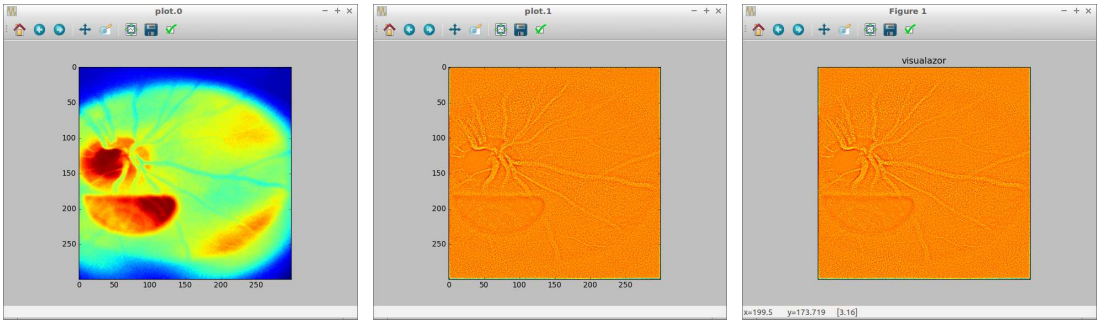
Figure 100. *RRtoolFC's* console



(a) How to open the console

(b) Demonstration of useful commands in local variables

Figure 101. 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"])`

14 References

About.com. (n.d.). Linux GUI Versus Linux Command Line: Which Is Better? Retrieved October 13, 2016, from http://linux.about.com/cs/softofficeutility/a/gui_cli.htm

Akram, M. U., Khalid, S., & Khan, S. A. (2013, January). "Identification and classification of microaneurysms for early detection of diabetic retinopathy." *Pattern Recognition*, *46*(1), 107–116. doi:10.1016/j.patcog.2012.07.002

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

Bob Fisher. (2004). Bilateral Filtering. Retrieved April 1, 2016, from http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/MANDUCHI1/Bilateral_Filtering.html

Bouhamidi, A., Enkhbat, R., & Jbilou, K. (2014, January). "Conditional gradient Tikhonov method for a convex optimization problem in image restoration." *Journal of Computational and Applied Mathematics*, *255*, 580–592. doi:10.1016/j.cam.2013.06.011

Chang, R.-C. & Tseng, F.-C. (2010, July). Automatic detection and correction for glossy reflections in digital photograph. (pp. 44–49). doi:10.1109/UMEDIA.2010.5543933

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

Collet, A., Berenson, D., Srinivasa, S. S., & Ferguson, D. (2009, May). Object recognition and full pose registration from a single image for robotic manipulation. (pp. 48–55). doi:10.1109/ROBOT.2009.5152739

Computerhope. (n.d.). Command line vs. GUI. Retrieved October 13, 2016, from http://www.computerhope.com/issues/ch000619.htm

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

Dai, S., Han, M., Wu, Y., & Gong, Y. (2007, July). Bilateral Back-Projection for Single Image Super Resolution. (pp. 1039–1042). doi:10.1109/ICME.2007.4284831

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

David Cortesi, Giovanni Bajo, William Caban, & Gordon McMillan. (n.d.). PyInstaller Manual — PyInstaller 3.2 documentation. Retrieved July 25, 2016, from http://pythonhosted.org/PyInstaller/

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

Deshpande, A. M. & Patnaik, S. (2014, January). "A novel modified cepstral based technique for blind estimation of motion blur." *Optik - International Journal for Light and Electron Optics*, *125*(2), 606–615. doi:10.1016/j.ijleo.2013.05.189

Dominus, M. J. (2005). Chapter 3 - Caching and Memoization. In *Higher-order perl* (pp. 63–113). San Francisco: Morgan Kaufmann. Retrieved from http://www.sciencedirect.com/science/article/pii/B9781558607019500039; http://www.sciencedirect.com.hemeroteca.lasalle.edu.co/science/article/pii/B9781558607019500039

Evening, M. (2007). *Adobe Photoshop CS3 for Photographers: A Professional Image Editor's Guide to the Creative Use of Photoshop for the Macintosh and PC*. Taylor & Francis. Retrieved from https://books.google.com.co/books?id=0XZO93ZcbqkC

Fathi, A. & Naghsh-Nilchi, A. R. (2013, January). "Automatic wavelet-based retinal blood vessels segmentation and vessel diameter estimation." *Biomedical Signal Processing and Control*, *8*(1), 71–80. doi:10.1016/j.bspc.2012.05.005

Faust, O., U, R. A., Ng, E. Y. K., Ng, K.-H. H., Suri, J. S., Acharya U., R., ... 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

Fraz, M. M., Barman, S. A., Remagnino, P., Hoppe, A., Basit, A., Uyyanonvara, B., ... Owen, C. G. (2012, November). "An approach to localize the retinal blood vessels using bit planes and centerline detection." *Computer Methods and Programs in Biomedicine*, *108*(2), 600–616. doi:10.1016/j.cmpb.2011.08.009

Fraz, M. M., Remagnino, P., Hoppe, A., Uyyanonvara, B., Rudnicka, A. R., Owen, C. G., & Barman, S. A. (2012, October). "Blood vessel segmentation methodologies in retinal images – A survey." *Computer Methods and Programs in Biomedicine*, *108*(1), 407–433. doi:10.1016/j.cmpb.2012.03.009

Gal, R., Kiryati, N., & Sochen, N. (2014, October). "Progress in the restoration of image sequences degraded by atmospheric turbulence." *Pattern Recognition Letters*. Celebrating the life and work of Maria Petrou, *48*, 8–14. doi:10.1016/j.patrec.2014.04.007

Garcia-Fidalgo, E. & Ortiz, A. (2015, February). "Vision-based topological mapping and localization methods: A survey." *Robotics and Autonomous Systems*, *64*, 1–20. doi:10.1016/j.robot.2014.11.009

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

Giancardo, L., Meriaudeau, F., Karnowski, T. P., Li, Y., Garg, S., Tobin Jr., K. W., & Chaum, E. (2012, January). "Exudate-based diabetic macular edema detection in fundus images using publicly available datasets." *Medical Image Analysis*, *16*(1), 216–226. doi:10.1016/j.media.2011.07.004

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

Haskell. (n.d.-a). Lazy evaluation - HaskellWiki. Retrieved October 14, 2016, from https://wiki.haskell.org/Lazy_evaluation

Haskell. (n.d.-b). Memoization - HaskellWiki. Retrieved October 14, 2016, from https://wiki.haskell.org/Memoization

Hearsay Social Engineering. (n.d.). Circular References in Python. Retrieved October 14, 2016, from http://engineering.hearsaysocial.com/2013/06/16/circular-references-in-python/

Hoover, A. & Goldbaum, M. (2003, August). “Locating the optic nerve in a retinal image using the fuzzy convergence of the blood vessels.” *IEEE Transactions on Medical Imaging*, *22*(8), 951–958. doi:10.1109/TMI.2003.815900

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

Imani, E., Pourreza, H.-R., & Banaee, T. (2015, July). “Fully automated diabetic retinopathy screening using morphological component analysis.” *Computerized Medical Imaging and Graphics*, *43*, 78–88. doi:10.1016/j.compmedimag.2015.03.004

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

James R. Bozeman, Matthew Pilling, 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; http://lyndonstate.edu/wp-content/uploads/2015/02/The-Convexity-Ratio-and-Applications.pdf

Jan, J., Odstrcilik, J., Gazarek, J., & Kolar, R. (2012, September). “Retinal image analysis aimed at blood vessel tree segmentation and early detection of neural-layer deterioration.” *Computerized Medical Imaging and Graphics*, *36*(6), 431–441. doi:10.1016/j.compmedimag.2012.04.006

Jiménez, S., Alemany, P., Núñez Benjumea, F., Serrano, C., Acha, B., Fondón, I., … Sánchez, C. (2011, September). “Automatic detection of microaneurysms in colour fundus images.” *Archivos de la Sociedad Española de Oftalmología (English Edition)*, *86*(9), 277–281. doi:10.1016/j.oftale.2011.04.009

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; http://opencvpython.blogspot.com.co/2013/03/histograms-2-histogram-equalization.html

Kallel, M., Aboulaich, R., Habbal, A., & Moakher, M. (2014, June). “A Nash-game approach to joint image restoration and segmentation.” *Applied Mathematical Modelling*, *38*(11–12), 3038–3053. doi:10.1016/j.apm.2013.11.034

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

Kumar, P. N. S., Deepak, R. U., Sathar, A., Sahasranamam, V., & Kumar, R. R. (2016). "Automated Detection System for Diabetic Retinopathy Using Two Field Fundus Photography." *Procedia Computer Science*. Proceedings of the 6th International Conference on Advances in Computing and Communications, *93*, 486–494. doi:10.1016/j.procs.2016.07.237

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

Li, Y., Liu, W., Li, X., Huang, Q., & Li, X. (2014, October). "GA-SIFT: A new scale invariant feature transform for multispectral image using geometric algebra." *Information Sciences*. Multimedia Modeling, *281*, 559–572. doi:10.1016/j.ins.2013.12.022

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

Lozano Forero, F. (2015, October). *Desarrollo de aplicación móvil para toma de fotografías retinales con dispositivos android* (Doctoral dissertation). Retrieved from http://scienti.colciencias.gov.co:8081/cvlac/visualizador/generarCurriculoCv.do?cod_rh=0000996700

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). Image Segmentation - MATLAB. Retrieved October 11, 2016, from http://www.mathworks.com/discovery/image-segmentation.html

MathWorks. (n.d.-c). Image Thresholding - MATLAB. Retrieved October 11, 2016, from http://www.mathworks.com/discovery/image-thresholding.html

MathWorks. (n.d.-d). Overview of Memory-Mapping - MATLAB & Simulink. Retrieved October 14, 2016, from https://www.mathworks.com/help/matlab/import_export/overview-of-memory-mapping.html

MathWorks. (n.d.-e). 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

Microsoft. (n.d.-a). Managing Memory-Mapped Files. Retrieved October 14, 2016, from https://msdn.microsoft.com/en-us/library/ms810613.aspx

Microsoft. (n.d.-b). Memory-Mapped Files. Retrieved October 14, 2016, from https://msdn.microsoft.com/en-us/library/dd997372(v=vs.110).aspx

Microsoft. (n.d.-c). Serialization (C# and Visual Basic). Retrieved from https://msdn.microsoft.com/en-us/library/ms233843.aspx

Moghimirad, E., Hamid Rezatofighi, S., & Soltanian-Zadeh, H. (2012, January). "Retinal vessel segmentation using a multi-scale medialness function." *Computers in Biology and Medicine*, *42*(1), 50–60. doi:10.1016/j.compbiomed.2011.10.008

Mookiah, M. R. K. [Muthu Rama Krishnan], Acharya, U. R., Chua, C. K., Lim, C. M., Ng, E. Y. K., & Laude, A. (2013, December). "Computer-aided diagnosis of diabetic retinopathy: A review." *Computers in Biology and Medicine*, *43*(12), 2136–2155. doi:10.1016/j.compbiomed.2013.10.007

Nguyen, U. T. V., Bhuiyan, A., Park, L. A. F., & Ramamohanarao, K. (2013, March). "An effective retinal blood vessel segmentation method using multi-scale line detection." *Pattern Recognition*, *46*(3), 703–715. doi:10.1016/j.patcog.2012.08.009

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

NumPy. (n.d.). NumPy Home Page. Retrieved October 14, 2016, from http://www.numpy.org/

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

OpenCV. (n.d.-c). Mat - The Basic Image Container — OpenCV 2.4.13.1 documentation. Retrieved October 11, 2016, from http://docs.opencv.org/2.4/doc/tutorials/core/mat_the_basic_image_container/mat_the_basic_image_container.html#storing-methods

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, Philippe Kruchten, Phillippe Kruchten, Kruchten, P., & 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. (n.d.-a). 11.1. pickle — Python object serialization — Python 2.7.12 documentation. Retrieved October 14, 2016, from https://docs.python.org/2/library/pickle.html

Python Software Foundation. (n.d.-b). 16.2. threading — Higher-level threading interface — Python 2.7.12 documentation. Retrieved October 14, 2016, from https://docs.python.org/2.7/library/threading.html

Python Software Foundation. (n.d.-c). 16.6. multiprocessing — Process-based " threading" interface — Python 2.7.12 documentation. Retrieved October 14, 2016, from https://docs.python.org/2.7/library/multiprocessing.html

Python Software Foundation. (n.d.-d). 16.7. mmap — Memory-mapped file support — Python 2.7.12 documentation. Retrieved October 14, 2016, from https://docs.python.org/2/library/mmap.html

Python Software Foundation. (n.d.-e). 25.3. unittest — Unit testing framework — Python 2.7.12 documentation. Retrieved October 23, 2016, from https://docs.python.org/2/library/unittest.html

Python Software Foundation. (n.d.-f). 5. Data Structures — Python 2.7.12 documentation. Retrieved October 14, 2016, from https://docs.python.org/2/tutorial/datastructures.html#dictionaries

Python Software Foundation. (n.d.-g). 8.11. weakref — Weak references — Python 2.7.12 documentation. Retrieved from https://docs.python.org/2/library/weakref.html

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

Python Software Foundation. (n.d.-i). Supporting Cyclic Garbage Collection — Python 2.7.12 documentation. Retrieved October 14, 2016, from https://docs.python.org/2/c-api/gcsupport.html

Python Software Foundation. (2015a). 11. Data Persistence — Python 2.7.12 documentation. Retrieved May 5, 2016, from https://docs.python.org/2/library/persistence.html

Python Software Foundation. (2015b). 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

Quellec, G., Lamard, M., Abràmoff, M. D., Decencière, E., Lay, B., Erginay, A., ... Cazuguel, G. (2012, August). "A multiple-instance learning framework for diabetic retinopathy screening." *Medical Image Analysis*, *16*(6), 1228–1240. doi:10.1016/j.media.2012.06.003

Ramlugun, G. S., Nagarajan, V. K., & Chakraborty, C. (2012, January). "Small retinal vessels extraction towards proliferative diabetic retinopathy screening." *Expert Systems with Applications*, *39*(1), 1141–1146. doi:10.1016/j.eswa.2011.07.115

Reis, M. S., de Oliveira, M. A. F., Körting, T. S., Pantaleão, E., Sant'Anna, S. J. S., Dutra, L. V., & Lu, D. (2015, July). Image segmentation algorithms comparison. (pp. 4340–4343). doi:10.1109/IGARSS.2015.7326787

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.-a). numpy.memmap — NumPy v1.11 Manual. Retrieved October 14, 2016, from http://docs.scipy.org/doc/numpy/reference/generated/numpy.memmap.html

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

Shang, L., Zhou, Y., & Su, P.-g. (2014, August). "Super-resolution restoration of MMW image based on sparse representation method." *Neurocomputing*. Advanced Intelligent Computing Theories and MethodologiesSelected papers from the 2012 Eighth International Conference on Intelligent Computing (ICIC 2012), *137*, 79–88. doi:10.1016/j.neucom.2013.02.056

Sinha, U. (n.d.). SIFT: Theory and Practice: Introduction - AI Shack - Tutorials for OpenCV, computer vision, deep learning, image processing, neural networks and artificial intelligence. Retrieved from http://aishack.in/tutorials/sift-scale-invariant-feature-transform-introduction/

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

Staal, J., Abramoff, M. D., Niemeijer, M., Viergever, M. A., & van Ginneken, B. (2004, April). "Ridge-based vessel segmentation in color images of the retina." *IEEE Transactions on Medical Imaging*, *23*(4), 501–509. doi:10.1109/TMI.2004.825627

Szeliski, R. (2010, November). *Computer Vision: Algorithms and Applications* (2011 edition). London ; New York: Springer. Retrieved from http://link.springer.com/book/10.1007%2F978-1-84882-935-0; http://szeliski.org/Book/

Thong, J. T., Sim, K. S., & Phang, J. C. (2001, October). “Single-image signal-to-noise ratio estimation.” *Scanning*, *23*(5), 328–36. doi:10.1002/sca.4950230506

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

Universidad De La Salle. (n.d.-a). AVARC - Automatizacion, Vision Artificial, Robotica y Control. Retrieved October 22, 2016, from http://scienti.colciencias.gov.co:8080/gruplac/jsp/visualiza/visualizagr.jsp?nro=00000000000921

Universidad De La Salle. (n.d.-b). Centro de Investigación en Salud y Visión (CISVI). Retrieved October 22, 2016, from http://www.lasalle.edu.co/wps/portal/Home/Principal/Investigaciones/centros_investigacion/centro-de-investigacion-en-salud-y-vision

Usman Akram, M., Khalid, S., Tariq, A., Khan, S. A., & Azam, F. (2014, February). “Detection and classification of retinal lesions for grading of diabetic retinopathy.” *Computers in Biology and Medicine*, *45*, 161–171. doi:10.1016/j.compbiomed.2013.11.014

Velandia Calderón, J. S. & Cifuentes Cifuentes, J. G. (2015, October). *Desarrollo de plataforma tecnológica de soporte en procesos de detección, atención y seguimiento de pacientes con retinopatía diabética* (Doctoral dissertation). Retrieved from http://repository.lasalle.edu.co/handle/10185/17440%0Ahttp://repository.lasalle.edu.co/bitstream/10185/17440/1/45101400_2015.pdf

Wang, G., Wang, Z., Chen, Y., & Zhao, W. (2015, May). “Robust point matching method for multimodal retinal image registration.” *Biomedical Signal Processing and Control*, *19*, 68–76. doi:10.1016/j.bspc.2015.03.004

Wang, Z., Kieu, H., Nguyen, H., & Le, M. (2015, February). “Digital image correlation in experimental mechanics and image registration in computer vision: Similarities, differences and complements.” *Optics and Lasers in Engineering*. Special Issue on Digital Image Correlation, *65*, 18–27. doi:10.1016/j.optlaseng.2014.04.002

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

Yang, M., Liu, Y., You, Z., Li, X., & Zhang, Y. (2014, April). “A homography transform based higher-order MRF model for stereo matching.” *Pattern Recognition Letters*, *40*, 66–71. doi:10.1016/j.patrec.2013.12.020

Yasukawa, S., Okuno, H., Ishii, K., & Yagi, T. (2016, September). “Real-time object tracking based on scale-invariant features employing bio-inspired hardware.” *Neural Networks*, *81*, 29–38. doi:10.1016/j.neunet.2016.05.002

Yu, G. [Guoshen] & 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

Yuan, Y., Huang, J., Peng, X., Xiong, C., Fang, J., & Yuan, F. (2014, January). “Accurate displacement measurement via a self-adaptive digital image correlation method based on a weighted ZNSSD criterion.” *Optics and Lasers in Engineering*, *52*, 75–85. doi:10.1016/j.optlaseng.2013.07.016

Zdešar, A., Škrjanc, I., & Klančar, G. (2014, October). “Homography estimation from circular motion for use in visual control.” *Robotics and Autonomous Systems*, *62*(10), 1486–1496. doi:10.1016/j.robot.2014.05.012

Zhang, C., Liu, J., Liang, C., Xue, Z., Pang, J., & Huang, Q. (2014, June). “Image classification by non-negative sparse coding, correlation constrained low-rank and sparse decomposition.” *Computer Vision and Image Understanding*, *123*, 14–22. doi:10.1016/j.cviu.2014.02.013

Zhang, Y., Zhou, L., Shang, Y., Zhang, X., & Yu, Q. (2016, April). “Contour model based homography estimation of texture-less planar objects in uncalibrated images.” *Pattern Recognition*, *52*, 375–383. doi:10.1016/j.patcog.2015.10.023

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

Zhu, C., Bichot, C.-E., & Chen, L. (2013, July). “Image region description using orthogonal combination of local binary patterns enhanced with color information.” *Pattern Recognition*, *46*(7), 1949–1963. doi:10.1016/j.patcog.2013.01.003

15 Further Reading

Chen, F., Chen, X., Xie, X., Feng, X., & Yang, L. (2013, September). “Full-field 3D measurement using multi-camera digital image correlation system.” *Optics and Lasers in Engineering*, *51*(9), 1044–1052. doi:10.1016/j.optlaseng.2013.03.001

Cunha-Vaz, J., Ribeiro, L., & Lobo, C. (2014, July). “Phenotypes and biomarkers of diabetic retinopathy.” *Progress in Retinal and Eye Research*, *41*, 90–111. doi:10.1016/j.preteyeres.2014.03.003

Dash, R. & Majhi, B. (2014, March). “Motion blur parameters estimation for image restoration.” *Optik - International Journal for Light and Electron Optics*, *125*(5), 1634–1640. doi:10.1016/j.ijleo.2013.09.026

Dupas, B., Walter, T., Erginay, A., Ordonez, R., Deb-Joardar, N., Gain, P., ... Massin, P. (2010, June). “Evaluation of automated fundus photograph analysis algorithms for detecting microaneurysms, haemorrhages and exudates, and of a computer-assisted diagnostic system for grading diabetic retinopathy.” *Diabetes & Metabolism*, *36*(3), 213–220. doi:10.1016/j.diabet.2010.01.002

Ege, B. M., Hejlesen, O. K., Larsen, O. V., Møller, K., Jennings, B., Kerr, D., & Cavan, D. A. (2000, July). “Screening for diabetic retinopathy using computer based image analysis and statistical classification.” *Computer Methods and Programs in Biomedicine*, *62*(3), 165–175. doi:10.1016/S0169-2607(00)00065-1

GitHub. (n.d.). Choose an open source license. Retrieved from http://choosealicense.com/

Hsiao, H.-K., Liu, C.-C., Yu, C.-Y., Kuo, S.-W., & Yu, S.-S. (2012, September). “A novel optic disc detection scheme on retinal images.” *Expert Systems with Applications*, *39*(12), 10600–10606. doi:10.1016/j.eswa.2012.02.157

Huang, Z., Kang, W., Wu, Q., & Chen, X. (2014, March). “A new descriptor resistant to affine transformation and monotonic intensity change.” *Computer Vision and Image Understanding*, *120*, 117–125. doi:10.1016/j.cviu.2013.10.010

Kauppi, T., Kamarainen, J. K., Lensu, L., Kälviäinen, H., & Uusitalo, H. (2011, June). “Detection and decision-support diagnosis of diabetic retinopathy using machine vision.” *Pattern Recognition and Image Analysis*, *21*(2), 140. doi:10.1134/S1054661811020465

Log0. (n.d.). Differences between L1 and L2 as Loss Function and Regularization. Retrieved from http://www.chioka.in/differences-between-l1-and-l2-as-loss-function-and-regularization/

Loncomilla, P., Ruiz-del-Solar, J., & Martínez, L. (2016, December). “Object recognition using local invariant features for robotic applications: A survey.” *Pattern Recognition*, *60*, 499–514. doi:10.1016/j.patcog.2016.05.021

Ma, X., Liu, D., Zhang, J., & Xin, J. (2015, March). “A fast affine-invariant features for image stitching under large viewpoint changes.” *Neurocomputing*, *151, Part 3*, 1430–1438. doi:10.1016/j.neucom.2014.10.045

McRitchie, I. N., Hart, P. M., & Winder, R. J. (2006, April). “Image registration and subtraction for the visualization of change in diabetic retinopathy screening.” *Computerized Medical Imaging and Graphics*, *30*(3), 139–145. doi:10.1016/j.compmedimag.2006.01.002

Melorose, J., Perroy, R., & Careas, S. (2015). “No Title No Title.” *Statewide Agricultural Land Use Baseline 2015*, *1*. doi:10.1017/CBO9781107415324.004. arXiv: arXiv:1011.1669v3

Mookiah, M. R. K. [M. R. K.], Acharya, U. R., Martis, R. J., Chua, C. K., Lim, C. M., Ng, E. Y. K., & Laude, A. (2013, February). “Evolutionary algorithm based classifier parameter tuning for automatic diabetic retinopathy grading: A hybrid feature extraction approach.” *Knowledge-Based Systems*, *39*, 9–22. doi:10.1016/j.knosys.2012.09.008

Niemeijer, M., Abràmoff, M. D., & van Ginneken, B. (2006, December). “Image structure clustering for image quality verification of color retina images in diabetic retinopathy screening.” *Medical Image Analysis*, *10*(6), 888–898. doi:10.1016/j.media.2006.09.006

OpenCV. (2014). *The OpenCV Reference Manual*. 2.4.9.0. Itseez.

Saleh, M. D. & Eswaran, C. (2012, October). “An automated decision-support system for non-proliferative diabetic retinopathy disease based on MAs and HAs detection.” *Computer Methods and Programs in Biomedicine*, *108*(1), 186–196. doi:10.1016/j.cmpb.2012.03.004

Shi, H., Ji, H., Yang, G., & He, X. (2013, January). “Shape and deformation measurement system by combining fringe projection and digital image correlation.” *Optics and Lasers in Engineering*, *51*(1), 47–53. doi:10.1016/j.optlaseng.2012.07.020

Takacs, G., Chandrasekhar, V., Tsai, S., Chen, D., Grzeszczuk, R., & Girod, B. (2013, April). “Rotation-invariant fast features for large-scale recognition and real-time

tracking." *Signal Processing: Image Communication*. Special Issue: VS&AR, *28*(4), 334–344. doi:10.1016/j.image.2012.11.004

Tavakoli, M., Shahri, R. P., Pourreza, H., Mehdizadeh, A., Banaee, T., & Bahreini Toosi, M. H. (2013, October). "A complementary method for automated detection of microaneurysms in fluorescein angiography fundus images to assess diabetic retinopathy." *Pattern Recognition*, *46*(10), 2740–2753. doi:10.1016/j.patcog.2013.03.011

Volodymyr Kindratenko. (2002). CVonline: Convexity ratio (Regions). Retrieved April 25, 2016, from http://homepages.inf.ed.ac.uk/cgi/rbf/CVONLINE/entries.pl?TAG156

WHO. (2015). WHO | Diabetes. Retrieved February 16, 2016, from http://www.who.int/mediacentre/factsheets/fs312/en/

Yu, G. [Gaohang], Xue, W., & Zhou, Y. (2014, October). "A nonmonotone adaptive projected gradient method for primal-dual total variation image restoration." *Signal Processing*. Image Restoration and Enhancement: Recent Advances and Applications, *103*, 242–249. doi:10.1016/j.sigpro.2014.02.025

Yu, H., Guo, R., Xia, H., Yan, F., Zhang, Y., & He, T. (2014, September). "Application of the mean intensity of the second derivative in evaluating the speckle patterns in digital image correlation." *Optics and Lasers in Engineering*, *60*, 32–37. doi:10.1016/j.optlaseng.2014.03.015

Zappa, E., Matinmanesh, A., & Mazzoleni, P. (2014, August). "Evaluation and improvement of digital image correlation uncertainty in dynamic conditions." *Optics and Lasers in Engineering*, *59*, 82–92. doi:10.1016/j.optlaseng.2014.03.007

Zappa, E., Mazzoleni, P., & Matinmanesh, A. (2014, May). "Uncertainty assessment of digital image correlation method in dynamic applications." *Optics and Lasers in Engineering*, *56*, 140–151. doi:10.1016/j.optlaseng.2013.12.016

Zhang, S., Wang, B., & Zhao, J. (2014, January). "High resolution optical image restoration for ground-based large telescope using phase diversity speckle." *Optik - International Journal for Light and Electron Optics*, *125*(2), 861–864. doi:10.1016/j.ijleo.2013.07.110

Zhong, S.-h., Liu, Y., & Chen, Q.-c. (2015, August). "Visual orientation inhomogeneity based scale-invariant feature transform." *Expert Systems with Applications*, *42*(13), 5658–5667. doi:10.1016/j.eswa.2015.01.012

Zhou, W., Li, H., & Tian, Q. (2014). Chapter 12 - Multimedia Content-Based Visual Retrieval. In S. T. Chellappa & Rama (Eds.), (Vol. 5, pp. 383–416). Academic Press Library in signal ProcessingImage and Video Compression and Multimedia. Elsevier. Retrieved from http://www.sciencedirect.com/science/article/pii/B9780124201491000120; http://www.sciencedirect.com.hemeroteca.lasalle.edu.co/science/article/pii/B9780124201491000120

Zhou, Y., Sun, C., & Chen, J. (2014, April). "Adaptive subset offset for systematic error reduction in incremental digital image correlation." *Optics and Lasers in Engineering*, *55*, 5–11. doi:10.1016/j.optlaseng.2013.10.014

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
$ pip install sympy
$ pip install pyqtgraph
```

The packages sympy and pyqtgraph are not needed to be installed if the *imrestore* program is going to be used. 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
```

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/2.4.11/opencv-2.4.11.exe/download and follow these steps:

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"
```

- Install the downloaded ‘opencv-2.4.11.exe’ file which will produce an ‘opencv’ folder.

- Go to the folder '/opencv/build/python/2.7/' and choose either 'x86' or 'x64' for the version of the installed python distribution.
- Once in the chosen folder (x86 or x64) copy the 'cv2.pyd' file.
- Paste the 'cv2.pyd' file in a registered path of python installation. It is usually 'C:/Python27/Lib/site-packages/'.
- To test that OpenCV is installed Open a cmd window.
- Type in the console 'python' to open a python console.
- Import OpenCV by typing 'import cv2'.
- If an ImportError is raised then OpenCV is not installed and must tried again changing the 'cv2.pyd' file location (e.g. in 'C:/Python27/').
- If cv2 can be imported but causes python to not respond then change the 'cv2.pyd' file version from x86 to x64 or vice versa to match the installed python distribution version.
- if the installation was correct extracted 'opencv' folder from 'opencv-2.4.11.exe' can be eliminated without worry due that it contains files for other languages like C++. Notice that examples for python 2 can be found in the '/opencv/sources/samples/python2' folder.

A.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
```

A.3 Procedures to download *imrestore* program and source

To download the execution files of *imrestore* just go to the *RRtools* repository found in (Toro, 2016). Click in the link as shown at Figure 102 in the red box then when the page is loaded as in Figure 103a scroll down to the download section as in Figure 103b. 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 104.

Figure 102. Finding release link in *RRtools* repository

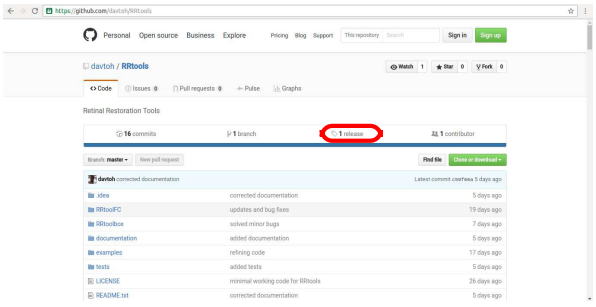
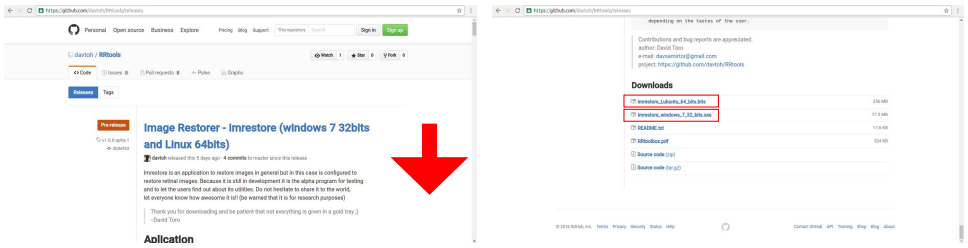
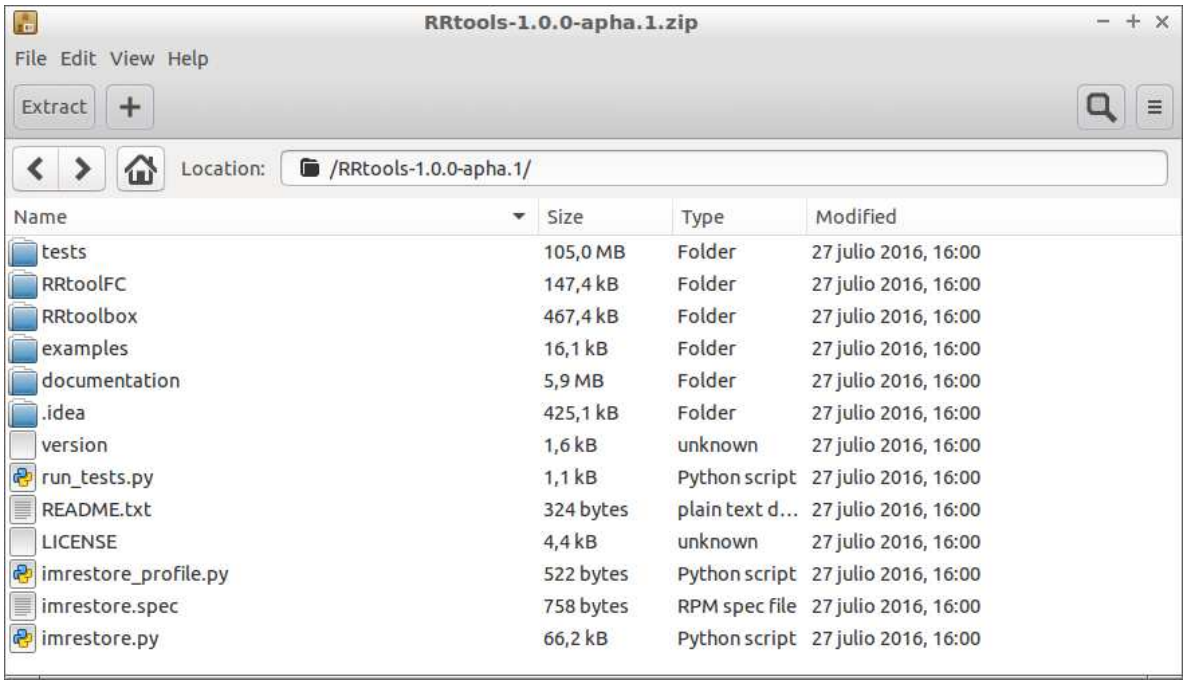


Figure 103. *Imrestore* pre-realease



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

Figure 104. *Imrestore* downloaded sources



16 Annexes

RRtoolbox Documentation

Release 1

David Toro

October 24, 2016

Contents

Contents:

CHAPTER 1

RRtoolbox package

1.1 Subpackages

1.1.1 RRtoolbox.lib package

Subpackages

RRtoolbox.lib.arrayops package

Submodules

RRtoolbox.lib.arrayops.basic module This module contains simple array operation methods

RRtoolbox.lib.arrayops.basic.**angle**(*v1*, *v2*, *deg=False*)
Angle between two N dimmensional vectors.

Parameters

- **v1** – vector 1.
- **v2** – vector 2.
- **deg** – if True angle is in Degrees, else radians.

Returns angle in radians.

Example:

```
>>> angle_between((1, 0, 0), (0, 1, 0))
1.5707963267948966
>>> angle_between((1, 0, 0), (1, 0, 0))
0.0
>>> angle_between((1, 0, 0), (-1, 0, 0))
3.141592653589793
```

Note: obtained from http://stackoverflow.com/a/13849249/5288758 and tested in http://onlinemschool.com/math/assistance/vector/angl/

RRtoolbox.lib.arrayops.basic.**angle2D**(*v1*, *v2*, *deg=False*, *absolute=None*)
Angle between two 2 dimensional vectors.

Parameters

- **v1** – vector 1.
- **v2** – vector 2.
- **deg** – if True angle is in Degrees, else radians.
- **absolute** – if None returns the angle (0 yo 180(pi)) between v1 and v2. if True returns the absolute angle (0 to 360(2pi)) from v1 as axis to v2. if False returns the angle (0 to 180 or 0 to -180) from v1 as axis to v2, where v2 angle relative to v1 is positive or negative if counter-clock or clock wise.

Returns angle in radians.

Note: implemented according to http://math.stackexchange.com/a/747992 and tested in http://onlinemschool.com/math/assistance/vector/angl/

RRtoolbox.lib.arrayops.basic.**angleXY**(*coorX*, *coorY*, *angle*)

Rotate coordinate.

Parameters

- **coorX** – x coordinate.
- **coorY** – y coordinate.
- **angle** – radian angle.

Returns rotated x,y

RRtoolbox.lib.arrayops.basic.**anorm**(*a*)

norm in array.

Parameters **a** –

Returns

RRtoolbox.lib.arrayops.basic.**anorm2**(*a*)

Summation of squares (helper function for *anorm()*)

Parameters **a** –

Returns

RRtoolbox.lib.arrayops.basic.**axesIntercept**(*coorSM*, *maxS*, *maxM*)

Intercept static axis (S) and mobile axis (M) with a coordinate connecting both axes from minS to minM.

S1 S2

S0 |<————————————|————> maxS

coorSM <————| |

M1 M2

M0 <————|————————————>| maxM

Parameters

- **coorSM** – coordinate of vector from S=0 to M=0.
- **maxS** – value representing end of estatic axis.
- **maxM** – value representing end of mobile axis.

Returns S1,S2,M1,M2.

RRtoolbox.lib.arrayops.basic.**boxPads**(*bx*, *points*)
Get box pads to fit all.

Parameters

- **bx** – box coordinates or previous boxPads [left_top, right_bottom]
- **points** – array of points

Returns [(left,top),(right,bottom)] where bx and points fit.

RRtoolbox.lib.arrayops.basic.**centerM**(*coor*, *maxM*)
Center vector coor in M axis.

Parameters

- **coor** – coordinate of vector from S=0 to M center
- **maxM** – value representing end of mobile axis

Returns M centered coordinate

RRtoolbox.lib.arrayops.basic.**centerS**(*coor*, *maxS*)
Center vector coor in S axis.

Parameters

- **coor** – coordinate of vector from S center to M=0
- **maxS** – value representing end of estatic axis

Returns S centered coordinate

RRtoolbox.lib.arrayops.basic.**centerSM**(*coorSM*, *maxS*, *maxM*)
Center vector coorSM in both S and M axes.

Parameters

- **coorSM** – coordinate of vector from S to M centers.
- **maxS** – value representing end of estatic axis.
- **maxM** – value representing end of mobile axis.

Returns SM centered coordinate.

RRtoolbox.lib.arrayops.basic.**contours2mask**(*contours*, *shape=None*, *astype=<type 'bool'>*)
Creates an array with filled polygons formed by contours.

Parameters

- **contours** – list of contour or points forming objects
- **shape** – (None) shape of array. If None it creates an array fitted to contours.
- **astype** – ("bool") numpy type

Returns

RRtoolbox.lib.arrayops.basic.**contoursArea**(*contours*)
Accumulates areas from list of contours.

Parameters **contours** – list of contours or binary array.

Returns area.

RRtoolbox.lib.arrayops.basic.**convertXY**(*x*, *y*, *backshape*, *foreshape*, *flag=0*, *quartile=0*, *angle=None*)

Convert absolute XY 0,0 coordinates to new system WZ.

Parameters

- **x** – x coordinate.
- **y** – y coordinate.
- **backshape** – shape of background image.

:param foreshape:shape of foreground image. :param flag: flag for position (default=0).

•flag==0 : foreground to left up.

•flag==1 : foreground to left down.

•flag==2 : foreground to right up.

•flag==3 : foreground to right down.

•flag==4 : foreground at center of background.

•flag==5 : XY 0,0 is at center of background.

•flag==6 : XY 0,0 is at center of foreground.

•flag==7 : XY 0,0 is at right down of foreground.

Parameters

- **quartile** – place Mobile image at quartile 1,2,3,4. if left quartile=0 image won't be moved.
- **angle** – angle in radians (defalut=None). if None it does not apply.

Returns W,Z

RRtoolbox.lib.arrayops.basic.**convexityRatio**(*cnt*, *hull=None*)

Ratio to test if contours are irregular

Parameters **cnt** – contour

:param hull:(None) convex hull :return: ratio

RRtoolbox.lib.arrayops.basic.**entroyTest**(*arr*)

Entropy test of intensity arrays. (Helper function for `entropy()`)

Parameters **arr** – array MxN of dim 2.

Returns entropy.

RRtoolbox.lib.arrayops.basic.**find_near**(*m*, *thresh=None*, *side=None*)

helper function for findminima and findmaxima :param m: minima or maxima points :param thresh: guess or seed point :param side: left or right :return: value

RRtoolbox.lib.arrayops.basic.**findmaxima**(*hist*, *thresh=None*, *side=None*)

Get nearest peak value to a thresh point from a histogram.

Parameters

- **hist** – histogram
- **thresh** – initial seed
- **side** – find valley from left or right of thresh

Returns

RRtoolbox.lib.arrayops.basic.**findminima**(*hist*, *thresh=None*, *side=None*)
Get nearest valley value to a thresh point from a histogram.

Parameters

- **hist** – histogram
- **thresh** – initial seed
- **side** – find valley from left or right of thresh

Returns

RRtoolbox.lib.arrayops.basic.**getOtsuThresh**(*hist*)
From histogram calculate Otsu threshold value.

Parameters **hist** – histogram

Returns otsu threshold value

RRtoolbox.lib.arrayops.basic.**getTransformedCorners**(*shape*, *H*)
from shape gets transformed corners of array.

Parameters

- **shape** – H,W array shape
- **H** – transformation matrix

Returns upper_left, upper_right, lower_right, lower_lef transformed corners.

RRtoolbox.lib.arrayops.basic.**getTransparency**(*array*)
Convert foreground to background.

Parameters **array** – image array.

Returns alfa (int or array)

RRtoolbox.lib.arrayops.basic.**get_x_space**(*funcs*, *step=10*, *xleft=-300*, *xright=300*)
get X axis space by brute force. This can be used to find the x points where the points in the y axis of any number of functions become stable.

Parameters

- **funcs** – list of functions
- **step** –

10. step to close guess to maximum

- **xleft** – maximum left limit
- **xright** – maximum right limit

Returns linspace

RRtoolbox.lib.arrayops.basic.**getdataVH**(*array*, *ypad=0*, *xpad=0*, *bgrcolor=None*, *alfa=None*)
Get data from array according to padding (Helper function for *padVH()*).

Parameters

- **array** – list of arrays to get data
- **ypad** – how much to pad in y axis
- **xpad** – how much to pad in x axis

Returns matrix_shapes, grid_div, row_grid, row_gridpad, globalgrid

RRtoolbox.lib.arrayops.basic.**histogram**(*img*)
Get image histogram.

Parameters **img** – gray or image with any bands

Returns histogram of every band

RRtoolbox.lib.arrayops.basic.**im2imFormat**(*src*, *dst*)
Tries to convert source image to destine image format.

Parameters

- **src** – source image.
- **dst** – destine image.

Returns reshaped source image.

RRtoolbox.lib.arrayops.basic.**im2shapeFormat**(*im*, *shape*)
Tries to convert image to intuited format from shape.

Parameters

- **im** – image.
- **shape** – shape to get format.

  shapes: * (None, None): converts to gray * (None, None, 2): converts to GR555 * (None, None, 3): converts to BGR * (None, None, 4): converts to BGRA

Returns reshaped image.

RRtoolbox.lib.arrayops.basic.**instability_bf**(*funcs*, *step=10*, *maximum=300*, *guess=0*, *tolerance=0.01*)
Find the instability of function approaching value by brute force,

Parameters

- **funcs** – list of functions
- **step** –

  10. step to close guess to maximum

- **maximum** –

  300. maximum value, if guess surpass this value then calculations are stopped.

- **guess** –

  0. initial guess

- **tolerance** – (0.01) tolerance with last step to check instability.

Returns (state, updated guess). state is True if successful, else False.

RRtoolbox.lib.arrayops.basic.**invertM**(*coorSM*, *maxM*)
Invert M axis.

Parameters

- **coorSM** – coordinate of vector for M inverted axes.
- **maxS** – value representing end of estatic axis.
- **maxM** – value representing end of mobile axis.

Returns SM coordinate on S axis and inverted M axis.

RRtoolbox.lib.arrayops.basic.**invertSM**(*coorSM*, *maxS*, *maxM*)
Invert S and M axes.

Parameters

- **coorSM** – coordinate of vector for SM inverted axes.
- **maxS** – value representing end of estatic axis.
- **maxM** – value representing end of mobile axis.

Returns SM coordinate on inverted SM axes.

RRtoolbox.lib.arrayops.basic.**isnumpy**(*arr*)
Test whether an object is a numpy array.

Parameters **arr** –

Returns True if numpy array, else false.

RRtoolbox.lib.arrayops.basic.**makeVis**(*globalgrid*, *bgrcolor=None*)
Make visualization (Helper function for *padVH()*)

Parameters

- **globalgrid** – shape
- **bgrcolor** – color of visualization

Returns array of shape globalgrid

RRtoolbox.lib.arrayops.basic.**matrixIntercept**(*x*, *y*, *staticm*, **mobilem*)
Intercepts planes x and y of a static matrix (staticm) with N mobile matrices (mobilem) translated from the origin to x,y coordinates.

Parameters

- **x** – x coordinate.
- **y** – y coordinate.
- **staticm** – static matrix.
- **mobilem** – mobile matrices.

Returns ROI of intercepted matrices [staticm,*mobilem].

RRtoolbox.lib.arrayops.basic.**multiple_superpose**(*base*, *fore*, *H*, *foremask=None*)
Superpose multiple foreground images to a single base image.

Parameters

- **base** – backgraound, base or dipest level image (level -1)
- **fore** – foreground image list (in order of level i = 0, ... , N)
- **H** – transformation matrix of fore in level i to overlay in base
- **foremask** – foreground alfa mask in level i

Returns generator of each overlay

RRtoolbox.lib.arrayops.basic.**noisy**(*arr*, *mode*)
Add noise to arrays

Parameters

- **arr** – Input ndarray data (it will be converted to float).

- **mode** – noise method:
  - 'gauss' - Gaussian-distributed additive noise.
  - 'poisson' - Poisson-distributed noise generated from the data.
  - 's&p' - Replaces random pixels with 0 or 1.
  - **'speckle' - Multiplicative noise using out = arr + n*arr,where** n is uniform noise with specified mean & variance.

:return noisy arr

RRtoolbox.lib.arrayops.basic.**normalize**(*arr*)
Normalize array to ensure range [0,1]

RRtoolbox.lib.arrayops.basic.**normalize2**(*arr*)
Normalize with factor of absolute maximum value.

RRtoolbox.lib.arrayops.basic.**normalizeCustom**(*arr*, *by=<function amax>*, *axis=None*)
Normalize array with custom operations.

Parameters

- **arr** – array (it does not correct negative values, use preferable NxM).
- **by** – np,max, np.sum or any function that gets an array to obtain factor.
- **axis** – if None it normalizes in all axes else in the selected axis.

Returns normalized to with factor.

RRtoolbox.lib.arrayops.basic.**overlay**(*back*, *fore*, *alpha=None*, *alfainverted=False*, *under=False*, *flag=0*)
Try to Overlay any dimension array.

Parameters

- **back** – BGRA background image.
- **fore** – BGRA foreground image.
- **alpha** – transparency channel.
- **alfainverted** – if True inverts alpha transparency.
- **under** – if True, place back as fore and fore as back.
- **flag** – (experimental)
  0. **Normally replace inverted transparency of alpha in back (N);** superpose alpha in back (V).
  1. **Bloat and replace inverted transparency of alpha in back;** superpose bgr in back (V).
  2. Superpose inverted transparent COLOR of alpha in back.
  3. Superpose inverted transparent COLOR of alpha in back.
  4. **Superpose transparent of alpha in back;** superpose transparent COLOR of alpha in back.
  5. **Superpose transparent of alpha in back;** superpose transparent COLOR of alpha in back.

Returns overlayed array

See also:

overlay2()

RRtoolbox.lib.arrayops.basic.**overlay2**(*back*, *fore*)

Overlay foreground to x,y coordinates in background image.

Parameters

- **back** – background image (numpy array dim 3).
- **fore** – foreground image (numpy array dim 4). the fourth dimension is used for transparency.

Returns back (with overlay).

#Example:

```
import cv2
import numpy as np
import time
a= time.time()
back = cv2.imread("t1.jpg")
temp = back.shape
bgr = np.zeros((temp[0],temp[1],4), np.uint8)
points = [(86, 162), (1219, 1112), (2219, 2112), (1277,3000),(86, 162)]
col_in = (0, 0, 0,255)
thickness = 10
for i in range(len(points)-1):
    pt1 = (points[i][0], points[i][1])
    pt2 = (points[i+1][0], points[i+1][1])
    cv2.line(bgr, pt1, pt2, col_in, thickness)

overlay(back,bgr)

win = "overlay"
cv2.namedWindow(win,cv2.WINDOW_NORMAL)
cv2.imshow(win, back)
print time.time()-a
cv2.waitKey()
cv2.destroyAllWindows()
```

See also:

overlay()

RRtoolbox.lib.arrayops.basic.**overlayXY**(*x*, *y*, *back*, *fore*, *alfa=None*, *alfainverted=False*, *under=False*, *flag=0*)

Overlay foreground image to x,y coordinates in background image. This function support images of different sizes with formats: BGR background and BGRA foreground of Opencv or numpy images.

Parameters

- **x** – x position in background.
- **y** – y position in background.
- **back** – background image (numpy array dim 3).
- **fore** – foreground image (numpy array dim 4). the fourth dimension is used for transparency.

Returns back (with overlay)

Example:

```
import cv2
back = cv2.imread("t1.jpg")
bgr = cv2.imread("mustache.png",-1)
x,y=convertXY(0,0,back.shape,bgr.shape,flag=1)
overlayXY(x,y,back,bgr)
win = "overlay"
cv2.namedWindow(win,cv2.WINDOW_NORMAL)
cv2.imshow(win, back)
cv2.waitKey()
cv2.destroyAllWindows()
```

RRtoolbox.lib.arrayops.basic.**overlaypng**(*back*, *fore*, *alpha=None*, *alfainverted=False*, *under=False*, *flag=0*)

Overlay only BGRA.

Parameters

- **back** – BGRA background image.
- **fore** – BGRA foreground image.
- **alpha** – transparency channel.
- **alfainverted** – if True inverts alpha transparency.
- **under** – if True, place back as fore and fore as back.
- **flag** – (experimental)
  0. **Normally replace inverted transparency of alpha in back (N);** superpose alpha in back (V).
  1. **Bloat and replace inverted transparency of alpha in back;** superpose bgr in back (V).
  2. Superpose inverted transparent COLOR of alpha in back.
  3. Superpose inverted transparent COLOR of alpha in back.
  4. **Superpose transparent of alpha in back;** superpose transparent COLOR of alpha in back.
  5. **Superpose transparent of alpha in back;** superpose transparent COLOR of alpha in back.

Returns overlayed array

See also:

overlay(), *overlay2()*

RRtoolbox.lib.arrayops.basic.**padVH**(*imgs*, *ypad=0*, *xpad=0*, *bgrcolor=None*, *alfa=None*)

Pad Vertically and Horizontally image or group of images into an array.

Parameters

- **imgs** – image to pad or list of horizontal images (i.e. piled up horizontally as [V1,..,VN] where each can be a list of vertical piling VN = [H1,..,HM]. It can be successive like horizontals, verticals, horizontals, etc.
- **ypad** – padding in axis y
- **xpad** – padding in axis x
- **bgrcolor** – color of spaces

- **alfa** – transparency of imgs over background of bgrcolor color.

Returns visualization of paded and piled images in imgs.

RRtoolbox.lib.arrayops.basic.**pad_to_fit_H**(*shape1*, *shape2*, *H*)
get boxPads to fit transformed shape1 in shape2.

Parameters

- **shape1** – shape of array 1
- **shape2** – shape of array 2
- **H** – transformation matrix to use in shape1

Returns [(left,top),(right,bottom)]

RRtoolbox.lib.arrayops.basic.**points2mask**(*pts*, *shape=None*, *astype=<type 'bool'>*)
Creates an array with the filled polygon formed by points.

Parameters

- **pts** – points.
- **shape** – (None) shape of array. If None it creates an array fitted to points.
- **astype** – ("bool") numpy type

Returns array.

Example:

```
pts = random_points([(-100, 100), (-100, 100)])
img = points2mask(pts)
Plotim("filled",img).show()
```

RRtoolbox.lib.arrayops.basic.**points_generator**(*shape=(10, 10)*, *nopoints=None*, *convex=False*, *erratic=False*, *complete=False*)
generate points.

Parameters

- **shape** – enclosed frame (width, height)
- **nopoints** – number of points
- **convex** – if True make points convex, else points follow a circular pattern.

Returns

RRtoolbox.lib.arrayops.basic.**polygonArea**(*pts*)
Area of points calculating polygon Area.

Parameters **pts** – points.

Returns area value.

..note::

- If polygon is incomplete (last is not first point) it completes the array.
- If the polygon crosses over itself the algorithm will fail.
- Based on http://www.mathopenref.com/coordpolygonarea.html

RRtoolbox.lib.arrayops.basic.**polygonArea_calcule**(*pts*)
Area of points calculating polygon Area.

Parameters **pts** – points.

Returns area value.

..note::

- If polygon is incomplete (last is not first point) it completes the array.
- If the polygon crosses over itself the algorithm will fail.
- Based on http://www.mathopenref.com/coordpolygonarea.html

RRtoolbox.lib.arrayops.basic.**polygonArea_contour**(*pts*)
Area of points using contours.

Parameters **pts** – points.

Returns area value.

..note:: if polygon is incomplete (last is not first point) it completes the array.

RRtoolbox.lib.arrayops.basic.**polygonArea_fill**(*pts*)
Area of points using filled polygon and pixel count.

Parameters **pts** – points.

Returns area value.

..note:: if polygon is incomplete (last is not first point) it completes the array.

RRtoolbox.lib.arrayops.basic.**process_as_blocks**(*arr*, *func*, *block_shape=(3, 3)*, *mask=None*, *asWindows=False*)
process with function over an array using blocks (using re-striding).

Parameters

- **arr** – array to process
- **func** – function to feed blocks
- **block_shape** – (3,3) shape of blocks
- **mask** – (None) mask to process arr
- **asWindows** – (False) if True all blocks overlap each other to give a result for each position of arr, if False the results are given in blocks equivalent for each processed blocks of arr (faster).

Returns processed array.

RRtoolbox.lib.arrayops.basic.**quadrant**(*coorX*, *coorY*, *maxX*, *maxY*, *quadrant=0*)
Moves a point to a quadrant

Parameters

- **coorX** – point in x coordinate
- **coorY** – point in y coordinate
- **maxX** – max value in x axis
- **maxY** – max value in y axis
- **quadrant** – Cartesian quadrant, if 0 or False it leaves coorX and coorY unprocessed.

Returns

RRtoolbox.lib.arrayops.basic.**random_points**(*axes_range=((-50, 50),)*, *nopoints=4*, *complete=False*)

Get random points.

Parameters

- **axes_range** – [x_points_range, y_points_range] where points_range is (min,max) range in axis.
- **nopoints** – number of points.
- **complete** – last point is the first point (adds an additional point i.e. nopoints+1).

Returns numpy array.

RRtoolbox.lib.arrayops.basic.**recursiveMap**(*function*, *sequence*)

Iterate recursively over a structure using a function.

Parameters

- **function** – function to apply
- **sequence** – iterator

Returns

RRtoolbox.lib.arrayops.basic.**relativeQuadrants**(*points*)

Get quadrants of relative vectors obtained from points.

Parameters **points** – array of points.

Returns quadrants.

RRtoolbox.lib.arrayops.basic.**relativeVectors**(*pts*, *all=True*)

Form vectors from points.

Parameters

- **pts** – array of points [p0, ... ,(x,y)].
- **all** – (True) if True adds last vector from last and first point.

Returns array of vectors [V0, ... , (V[n] = x[n+1]-x[n],y[n+1]-y[n])].

RRtoolbox.lib.arrayops.basic.**rescale**(*arr*, *max=1*, *min=0*)

Rescales array values to range [min,max].

Parameters

- **arr** – array.
- **max** – maximum value in range.
- **min** – minimum value in range.

Returns rescaled array.

RRtoolbox.lib.arrayops.basic.**separePointsByAxis**(*pts*, *ptaxis=(1, 0)*, *origin=(0, 0)*)

Separate scattered points with respect to axis (splitting line).

Parameters

- **pts** – points to separate.
- **ptaxis** – point to form axis from origin
- **origin** – origin

Returns left, right points from axis.

RRtoolbox.lib.arrayops.basic.**splitPoints**(*pts*, *aslist=None*)
from points get x,y columns

Parameters

- **pts** – array of points
- **aslist** – True to return lists instead of arrays

Returns x, y columns

RRtoolbox.lib.arrayops.basic.**standarizePoints**(*pts*, *aslist=False*)
converts points to a standard form :param pts: list or array of points :param aslist: True to return list instead of array :return: standard points

RRtoolbox.lib.arrayops.basic.**superpose**(*back*, *fore*, *H*, *foreMask=None*, *grow=True*)
Superpose foreground image to background image.

Parameters

- **back** – background image
- **fore** – foreground image
- **H** – transformation matrix of fore to overlay in back
- **foreMask** – (None) foreground alpha mask, None or function. foreMask values are from 1 for solid to 0 for transparency. If a function is provided the new back,fore parameters are provided to produce the foreMask. If None is provided as foreMask then it is equivalent to a foreMask with all values to 1 where fore is True.
- **grow** – If True, im can be bigger than back and is calculated according to how fore is superposed in back; if False im is of the same shape as back.

Returns im, H_back, H_fore

RRtoolbox.lib.arrayops.basic.**transformPoint**(*p*, *H*)
Transform individual x,y point with Transformation Matrix.

Parameters

- **p** – x,y point
- **H** – transformation matrix

Returns transformed x,y point

RRtoolbox.lib.arrayops.basic.**transformPoints**(*p*, *H*)
Transform x,y points in array with Transformation Matrix.

Parameters

- **p** – array of points
- **H** – transformation matrix

Returns transformed array of x,y point

RRtoolbox.lib.arrayops.basic.**unit_vector**(*vector*)
Returns the unit vector of the vector.

RRtoolbox.lib.arrayops.basic.**vectorsAngles**(*pts*, *ptaxis=(1, 0)*, *origin=(0, 0)*, *dtype=None*, *deg=False*, *absolute=None*)
Angle of formed vectors in Cartesian plane with respect to formed axis vector.

i.e. angle between vector "Vn" (formed by point "Pn" and the "origin")

and vector "Vaxis" formed by "ptaxis" and the "origin".

where pts-origin = (P0-origin ... Pn-origin) = V0 ... Vn

Parameters

- **pts** – points to form vectors from origin
- **ptaxis** – point to form axis from origin
- **origin** – origin
- **dtype** – return array of type supported by numpy.
- **deg** – if True angle is in Degrees, else radians.
- **absolute** – if None returns angles (0 yo 180(pi)) between pts-origin (V0 .. Vn) and Vaxis. if True returns any Vn absolute angle (0 to 360(2pi)) from Vaxis as axis to Vn. if False returns any Vn angle (0 to 180 or 0 to -180) from Vaxis as axis to Vn, where any Vn angle is positive or negative if counter-clock or clock wise from Vaxis.

Returns

RRtoolbox.lib.arrayops.basic.**vectorsQuadrants**(*vecs*)

Get quadrants of vectors.

Parameters **vecs** – array of vectors.

Returns quadrants.

RRtoolbox.lib.arrayops.basic.**vertexesAngles**(*pts*, *dtype=None*, *deg=False*)

Relative angle of vectors formed by vertexes (where vectors cross).

i.e. angle between vectors "v01" formed by points "p0-p1" and "v12" formed by points "p1-p2" where "p1" is seen as a vertex (where vectors cross).

Parameters

- **pts** – points seen as vertexes (vectors are recreated from point to point).
- **dtype** – return array of type supported by numpy.
- **deg** – if True angle is in Degrees, else radians.

Returns angles.

RRtoolbox.lib.arrayops.basic.**view_as_blocks**(*arr_in*, *block_shape=(3, 3)*)

Provide a 2D block_shape view to 2D array. No error checking made. Therefore meaningful (as implemented) only for blocks strictly compatible with the shape of arr_in.

Parameters

- **arr_in** –
- **block_shape** –

Returns

RRtoolbox.lib.arrayops.basic.**view_as_windows**(*arr_in*, *window_shape*, *step=1*)

Provide a 2D block_shape rolling view to 2D array. No error checking made. Therefore meaningful (as implemented) only for blocks strictly compatible with the shape of arr_in.

Parameters

- **arr_in** –

- **window_shape** –
- **step** –

Returns

RRtoolbox.lib.arrayops.convert module This module unlike common and basic array operations classifies just the from-to-conversions methods

class RRtoolbox.lib.arrayops.convert.**SimKeyPoint**(**args*)
Simulates opencv keypoint (it allows manipulation, conversion and serialization of keypoints).

Note: Used for conversions and data persistence.

RRtoolbox.lib.arrayops.convert.**apply2kp_pairs**(*kp_pairs*, *kp1_rel*, *kp2_rel*, *func=None*)
Apply to kp_pairs.

Parameters

- **kp_pairs** – list of (kp1,kp2) pairs
- **kp1_rel** – x,y relation or function to apply to kp1
- **kp2_rel** – x,y relation or function to apply to kp2
- **func** – function to build new copy of keypoint

Returns transformed kp_pairs

RRtoolbox.lib.arrayops.convert.**cnt2pts**(*contours*)
Convert contours to points. (cnt2pts)

Parameters **contours** – array of contours (cnt) ([[x,y]] only for openCV)

Returns

Example:

```
contours = np.array([[[0, 0]], [[1, 0]]]) # contours
points = contour2points(contours)
print points # np.array([[0, 0], [1, 0]])
```

RRtoolbox.lib.arrayops.convert.**contour2points**(*contours*)
Convert contours to points. (cnt2pts)

Parameters **contours** – array of contours (cnt) ([[x,y]] only for openCV)

Returns

Example:

```
contours = np.array([[[0, 0]], [[1, 0]]]) # contours
points = contour2points(contours)
print points # np.array([[0, 0], [1, 0]])
```

RRtoolbox.lib.arrayops.convert.**conv3H4H**(*M*)
Convert a 3D transformation matrix (TM) to 4D TM.

Parameters **M** – Matrix

Returns 4D Matrix

RRtoolbox.lib.arrayops.convert.**dict2keyPoint**(*d*, *func=<built-in function KeyPoint>*)
KeyPoint([x, y, _size[, _angle[, _response[, _octave[, _class_id]]]]]) -> <KeyPoint object>

RRtoolbox.lib.arrayops.convert.**getSOpointRelation**(*source_shape*, *destine_shape*, *asMatrix=False*)
Return parameters to change scaled point to original point.

destine_domain = relation*source_domain

Parameters

- **source_shape** – image shape for source domain
- **destine_shape** – image shape for destine domain
- **asMatrix** – if true returns a Transformation Matrix H

Returns x, y coordinate relations or H if asMatrix is True

Note: Used to get relations to convert scaled points to original points of an Image.

RRtoolbox.lib.arrayops.convert.**invertH**(*H*)
Invert Transformation Matrix.

Parameters **H** –

Returns

RRtoolbox.lib.arrayops.convert.**keyPoint2tuple**(*keypoint*)
obj.angle, obj.class_id, obj.octave, obj.pt, obj.response, obj.size

RRtoolbox.lib.arrayops.convert.**points2contour**(*points*)
Convert points to contours. (pts2cnt)

Parameters **points** – array of points ([x,y] for openCV, [y,x] for numpy)

Returns

Example:

```
points = np.array([[0, 0], [1, 0]]) # points
contours = points2contour(points)
print contours # np.array([[[0, 0]], [[1, 0]]])
```

RRtoolbox.lib.arrayops.convert.**points2vectos**(*pts*, *origin=None*)
Convert points to vectors with respect to origin.

Parameters

- **pts** – array of points.
- **origin** – point of origin.

Returns vectors.

RRtoolbox.lib.arrayops.convert.**pts2cnt**(*points*)
Convert points to contours. (pts2cnt)

Parameters **points** – array of points ([x,y] for openCV, [y,x] for numpy)

Returns

Example:

```
    points = np.array([[0, 0], [1, 0]]) # points
    contours = points2contour(points)
    print contours # np.array([[[0, 0]], [[1, 0]]])
```

RRtoolbox.lib.arrayops.convert.**sh2oh**(*sH*, *osrc_sh*, *sscr_sh*, *odst_sh*, *sdst_sh*)

Convert scaled transformation matrix (sH) to original (oH).

Parameters

- **sH** – scaled transformation matrix
- **osrc_sh** – original source's shape
- **sscr_sh** – scaled source's shape
- **odst_sh** – original destine's shape
- **sdst_sh** – scaled destine's shape

Returns

RRtoolbox.lib.arrayops.convert.**spairs2opairs**(*kp_pairs*, *osrc_sh*, *sscr_sh*, *odst_sh*, *sdst_sh*, *func=None*)

Convert scaled kp_pairs to original kp_pairs.

Parameters

- **kp_pairs** – list of kp_pairs
- **osrc_sh** – original source's shape
- **sscr_sh** – scaled source's shape
- **odst_sh** – original destine's shape
- **sdst_sh** – scaled destine's shape
- **func** – function to build new copy of keypoint

Returns

RRtoolbox.lib.arrayops.convert.**spoint2opointfunc**(*source_shape*, *destine_shape*)

Return function with parameters to change scaled point to original point.

Parameters

- **source_shape** –
- **destine_shape** – shape of

Returns

Example:

```
    forefunc = scaled2realfunc(imgf.shape,bgr.shape)
    backfunc = scaled2realfunc(imgb.shape,back.shape)
    p1fore = np.array([forefunc(i) for i in p1])
    p2back = np.array([backfunc(i) for i in p2])
```

RRtoolbox.lib.arrayops.convert.**toTupple**(*obj*)

Converts recursively to tuple

Parameters **obj** – numpy array, list structure, iterators, etc.

Returns tuple representation obj.

RRtoolbox.lib.arrayops.convert.**translateQuadrants**(*quadrants*, *quadrantmap={(0, 1): 'up', (-1, 1): 'left-up', (0, 0): 'origin', (-1, 0): 'left', (-1, -1): 'left-down', (0, -1): 'down', (1, 0): 'right', (1, -1): 'right-down', (1, 1): 'right-up'}*)

Convert quadrants into human readable data.

Parameters

- **quadrants** – array of quadrants.
- **quadrantmap** – dictionary map to translate quadrants. it is of the form:

```
{(0,0):"origin",(1,0):"right",(1,1):"top-right",(0,1):"top",(-1,1):"top-left",
```

(-1,0):"left",(-1,-1):"bottom-left",(0,-1):"bottom",(1,-1):"bottom-right"}

Returns list of translated quadrants.

RRtoolbox.lib.arrayops.convert.**tuple2keyPoint**(*points*, *func=<built-in function KeyPoint>*)

KeyPoint([x, y, _size[, _angle[, _response[, _octave[, _class_id]]]]]) -> <KeyPoint object>

RRtoolbox.lib.arrayops.convert.**vectos2points**(*vecs*, *origin=None*)

Convert points to vectors with respect to origin.

Parameters

- **vecs** – array of vectors.
- **origin** – point of origin.

Returns points.

RRtoolbox.lib.arrayops.filters module This module contains custom 1D adn 2D-array filters and pre-processing (as in filtering phase) methods

class RRtoolbox.lib.arrayops.filters.**Bandpass**(*alpha*, *beta1*, *beta2*)

Bases: *RRtoolbox.lib.arrayops.filters.FilterBase*

Bandpass filter (recommended to use float types)

class RRtoolbox.lib.arrayops.filters.**Bandstop**(*alpha*, *beta1*, *beta2*)

Bases: *RRtoolbox.lib.arrayops.filters.FilterBase*

Bandstop filter (recommended to use float types)

class RRtoolbox.lib.arrayops.filters.**BilateraParameter**(*scale*, *shift=33*, *name=None*, *alpha=100*, *beta1=-400*, *beta2=200*)

Bases: *RRtoolbox.lib.arrayops.filters.Bandstop*

bilateral parameter

class RRtoolbox.lib.arrayops.filters.**BilateralParameters**(*d=None*, *sigmaColor=None*, *sigmaSpace=None*)

Bases: object

create instance to calculate bilateral parameters from image shape.

d -> inf then:

- computation is slower
- filtering is better to eliminate noise
- images look more cartoon-like

Parameters

- **d** – distance
- **sigmaColor** – sigma in color
- **sigmaSpace** – sigma in space

d = <RRtoolbox.lib.arrayops.filters.BilateraParameter object>

filters
list of filters

sigmaColor = <RRtoolbox.lib.arrayops.filters.BilateraParameter object>

sigmaSpace = <RRtoolbox.lib.arrayops.filters.BilateraParameter object>

class RRtoolbox.lib.arrayops.filters.**FilterBase**(*alpha=None*, *beta1=None*, *beta2=None*)
Bases: object

base filter to create custom filters

alpha

beta1

beta2

class RRtoolbox.lib.arrayops.filters.**Highpass**(*alpha*, *beta1*)
Bases: *RRtoolbox.lib.arrayops.filters.FilterBase*

Highpass filter (recommended to use float types)

class RRtoolbox.lib.arrayops.filters.**InvertedBandpass**(*alpha*, *beta1*, *beta2*)
Bases: *RRtoolbox.lib.arrayops.filters.Bandpass*

inverted Bandpass filter (recommended to use float types)

class RRtoolbox.lib.arrayops.filters.**InvertedBandstop**(*alpha*, *beta1*, *beta2*)
Bases: *RRtoolbox.lib.arrayops.filters.Bandstop*

inverted Bandstop filter (recommended to use float types)

class RRtoolbox.lib.arrayops.filters.**Lowpass**(*alpha*, *beta1*)
Bases: *RRtoolbox.lib.arrayops.filters.FilterBase*

Lowpass filter (recommended to use float types)

RRtoolbox.lib.arrayops.filters.**bilateralFilter**(*im*, *d*, *sigmaColor*, *sigmaSpace*)
Apply bilateral Filter.

Parameters

- **im** –
- **d** –
- **sigmaColor** –
- **sigmaSpace** –

Returns filtered image

RRtoolbox.lib.arrayops.filters.**filterFactory**(*alpha*, *beta1*, *beta2=None*)
Make filter.

Parameters

- **alpha** – steepness of filter
- **beta1** – first shift from origin
- **beta2** – second shift from origin:

  alpha must be != 0 if beta2 = None:

  if alpha > 0: high-pass filter, if alpha < 0: low-pass filter

  else:

  if beta2 > beta1: if alpha > 0: band-pass filter, if alpha < 0: band-stop filter

  else: if alpha > 0: inverted-band-pass filter, if alpha < 0: inverted-band-stop filter

Returns filter funtion with intup levels

Example:

```
alpha,beta1,beta2 = 10,20,100
myfilter = filter(alpha,beta1,beta2)
print myfilter,type(myfilter)
print myfilter.alpha,myfilter.beta1,myfilter.beta2
```

RRtoolbox.lib.arrayops.filters.**getBilateralParameters**(*shape=None*, *mode=None*)
Calculate from shape bilateral parameters.

Parameters

- **shape** – image shape. if None it returns the instace to use with shapes.
- **mode** – "mild", "heavy" or "normal" to process noise

Returns instance or parameters

RRtoolbox.lib.arrayops.filters.**normsigmoid**(*x*, *alpha*, *beta*)
Apply normalized sigmoid filter.

Parameters

- **x** – data to apply filter
- **alpha** – if alpha > 0: pass high filter, if alpha < 0: pass low filter, alpha must be != 0
- **beta** – shift from origin

Returns filtered values normalized to range [-1 if x<0, 1 if x>=0]

RRtoolbox.lib.arrayops.filters.**sigmoid**(*x*, *alpha*, *beta*, *max=255*, *min=0*)
Apply sigmoid filter.

Parameters

- **x** – data to apply filter
- **alpha** – if alpha > 0: pass high filter, if alpha < 0: pass low filter, alpha must be != 0
- **beta** – shift from origin
- **max** – maximum output value
- **min** – minimum output value

Returns filtered values ranging as [min,max]

Note: Based from http://www.itk.org/Doxygen/html/classitk_1_1SigmoidImageFilter.html

RRtoolbox.lib.arrayops.filters.**smooth**(*x*, *window_len=11*, *window='hanning'*, *correct=False*)

Smooth the data using a window with requested size.

This method is based on the convolution of a scaled window with the signal. The signal is prepared by introducing reflected copies of the signal (with the window size) in both ends so that transient parts are minimized in the beginning and end part of the output signal.

input: x: the input signal window_len: the dimension of the smoothing window; should be an odd integer window: the type of window from 'flat', 'hanning', 'hamming', 'bartlett', 'blackman'

flat window will produce a moving average smoothing.

output: the smoothed signal

Example:

```
t=linspace(-2,2,0.1)
x=sin(t)+randn(len(t))*0.1
y=smooth(x)
```

See also:

numpy.hanning, numpy.hamming, numpy.bartlett, numpy.blackman, numpy.convolve, scipy.signal.lfilter

Note: length(output) != length(input), to correct this: return y[(window_len/2-1):-(window_len/2)] instead of just y.

RRtoolbox.lib.arrayops.mask module This module contains all basic masking and pre-processing (as in segmenting phase) methods

RRtoolbox.lib.arrayops.mask.**background**(*gray*, *mask=None*, *iterations=3*)

get the background mask of a gray image. (this it the inverted of *foreground()*)

Parameters

- **gray** – gray image
- **mask** – (None) input mask to process gray
- **iterations** – (3) number of iterations to detect background with otsu threshold.

Returns output mask

RRtoolbox.lib.arrayops.mask.**biggestCnt**(*contours*)

Filters contours to get biggest contour.

Parameters **contours** –

Returns cnt

RRtoolbox.lib.arrayops.mask.**biggestCntData**(*contours*)

Gets index and area of biggest contour.

Parameters **contours** –

Returns index, area

RRtoolbox.lib.arrayops.mask.**brightness**(*img*)
get brightness from an image :param img: BGR or gray image :return:

RRtoolbox.lib.arrayops.mask.**cnt_hist**(*gray*)
Mask of a ellipse enclosing retina using histogram threshold.

Parameters

- **gray** – gray image
- **invert** – invert mask

Returns mask

RRtoolbox.lib.arrayops.mask.**foreground**(*gray*, *mask=None*, *iterations=3*)
get the foreground mask of a gray image. (this it the inverted of *background()*)

Parameters

- **gray** – gray image
- **mask** – (None) input mask to process gray
- **iterations** – (3) number of iterations to detect foreground with otsu threshold.

Returns output mask

RRtoolbox.lib.arrayops.mask.**gethull**(*contours*)
Get convex hull.

Parameters **contours** – contours or mask array

Returns cnt

RRtoolbox.lib.arrayops.mask.**hist_cdf**(*img*, *window_len=0*, *window='hanning'*)
Get image histogram and the normalized cumulative distribution function.

Parameters

- **img** – imaeg
- **window_len** –
- **window** –

Returns histogram (int), normalized cdf (float)

RRtoolbox.lib.arrayops.mask.**mask_watershed**(*BGR*, *GRAY=None*)
Get retinal mask with watershed method.

Parameters

- **BGR** –
- **GRAY** –

Returns mask

RRtoolbox.lib.arrayops.mask.**multiple_otsu**(*gray*, *mask=None*, *flag=0L*, *iterations=1*)
get the mask of a gray image applying Otsu threshold.

Parameters

- **gray** – gray image
- **mask** – (None) input mask to process gray
- **iterations** –

1. number of iterations to detect Otsu threshold.

Returns thresh, mask

RRtoolbox.lib.arrayops.mask.**thresh_biggestCnt**(*thresh*)
From threshold obtain biggest contour.

Parameters **thresh** – binary image

Returns cnt

RRtoolbox.lib.arrayops.mask.**thresh_hist**(*gray*)
Get best possible thresh to threshold object from the gray image.

Parameters **gray** – gray image.

Returns thresh value.

RRtoolbox.lib.arrayops.mask.**threshold_opening**(*src*, *thresh*, *maxval*, *type*)
Eliminate small objects from threshold.

Parameters

- **src** –
- **thresh** –
- **maxval** –
- **type** –

Returns

Module contents

Submodules

RRtoolbox.lib.cache module

platform Unix, Windows

synopsis Serialize and Memoize.

Contains memoizing, caching, serializing and memory-mapping methods so as to let the package save its state (persistence) and to let a method "remember" what it processed in a session (with cache) or between sessions (memoization and serialization) of the same input contend once processed. It also wraps mmapping functions to let objects "live" in the disk (slower but almost unlimited) rather than in memory (faster but limited).

@cache is used as replacement of *@property* to compute a class method once. It is computed only one time after which an attribute of the same name is generated in its place.

@cachedProperty is used as replacement of *@property* to compute a class method depending on changes in its watched variables.

@memoize used as a general memoizer decorator for functions where metadata is generated to disk for persistence.

Made by Davtoh, powered by joblib. Dependent project: https://github.com/joblib/joblib

class RRtoolbox.lib.cache.**Cache**(*func*)
Bases: object

Descriptor (non-data) for building an attribute on-demand at first use. @cache decorator is used for class methods without inputs (only self reference to the object) and it caches on first compute. ex:

```
class x(object):
    @cache
    def method_x(self):
        return self.data
```

Note: Cached data can be deleted in the decorated object to recalculate its value.

class RRtoolbox.lib.cache.**DynamicMemoizedFunc**(*func*, *cachedir=None*, *ignore=None*, *mmap_mode=None*, *compress=False*, *verbose=1*, *timestamp=None*, *banned=False*)

Bases: object

cachedir

call_and_shelve(**args*, ***kwargs*)

clear(*warn=True*)

compress

enabled

func

ignore

mmap_mode

verbose

class RRtoolbox.lib.cache.**LazyDict**(*getter*, *dictionary=None*)

Bases: _abcoll.MutableMapping

Create objects on demand if needed. call the instance with keys to prevent it from using lazy evaluations (e.g. instead of self[key] use self(key) to prevent recursion). Containing operations are safe to prevent recursion (e.g. if key in self instead of self[key]). In addition use self.isLazy flag to enable or disable lazy operations to prevent possible recursions when getter is called.

class RRtoolbox.lib.cache.**MemoizedDict**(*path*, *mode=None*)

Bases: _abcoll.MutableMapping

memoized dictionary with keys and values persisted to files.

Parameters

- **path** – path to save memo file
- **mode** – loading mode from memo file {None, 'r+', 'r', 'w+', 'c'}

Warning: Some data structures cannot be memoize, so this structure is not save yet. Use at your own risk.

clear()

Remove all items from D.

class RRtoolbox.lib.cache.**Memoizer**(*ignore=()*, *ignoreAll=False*)

Bases: object

ignore

makememory(*cachedir=None*, *mmap_mode=None*, *compress=False*, *verbose=0*)

Make memory for *memoize()* decorator.

Parameters

- **cachedir** – path to save metadata, if left None function is not cached.
- **mmap_mode** – {None, 'r+', 'r', 'w+', 'c'}, optional. The memmapping mode used when loading from cache numpy arrays. See numpy.load for the meaning of the arguments.
- **compress** – (boolean or integer) Whether to zip the stored data on disk. If an integer is given, it should be between 1 and 9, and sets the amount of compression. Note that compressed arrays cannot be read by memmapping.
- **verbose** – (int, optional) Verbosity flag, controls the debug messages that are issued as functions are evaluated.

Returns

memoize(*memory=None*, *ignore=None*, *verbose=0*, *mmap_mode=False*)

Decorated functions are faster by trading memory for time, only hashable values can be memoized.

Parameters

- **memory** – (Memory or path to folder) if left None function is not cached.
- **ignore** – (list of strings) A list of arguments name to ignore in the hashing.
- **verbose** – (integer) Verbosity flag, controls the debug messages that are issued as functions are evaluated.
- **mmap_mode** – {None, 'r+', 'r', 'w+', 'c'}, optional. The memmapping mode used when loading from cache numpy arrays. See numpy.load for the meaning of the arguments.

Returns decorator

memoizers = {140410707069776: <weakref at 0x7fb3ea29eaf8; to 'Memoizer' at 0x7fb3ea514b50>}

class RRtoolbox.lib.cache.**MemorizedFunc**(*func*, *cachedir*, *ignore=None*, *mmap_mode=None*, *compress=False*, *verbose=1*, *timestamp=None*)

Bases: joblib.memory.MemorizedFunc

class RRtoolbox.lib.cache.**Memory**(*cachedir*, *mmap_mode=None*, *compress=False*, *verbose=1*)

Bases: joblib.memory.Memory

A wrapper to joblib Memory to have better control.

class RRtoolbox.lib.cache.**NotMemorizedFunc**(*func*)

Bases: joblib.memory.NotMemorizedFunc

class RRtoolbox.lib.cache.**ObjectGetter**(*callfunc=None*, *obj=None*, *callback=None*, ***annotations*)

Bases: object

Creates or get instance object depending if it is alive.

create(*throw=False*)

Creates an object and keep reference.

Parameters **throw** – if there is not creation function throws error.

Returns created object.

Warning: previous object reference is lost even if it was alive.

Note: Recommended only to use when object from current reference is dead.

getObj(*throw=False*)

isAlive()
test if object of reference is alive

isCreatable()
test if can create object

isGettable()
test if object can be gotten either by reference or creation.

raw()
get object from reference. :return: None if object is dead, object itself if is alive.

update(***kwargs*)

class RRtoolbox.lib.cache.**ResourceManager**(*maxMemory=None*, *margin=0.8*, *unit='MB'*, *all=True*)
Bases: *RRtoolbox.lib.cache.Retriever*

keep track of references, create objects on demand, manage their memory and optimize for better performance.

Parameters

- **maxMemory** – (None) max memory in specified unit to keep in check optimization (it does not mean that memory never surpasses maxMemory).
- **margin** – (0.8) margin from maxMemory to trigger optimization. It is in percentage of maxMemory ranging from 0 (0%) to maximum 1 (100%). So optimal memory is inside range: maxMemory*margin < Memory < maxMemory
- **unit** – (MB) maxMemory unit, it can be GB (Gigabytes), MB (Megabytes), B (bytes)
- **all** – if True used memory is from all alive references, if False used memory is only from keptAlive references.

all

Returns all flag, if True: used memory is from all alive references, if False: used memory is only from keptAlive references.

bytes2units(*value*)
converts value from bytes to user units

getSizeOf(*item*)

keepAlive(*key*, *obj*)

margin

Returns margin used for triggering memory optimization from maxMemory.

maxMemory

optimizeObject(*key*, *getter*, *toWhiteList=False*)

register(*key*, *method=None*, *instance=None*)
Register object to retrieve.

Parameters

- **key** – hashable key to retrieve
- **method** – callable method to get object
- **instance** – object instance already created from method

Note: This method is used in __setitem__ as self.register(key, value). Overwrite this method to change key assignation behaviour.

Example:

```
def mymethod():
    class constructor: pass
    return constructor()

ret = retriever()
ret["obj"] = mymethod # register creating method in "obj"
im = ret["obj"] # get object (created obj +1, with reference)
assert im is ret["obj"] # check that it gets the same object
# it remembers that "obj" is last registered or fetched object too
assert ret() is ret()
# lets register with better control (created obj2 +1, no reference)
ret.register("obj2",mymethod(),mymethod)
# proves that obj2 is not the same as obj (created obj2 +1, no reference)
assert ret() is not ret["obj"]
print list(ret.iteritems()) # get items
```

static **resetGetter**(*getter*)
Helper function to reset getter parameters.

Parameters **getter** – any instance of objectGetter

unit

Returns user defined units

units2bytes(*value*)
converts value from user units two bytes

usedMemory

Returns used memory in user units

class RRtoolbox.lib.cache.**Retriever**
Bases: _abcoll.MutableMapping

keep track of references and create objects on demand if needed.

register(*key*, *method=None*, *instance=None*)
Register object to retrieve.

Parameters

- **key** – hashable key to retrieve
- **method** – callable method to get object
- **instance** – object instance already created from method

Returns

Example:

```
def mymethod():
    class constructor: pass
    return constructor()

ret = retriever()
```

```
        ret["obj"] = mymethod # register creating method in "obj"
        im = ret["obj"] # get object (created obj +1, with reference)
        assert im is ret["obj"] # check that it gets the same object
        # it remembers that "obj" is last registered or fetched object too
        assert ret() is ret()
        # lets register with better control (created obj2 +1, no reference)
        ret.register("obj2",mymethod(),mymethod)
        # proves that obj2 is not the same as obj (created obj2 +1, no reference)
        assert ret() is not ret["obj"]
        print list(ret.iteritems()) # get items
```

RRtoolbox.lib.cache.**cachedProperty**(*watch=[]*, *handle=[]*)
A memoize decorator of @property decorator specifying what to trigger caching.

Parameters

- **watch** – (list of strings) A list of arguments name to watch in the hashing.
- **handle** – (list of handles or empty list) Provided list is appended with the Memo handle were data is stored for the method and where a clear() function is provided.

Returns

RRtoolbox.lib.cache.**mapper**(*path*, *obj=None*, *mode=None*, *onlynumpy=False*)
Save and load or map live objects to disk to free RAM memory.

Parameters

- **path** – path to save mapped file.
- **obj** – the object to map, if None it tries to load obj from path if exist
- **mode** – {None, 'r+', 'r', 'w+', 'c'}.
- **onlynumpy** – if True, it saves a numpy mapper from obj.

Returns mmap image, names of mmap files

RRtoolbox.lib.config module

platform Unix, Windows

synopsis Looking for a reference? look here!.

This module contains all config data to the package.

class RRtoolbox.lib.config.**ConfigTool**
Manage the configured Tools.

static getTools(*package*)
Obtains the tools of a directory for the RRtoolbox.

Parameters **package** – path to the directory or package object.

Returns a dictionary of imported modules.

class RRtoolbox.lib.config.**DirectoryManager**(*path=None*, *raiseError=True*, *autosave=False*)
Bases: object

Manage the configured variables, paths and files.

Parameters

- **path** – (None) path to configuration file. If None uses default path.

- **raiseError** – True to raise when not attribute in ConfigFile.
- **autosave** – (True) if True saves at each change.

Note: Any attribute that is not in ConfigFile returns None. Use raiseError to control this behaviour.

default
get directories from dictionary representing environment variables.

Returns dictionary of directories.

Note: Only directories in the scope of the module are detected.

load()
loads the configuration file and update.

Returns loaded configuration file dictionary.

Warning: Unsaved instance variables will be replaced by configuration file variables.

reset()
Returns the configuration file to default variables.

Returns False, if error. Dictionary of new data, if successful.

Warning: All custom data is lost in configuration file.

Warning: ConfigFile is purposely not updated. Call manually method load()

save(*mode=0*)
saves configuration file.

Parameters **mode** – 0- delete and save, 1- update without replace, 2- update replacing variables.

Returns False, if error. Dictionary of new data, if successful.

RRtoolbox.lib.config.**findModules**(*package*, *exclude=None*)
Find modules from a package.

Parameters

- **package** – imported packaged or path (str).
- **exclude** – list of modules to exclude.

Returns dictionary containing importer, ispkg

RRtoolbox.lib.config.**getModules**(*package*, *exclude=None*)
Import modules from a package.

Parameters **package** – imported packaged or path (str).

Returns dictionary containing imported modules.

RRtoolbox.lib.config.**getPackagePath**(*package*)
Get the path of a package object.

Parameters **package** – package object or path (str).

Returns path to the package.

RRtoolbox.lib.descriptors module

RRtoolbox.lib.descriptors.**ASIFT**(*feature_name*, *img*, *mask=None*, *pool=<multiprocessing.pool.ThreadPool object>*)

asift(feature_name, img, mask=None, pool=None) -> keypoints, descrs

Apply a set of affine transformations to the image, detect keypoints and reproject them into initial image coordinates. See http://www.ipol.im/pub/algo/my_affine_sift/ for the details.

ThreadPool object may be passed to speedup the computation.

Parameters

- **feature_name** – feature name to create detector.
- **img** – image to find keypoints and its descriptors
- **mask** – mask to detect keypoints (it uses default, mask[:] = 255)
- **pool** – multiprocessing pool (dummy, it uses multithreading)

Returns keypoints,descriptors

RRtoolbox.lib.descriptors.**ASIFT_iter**(*imgs*, *feature_name='sift-flann'*)

Affine-SIFT for N images.

Parameters

- **imgs** – images to apply asift
- **feature_name** – eg. SIFT SURF ORB

Returns [(kp1,desc1),...,(kpN,descN)]

RRtoolbox.lib.descriptors.**ASIFT_multiple**(*imgs*, *feature_name='sift-flann'*)

Affine-SIFT for N images.

Parameters

- **imgs** – images to apply asift
- **feature_name** – eg. SIFT SURF ORB

Returns [(kp1,desc1),...,(kpN,descN)]

class RRtoolbox.lib.descriptors.**Feature**(*pool=<multiprocessing.pool.ThreadPool object>*, *useASIFT=True*, *debug=True*)

Bases: `object`

Class to manage detection and computation of features

Parameters

- **pool** – multiprocessing pool (dummy, it uses multithreading)
- **useASIFT** – if True adds Affine perspectives to the detector.
- **debug** – if True prints to the stdout debug messages.

config(*name*, *separator='-'*)

This function takes parameters from a command to initialize a detector and matcher.

Parameters

- **name** – "[a-]<sift|surf|orb>[-flann]" (str) Ex: "a-sift-flann"

- **features** – it is a dictionary containing the mapping from name to the initialized detector, matcher pair. If None it is created. This feature is to reduce time by reusing created features.

Returns detector, matcher

detectAndCompute(*img*, *mask=None*)
detect keypoints and descriptors

Parameters

- **img** – image to find keypoints and its descriptors
- **mask** – mask to detect keypoints (it uses default, mask[:] = 255)

Returns keypoints,descriptors

RRtoolbox.lib.descriptors.**MATCH**(*feature_name*, *kp1*, *desc1*, *kp2*, *desc2*)
Use matcher and asift output to obtain Transformation matrix (TM).

Parameters

- **feature_name** – feature name to create detector. It is the same used in the detector which is used in init_feature function but the detector itself is ignored. e.g. if 'detector' uses BFMatcher, if 'detector-flann' uses FlannBasedMatcher.
- **kp1** – keypoints of source image
- **desc1** – descriptors of kp1
- **kp2** – keypoints of destine image
- **desc2** – descriptors of kp2

Returns TM

http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_feature_homography/py_feature_homography.html

RRtoolbox.lib.descriptors.**MATCH_multiple**(*pairlist*, *feature_name='sift-flann'*)

Parameters

- **pairlist** – list of keypoint and descriptors pair e.g. [(kp1,desc1),...,(kpN,descN)]
- **feature_name** – feature name to create detector

Returns [(H1, mask1, kp_pairs1),....(HN, maskN, kp_pairsN)]

RRtoolbox.lib.descriptors.**affine_skew**(*tilt*, *phi*, *img*, *mask=None*)
Increase robustness to descriptors by calculating other invariant perspectives to image.

Parameters

- **tilt** – tilting of image
- **phi** – rotation of image (in degrees)
- **img** – image to find Affine transforms
- **mask** – mask to detect keypoints (it uses default, mask[:] = 255)

Returns skew_img, skew_mask, Ai (invert Affine Transform)

Ai - is an affine transform matrix from skew_img to img

RRtoolbox.lib.descriptors.**filter_matches**(*kp1*, *kp2*, *matches*, *ratio=0.75*)
This function applies a ratio test.

Parameters

- **kp1** – raw keypoints 1
- **kp2** – raw keypoints 2
- **matches** – raw matches
- **ratio** – filtering ratio of distance

Returns filtered keypoint 1, filtered keypoint 2, keypoint pairs

RRtoolbox.lib.descriptors.**init_feature**(*name*, *features=None*)
This function takes parameters from a command to initialize a detector and matcher.

Parameters

- **name** – "<sift|surf|orb>[-flann]" (str) Ex: "sift-flann"
- **features** – it is a dictionary containing the mapping from name to the initialized detector, matcher pair. If None it is created. This feature is to reduce time by reusing created features.

Returns detector, matcher

RRtoolbox.lib.descriptors.**inlineRatio**(*inlines*, *lines*, *thresh=30*)
Probability that a match was correct.

Parameters

- **inlines** – number of matched lines
- **lines** – number lines
- **thresh** – threshold for lines (i.e. very low probability <= thresh < good probability)

Returns

RRtoolbox.lib.directory module

This module holds all path manipulation methods and a string concept called directory (referenced paths and strings) designed to support `config` and be used with `session`.

keywords:

path: it can be to a folder or file or url if specified *filename*: the file name without its path *filepath*: the path to a file *dirname*: the path to a folder *url*: Universal Resource Locator

class RRtoolbox.lib.directory.**Directory**
Bases: `str`

semi-mutable string representation of a inmutable string with support for path representations.

Parameters

- **data** – list, directory instance, dictionary or string.
- **ispath** – True to add support for paths.
- **copy** – when data is a directory if copy is True then this instance data is independent of the passed directory otherwise both directories are a reference to the same dictionary data but they are not the same object.
- **kwargs** – additional data to add in directory.

copy()
Creates copy of itself.

Returns non-referenced directory copy.

correctSTRBuiltin()
Decorate all the built-in functions of class directory.

Returns built-in decorated function.

static filterdata(*data*, *ispath=None*, *kwargs=None*)
Adequate data for dictionary creation.

Parameters

- **data** – any supported object.
- **ispath** – True to add support for paths.
- **kwargs** – additional data to add in directory.

Returns dictionary

static repr2list(*data*, *level=0*)
Converts the representation of a directory.repr to pickleable.

Parameters **data** – directory.repr of the form ["string",directory,...,directory.repr].

Returns pickleable list.

static repr2str(*data*, *ispath=True*)
Converts the representation of a directory.repr to string.

Parameters **data** – directory.repr of the form ["string",directory,...,directory.repr].

Returns converted string.

update(*data=None*)
Return an updated copy with provided data.

Parameters **data** – any supported object. If None return updated and referenced copy of itself.

Returns new directory referenced to itself.

update_left(*other*)
Updates representation a the left.

Parameters **other** – any supported object.

Returns new directory referenced to itself.

Note: Equivalent to self - other e.g. directory([other, self])

---

update_right(*other*)
Updates representation a the right.

Parameters **other** – any supported object.

Returns new directory referenced to itself.

Note: Equivalent to self + other e.g. directory([self, other])

---

class RRtoolbox.lib.directory.**FileDirectory**
Bases: *RRtoolbox.lib.directory.Directory*

Saves contents of a file as with directories.

Parameters

- **data** – list, directory instance, dictionary or string.
- **filename** – name of file.
- **path** – path to folder where file is (it must finish in /).
- **notes** – optional description string
- **kwargs** – additional data to add in directory.

makeFile()
Makes a file with its contents to path/filename.

Returns True if successful

RRtoolbox.lib.directory.**changedir**(*filepath*, *dirname*, *ext=True*)
Change path to file with dirname.

Parameters

- **filepath** – path to file.
- **dirname** – new path to replace in filepath.
- **ext** – True to keep extension of file if any.

Returns directory object of changed path.

RRtoolbox.lib.directory.**checkDir**(*dirname*)
checks if dirname exists.

Parameters **dirname** – path to folder

Returns True if exits, False if not

RRtoolbox.lib.directory.**checkFile**(*path*)
checks if filepath or filename exists.

Parameters **path** – filepath or filename

Returns True if exits, False if not

RRtoolbox.lib.directory.**checkPath**(*path*)
checks if path exists.

Parameters **path** – path to folder or file.

Returns True if exits, False if not

RRtoolbox.lib.directory.**checkURL**(*url*)
checks if url exists. :param url: path to url :return: True if exits, False if not

RRtoolbox.lib.directory.**correctPath**(*path*, *relative*)
Get path corrected from its relative path or level index.

Parameters

- **path** – path or file name.
- **relative** – pattern or level in directory.

Returns corrected path.

RRtoolbox.lib.directory.**correctSep**(*path='/mnt/4E443F99443F82AF/Dropbox/PYTHON/RRtools/RRtoolbox/lib/direct*
separator='/')
Replaces the path separators by custom or OS standard separator.

Parameters

- **path** – relative or absolute path (str). Default is __file__ or module's path.
- **separator** – desired separators, By default uses system separator (os.path.sep).

Returns path with corrected separator.

RRtoolbox.lib.directory.**decoratePath**(*relative*, *sep='/'*)
Decorated path is controlled to give absolute path from relative path.

Parameters

- **relative** – int or path.
- **sep** – path separator

Returns decorator

RRtoolbox.lib.directory.**getData**(*path='/mnt/4E443F99443F82AF/Dropbox/PYTHON/RRtools/RRtoolbox/lib/directory.p*
Get standard path from path.

Parameters **path** – it can be to a folder or file. Default is __file__ or module's path.

Returns [drive,dirname,filename,ext]. 1. drive or UNC (Universal Naming Convention) 2. dirname is path to folder. 3. filename is name of file. 4. ext is extension of file.

RRtoolbox.lib.directory.**getFileHandle**(*path*)
Gets a file handle from url or disk file.

Parameters **path** – filepath or url

Returns file object

RRtoolbox.lib.directory.**getFileSize**(*path*)
Gets a size from url or disk file.

Parameters **path** – filepath or url

Returns size in bytes

RRtoolbox.lib.directory.**getPath**(*path='/mnt/4E443F99443F82AF/Dropbox/PYTHON/RRtools/RRtoolbox/lib/directory.p*
Get standard path from path.

Parameters **path** – it can be to a folder or file. Default is __file__ or module's path. If file exists it selects its folder.

Returns dirname (path to a folder)

Note: It is the same as os.path.dirname(os.path.abspath(path)).

RRtoolbox.lib.directory.**getSep**(*path*, *pattern='/\\'*)
Get path separator or indicator.

Parameters

- **path** – relative or absolute path (str).
- **pattern** – guess characters to compare path (str).

Returns sep (str).

Note: It is equivalent to os.path.sep but obtained from the given path and patterns.

RRtoolbox.lib.directory.**getShortenedPath**(*path*, *comp*)
Path is controlled to give absolute path from relative path or integer.

Parameters

- **path** – absolute path (str).
- **comp** – pattern or relative path (str) or integer representing level of folder determined by the separator Ex. "/level 1/level 2/.../level N or -1".

Returns path before matched to comp Ex: "C://level 1//comp –> C://level 1"

Example:

```
>>> path = 'LEVEL1/LEVEL2/LEVEL3/LEVEL4/LEVEL5'
>>> print getShortenedPath(path,-2) # minus two levels
LEVEL1/LEVEL2/LEVEL3
>>> print getShortenedPath(path,2) # until three levels
LEVEL1/LEVEL2
>>> print getShortenedPath(path,'LEVEL1/LEVEL2/LEVEL3/')
LEVEL1/LEVEL2/LEVEL3/
>>> print getShortenedPath(path,'LEVEL4/REPLACE5/NEWLEVEL')
LEVEL1/LEVEL2/LEVEL3/LEVEL4/REPLACE5/NEWLEVEL
>>> print getShortenedPath(path,'../../SHOULD_BE_LEVEL4')
LEVEL1/LEVEL2/LEVEL3/SHOULD_BE_LEVEL4
```

RRtoolbox.lib.directory.**getSplitted**(*path='/mnt/4E443F99443F82AF/Dropbox/PYTHON/RRtools/RRtoolbox/lib/direc*
Splits a file path by its separators.

Parameters **path** – it can be to a folder or file. Default is __file__ or module's path.

Returns splitted path.

RRtoolbox.lib.directory.**increment_if_exits**(*path*, *add='_{num}'*, *force=None*)
Generates new name if it exits.

Parameters

- **path** – absolute path or filename
- **add** – if fn exists add pattern
- **force** – (None) force existent files even if they don't. if True treats fn as existent or if it is a list it treats names from the list as existent names.

Returns un-existent fn

RRtoolbox.lib.directory.**joinPath**(*absolute*, *relative*)
Joins an absolute path to a relative path.

Parameters

- **absolute** – directory or path.
- **relative** – directory or path.

Returns joined path.

Note: It is equivalent to os.path.join but works with directories.

RRtoolbox.lib.directory.**mkPath**(*path*)
Make path (i.e. creating folder) for filepath.

Parameters **path** – path to nonexistent folder or file.

Returns created path.

RRtoolbox.lib.directory.**quickOps**(*path*, *comp*)
(IN DEVELOPMENT) make quick matching operations in path.

Parameters

- **path** – path to folder
- **comp** – pattern

Returns

Requirements:

```
path = 'LEVEL1/LEVEL2/LEVEL3/LEVEL4/LEVEL5'
print quickOps(path,'../ROOT/../LEVEL1/../LEVEL2/LEVEL3/../../LEVEL4')
'LEVEL4'
print quickOps(path,'ROOT/../LEVEL1/LEVEL2/../../LEVEL4')
'LEVEL3/LEVEL4'
print quickOps(path,'../LEVEL2/../')
'LEVEL1/LEVEL3/LEVEL4/LEVEL5'
print quickOps(path,'../LEVEL2/..')
'LEVEL1/LEVEL3/LEVEL4/LEVEL5/'
print quickOps(path,'LEVEL2/../../LEVEL4/')
'LEVEL2/LEVEL3/LEVEL4/'
print quickOps(path,'ROOT/../LEVEL2/../../LEVEL4')
'ROOT/LEVEL3/LEVEL4'
print quickOps(path,'LEVEL-1/../NEW7/LEVEL8')
'LEVEL-1/LEVEL1/LEVEL2/LEVEL3/LEVEL4/LEVEL5/NEW7/LEVEL8'
print
```

RRtoolbox.lib.directory.**resource_path**(*relative_path=''*)
Get absolute path to resource, works for dev and for PyInstaller

RRtoolbox.lib.directory.**rmFile**(*filepath*)
Remove file.

Parameters **filepath** – path to file.

Returns None

RRtoolbox.lib.directory.**rmPath**(*path*, *ignore_errors=False*, *onerror=None*)
Remove path from path.

Parameters **path** – path to nonexistent folder or file.

Returns None

See also:

shutil.rmtree

RRtoolbox.lib.directory.**strdifference**(*s1*, *s2*)
Get string differences.

Parameters

- **s1** – string 1
- **s2** – string 2

Returns (splitted string 1, splitted string 2, index). A splitted string is a list with the string parts. Index is a list containing the indexes of different parts of the two splitted strings.

RRtoolbox.lib.image module

Bundle of methods for handling images. Rather than manipulating specialized operations in images methods in this module are used for loading, outputting and format-converting methods, as well as color manipulation.

SUPPORTED FORMATS

see http://docs.opencv.org/2.4/modules/highgui/doc/reading_and_writing_images_and_video.html#imread

Windows bitmaps - *.bmp, *.dib (always supported) JPEG files - *.jpeg, *.jpg, *.jpe (see the Notes section) JPEG 2000 files - *.jp2 (see the Notes section) Portable Network Graphics - *.png (see the Notes section) Portable image format - *.pbm, *.pgm, *.ppm (always supported) Sun rasters - *.sr, *.ras (always supported) TIFF files - *.tiff, *.tif (see the Notes section)

class RRtoolbox.lib.image.**GetCoors**(*im*, *win='get coordinates'*, *updatefunc=<function drawcoorpoints>*, *unique=True*, *col_out=(0, 0, 0)*, *col_in=(0, 0, 255)*)

Bases: *RRtoolbox.lib.plotter.Plotim*

Create window to select points from image.

Parameters

- **im** – image to get points.
- **win** – window name.
- **updatefunc** – function to draw interaction with points. (e.g. limitaxispoints, drawcoorperspective, etc.).
- **prox** – proximity to identify point.
- **radius** – radius of drawn points.
- **unique** – If True no point can be repeated, else selected points can be repeated.
- **col_out** – outer color of point.
- **col_in** – inner color of point.

coors

drawstats(*points*, *col_out=(0, 0, 0)*, *col_in=(0, 255, 0)*, *radius=2*)

Parameters

- **self** –
- **points** –
- **col_out** –
- **col_in** –
- **radius** –

Returns

mousefunc()

Parameters **self** –

Returns

updatecoors()

Parameters **self** –

Returns

class RRtoolbox.lib.image.**ImCoors**(*pts*, *dtype=<type 'numpy.float32'>*, *deg=False*)
Bases: object

Image's coordinates class. Example:

```
a = ImCoors(np.array([(116, 161), (295, 96), (122, 336), (291, 286)]))
print a.__dict__
print "mean depend on min and max: ", a.mean
print a.__dict__
print "after mean max has been already been calculated: ", a.max
a.data = np.array([(116, 161), (295, 96)])
print a.__dict__
print "mean and all its dependencies are processed again: ", a.mean
```

dtype

pts

class RRtoolbox.lib.image.**ImFactory**(***kwargs*)
image factory for RRToolbox to create scripts to standardize loading images and provide lazy loading (it can load images from disk with the customized options and/or create mapping images to load when needed) to conserve memory.

> **Warning:** In development.

get_Func()
gets the loading function

get_code()
get the script code

get_convertionFunc(*code*)

get_errorFunc(*path=None*, *throw=None*)

get_loadFunc(*flag=None*)

get_mapFunc(*flag=None*, *RGB=None*, *mpath=None*, *mode=None*, *func=None*, *dsize=None*, *dst=None*, *fx=None*, *fy=None*, *interpolation=None*)

get_np2qi()

get_resizeFunc(*dsize=None*, *dst=None*, *fx=None*, *fy=None*, *interpolation=None*)

get_transposeFunc()

update(***kwargs*)

class RRtoolbox.lib.image.**ImLoader**(*path*, *flag=0*, *dsize=None*, *dst=None*, *fx=None*, *fy=None*, *interpolation=None*, *mmode=None*, *mpath=None*, *throw=True*)
Class to load image array from path, url, server, string or directly from numpy array (supports databases).

Parameters

- **flag** – (default: 0) 0 to read as gray, 1 to read as BGR, -1 to read as BGRA, 2 to read as RGB, -2 to read as RGBA.

  It supports openCV flags:

  - cv2.CV_LOAD_IMAGE_COLOR
  - cv2.CV_LOAD_IMAGE_GRAYSCALE
  - cv2.CV_LOAD_IMAGE_UNCHANGED

value	openCV flag	output
2.	N/A	RGB
1.	cv2.CV_LOAD_IMAGE_COLOR	BGR
0.	cv2.CV_LOAD_IMAGE_GRAYSCALE	GRAYSCALE
(-1)	cv2.CV_LOAD_IMAGE_UNCHANGED	BGRA
(-2)	N/A	RGBA

- **dsize** – (None) output image size; if it equals zero, it is computed as:

  exttt{dsize = Size(round(fx*src.cols), round(fy*src.rows))}

- **dst** – (None) output image; it has the size dsize (when it is non-zero) or the size computed from src.size(), fx, and fy; the type of dst is uint8.

- **fx** – scale factor along the horizontal axis; when it equals 0, it is computed as

  exttt{(double)dsize.width/src.cols}

- **fy** – scale factor along the vertical axis; when it equals 0, it is computed as

  exttt{(double)dsize.height/src.rows}

- **interpolation** – interpolation method compliant with opencv:

flag	Operation	Description
0.	INTER_NEAREST	nearest-neighbor interpolation
1.	INTER_LINEAR	bilinear interpolation (used by default)
2.	INTER_CUBIC	bicubic interpolation over 4x4 pixel neighborhood
3.	INTER_AREA	resampling using pixel area relation. It may be a preferred method for image decimation, as it gives moire'-free results. But when the image is zoomed, it is similar to the INTER_NEAREST method.
4.	INTER_LANCZOS4	Lanczos interpolation over 8x8 pixel neighborhood

- **mmode** – (None) mmode to create mapped file. if mpath is specified loads image, converts to mapped file and then loads mapping file with mode {None, 'r+', 'r', 'w+', 'c'} (it is slow for big images). If None, loads mapping file to memory (useful to keep image copy for session even if original image is deleted or modified).

- **mpath** – (None) path to create mapped file. None, do not create mapping file "", uses path directory; "*", uses working directory; else, uses specified directory.

Note: If mmode is None and mpath is given it creates mmap file but loads from it to memory. It is useful to create physical copy of data to keep loading from (data can be reloaded even if original file is moved or deleted).

getConfiguration(***kwargs*)
get Custom configuration from default configuration :param kwargs: keys to customize default configuration.

If no key is provided default configuration is returned.

Returns dictionary of configuration

temp(***kwargs*)
loads from temporal loader created with customized and default parameters.

Parameters **kwargs** – keys to customize default configuration.

Returns loaded image.

class RRtoolbox.lib.image.**Image**(*name=None*, *ext=None*, *path=None*, *shape=None*, *verbosity=False*)
Bases: object

Structure to load and save images

BGR

BGRA

RGB

RGBA

ext

gray

load(*name=None*, *path=None*, *shape=None*)

path

save(*name=None*, *image=None*, *overwrite=None*)
save restored image in path.

Parameters

- **name** – 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.
- **image** – (self.BGRA)
- **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}"

Returns saved path, status (True for success and False for fail)

shape

class RRtoolbox.lib.image.**LoaderDict**(*loader=None*, *maxMemory=None*, *margin=0.8*, *unit='MB'*, *all=True*, *config=None*)
Bases: *RRtoolbox.lib.cache.ResourceManager*

Class to standardize loading objects and manage memory efficiently.

Parameters

- **loader** – default loader for objects (e.g. load from file or create instance object)
- **maxMemory** – (None) max memory in specified unit to keep in check optimization (it does not mean that memory never surpasses maxMemory).
- **margin** – (0.8) margin from maxMemory to trigger optimization. It is in percentage of maxMemory ranging from 0 (0%) to maximum 1 (100%). So optimal memory is inside range: maxMemory*margin < Memory < maxMemory
- **unit** – (MB) maxMemory unit, it can be GB (Gigabytes), MB (Megabytes), B (bytes)
- **all** – if True used memory is from all alive references, if False used memory is only from keptAlive references.
- **config** – (Not Implemented)

register(*key*, *path=None*, *method=None*)

class RRtoolbox.lib.image.**PathLoader**(*fns=None*, *loader=None*)
Bases: _abcoll.MutableSequence

Class to standardize loading images from list of paths and offer lazy evaluations.

Parameters

- **fns** – list of paths
- **loader** – path loader (loadcv,loadsfrom, or function from loadFunc)

Example:

```
fns = ["/path to/image 1.ext","/path to/image 2.ext"]
imgs = pathLoader(fns)
print imgs[0] # loads image in path 0
print imgs[1] # loads image in path 1
```

insert(*index*, *value*)

RRtoolbox.lib.image.**bgra2bgr**(*im*, *bgrcolor=(255*, *255*, *255)*)
Convert BGR to BGRA image.

Parameters

- **im** – image
- **bgrcolor** – BGR color representing transparency. (information is lost when converting BGRA to BGR) e.g. [200,200,200].

Returns

RRtoolbox.lib.image.**checkLoaded**(*obj*, *fn=''*, *raiseError=False*)
Simple function to determine if variable is valid.

Parameters

- **obj** – loaded object
- **fn** – path of file
- **raiseError** – if True and obj is None, raise

Returns None

RRtoolbox.lib.image.**convertAs**(*fns*, *base=None*, *folder=None*, *name=None*, *ext=None*, *overwrite=False*, *loader=None*, *simulate=False*)
Reads a file and save as other file based in a pattern.

Parameters

- **fns** – file name or list of file names. It supports glob operations. By default glob operations ignore folders.
- **base** – path to place images.
- **folder** – (None) folder to place images in base's path. If True it uses the folder in which image was loaded. If None, not folder is used.
- **name** – string for formatting new name of image with the {name} tag. Ex: if name is 'new_{name}' and image is called 'img001' then the formatted new image's name is 'new_img001'
- **ext** – (None) extension to save all images. If None uses the same extension as the loaded image.
- **overwrite** – (False) If True and the destine filename for saving already exists then it is replaced, else a new filename is generated with an index "{name}_{index}.{extension}"
- **loader** – (None) loader for the image file to change image attributes. If None reads the original images untouched.
- **simulate** – (False) if True, no saving is performed but the status is returned to confirm what images where adequately processed.

Returns list of statuses (0 - no error, 1 - image not loaded, 2 - image not saved, 3 - error in processing image)

RRtoolbox.lib.image.**drawcoorarea**(*vis*, *points*, *col_out=(0, 0, 0)*, *col_in=(0, 0, 255)*, *radius=2*)

Function to draw interaction with points to obtain area.

Parameters

- **vis** – image array.
- **points** – list of points.
- **col_out** – outer color of point.
- **col_in** – inner color of point.
- **radius** – radius of drawn points.

Returns

RRtoolbox.lib.image.**drawcooraxes**(*vis*, *points*, *col_out=(0, 0, 0)*, *col_in=(0, 255, 0)*, *radius=2*)

Function to draw axes instead of points.

Parameters

- **vis** – image array.
- **points** – list of points.
- **col_out** – outer color of point.
- **col_in** – inner color of point.
- **radius** – radius of drawn points.

Returns

RRtoolbox.lib.image.**drawcoorperspective**(*vis*, *points*, *col_out=(0, 0, 0)*, *col_in=(0, 0, 255)*, *radius=2*)

Function to draw interaction with points to obtain perspective.

Parameters

- **vis** – image array.
- **points** – list of points.
- **col_out** – outer color of point.
- **col_in** – inner color of point.
- **radius** – radius of drawn points.

Returns

RRtoolbox.lib.image.**drawcoorpoints**(*vis*, *points*, *col_out=(0, 0, 0)*, *col_in=(0, 0, 255)*, *radius=2*)

Funtion to draw points.

Parameters

- **vis** – image array.
- **points** – list of points.
- **col_out** – outer color of point.
- **col_in** – inner color of point.
- **radius** – radius of drawn points.

Returns

RRtoolbox.lib.image.**drawcoorpolyArrow**(*vis*, *points*, *col_out=(0, 0, 0)*, *col_in=(0, 0, 255)*, *radius=2*)

Function to draw interaction with vectors to obtain polygonal.

Parameters

- **vis** – image array.
- **points** – list of points.
- **col_out** – outer color of point.
- **col_in** – inner color of point.
- **radius** – radius of drawn points.

Returns

RRtoolbox.lib.image.**drawcoorpolyline**(*vis*, *points*, *col_out=(0, 0, 0)*, *col_in=(0, 0, 255)*, *radius=2*)

Function to draw interaction with points to obtain polygonal.

Parameters

- **vis** – image array.
- **points** – list of points.
- **col_out** – outer color of point.
- **col_in** – inner color of point.
- **radius** – radius of drawn points.

Returns

RRtoolbox.lib.image.**fig2bgr**(*fig*)

Convert a Matplotlib figure to a RGB image.

Parameters **fig** – a matplotlib figure

Returns RGB image.

RRtoolbox.lib.image.**fig2bgra**(*fig*)
Convert a Matplotlib figure to a RGBA image.

Parameters **fig** – a matplotlib figure

Returns RGBA image.

RRtoolbox.lib.image.**getcoors**(*im*, *win='get coordinates'*, *updatefunc=<function drawcoorpoints>*, *coors=None*, *prox=8*, *radius=3*, *unique=True*, *col_out=(0, 0, 0)*, *col_in=(0, 0, 255)*)

RRtoolbox.lib.image.**getgeometrycoors**(**data*)
Get filled object coordinates. (function in progress)

RRtoolbox.lib.image.**getrectcoors**(**data*)

Get ordered points.

Parameters **data** – list of points

Returns [Top_left,Top_right,Bottom_left,Bottom_right]

RRtoolbox.lib.image.**gray2qi**(*gray*)
Convert the 2D numpy array *gray* into a 8-bit QImage with a gray colormap. The first dimension represents the vertical image axis.

ATTENTION: This QImage carries an attribute *ndimage* with a reference to the underlying numpy array that holds the data. On Windows, the conversion into a QPixmap does not copy the data, so that you have to take care that the QImage does not get garbage collected (otherwise PyQt will throw away the wrapper, effectively freeing the underlying memory - boom!).

source from: https://kogs-www.informatik.uni-hamburg.de/~meine/software/vigraqt/qimage2ndarray.py

RRtoolbox.lib.image.**hist_match**(*source*, *template*, *alpha=None*)
Adjust the pixel values of an image to match those of a template image.

Parameters

- **source** – image to transform colors to template
- **template** – template image ()
- **alpha** –

Returns transformed source

RRtoolbox.lib.image.**interpretImage**(*toparse*, *flags*)
Interprets to get image.

Parameters

- **toparse** – string to parse or array. It can interpret:

  *connection to server (i.e. host:port) *path to file (e.g. /path_to_image/image_name.ext) *URL to image (e.g. http://domain.com/path_to_image/image_name.ext) *image as string (i.g. numpy converted to string) *image itself (i.e. numpy array)

- **flags** – openCV flags:

value	openCV flag	output
1.	cv2.CV_LOAD_IMAGE_COLOR	BGR
0.	cv2.CV_LOAD_IMAGE_GRAYSCALE	GRAYSCALE
(-1)	cv2.CV_LOAD_IMAGE_UNCHANGED	Normal

Returns image or None if not successfull

RRtoolbox.lib.image.**limitaxispoints**(*c*, *maxc*, *minc=0*)

Limit a point in axis.

Parameters

- **c** – list of points..
- **maxc** – maximum value of point.
- **minc** – minimum value of point.

Returns return limited points.

RRtoolbox.lib.image.**loadFunc**(*flag=0*, *dsize=None*, *dst=None*, *fx=None*, *fy=None*, *interpolation=None*, *mmode=None*, *mpath=None*, *throw=True*, *keepratio=True*)

Creates a function that loads image array from path, url, server, string or directly from numpy array (supports databases).

Parameters

- **flag** – (default: 0) 0 to read as gray, 1 to read as BGR, -1 to read as BGRA, 2 to read as RGB, -2 to read as RGBA.

  It supports openCV flags:

  - cv2.CV_LOAD_IMAGE_COLOR
  - cv2.CV_LOAD_IMAGE_GRAYSCALE
  - cv2.CV_LOAD_IMAGE_UNCHANGED

value	openCV flag	output
2.	N/A	RGB
1.	cv2.CV_LOAD_IMAGE_COLOR	BGR
0.	cv2.CV_LOAD_IMAGE_GRAYSCALE	GRAYSCALE
(-1)	cv2.CV_LOAD_IMAGE_UNCHANGED	BGRA
(-2)	N/A	RGBA

- **dsize** – (None) output image size; if it equals zero, it is computed as:

  exttt{dsize = Size(round(fx*src.cols), round(fy*src.rows))}

  If (integer,None) or (None,integer) it completes the values according to keepratio parameter.

- **dst** – (None) output image; it has the size dsize (when it is non-zero) or the size computed from src.size(), fx, and fy; the type of dst is uint8.
- **fx** – scale factor along the horizontal axis
- **fy** – scale factor along the vertical axis
- **interpolation** – interpolation method compliant with opencv:

flag	Operation	Description
0.	INTER_NEAREST	nearest-neighbor interpolation
1.	INTER_LINEAR	bilinear interpolation (used by default)
2.	INTER_CUBIC	bicubic interpolation over 4x4 pixel neighborhood
3.	INTER_AREA	resampling using pixel area relation. It may be a preferred method for image decimation, as it gives moire'-free results. But when the image is zoomed, it is similar to the INTER_NEAREST method.
4.	INTER_LANCZOS4	Lanczos interpolation over 8x8 pixel neighborhood

- **mmode** – (None) mmode to create mapped file. if mpath is specified loads image, converts to mapped file and then loads mapping file with mode {None, 'r+', 'r', 'w+', 'c'} (it is slow for big images). If None, loads mapping file to memory (useful to keep image copy for session even if original image is deleted or modified).
- **mpath** – (None) path to create mapped file. None, do not create mapping file "", uses path directory; "*", uses working directory; else, uses specified directory.
- **keepratio** – True to keep image ratio when completing data from dsize,fx and fy, False to not keep ratio.

Note: If mmode is None and mpath is given it creates mmap file but loads from it to memory. It is useful to create physical copy of data to keep loading from (data can be reloaded even if original file is moved or deleted).

:return loader function

RRtoolbox.lib.image.**loadcv**(*path*, *flags=-1*, *shape=None*)

Simple function to load using opencv.

Parameters

- **path** – path to image.
- **flag** – openCV flags:

value	openCV flag	output
1.	cv2.CV_LOAD_IMAGE_COLOR	BGR
0.	cv2.CV_LOAD_IMAGE_GRAYSCALE	GRAYSCALE
(-1)	cv2.CV_LOAD_IMAGE_UNCHANGED	Unmodified

- **shape** – shape to resize image.

Returns loaded image

RRtoolbox.lib.image.**loadsfrom**(*path*, *flags=1L*)

Loads Image from URL or file.

Parameters

- **path** – filepath or url
- **flags** – openCV flags:

value	openCV flag	output
1.	cv2.CV_LOAD_IMAGE_COLOR	BGR
0.	cv2.CV_LOAD_IMAGE_GRAYSCALE	GRAYSCALE
(-1)	cv2.CV_LOAD_IMAGE_UNCHANGED	Unmodified

Returns

RRtoolbox.lib.image.**myline**(*img*, *pt1*, *pt2*, *color*, *thickness=None*)

Funtion to draw points (experimental).

Parameters

- **img** –
- **pt1** –
- **pt2** –
- **color** –
- **thickness** –

Returns

RRtoolbox.lib.image.**np2qi**(*array*)

Convert numpy array to Qt Image.

source from: https://kogs-www.informatik.uni-hamburg.de/~meine/software/vigraqt/qimage2ndarray.py

Parameters **array** –

Returns

RRtoolbox.lib.image.**np2str**(*arr*)

RRtoolbox.lib.image.**plt2bgr**(*image*)

RRtoolbox.lib.image.**plt2bgra**(*image*)

RRtoolbox.lib.image.**qi2np**(*qimage*, *dtype='array'*)

Convert QImage to numpy.ndarray. The dtype defaults to uint8 for QImage.Format_Indexed8 or *bgra_dtype* (i.e. a record array) for 32bit color images. You can pass a different dtype to use, or 'array' to get a 3D uint8 array for color images.

source from: https://kogs-www.informatik.uni-hamburg.de/~meine/software/vigraqt/qimage2ndarray.py

RRtoolbox.lib.image.**quadrants**(*points*)

Separate points respect to center of gravity point.

Parameters **points** – list of points

Returns [[Top_left],[Top_right],[Bottom_left],[Bottom_right]]

RRtoolbox.lib.image.**random_color**(*channels=1*, *min=0*, *max=256*)

Random color.

Parameters

- **channels** – number of channels
- **min** – min color in any channel
- **max** – max color in any channel

Returns random color

RRtoolbox.lib.image.**rgb2qi**(*rgb*)

Convert the 3D numpy array *rgb* into a 32-bit QImage. *rgb* must have three dimensions with the vertical, horizontal and RGB image axes.

ATTENTION: This QImage carries an attribute *ndimage* with a reference to the underlying numpy array that holds the data. On Windows, the conversion into a QPixmap does not copy the data, so that you have to take care that the QImage does not get garbage collected (otherwise PyQt will throw away the wrapper, effectively freeing the underlying memory - boom!).

source from: https://kogs-www.informatik.uni-hamburg.de/~meine/software/vigraqt/qimage2ndarray.py

RRtoolbox.lib.image.**separe**(*values*, *sep*, *axis=0*)

Separate values from separator or threshold.

Parameters

- **values** – list of values
- **sep** – peparator value
- **axis** – axis in each value

:return:lists of greater values, list of lesser values

RRtoolbox.lib.image.**str2np**(*string*, *shape*)

RRtoolbox.lib.image.**transposeIm**(*im*)

RRtoolbox.lib.image.**try_loads**(*fns*, *func=<built-in function imread>*, *paths=None*, *debug=False*, *addpath=False*)

Try to load images from paths.

Parameters

- **fns** – list of file names
- **func** – loader function
- **paths** – paths to try. By default it loads working dir and test path
- **debug** – True to show debug messages

- **addpath** – add path as second argument

Returns image else None

RRtoolbox.lib.inspector module

This module is an all purpose intended for debugging, tracking, auto-documenting and self-introspecting the package

Made by Davtoh. Powered partially by pycallgraph. Dependent project: https://github.com/gak/pycallgraph/#python-call-graph

class RRtoolbox.lib.inspector.**Asyncronous**(*outputs*, *config*)
Bases: *RRtoolbox.lib.inspector.Syncronous*

done()

start()

tracer(*frame*, *event*, *arg*)

class RRtoolbox.lib.inspector.**GraphTrace**(*output=None*, *config=None*)
Bases: pycallgraph.pycallgraph.PyCallGraph

get_tracer_class()

saveSource(*file*)

source

class RRtoolbox.lib.inspector.**GraphTraceOutput**(*source=None*, *saveflag=True*, *label=''*, ***kwargs*)
Bases: pycallgraph.output.graphviz.GraphvizOutput

done()

save(*file=None*, *source=None*)

saveSource(*file*, *source=None*)

class RRtoolbox.lib.inspector.**Logger**(***kwargs*)
Bases: object

Logger for decorated functions. Holds important information of an instanced object and can be used with @trace decorator for traceback purposes.

Parameters

- **func** – object reference.
- **funcname** – object name.
- **inputs** – inputs pass to the object.
- **outputs** – outputs given by the object execution.
- **time** – initial time of execution.
- **exectime** – time of execution in seconds.
- **writer** – writer function where messages are passed.
- **eventHandle** – event function where object is passed when Logger.broadcast() is called.
- **msg_report** – message format to use in reports.
- **msg_no_executed** – massage format to pass to writer when object has not been executed and Logger.report() is called.

- **msg_executed** – massage format to use when object is executed and Logger.broadcast() is called.

Time_
returns formated time (str)

Type_
returns type name (str)

broadcast()
pass a notification message on object execution to the writer

eventHandle = None

file = <open file '<stdout>', mode 'w'>

renew()
renew Instance

report()
pass a report of the last executed object to the writer

throwError()
throw caught error :return:

tracer

writer(*sender*, **arg*)

class RRtoolbox.lib.inspector.**Syncronous**(*outputs*, *config*)
Bases: pycallgraph.tracer.SyncronousTracer

start()

stop()

RRtoolbox.lib.inspector.**funcData**(*func*)

RRtoolbox.lib.inspector.**load**(*mod_name*, *obj_name*)
Convert a string version of a class name to the object.

For example, get_class('sympy.core.Basic') will return class Basic located in module sympy.core

RRtoolbox.lib.inspector.**reloadFunc**(*func*)

RRtoolbox.lib.inspector.**tracer**(*instance*, *broadcast=True*, *report=True*)
Tracer for decorated functions.

Parameters

- **instance** – Logger instance
- **broadcast** –
- **report** –

Returns

RRtoolbox.lib.plotter module

This module holds the plotting and data-visualization tools. Motto: don't know how it is interpreted? i'll show you!

#Plotim example filename = "t2.jpg" win = "test" img = cv2.resize(cv2.imread(filename), (400, 400)) # (height, width) plot = Plotim(win,img) plot.show()

class RRtoolbox.lib.plotter.**Edger**(*img*, *isSIZE=True*, *isEQUA=False*, *isCLAHE=False*, *isBFILTER=False*)

Bases: *RRtoolbox.lib.plotter.Plotim*

Test visualization for edges

self.edge -> the edges in processed image self.img -> the processed image self.sample -> the rendered precessed image

computeAll()

computeEdge()

getParameters(*params=('d', 'sigmaColor', 'sigmaSpace', 'clipLimit', 'tileGridSize', 'isSIZE', 'isEQUA', 'isCLAHE', 'isBFILTER', 'th1', 'th2', 'size', 'apertureSize', 'L2gradient')*)

isActiveWindow()

isBFILTER

isCLAHE

isEQUA

isSIZE

load(*img*, *compute=True*)

maxth

onTrackbar1(**args*)

onTrackbar2(**args*)

save(*strname=None*, *ext='.png'*, *name='img'*)

showgray

size

th1

th2

windowfunc()

class RRtoolbox.lib.plotter.**Imtester**(*img*, *win='Imtester plot'*)

Bases: *RRtoolbox.lib.plotter.Plotim*

Used to test some concepts as thresholds and filters

static applythresh(*img*, *type*, *adaptativetoggle*, *threshtoggle*, *th*, *blocksz*, *c*, *i=''*, *ti=''*, *info=''*, *title=''*)

builtcmd()

computefunc(*image=None*)

detectType(*type*, *i=''*, *info=''*)

static formatinfo(*info*, *words=9*)

updatevisualization(*image*, *channel*, *th=None*, *items=None*, *thresh1=None*, *thresh2=None*)

visualize()

windowfunc()

class RRtoolbox.lib.plotter.**MatchExplorer**(*win*, *img1*, *img2*, *kp_pairs=()*, *status=None*, *H=None*, *show=True*, *block=True*, *daemon=True*)

Bases: *RRtoolbox.lib.plotter.Plotim*

Draws a set of keypoint pairs obtained on a match method of a descriptor on two images imgf and imgb. (backend: Plotim).

Parameters

- **win** – window's name (str)
- **img1** – image1 (numpy array)
- **img2** – image2 (numpy array)
- **kp_pairs** – zip(keypoint1, keypoint2)
- **status** – obtained from cv2.findHomography
- **H** – obtained from cv2.findHomography (default=None)
- **show** – if True shows Plotim using block and daemon, else do not show
- **block** – if True it wait for window close, else it detaches
- **daemon** – if True window closes if main thread ends, else windows must be closed to main thread to end

Returns Plotim object with visualization as self.rimg (image with matching result) (default=None)

Note: It supports BGR and gray images.

---

drawline()
Draws background visualization without interaction

drawrelation()
Draw keypoints where pointer is placed and pressed

keyfunc()

mousefunc()

static randomColor()

updaterenderer(*img=None*, *zoom=True*)
update renderer when called.

Parameters

- **img** – image to update in renderer, if None use self.img
- **zoom** – True to enable zoom, else updates with original img.

Returns None

class RRtoolbox.lib.plotter.**Plotim**(*win*, *im=array([[1]])*, *bgrcolor=(250*, *243*, *238)*)

Bases: object

Show and image with events, animations, controls, internal commands and highly customizable by code.

Parameters

- **win** – window name
- **im** – image of numpy array
- **bgrcolor** – default color to use for transparent or background color.

Warning: Plotim is deprecated and will be replaced in the future (it was made to test concepts). Originally it was made for windows but some functions were removed to let it be multi-platform.

builtincmd()
Internal cmd control

builtincontrol(*control=False*)
Internal control. use self.usecontrol = True to activate.

Parameters **control** – if True, use control key.

Returns

builtinplot(*pixel=None*, *useritems=None*, *flag=1*, *xpad=0*, *ypad=0*, *bgrcolor=None*, *alpha=None*)
Internal plot.

Parameters

- **pixel** – pixel color where mouse is placed (placed for better control). Color can be from real image, showed image, original image or rendered image, or any color.
- **useritems** – items to show.
- **flag** – flag for position (default=0).
  - flag==0 : foreground to left up.
  - flag==1 : foreground to left down.
  - flag==2 : foreground to right up.
  - flag==3 : foreground to right down.
  - flag==4 : foreground at center of background.
  - flag==5 : XY 0,0 is at center of background.
  - flag==6 : XY 0,0 is at center of foreground.
  - flag==7 : XY 0,0 is at right down of foreground.
- **xpad** – padding in x
- **ypad** – padding in y
- **bgrcolor** – background color
- **alpha** – alpha mask or value for transparency

Returns

builtinwindow()
loads windowfunc, showfunc, starts window thread and mousecallback.

clean()
Attempt to clean the plotter dictionary for an error in garbage collection. :return:

closefunc()
Decoupled close function for Plotim (replace self.closefunc).

Parameters **self** – Plotim instance

cmdfunc(*execute=False*)
command function and decoupled cmd solver for Plotim. (repalce self.cmdfunc)

Parameters

- **self** –
- **execute** – True, enable execution of commands, False, disable execution.

errorbackground

formatcmd(*cmd*, *references=('+', '-', '*', '=')*, *lmissing='self.'*)
Decoupled cmd formatter for cmdfunc and Plotim.

Parameters

- **self** – Plotim instance
- **cmd** – command
- **references** –
- **lmissing** – assumed missing part in command

Returns

help(*showAll=False*)
function to print the quick help for the user with all the commands

init()
Pseudo __init__. it is used to restart default values without destroying configurations.

keyfunc()
Decoupled key function for Plotim (replace self.keyfunc).

Parameters **self** – Plotim instance

makeoverlay(*items*, *xpad=0*, *ypad=0*, *bgrcolor=None*, *alpha=None*)
overlay items over image.

Parameters

- **self** – instance
- **items** – list of object to overlay
- **xpad** – pad in x
- **ypad** – pad in y
- **bgrcolor** – background color
- **alpha** – transparency color

Returns overlayed

mousefunc()
Decoupled mouse function for Plotim (replace self.mousefunc).

Parameters **self** – Plotim instance

static onmouse(*event*, *x*, *y*, *flags*, *self*)
Mouse event function for Plotim. (replace self.mousefunc)

Parameters

- **event** – mouse event
- **x** – x position
- **y** – y postion
- **flags** – mouse flag to use in control (it represents clicks)

- **self** – Plotim object

Returns

plotatpointer(*items*, *img=None*, *x=0*, *y=0*, *flag=6*, *xpad=0*, *ypad=0*, *bgrcolor=None*, *alpha=None*, *pixel=None*)

Plot message where mouse pointer is.

Parameters

- **items** – list of items supported by `self.makeoverlay()`
- **img** – image to place in items. If None it uses self.remg
- **x** – x position
- **y** – y position
- **flag** – flag for position (default=0).
  - flag==0 : foreground to left up.
  - flag==1 : foreground to left down.
  - flag==2 : foreground to right up.
  - flag==3 : foreground to right down.
  - flag==4 : foreground at center of background.
  - flag==5 : XY 0,0 is at center of background.
  - flag==6 : XY 0,0 is at center of foreground.
  - flag==7 : XY 0,0 is at right down of foreground.
- **xpad** – padding in x
- **ypad** – padding in y
- **bgrcolor** – background color
- **alpha** – alpha mask or value for transparency
- **pixel** – color to add as item im items,

Returns

plotatxy(*items*, *img=None*, *x=0*, *y=0*, *flag=0*, *xpad=0*, *ypad=0*, *bgrcolor=None*, *alpha=None*)

Plot message in xy position.

Parameters

- **items** – list of items supported by *makeoverlay()*
- **img** – image to place in items. If None it uses self.remg
- **x** – x position
- **y** – y position
- **flag** – flag for position (default=0).
  - flag==0 : foreground to left up.
  - flag==1 : foreground to left down.
  - flag==2 : foreground to right up.
  - flag==3 : foreground to right down.

- flag==4 : foreground at center of background.
- flag==5 : XY 0,0 is at center of background.
- flag==6 : XY 0,0 is at center of foreground.
- flag==7 : XY 0,0 is at right down of foreground.

- **xpad** – padding in x
- **ypad** – padding in y
- **bgrcolor** – background color
- **alpha** – alpha mask or value for transparency

Returns

plotintime(*items=None*, *wait=2*, *img=None*, *bgrcolor=None*)
plots messages and events.

Parameters

- **items** – list of items supported by *makeoverlay()*
- **wait** – time of message.
- **img** – image to place in items. If None it uses self.remg
- **bgrcolor** – color of message.

Returns

real2render(*x*, *y*, *astype=None*)
from real coordinates get rendered coordinates.

Parameters

- **x** – real x
- **y** – real y
- **astype** – (np.int32) return as the specified type

Returns rendered x, rendered y

render2real(*rx*, *ry*, *astype=<type 'numpy.int32'>*)
from rendered coordinates get real coordinates.

Parameters

- **rx** – rendered x
- **ry** – rendered y
- **astype** – (np.int32) return as the specified type

Returns real x, real y

rx1

rx2

ry1

ry2

save(*strname=None*, *ext='.png'*, *name='img'*)
Save image (save image if not Qt backend is installed) :param strname: name to save, a label with {win} can be used to be replaced with the plot win name :param ext: (".png") extension. :param name: ("img") name of image object from self. default is "img" that is self.img

(it allows better control to get custom image)

Returns True if saved, False if not saved (possibly because folder does not exists)

show(*frames=None*, *block=True*, *daemon=False*, *clean=True*)
Show function. calls buildinwindow, handles key presses and close events.

Parameters

- **frames** – show number of frames and close.
- **block** – if True it wait for window close, else it detaches (Experimental)
- **daemon** – if True window closes if main thread ends, else windows must be closed to main thread to end (Experimental)

Returns

showfunc(*img=None*)
Decoupled show function for Plotim (replace self.showfunc).

Parameters

- **self** – Plotim instance
- **img** – image to show

textbackground

updaterenderer(*img=None*, *zoom=True*)
update renderer when called.

Parameters

- **img** – image to update in renderer, if None use self.img
- **zoom** – True to enable zoom, else updates with original img.

Returns None

windowfunc()
Decoupled window function for Plotim (replace self.windowfunc).

Parameters **self** – Plotim instance

RRtoolbox.lib.plotter.**background**(*color*, *x=1*, *y=1*, *flag=0*)
Creates background rectangle.

Parameters

- **color** – main color.
- **x** – x pixels in axis x.
- **y** – y pixels in axis y.
- **flag** – Not implemented.

Returns image of shape y,x and ndim == color.ndim.

RRtoolbox.lib.plotter.**convert2bgr**(*src*, *bgrcolor=None*)
Tries to convert any image format to BGR.

Parameters

- **src** – source image.
- **bgrcolor** – background or transparent color.

Returns BGR array image.

RRtoolbox.lib.plotter.**convert2bgra**(*src*, *bgracolor=None*, *transparency=None*)
Tries to convert any image format to BGRA.

Parameters

- **src** – source image.
- **bgracolor** – background or transparent color.
- **transparency** – mask or A channel. (typically source image has not A channel, so user can provide it)

Returns BGRA array image.

RRtoolbox.lib.plotter.**echo**(*obj*)
Printer (used when user wants to print an object from Plotim) :param obj: object

RRtoolbox.lib.plotter.**fastplt**(*image*, *cmap=None*, *title='visualazor'*, *win=None*, *block=False*, *daemon=False*)
Fast plot.

Parameters

- **image** – image to show
- **cmap** – "gray" or None
- **title** – title of subplot
- **win** – title of window
- **block** – if True it wait for window close, else it detaches (Experimental)
- **daemon** – if True window closes if main thread ends, else windows must be closed to main thread to end (Experimental)

Returns plt

RRtoolbox.lib.plotter.**graph_filter**(*filters*, *levels=None*, *titles=None*, *win=None*, *single=True*, *legend=True*, *annotate=True*, *cols=3*, *scale=0.07*, *show=True*, *lxp=None*, *lyp=None*)
Graph filter with standard data to watch response.

Parameters

- **filters** – list of filters
- **levels** – numpy array with values. if None tries to fit data or assumes from 0 to 255
- **titles** – list of titles for each filter in filters. if None creates the titles
- **win** – window name
- **single** – True to plot all filters in one plot. else separate each filter in a plot.
- **legend** – True to add legends.
- **annotate** – True to add annotations.
- **cols** – number of columns to create plots

- **scale** – factor from maximum to draw annotations
- **show** – to show the figure

Returns figure

RRtoolbox.lib.plotter.**limitaxis**(*c*, *maxc*, *minc=0*)

Limit value in axis.

Parameters

- **c** – value
- **maxc** – max c value.
- **minc** – min c value.

Returns limited c value c E [minc,maxc]

RRtoolbox.lib.plotter.**plotPointsContour**(*pts*, *ax=None*, *lcor='k'*, *pcor=None*, *deg=None*, *annotate=True*, *width=0.004*, *label='pt{pt}({x}, {y}, {a})'*, *arrowprops=None*)

Plots points and joining lines in axes.

Parameters

- **pts** – points. [(x0,y0)...(xN,yN)]
- **ax** – axes handle to draw points.
- **lcor** – color of joining lines.
- **pcor** – color of points. If specified uses lines, else vectors.
- **deg** – angle of vertex, if True in degrees, if False in radians, if None do not add.
- **annotate** – (True) True to annotate
- **width** – adjust width of lines
- **label** – string to format point labels. add the point with {pt}, x and y coordinates with {x} and {y}, and angle with {a}. By default label is 'pt{pt}({x}, {y}, {a})'.
- **arrowprops** – dictionary to modify array properties

Returns axes

RRtoolbox.lib.root module

This module holds core-like methods for library modules but not for the hole package

class RRtoolbox.lib.root.**Controlstdout**(*disable=True*, *buffer=None*)

Bases: object

Context manager to control output to stdout

Parameters

- **disable** – if True suppress output.
- **buffer** – (None) if True creates a buffer to collect all data printed to the stdout which can be retrieved with self.buffered. A file can be given but if it is write-only it cannot retrieve data to self.buffered so "w+" is recommended to be used with self.buffered.

Warning: If a references to sys.stdout is kept before the Controlstdout instance then output can be printed trough it and cannot be controlled by the Controlstdout context.

class RRtoolbox.lib.root.**FactorConvert**(*factor=None*, *abbreviate=True*)
Bases: object

Keep track of factor and converts to any available factor.

convert(*factor*, *to=None*)
Convert from actual factor to another factor.

Parameters

- **factor** – number
- **to** – factor to convert

Returns converted value, units

convert2sample(*factor*, *to=None*)
Convert to resemble sample.

Parameters

- **factor** – number
- **to** – sample factor.

Returns converted value, units

exactFactorIndex(*key*)
Find the index of a factor that contains a key.

Parameters **key** – anything to look in factors (i.e. factor name, factor value, abbreviation).

Returns factor structure, else None.

factor

factors

getFactor(*key*)
Tries to find factor value in factors.

Parameters **key** – anything to look in factors (i.e. factor name, factor value, abbreviation). If key is a factor value it will look for the nearest factor value.

Returns factor structure, else raises error.

nearFactorIndex(*factor*)
Find the index of nearest factor value.

Parameters **factor** – factor value.

Returns factor structure near factor value.

static parts(*value*, *precision=4*)
Get number parts.

Parameters

- **value** – number
- **precision** – decimal precision

Returns ([... ,Hundreds, Tens, Ones],[Tenths, ...])

static split(*value*)
Get number fraction.

Parameters **value** – number

Returns integer, fraction

class RRtoolbox.lib.root.**Magnitude**(*value=0*, *factor=None*, *unit=None*, *precision=None*, *abbreviate=False*)
Bases: object

format_value(*value*)

class RRtoolbox.lib.root.**NameSpace**
Bases: object

used to store variables

exception RRtoolbox.lib.root.**NoParserFound**
Bases: exceptions.Exception

exception RRtoolbox.lib.root.**NotCallable**
Bases: exceptions.Exception

Defines objectGetter error: given object is not callable.

exception RRtoolbox.lib.root.**NotCreatable**
Bases: exceptions.Exception

Defines objectGetter error: objectGetter cannot create new object.

class RRtoolbox.lib.root.**Profiler**(*msg=None*, *tag=None*)
Bases: object

profiler for code points

param msg custom comment for profiling point

param tag classification tag

parameter space (" ")

parameter format_line ("{space}{tag}{msg}{time}")

parameter format_structure ("

{space}[{tag}{msg}{time}{child}]{side}")

parameter points profile instaces which are divided in "side" or "children" points according if they are side by side or are inside of the profiler.

close()
close profiler and all their points

formatter(*level*, *tag*, *msg*, *time*)
format profiling point arguments.

Parameters

- **level** –
- **tag** – classification tag
- **msg** – custom comment of profiling point
- **time** – time of profiling

Returns formatted (spacing, tag, msg, time)

lines_formatted(*collapse=None*)
generate string lines

Parameters **collapse** – list for collapsing repeated tags or messages.

Returns list of lines

lines_unformatted(*collapse=None*)
generate structure lines

Parameters **collapse** – list for collapsing repeated tags or messages.

Returns generator with outputs (level, tag, msg, time)

open_point(*msg=None*, *tag=None*)
Open a profiling point to track time.

Parameters

- **msg** – custom comment for profiling point
- **tag** – classification tag

Returns

restructure(*structure*, *collapse*)
reprocess an already created structure.

Parameters

- **structure** – structure.
- **collapse** – list for collapsing repeated tags or messages.

Returns reprocessed structure

string_lines()
string with plain structure of profiling

string_structured(*collapse=None*, *structure=None*)
string with plain structure of profiling

Parameters

- **collapse** – list for collapsing repeated tags or messages.
- **structure** – (None) uses and already created structure. If None it creates the structure.

Returns string

structure(*collapse=None*)
profiling structure.

Parameters **collapse** – list for collapsing repeated tags or messages.

Returns structure with format [tag,msg,time,children]

time

Returns overall time of profiling

class RRtoolbox.lib.root.**StdoutLOG**(*path*, *mode='w+'*, *chain=False*)
simple logger to save stdout output so anything printed in the console is logged to a file.

Parameters

- **path** – path to logging file

- **mode** – mode for opening the file.
- **chain** – if True closes previous logs and continues with new log

close(***kwargs*)

flush(***kwargs*)

printline(*text*, ***kwargs*)

printlines(*lines*, ***kwargs*)

write(*text*, ***kwargs*)

class RRtoolbox.lib.root.**StdoutMULTI**(*filelist*)
Enclose several file-like objects.

:param filelist = list of file-like objects

close(***kwargs*)

flush(***kwargs*)

printline(*text*, ***kwargs*)

printlines(*lines*, ***kwargs*)

write(*text*, ***kwargs*)

class RRtoolbox.lib.root.**StdoutSIM**(*disable=False*, *stdout=None*)
simple logger to simulate stdout output

close()

flush()

printline(*text*, ***kwargs*)

printlines(*lines*, ***kwargs*)

write(*text*, ***kwargs*)

class RRtoolbox.lib.root.**TimeCode**(*msg=None*, *factor=None*, *precision=None*, *abv=None*, *endmsg='{time}\n'*, *enableMsg=True*, *printfunc=None*, *profiler=None*, *profile_point=None*)
Bases: object

Context to profile code by printing a prelude and prologue with time.

Parameters

- **msg** – prelude or description message
- **factor** – factor supported by FactorConvert class
- **precision** – number of digits after a float point
- **abv** – if True prints "s", if False "seconds" for time
- **endmsg** – prologue message
- **enableMsg** – (True) A flag specifying if context should be printed or not.
- **printfunc** – function to print messages. By default it is sys.stdout.write

time

time_end

exception RRtoolbox.lib.root.**TimeOutException**
Bases: exceptions.Exception

exception RRtoolbox.lib.root.**TransferExeption**
Bases: exceptions.Exception

exception RRtoolbox.lib.root.**VariableNotDeletable**
Bases: exceptions.Exception

exception RRtoolbox.lib.root.**VariableNotSettable**
Bases: exceptions.Exception

RRtoolbox.lib.root.**addto**(*instance*, *funcname=None*)
Decorator: Add function as method to instance.

Parameters

- **instance** – class instance.
- **funcname** – name to register in instance.

Returns

RRtoolbox.lib.root.**decorateInstanceMethods**(*self*, *decorator*, *excludeMth='__init__'*, *includeMth=None*)
Decorate methods in an instance. It should be used in the __init__ method of a class.

Parameters

- **self** – class instance.
- **decorator** – decorator function to apply to self.
- **excludeMth** – list of methods to exclude.
- **includeMth** – list of methods to include if not in exclude. if excludeMth is None then decorateInstanceMethods checks for includeMth list. if includeMth and excludeMth is None then all methods of self are decorated.

Returns self

Note: It must be used at instance initialization (i.e. inside __init__ method)

RRtoolbox.lib.root.**ensureList**(*obj*)
ensures that object is list

RRtoolbox.lib.root.**formatConsume**(*format_string*, *kwargs*, *formatter=None*, *handle=None*)
Format with dictionary and consume keys.

Parameters

- **format_string** – string to format
- **kwargs** – dictionary containing the keys and values to format string. The keys must be supported by the string formatter
- **formatter** – (None) formatter function to format string

Returns formatted string

RRtoolbox.lib.root.**formatOnly**(*format_string*, ***kwargs*)
Format string only with provided keys

Parameters

- **format_string** – string to format
- **kwargs** – format keys

Returns formatted string

RRtoolbox.lib.root.**glob**(*path*, *contents='*'*, *check=<function isfile>*)
Return a list of paths matching a pathname pattern with valid files.

Parameters

- **path** – path to process ing glob filter
- **contents** – If path is a folder then looks for contents using
- **check** – function to filter contents. it must receive the path and return True to let it pass and False to suppress it.

Returns return list of files

class RRtoolbox.lib.root.**globFilter**(*include=None*, *exclude=None*, *case=False*)
Bases: object

glob filter for patterns

RRtoolbox.lib.root.**lookinglob**(*pattern*, *path=None*, *ext=None*, *forward=None*, *filelist=None*, *aslist=False*, *raiseErr=False*)
Look for patterns in Path. It looks as {if path}{if pattern}{if forward}{if ext}.

Parameters

- **pattern** – string to look for pattern.
- **path** – (None) path to look for pattern
- **ext** – (None) extension of pattern in path
- **forward** – (None) look changes after pattern and before ext parameter.
- **filelist** – (None) simulates the files in path and look patterns in this list.
- **aslist** – (False) if False it returns the first match case string else the list of matching cases.
- **raiseErr** – If true raises Exception if patter is not found in path or there are more than one match

Returns matched case if returnAll is False else the list of matched cases or if no match is found None

RRtoolbox.lib.serverServices module

class RRtoolbox.lib.serverServices.**Conection**(*conn*)
represent a connection to interchange objects between servers and clients.

getLen(*timeout=None*)

rcv()

recvall()

send(*obj*)

sendLen(*length*, *timeout=None*)

RRtoolbox.lib.serverServices.**generateServer**(*host='localhost'*, *to=63342*)
generates a simple Server in available address.

Parameters **to** – until port.

Returns socket, address

RRtoolbox.lib.serverServices.**initClient**(*addr*, *timeout=None*)
Inits a simple client from address. :param addr: (host, port) :return: socket

RRtoolbox.lib.serverServices.**initServer**(*addr*)
Inits a simple server from address.

Parameters **addr** – (host, port)

Returns socket

RRtoolbox.lib.serverServices.**parseString**(*string*, *timeout=3*)

Parameters

- **string** –
- **timeout** –

Returns

RRtoolbox.lib.serverServices.**ping**(*host*, *port*)
Ping to.

Parameters

- **host** – IP address
- **port** – port address

Returns

RRtoolbox.lib.serverServices.**rcvPickle**(*addr=('localhost', 50007)*, *timeout=None*)
Receive potentially any data using sockets.

Parameters

- **addr** – socket or address.
- **timeout** – NotImplemented

Returns data, else throws error.

RRtoolbox.lib.serverServices.**recv_into**(*viewable*, *socket*)
Receive from socket into viewable object.

Parameters

- **viewable** – viewable object
- **socket** – source socket

Returns None

RRtoolbox.lib.serverServices.**scan_ports**(*host*)
Scan opened ports in address.

Parameters **host** – host IP to filter opened ports.

Returns generator

RRtoolbox.lib.serverServices.**sendPickle**(*obj*, *addr=('localhost', 50007)*, *timeout=None*, *threaded=False*)

Send potentially any data using sockets.

Parameters

- **obj** – packable object.
- **addr** – socket or address.
- **timeout** – NotImplemented

Returns True if sent successfully, else Throw error.

RRtoolbox.lib.serverServices.**send_from**(*viewable*, *socket*)

Send from viewable object.

Parameters

- **viewable** – viewable object
- **socket** – destine socket

Returns None

RRtoolbox.lib.serverServices.**string_is_socket_address**(*string*)

RRtoolbox.lib.session module

This module have serializing methods for data persistence so to let the package "save" custom objects

session module made by Davtoh and powered by dill Dependency project: https://github.com/uqfoundation/dill

RRtoolbox.lib.session.**checkFromSession**(*filepath*, *varlist*)

Check that variables exits in session file.

Parameters

- **filepath** – path to session file.
- **varlist** – list of variables to checkLoaded.

Returns list checkLoaded results

RRtoolbox.lib.session.**deleteFromSession**(*filepath*, *varlist*)

Delete variables from session file.

Parameters

- **filepath** – path to session file.
- **varlist** – list of variables to delete.

Returns None

RRtoolbox.lib.session.**flushSession**(*filepath*)

Empty session in file.

Parameters **filepath** – path to session file.

Returns

RRtoolbox.lib.session.**getEnviromentSession**(*enviroment=None*)

Gets the filtered session from the global variables.

Returns dictionary containing filtered session.

RRtoolbox.lib.session.**readSession**(*filepath*, *helper=None*)
Loads a dictionary session from file.

Parameters

- **filepath** – path to load session file.
- **helper** – function to pos-process session file

Returns session

RRtoolbox.lib.session.**saveSession**(*filepath*, *session*, *helper=None*)
Saves dictionary session to file.

Parameters

- **filepath** – path to save session file.
- **session** – dictionary
- **helper** – function to pre-process session

Returns filename of saved session

RRtoolbox.lib.session.**updateSession**(*filepath*, *session*, *replace=True*, *rdhelper=None*, *svhelper=None*)
Updates a dictionary session in file.

Parameters

- **filepath** – path to session file.
- **session** – dictionary.
- **replace** – if True key values are replaced else old key values ar kept.
- **rdhelper** – read helper.
- **svhelper** – save helper.

Returns None

Module contents

This module contains core-like, too-much-used and too-much-referenced modules

1.1.2 RRtoolbox.tools package

Submodules

RRtoolbox.tools.lens module

RRtoolbox.tools.lens.**drawCircle**(*array*, *cnt*, *color=0*)
project circle over array.

Parameters

- **array** – array to draw circle
- **cnt** – contours of segmentation to fit circle
- **color** – color of lens

Returns array

RRtoolbox.tools.lens.**drawEllipse**(*array*, *cnt*, *color=0*)
project ellipse over array.

Parameters

- **array** – array to draw ellipse
- **cnt** – contours of segmentation to fit ellipse
- **color** – color of lens

Returns array

RRtoolbox.tools.lens.**fitLens**(*img*, *mask*, *color=0*, *asEllipse=False*, *addmask=False*)
Place lens-like object in image.

Parameters

- **img** – image to place lens
- **mask** – mask to fit lens
- **color** – color of the lens
- **asEllipse** – True to fit lens as a ellipse, False to fit circle.
- **addmask** – return additional mask parameter

Returns image with simulated lens

RRtoolbox.tools.lens.**simulateLens**(*img*, *threshfunc=None*, *pshape=(300, 300)*, *color=0*, *asEllipse=True*)
Place lens-like object in image.

Parameters

- **img** – image to place lens.
- **threshfunc** – function to segment retinal area and get its mask.
- **pshape** – shape to resize processing image to increase performance.
- **color** – color of the lens.
- **asEllipse** – True to fit lens as a ellipse, False to fit circle.

Returns image with simulated lens.

RRtoolbox.tools.segmentation module

RRtoolbox.tools.segmentation.**find_optic_disc_watershed**(*img*, *P*)
Find optic disk in image using a watershed method.

Parameters

- **img** – BGR image
- **P** – gray image

Returns optic_disc, Crs, markers, watershed

RRtoolbox.tools.segmentation.**get_beta_params_Otsu**(*P*)
Automatically find parameters for alpha masks using Otsu threshold value.

Parameters **P** – gray image

Returns beta1 for minimum histogram value, beta2 for Otsu value

RRtoolbox.tools.segmentation.**get_beta_params_hist**(*P*)
Automatically find parameters for bright alpha masks using a histogram analysis method.

Parameters **P** – gray image

Returns beta1 for minimum valley left of body, beta2 for brightest valley right of body where the body starts at the tallest peak in the histogram.

RRtoolbox.tools.segmentation.**get_bright_alpha**(*backgray*, *foregray*, *window=None*)
Get alpha transparency for merging foreground to background gray image according to brightness.

Parameters

- **backgray** – background image. (as float)
- **foregray** – foreground image. (as float)
- **window** – window used to customizing alfa. It can be a binary or alpha mask, values go from 0 for transparency to any value where the maximum is visible i.e a window with all the same values does nothing. A binary mask can be used, where 0 is transparent and 1 is visible. If not window is given alfa is not altered and the intended alpha is returned.

Returns alfa mask

RRtoolbox.tools.segmentation.**get_layered_alpha**(*back*, *fore*)
Get bright alpha mask (using Otsu method)

Parameters

- **back** – BGR background image
- **fore** – BGR foreground image

Returns alpha mask

RRtoolbox.tools.segmentation.**layeredfloods**(*img*, *gray=None*, *backmask=None*, *step=1*, *connectivity=4*, *weight=False*)
Create an alpha mask from an image using a weighted layered flooding algorithm,

Parameters

- **img** – BGR image
- **gray** – Gray image
- **backmask** – background mask
- **step** – step to increase upDiff in the floodFill algorithm. If weight is True step also increases the weight of the layers.
- **connectivity** – pixel connectivity of 4 or 8 to use in the floodFill algorithm
- **weight** – Increase progressively the weight of the layers using the step parameter.

Returns alpha mask

RRtoolbox.tools.segmentation.**retina_markers_thresh**(*P*)
Retinal markers thresholds to find background, retinal area and optic disc with flares based in the histogram.

Parameters **P** – gray image

Returns min,b1,b2,max

RRtoolbox.tools.segmentation.**retinal_mask**(*img*, *biggest=False*, *addalpha=False*)
Obtain the mask of the retinal area in an image. For a simpler and lightweight algorithm see *retinal_mask_watershed()*.

Parameters

- **img** – BGR or gray image
- **biggest** – True to return only biggest object
- **addalpha** – True to add additional alpha mask parameter

Returns

if addalpha: binary mask, alpha mask

else: binary mask

RRtoolbox.tools.segmentation.**retinal_mask_watershed**(*img*, *parameters=(10*, *30*, *None)*, *addMarkers=False*)

Quick and simple watershed method to obtain the mask of the retinal area in an image. For a more robust algorithm see *retinal_mask()*.

Parameters

- **img** – BGR or gray image
- **parameters** – tuple of parameters to pass to `filterFactory()`
- **addMarkers** – True to add additional Marker mask. It contains 0 for unknown areas, 1 for background and 2 for retinal area.

Returns

if addMarkers: binary mask, Markers mask

else: binary mask

RRtoolbox.tools.selectors module

class RRtoolbox.tools.selectors.**EntropyPlot**(*images*, *win='Entropy tests'*, *func=None*)

Bases: *RRtoolbox.lib.plotter.Plotim*

Plot entropy test

getData(*im*)

getImage(*im*)

keyfunc()

nextim()

previousim()

selectlist(*imlist*)

RRtoolbox.tools.selectors.**entropy**(*imlist*, *loadfunc=None*, *invert=False*)

Entropy function modified from:

Yan Liu, Feihong Yu, An automatic image fusion algorithm for unregistered multiply multi-focus images, Optics Communications, Volume 341, 15 April 2015, Pages 101-113, ISSN 0030-4018, http://dx.doi.org/10.1016/j.optcom.2014.12.015. (http://www.sciencedirect.com/science/article/pii/S0030401814011559)

Parameters **imlist** – list of path to images or arrays

Returns sortedD,sortedImlist,D,fns

where sortedD is the ranking of the Entropy test, D = [D0,...,DN] D0>DN sortedImlist is fns sorted to match sortedD, D is the list of the absolute difference between entropy and the root mean square, D = **|E-RMS|**

RRtoolbox.tools.selectors.**hist_comp**(*imlist*, *loadfunc=None*, *method='correlation'*)
Histogram comparison

Parameters **imlist** – list of path to images or arrays

Returns comparison

RRtoolbox.tools.sticher module

RRtoolbox.tools.sticher.**stich**(***opts*)

Module contents

1.2 Submodules

1.3 RRtoolbox.core module

RRtoolbox.core.**f**(**args*, ***kwargs*)

class RRtoolbox.core.**rrbox**(**args*)
Bases: object

asift(*fn*)

RRtoolbox.core.**tools**(*instance*, *modules*)

RRtoolbox.core.**tools2**(*instance*, *modules*)

1.4 RRtoolbox.run module

1.5 RRtoolbox.shell module

class RRtoolbox.shell.**Shell**

generateParser(*func*)

getParser(*func*)

parse(*func*, *args=None*, *namespace=None*)

parser_fastplt()

parser_loadFunc()

RRtoolbox.shell.**getDocParamLines**(*doc*)
gets each parameter line from reStructured doc.

Parameters **doc** – documentation

Returns lines

RRtoolbox.shell.**getDocParameters**(*doc*)
gets param and comment from reStructured doc.

Parameters **doc** – documentation

Returns list of (param,comment) items.

RRtoolbox.shell.**shell_processor**(*commands*)

RRtoolbox.shell.**shell_processor_parser**(*syslist*, *flags=''*, *longopts=('feature=', 'nnn=')*)

RRtoolbox.shell.**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

RRtoolbox.shell.**tuple_creator**(*string*)
Process string to get tuple.

Parameters **string** – string parameters with "," (colon) as separator Ex: param1, param2, ..., paramN

Returns tuple

1.6 Module contents

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

r

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W