TOOLBOX OF IMAGE PROCESSING USING THE PYTHON LANGUAGE

Alexandre G. Silva[1], *Roberto de A. Lotufo*[1], *Rubens C. Machado*[2], *André V. Saúde*[1,2]

[1]DCA–FEEC, University of Campinas, P. O. Box 6101, 13083-970, Campinas, SP, Brazil
{alexgs,lotufo,andrevit}@dca.fee.unicamp.br
[2]Renato Archer Research Center, P. O. Box 6162, 13089-120, Campinas, SP, Brazil
rubens.machado@cenpra.gov.br

ABSTRACT

This work consists in the study, development and implementation of a toolbox for image processing using the *Python* language and the *Numerical Python* package. This set has "open source" distribution and is adequate for multidimensional mathematical processing. Python is a modern and well projected language, interpreted, "very-high-level", object oriented and extremely portable, apart from being suitable for rapid application development (RAD). The system is generated using the methodology of the *Adesso* project for construction of scientific software. This environment is useful in education, research and development of final applications.

1. INTRODUCTION

A generic computational tool for image processing usually offers two levels of programming: a script interpreted language serving as interface to a high performance library normally written in *C* or *Fortran*. There are several appropriate tools for image processing nowadays, considering easiness of use and performance.

Our intention is to congregate the positive features of platforms such as *MATLAB* [3] and on-line image processing courses such as the Digital Image Processing (DIP) using *Khoros* [8, 4] into a freely available open system. In this sense, there is the effort associated to development of the *Adesso* project [7] that consists in an environment of authoring scientific software. Adesso is strongly based on *XML* [10] technologies and has support for automatic generation of documentation and code interface of computational libraries. Its current stage of development, it support *MATLAB* and *Tcl/Tk* [1] platforms. This work has also extended Adesso to provide support for *Python* [6].

Recently, the *Python* language is having a great growth, specially in the academic community. It is a sufficiently modern scripting language, with refined concepts of object orientation. In addition, there is the *Numerical* package [2] that presents a set of modules which can be easily incorporated to *Python*. These modules were projected under the strong influence of the basic functions of *MATLAB*. *Python*, as well as the *Numerical* package, simplify the programming in two levels through a well-defined modular form. This association guarantees efficient multidimensional matrix computation that is essential for image processing. In this way, the use of *Python*, associated to this numeric package, is becoming a very attractive solution for image processing.

The following sections introduce the adopted programming model, the tools used in this work, and explain the development of the image processing toolbox. Finally, the contents of the toolbox is commented.

This work was supported by FAPESP, Brazil, process 00/13671-0.

2. PROGRAMMING MODEL

Prechelt [5] proposed a study to compare the performance of several programming languages. While *scripting* (or interpreted) languages have simplified programming that allows rapid prototyping, compiled languages generate faster codes but demand considerably more developing time. In this section, we want to illustrate that an efficient multidimensional image processing program can be written in *scripting* languages, using the matrix support of Python and MATLAB.

Let us consider an example to compute the MSE (Mean Square Error) between two color images. The adopted equation is

$$MSE = \frac{1}{N_h N_w} \sum_{i=0}^{N_h-1} \sum_{j=0}^{N_w-1} \sum_{k=0}^{2} \left[(f_1(i,j,k) - f_2(i,j,k))^2 \right]$$

where f_1 and f_2 are the input images, in the RGB model, with dimensions N_hxN_wx3.

We define two forms of implementation of this equation according to the way of the pixels are scanned: ***(i)*** *explicit* where loops are used; ***(ii)*** *implicit* where the loops are intrinsic from the matrix operators. Figure 1 illustrates the algorithms in question. Table 1 shows the results of the

```
Input: images f1 e f2 (same dimensions)
Output: MSE value
```

Explicit scan:

```
MSE = 0
For all i between 0 and height(f1)−1 do:
    For all j between 0 and width(f1)−1 do:
        For all k between 0 and 2 do:
            MSE = MSE + (f1[i,j,k] − f2[i,j,k])^2
MSE = MSE / (height(f1)*width(f1))
```

Implicit scan:

```
MSE = (ravel(f1) − ravel(f2))^2

MSE = sum(MSE) / (height(f1)*width(f1))
```

Fig. 1. Algorithms of explicit and implicit scan for the calculation of the MSE.

	Time		**Time**
MATLAB 6 (*implicit*)	$0,038s$	**Java** (*explicit*)	$0,198s$
C (*explicit*)	$0,051s$	**Python** (*explicit*)	$4,473s$
Python (*implicit*)	$0,136s$	**MATLAB 6** (*explicit*)	$14,628s$

Table 1. Comparison of average times (Sun Ultra 60) for the calculation of the MSE between two $256x256$ colorful images.

implementation of the MSE in different languages in terms of speed performance. Note that when an program can be expressed in matrix computation, these languages can provide a comparable computational efficiency (or better) when compared with a standard C program without optimization. This is the case when we compare the speed of the MATLAB[1] implicit code solution with the non-optimized C code.

As an initial exercise in image processing class, we suggest to implement the synthetic generation of simple 2-D images such as to create a chessboard like arrangement of pixels. The idea is to explore the many ways one can code the exercises using implicit matrix operations. Normally five or more conceptually different solutions are possible. One of the most general approaches is to create two matrices of x and y indices and compute the image pixels values as a mathematical expression of the indices. To create the chessboard image, the following equation can be used:

$$z(x,y) = (x(i,j) + y(i,j)) \,\%\, 2$$

where $x(i,j) = i+1$ e $y(i,j) = j+1$, for $\forall i \in [0, N-1]$ e $\forall j \in [0, M-1]$. Figure 2 illustrates this operation. In this case, the operators "$+$" (sum) and "$\%$" (remaining portion of the integer division) must belong to the language and perform the calculation pixel by pixel. Matrices x and y can be efficiently constructed in Python[2], and in MATLAB[3]. The same methodology can be used to synthesize other images such as ramps, ellipses, sinoidal, among others examples.

[1]New MATLAB version 6.5 uses JIT compiler and is better in explicit processing. These times had probably changed.

[2]`Numeric.indices`

[3]`meshgrid`

$$\left(\underbrace{\boxed{\begin{matrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ N & \cdots & N \end{matrix}}}_{x_{NxM}} + \underbrace{\boxed{\begin{matrix} 1 & \cdots & M \\ \vdots & \ddots & \vdots \\ 1 & \cdots & M \end{matrix}}}_{y_{NxM}} \right) \,\%\, 2 = \underbrace{\boxed{\begin{matrix} 0 & 1 & \cdots \\ 1 & 0 & \cdots \\ \vdots & \vdots & \ddots \end{matrix}}}_{z_{NxM}}$$

Fig. 2. Example of generation of a chessboard image using matrix manipulations.

3. DEVELOPMENT ENVIRONMENT

3.1. Python language

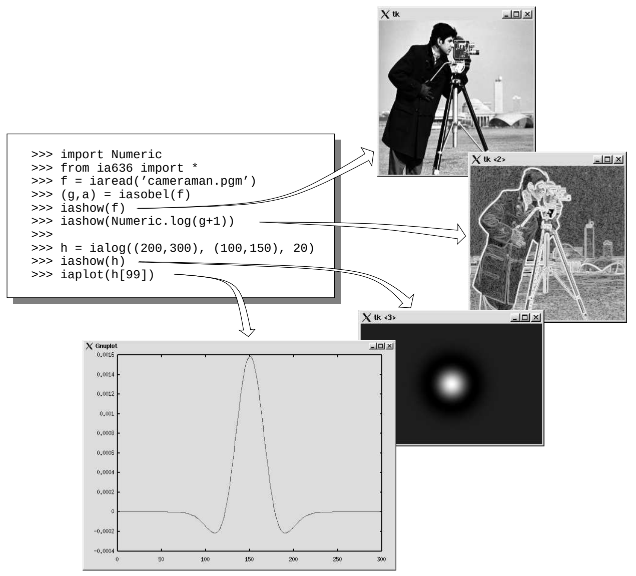
Python associated with the Numerical package is a generic language, extremely portable and efficient enough for image processing. Python has the flexibility of Perl, associated with the numerical power and ease of use of MATLAB, but available as an open source environment. The source code is generally small when compared to compiled languages by several reasons: high-level data types and operations, no type declarations (dynamic typing), automatic memory management, and command blocks marked by indentation. In C/C++, equivalent data structures and functionalities with the same optimization would cost considerable more programming time. There is also a great native set of libraries implemented in C/C++ (built-in), for Python, that practically discard the process of compilation / correction / re-compilation (except for API extensions of the language). These characteristics generate a high productivity gain.

The images are displayed using the *Tkinter* module, a default Python graphical user interface (GUI) with the functionalities of the package *Tk* [1], but there are many other options to choose from, such as *wxPython*, *PyGTK*, *PyQt*, ...). For graph plotting, we choose *Gnuplot* [9]. Figure 3 shows a script with its output displays.

3.2. Adesso system

The Python image processing toolbox is built using the Adesso authoring system [7], that is an environment to design scientific components. The essential information of a toolbox is stored in a XML database. Code, documentation and packaging are generated automatically using the Adesso style sheet processing. The Adesso methodology allows easiness of maintenance and management of the information, consistency in the user interface, flexibility in the presentation based on designed style sheets, greater immunity to errors, among other characteristics. Figure 4 shows a block diagram of the Adesso automatic transformation process.

The XML files of the Adesso system form a database containing algorithms and their descriptions, input and output parameters and their descriptions, examples, equations (in LaTeX notation), tests, function dependency list, among



```
>>> import Numeric
>>> from ia636 import *
>>> f = iaread('cameraman.pgm')
>>> (g,a) = iasobel(f)
>>> iashow(f)
>>> iashow(Numeric.log(g+1))
>>>
>>> h = ialog((200,300), (100,150), 20)
>>> iashow(h)
>>> iaplot(h[99])
```

Fig. 3. Typical script to show images and plotting graphics.

other fields. The following style sheets were implemented to expand the Adesso capabilities to deal with the Python language:

Code generation. This style sheet produces three main outputs: *functions* with on-line documentation; *demonstrations* and *lessons* used to illustrate several numeric or graphical outputs, step by step; and *test suites* that are examples that check if the functions are implemented correctly.

Wrapper generation. This style sheet generates the interface code between Python and libraries in C/C++ language. This makes the C/C++ libraries already in Adesso available in Python. This is one of the most benefits of using Adesso in the development of toolboxes. This automatic code generation avoid the common use of copy and paste to create such interfaces manually.

Setup generation. This style sheet generates the ”setup.py” file with compilation and installation rules.

Documentation generation. This style sheet creates the documentation and the illustrations, such as numerical outputs, images, graphics, and equations. These illustrations are actually generated during the document creation by invoking Python to execute the script examples. This feature guarantees that the illustrations in the documentation reflects the toolbox version. This is the ”wysiwyg” equivalent for software documentation. An example of a typical HTML function documentation is shown in Figure 5.

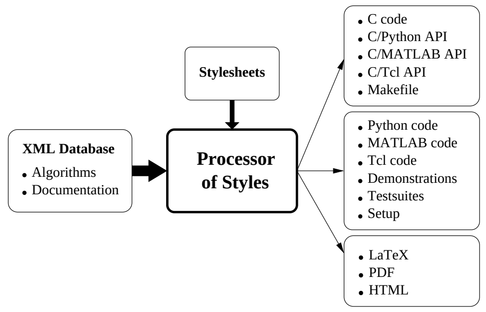


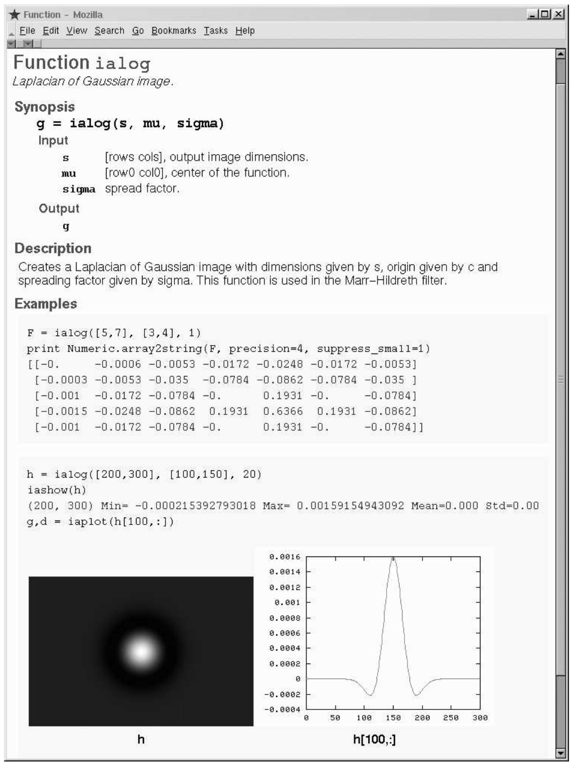
Fig. 4. Block diagram of the Adesso information transformation.

4. RESULTS

The image processing toolbox is called `ia636` as a reference to the code of the computer vision graduate course taught at the Faculty of Electrical and Computer Engineering, UNICAMP. The last version of the toolbox can be seen at `http://marahu.dca.fee.unicamp.br/course/ia636.html`. Its current stage of development contains the following:

Lessons have been implemented to: ***(i)*** illustrate the generation of different images; ***(ii)*** illustrate the contrast transform function; ***(iii)*** illustrate histogram equalization; ***(iv)*** illustrate the template matching technique; ***(v)*** illustrate the decomposition of the image in primitive 2-D waves; ***(vi)*** compute the Discrete Fourier Transfor spectrum of simple synthetic images; ***(vii)*** compute the kernel matrix for the DFT; ***(viii)*** illustrate the scale property of the DFT; ***(ix)*** illustrate the convolution theorem; ***(x)*** illustrate the Hotelling Transform; ***(xi)*** illustrate the inverse filtering for restoration; ***(xii)*** illustrate the interpolation of magnified images; ***(xiii)*** illustrate the Otsu automatic thresholding selection method.

Several *functions* divided in the following topics have been implemented: ***(i)*** *image creation*: low-pass Butterworth frequency filter; circle; impulses; cossenoidal 2-D; Gaussian 2-D; Laplacian of Gaussian; vertical bands of increasing gray values; rectangle; ***(ii)*** *image information and manipulation*: image cropping to find the minimum rectangle; linear index conversion from double subscripts and vice-versa; 2-D matrices index creation; image negation; normalization of the pixel values between the specified range; insertion of a frame around an image; cutting a rectangle out of an image; image replication until reaching a new size; ***(iii)*** *image I/O*: image reading and writing; ***(iv)*** *contrast manipulation*: intensity image transform; color map; ***(v)*** *color processing*: RGB to HSV color model conversion and vice-versa; true color RGB to index and color map conversion; RGB to YCbCr color model conversion and vice-versa; ***(vi)*** *geometric manipulations*: affine transform; 2-D rigid body geometric transformation and scaling; image resizing; periodic translation; ***(vii)*** *image transformation*: Discrete Cos-



Function ialog

Laplacian of Gaussian image.

Synopsis

```
g = ialog(s, mu, sigma)
```

Input

s	[rows cols], output image dimensions.
mu	[row0 col0], center of the function.
sigma	spread factor.

Output

g

Description

Creates a Laplacian of Gaussian image with dimensions given by s, origin given by c and spreading factor given by sigma. This function is used in the Marr–Hildreth filter.

Examples

```
F = ialog([5,7], [3,4], 1)
print Numeric.array2string(F, precision=4, suppress_small=1)
[[-0.     -0.0006 -0.0053 -0.0172 -0.0248 -0.0172 -0.0053]
 [-0.0003 -0.0053 -0.035  -0.0784 -0.0862 -0.0784 -0.035 ]
 [-0.001  -0.0172 -0.0784 -0.      0.1931 -0.     -0.0784]
 [-0.0015 -0.0248 -0.0862  0.1931  0.6366  0.1931 -0.0862]
 [-0.001  -0.0172 -0.0784 -0.      0.1931 -0.     -0.0784]]
```

```
h = ialog([200,300], [100,150], 20)
iashow(h)
(200, 300) Min= -0.000215392793018 Max= 0.00159154943092 Mean=0.000 Std=0.00
g,d = iaplot(h[100,:])
```

Fig. 5. HTML documentation generated automatically.

sine Transform; Discrete Fourier Transform; Haar Wavelet Transform; Hadamard Transform; and respective inverses; shifting zero-frequency component to the center of spectrum; conjugate symmetry checking; ***(viii)*** *image filtering*: Laplacian of Gaussian filter; contours of binary images; 2-D convolution and periodic convolution; Sobel edge detection; variance filter; ***(ix)*** *automatic thresholding techniques*: thresholding by Otsu; ***(x)*** *measurements*: color and gray-scale image histograms; labeling; reconstruction of a connect component; MSE, PSNR and Pearson correlation between two images; ***(xi)*** *halftoning approximation*: ordered Dither; Floyd-Steinberd error diffusion; ***(xii)*** *visualization*: optical Fourier Spectrum display from DFT data; isolines of a gray-scale image; display labeled images by assigning a random color for each label; plotting a function or a surface; image display.

5. CONCLUSIONS

This paper describes the development of an image processing toolbox for Python using the Numeric package. The toolbox was developed using Adesso, an authoring system that helps the creation of such toolboxes. For such, Adesso was extended to support the Python language in the addition to MATLAB and Tcl-Tk. The toolbox is intended to be used as a practical resource in image processing courses. The Python toolbox is compatible to the equivalent MATLAB ”ia636” image processing toolbox being used and developed.

6. REFERENCES

[1] B. B. Welch. *Pratical Programming in Tcl and Tk*. Prentice Hall PTR, 3rd edition, 1999.

[2] D. Ascher, P. F. Dubois, K. Hinsen, J. Hugunin and T. Oliphant. *Numerical Python*. September 2001.

[3] S. L. Eddins and M. T. Orchard. Using MATLAB and C in an Image Processing Lab Course. In *Proceedings of ICIP-94*, pages 515–519, Austin, USA, November 1994.

[4] R. Jordan and R. A. Lotufo. Interactive Digital Imagem Processing Course on the World-Wide Web. In *Proceedings of the 1996 International Conference on Image Processing*, pages 433–436, Lausanne, Switzerland, September 1996. IEEE Signal Processing Society.

[5] L. Prechelt. An Empirical Comparison of Seven Programming Languages. *Computing Pratices IEEE*, pages 23–29, October 2000.

[6] M. Lutz and D. Ascher. *Learning Python*. O’Reilly & Associates, April 1998.

[7] R. C. Machado. Adesso: Ambiente para Desenvolvimento de Software Cientı́fico. Dissertação de Mestrado, Faculdade de Engenharia Elétrica e de Computação - Universidade Estadual de Campinas, Junho 2002.

[8] J. Rasure, R. Jordán, and R. Lotufo. Teaching Image Processing with KHOROS. In *IEEE International Conference on Image Processing*, pages 13–16, 1994. (kieee94.ps.gz).

[9] T. Williams and C. Kelley. *gnuplot, An Interactive Plotting Program*. December 1998.

[10] W3C Recommendation 6 October 2000. Extensible Markup Language (XML) 1.0. 2000.