Python for Prototyping Computer Vision Applications

Brian Thorne
HitLabNZ
University of Canterbury
Private Bag 4800, Christchurch
brian.thorne@hitlabnz.org

Raphaël Grasset
HitLabNZ
University of Canterbury
Private Bag 4800, Christchurch
raphael.grasset@hitlabnz.org

ABSTRACT
Python is a popular language widely adopted by the scientific community due to its clear syntax and an extensive number of specialized packages. For image processing or computer vision development, two libraries are prominently used: *NumPy/SciPy* and *OpenCV* with a Python wrapper. In this paper, we present a comparative evaluation of both libraries, assessing their performance and their usability. We also investigate the performance of OpenCV when accessed through a python wrapper versus directly using the native C implementation.

Categories and Subject Descriptors
I.4.9 [**Image Processing and Computer Vision**]: [Performance]; I.4.9 [**Image Processing and Computer Vision**]: Applications—*Computer Vision*; I.5.4 [**Pattern Recognition**]: General

General Terms
Computer Vision, Human Computer Interaction

Keywords
Python, SciPy, OpenCV

1. INTRODUCTION
Python[1] has been of growing interest to the academic community over the last decade, especially in the area of computational science. The simple syntax of Python, high level dynamic data types, and automated memory management has captured the research communities attention and forged it as a popular tool.

The field of image processing and computer vision (CV) has been driven for the last decade by development in C/C++ and the usage of MATLAB® software [2]. Although MATLAB® offers an efficient high level platform for prototyping and testing algorithms, its performance doesn't compete with a well designed and optimized C/C++ implementation[3]. Recently, potential and valuable solutions have emerged for developing image processing and computer vision algorithms in Python.

This paper evaluates the performances and usability of the most common methods for developing CV algorithms and CV applications in Python. Indeed, we aim to offer a comprehensive overview of the advantages/disadvantages of using Python for CV development is given for the benefit of any interested researcher in the field.

We have focused our attention on the performances of two widely used open source libraries in Python: The computer vision specific *OpenCV* [4] and the two closely related scientific packages *NumPy/SciPy*[5].

For this matter, we analyzed their performances through a list of common tasks and processes regularly employed in computer vision (e.g. video capture, filtering algorithms, feature detection, etc). We were particularly interested to learn how Python performs in comparison with the native C implementation of OpenCV considering the low level calling of the original OpenCV C functions.

This paper outlines the experimental process and tests employed, and discusses the results of these tests. From the findings, recommendations are given for academics and novices faced with selecting a setup for CV.

2. EXPERIMENTAL PROCESS
In this section, we briefly introduce the different libraries we tested and our experimental apparatus and protocol.

2.1 Libraries
Python. Python is a general purpose dynamic programming language [6]. Python is highly regarded not least because of its fast development time and the ease of integrating packages [7]. Python's performance makes it a viable programming language for scientific work [8], and it has also been used by members in the CV community for many years [9].

OpenCV. Originally an Intel research initiative, *OpenCV* is a cross-platform open source computer vision library, mostly employed for its real time image processing performance. It aims to provide well tested, optimized and open source implementation of state of the art image processing and computer vision algorithms.

The library is written in C, ensuring fast and portable code (optionally to embedded platforms). The library is built

NZCSRSC 2010, April 2010, Wellington, New Zealand. This paper was published in the proceedings of the New Zealand Computer Science Research Student Conference (NZCSRSC) 2010. Copyright is held by the authors.

above a core image library, which supports image structure and basic image manipulation. This core image library has two forms; a software implementation is provided freely whilst an accelerated version utilizing the *Integrated Performance Primitives* [10] can be optionally acquired from Intel. This latter option takes advantage of the extended multimedia instructions set available on Intel Processors (e.g. SSE3, SSE4).

Nowadays, bindings are available for OpenCV for multiple languages, such as OpenCVDotNet and EmguCV for the .NET platform. Multiple bindings to OpenCV such as OpenCV Python, and PyCV [11] have been created for Python, as well as the bindings automatically built with SWIG [12] which we tested in this paper. Complimentary, additional tools such as GPUCV [13] have been made for OpenCV using graphics hardware to accelerate CV performance on the GPU.

NumPy/SciPy. *NumPy* gives strongly typed N-dimensional array support to Python [14]. The library is well recognised and offers an easier approach for multidimensional array manipulation than in the C programming language. A large part of the low level algorithms are implemented in C and FORTRAN (and wrapped around Python), resulting in very fast and optimized raw data processing and iterating.

SciPy [15] is a set of Python libraries and tools for scientific and mathematical work built on top of NumPy [5]. SciPy offers many different modules including routines such as numerical integration, optimization, signal processing and image processing/computer vision functions. Two major tools are usually distributed with SciPy that are very useful for computer vision development; Matplotlib and IPython. Matplotlib [16] is an array and image plotting library, and IPython [17] is an improved interactive shell for Python. Some features of Matplotlib and IPython are further described in this paper.

2.2 Apparatus

We conducted our testing on a Intel Core 2 Duo 6600 machine, 4GB RAM, running Ubuntu 9.04 64-bit OS.

For the test we compared these different libraries (all builds were 64-bit version):

- OpenCV Native Language (OPENCV_C): We used snapshot built version 1.1.1, rev 1978. The code has been compiled with the GNU tool chain version (4.3.3), in Release mode with O3 compiler optimisations MMX, fast math, and SSE3. All additional packages, except 1393, are turned on (png, jpg, gtk, gstreamer, unicap, V4L).
- OpenCV Python Wrapper (OPENCV_PY): We used the SWIG [12] wrapper version 1.3.36. We used a similar OpenCV build as the OpenCV C version.
- SciPy/NumPy (SCIPY): We used the stable versions from the Ubuntu repositories: SciPy version 0.7.0 and NumPy 1.2.1

For the camera, we conducted our test with an off-the-shelf USB webcam Logitech Quickcam Pro for Notebooks. White balance, focus and exposure have been fixed to a constant value prior to the tests. The test environment was a large room with neon lamps at the ceiling and a low amount of ambient light.

2.3 Evaluation Protocol

For our testing, we cover different standard algorithms traditionally used in CV applications, as well as some major processes relevant to computer vision (e.g. image acquisition). Our tests were aiming to reproduce general high level processes applied during CV applications development, rather than low level function calls. The tests were chosen to represent algorithms underlying many complex computer vision applications. The particular set of tests exercise different aspects of the libraries, for example image aquisition is an IO-bound task whereas feature point detection is computationally bound.

For each test we describe the process, the difference in syntax between different libraries, and the performance and usability of each library. The performance measurements were taken a minimum of 3 times for a 2 minute period, the resulting standard deviation in measurement time is shown as error bars in the output graphs.

In most cases we provide implementations of the tests using each of the 3 libraries under analysis, however for some of the tests, an implementation was not possible due to differences in the libraries. As we are more focused on comparing the two Python libraries OPENCV_PY versus SCIPY, an OPENCV_C implementation was not necessary for every test.

3. QUANTITATIVE TESTS

3.1 Image Acquisition

Live image acquisition is widely utilized by the majority of CV applications. Hence frame acquisition and frame display was an initial test. Additional to performance results, we describe in this section the syntax between the different libraries for implementing this test.

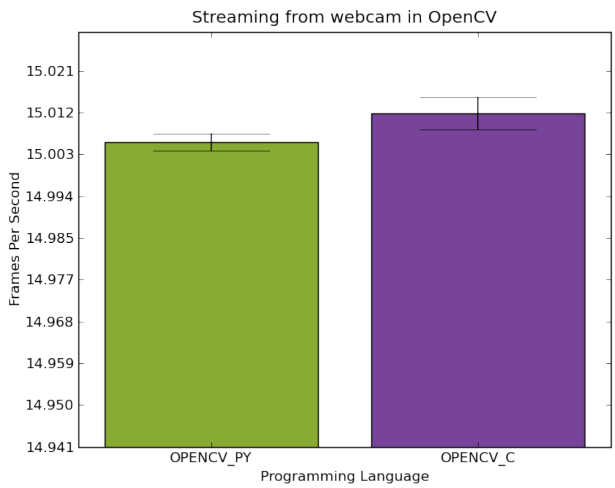


Figure 1: Comparison of capture performances between OPENCV_PY and OPENCV_C.

Acquisition/Display of an image with OpenCV C

Algorithm 1 describes how to open up a new camera capture device, capture one frame, open a new window and display the result[1].

Algorithm 1 Image capture and display with OpenCV in C

```
#include "cv.h"
#include "highgui.h"
int main(){
  IplImage  *frame;
  CvCapture *capture;
  capture = cvCreateCameraCapture(0);
  cvNamedWindow( "Snapshot", 0 );
  frame = cvQueryFrame( capture );
  cvShowImage( "Snapshot", frame );
}
```

Acquisition/Display of an image with OpenCV Python

Algorithm 2 shows the equivalent of the python wrapper. There is a high level of similarity with the previous version, however, in the Python code no variables types are declared.

Algorithm 2 Image capture and display with OpenCV in Python

```
from opencv import highgui as hg
capture = hg.cvCreateCameraCapture(0)
hg.cvNamedWindow("Snapshot")
frame = hg.cvQueryFrame(capture)
hg.cvShowImage("Snapshot", frame)
```

Comparison

Figure 1 shows the performance results for the previous algorithms. OPENCV_PY and OPENCV_C perform at very similar frame rates while carrying out an I/O bound task. OPENCV_C has a marginally higher frame rate output than OPENCV_PY.

The SciPy package does not currently have a direct method for image capture, so it was not possible to compare live acquisition. However, a solution was developed for using the OpenCV camera capture with SciPy; we created a Python decorator which converts the image data to a NumPy array before and after calling a Python function that processes and supports NumPy images. A 640x480 RGB image takes less than 2 ms to convert in either direction on the testing platform used throughout this report.

3.2 Image Blur

One of the simplest operations in image processing is blurring an image. As this can be achieved in different ways, we focused here on testing a basic Gaussian blur. This is easily achieved by convolving the image with a Gaussian filter. Because of the separability of multidimensional Gaussian filters [18], the convolution can be applied in two ways; applying a one-dimensional filter twice - once in each direction, or secondly the image can be convolved with a two-dimensional Gaussian filter created by the product of two one-dimensional filters.

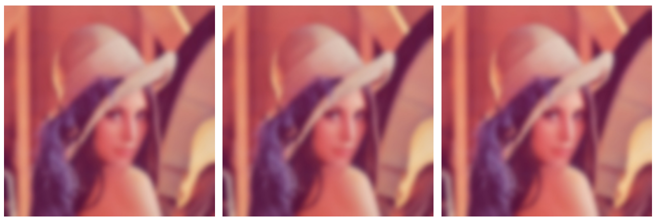
The Gaussian function for obtaining the filter in one dimension is given in Equation 1, Equation 2 gives the 2 dimensional case [19].

$$G(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}} \tag{1}$$

$$G(x,y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2}$$

Where σ is the standard deviation of the Gaussian distribution.

OpenCV includes a Gaussian filter implementation that can be applied to an image by calling the cvSmooth function and passing the desired filter size. SciPy has a n-dimensional Gaussian filter that acts on a NumPy array. Both libraries use the one-dimensional case, as it requires less computation.



(a) OPENCV_PY (b) OPENCV_C (c) SCIPY

Figure 2: Generated Images from Gaussian Blur filter using OPENCV_PY, OPENCV_C, and SCIPY on Lena dataset.

To ensure the same level of filtering is carried out for all the libraries, the filter parameters have been converted to be compatible with OpenCV's cvSmooth defaults [20].

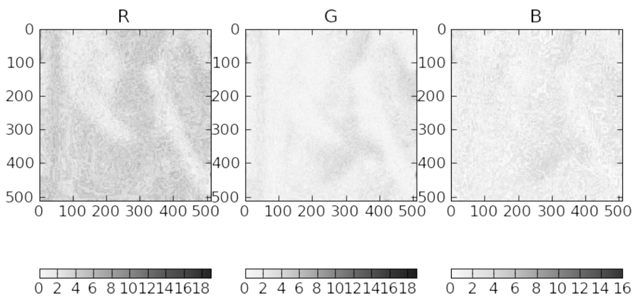


Figure 3: Channel Difference (RGB, 255 bits resolution) from Gaussian blur filter between OPENCV_PY and SCIPY.

Comparison

The blurred output images are shown on the Lena dataset in Figure 2. A basic image difference between output images confirmed exactly the same results between C++/Python OpenCV version (as expected), but small differences were found between SciPy and OpenCV Python code as presented in Figure 3. The graph in Figure 3 shows the pixel by pixel differences in each of the colour channels of a single image. The maximum intensity difference at any point was 7.8%, the mean difference was 0.8% of the full intensity scale.

[1]For presentation brevity we omitted in this paper the source code for error checking, cleanup and optimization. However they are present in the source code of our tests

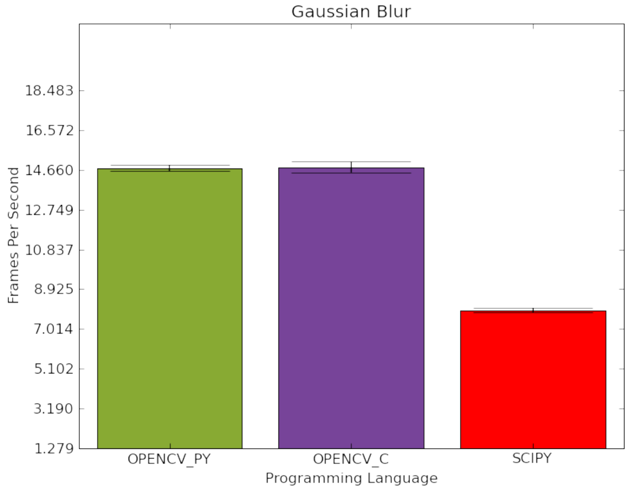


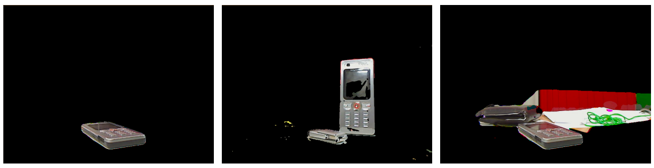
Figure 4: Comparison of gaussian blur performances between OPENCV_PY, OPENCV_C and SCIPY.

This discrepancy in Figure 3 could be simply explained by a difference in the implementation of the Gaussian kernel approximations. In SciPy the filter is created by a direct sampling of the Gaussian function; OpenCV on the other hand, uses the size of the filter, this is a good indication it probably uses the pascal triangle as an approximation for the Gaussian kernel [21]. These differences are minor, but it is worth noting that such a simple traditionally used algorithm provides such different results.

In terms of time performance, Figure 4 shows that OpenCV (either Python and C version) runs twice as fast as SciPy with the implementations given.

3.3 Background subtraction

A common task in security surveillance, human computer interaction is the detection of any visual changes in a video. This is done in its simplest form by a comparison of one frame to another previous frame [22]. If the image difference exceeds a specified threshold, something is deemed to have changed.



(a) Adding item (b) minor problems (c) Addition and removal

Figure 5: Background subtraction response after adding and removing items from a scene using OPENCV_PY.

An example is presented in Figure 5 after adding a cellphone to a scene for the OPENCV_PY implementation. As Figure 6 shows, the performance between Python and C are in the same order of magnitude, no significant difference were observable, however, the frame rate for SCIPY is significantly slower.

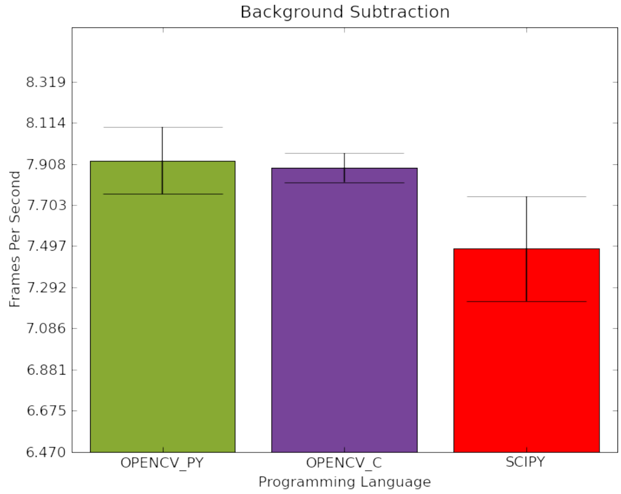


Figure 6: Comparison of background subtraction performances between OPENCV_PY, OPENCV_C and SCIPY.

3.4 Feature Point Detection

Many methods in CV for identifying the contents of an image rely on extracting *interesting* features. Generally used as feature points are corners of intersecting lines, line endings, or any isolated point where local image regions have a high degree of variation in all directions [23]. One such method of obtaining these features is the Harris & Stephens algorithm. According to [24] the algorithm is in short:

> *A matrix W is created from the outer product of the image gradient, this matrix is averaged over a region and then a corner response function is defined as the ratio of the determinant to the trace of W.*

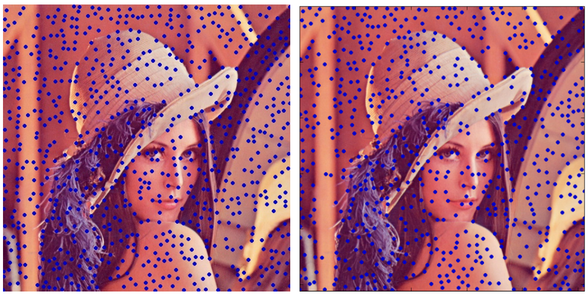
A threshold is then applied to this corner response image to pick the most likely candidates and then these points are plotted. We used this algorithm as the basis of the test to compare the different libraries.

We took and modified an existing implementation in SciPy from [24]. A filter kernel size of 3 pixels was used when computing the harris response. Results are visible in Figure 7 on the Lena dataset for OpenCV Python and SciPy. We observed that with a larger kernel SciPy seemed to slow down more than OpenCV. The threshold filtering and display of the corner response was implemented solely in OpenCV to reduce differences; the SciPy implementation therefore had an extra data conversion stage.

Visual assessment of the images show a difference of the features identified also reflecting a difference in terms of implementation between both libraries. Timing performances are available in table 1 (measured average over 300 iterations on the Lena dataset), OPENCV_PY performed the task roughly three times quicker than SciPy.

3.5 Face Detection

Face detection is the task of identifying the presence of any number of faces in an image, this is a specific case of general



(a) OPENCV_PY (b) SCIPY

Figure 7: Running the Harris & Stephens feature detection algorithm on the Lena test image with OPENCV_PY and SCIPY.

Library	Mean	Std
OPENCV_PY	65.7 ms	1.27 ms
SCIPY	191.5 ms	0.87 ms

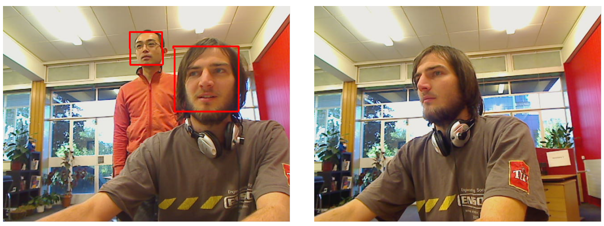
Table 1: Timings of running feature detection on the Lena image.

object detection. Figure 8 shows the output from our tests running on OpenCV Python under different conditions using the face Haar-Cascade classifier that comes with OpenCV.

The method gave an average frame rate of 7.16 ± 0.02 hz. The detection process itself gave very consistent timings of 107 ± 1 ms. There is no corresponding high level functionality for Face detection in SciPy, so a performance comparison was not possible.

However, we can note that a recent project PyCV [11] improves on the face detection in OpenCV utilizing SciPy.

(a) Single face in frame (b) Obscured face in frame



(c) Multiple faces (d) Rotated face

Figure 8: Face Detection with OPENCV_PY

Algorithm 3 Using IPython, the interactive shell can be used from deep inside a nested loop in a running program. Here we have called IPShellEmbed()() inside the background subtract algorithm to look at intermediate data and quickly assess timing. Note the variables from the running program are directly accessible from the shell.

```
In [1]: from opencv import cv
In [2]: cv.cvAnd(diffImage, image, temp)
In [3]: timeit cv.cvAnd(diffImage, image, temp)
1000 loops, best of 3: 229 µs per loop
In [4]: from pylab import *
In [5]: imshow(temp)
Out[5]: <AxesImage object at 0x42489d0>
In [6]: show()
In [7]: image.shape
Out[7]: (480, 640, 3)
In [8]: differenceImage = abs(np_image.astype(float) - original.astype(float)).astype(uint8)
In [9]: timeit differenceImage = abs(np_image.astype(float) - original.astype(float)).astype(uint8)
10 loops, best of 3: 27.7 ms per loop
```

4. QUALITATIVE COMPARISON

Comparing OpenCV Python versus OpenCV C, the development and testing process has been observed to be shorter and easier with Python. With regard to the development of image processing and CV, Python has been shown to have value as a rapid prototyping tool.

The documentation in both SciPy and OpenCV was found to be complete, but not as extensive as for a professional package like MATLAB®. The support for these open source packages relies almost entirely on experienced members of the community responding to requests on message boards or mailing lists. The community support is however of great response and valuable technical quality.

A major limitation of using Python is the portability on embedded platforms and hardware, generally requiring highly optimized C/C++ code. The stability of the actual packages is also questionable: OpenCV Python bindings are being rewritten manually to replace the SWIG produced bindings and SciPy is still a relatively new library. In some cases, we also noticed the absence of analogous functions in both libraries, generally explained by the actual different orientation of libraries (SciPy being more oriented for general scientific computing).

SciPy offers valuable tools for the development and the monitoring of an application. Graphs can be generated easily with IPython using the Python graphing library Matplotlib. A powerful feature of IPython is the embeddable interpreter which delivers access to a live interactive shell with full timing and plotting capabilities during program execution (see Algorithm 3).

Another advantage in favor of Python is its high interoperability with other libraries. For example, PyGame can be combined with OpenCV as illustrated Figure 9.

(a) detecting face objects (b) edge filtering and face detection

Figure 9: PyGame can be used to capture and display the video image, while OpenCV Python does the processing.

5. RELATED WORKS

Beyond the presented libraries different works have focused on accelerating the performances of Python interpretation.

For example, SciPy possesses the Weave module for in-lining C and C++ code that can produce code 100x faster than pure Python [25]. Cython[26] allows developers to generate C extensions for the Python language using a particular dialect of Python. From a different direction a tool named OMPC has been created for compiling existing MATLAB code into Python[27].

For parallel programming, mixed language solutions have been shown to exhibit the same performance gains as native language solutions [8]. A different direction for parallel implementation is aimed at utilizing the power of the graphics card (GPGPU), PyGPU and the GPUCV are two examples of projects leveraging this possibility from Python [28] [13] [29].

Another related area of research is the native performance of Python itself as demonstrated by the Psyco *just in time compiler* for Python [30] (unfortunately development of this project has ceased and it is technically limited to Python 2.6.X version for x86 machines only). We can also cite additional projects also aiming in a similar direction as: PyPy [31], a compliant, flexible and fast implementation of the Python Language, Google's UNLADEN SWALLOW project which aims to speed up Python by leveraging the LOW LEVEL VIRTUAL MACHINE (LLVM).

Pyro [32], a robotics simulation environment is another example of a platform including computer vision modules.

6. CONCLUSION

For the CV community, Python offers a valuable platform for experimenting with new algorithms very quickly. Our tests demonstrated the quantitative and qualitative value of Python with particular regard to the OpenCV Python library. Thus for beginners in CV development, we recommend Python. For advanced project development requiring real-time support and portability on embedded systems, OpenCV C offers a more reliable approach.

If a free software tool such as Python or OpenCV is to be used as a common standard in Computer Vision it must match or exceed the performance of the commercial product. Future work will undertake a comparative analysis of the performance of MATLAB® for carrying out Computer Vision algorithms versus the open source counterparts.

The benchmarks and source code for all the tests is freely available under a GNU General Public License from: `http://pycam.googlecode.com`

7. REFERENCES

[1] G. Van Rossum *et al.*, "Python programming language," *CWI, Department CST, The Netherlands*, 1994.

[2] P. Kovesi, "Matlab functions for computer vision and image analysis," *School of Computer Science & Software Engineering, The University of Western Australia. Available from:< http://www. csse. uwa. edu. au/pk/research/matlabfns*.

[3] L. A. DeRose, "Compiler techniques for matlab programs," 1996.

[4] G. Bradski, "The opencv library," *DOCTOR DOBBS JOURNAL*, vol. 25, no. 11, pp. 120–126, 2000.

[5] T. Oliphant, "Python for scientific computing," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 10–20, 2007.

[6] G. V. Rossum, F. Drake, and eds, *Python documentation: The Python Standard Library*. Python Software Foundation, v2.6.2 ed., 4 Sept. 2009. http://docs.python.org/library/index.html.

[7] M. Sanner, "Python: a programming language for software integration and development," *J. Mol. Graph. Model*, vol. 17, pp. 57–61, 1999.

[8] X. Cai, H. Langtangen, and H. Moe, "On the performance of the python programming language for serial and parallel scientific computations," *Scientific Programming*, vol. 13, no. 1, pp. 31–56, 2005.

[9] J. Doak and L. Prasad, "Python and Computer Vision," in *the Tenth International Python Conference*, 2002.

[10] S. Taylor, *Intel integrated performance primitives*. Intel Press, 2004.

[11] P. Tri, *Principled Asymmetric Boosting Approaches to Rapid Training and Classification in Face Detection*. PhD thesis, Nanyang Technological University, 2009.

[12] D. Beazley, "Swig: An easy to use tool for integrating scripting languages with c and c++," in *Proceedings of the 4th conference on USENIX Tcl/Tk Workshop, 1996-Volume 4*, pp. 15–15, USENIX Association Berkeley, CA, USA, 1996.

[13] J. Farrugia, P. Horain, E. Guehenneux, and Y. Alusse, "Gpucv: A framework for image processing acceleration with graphics processors," in *2006 IEEE International Conference on Multimedia and Expo*, pp. 585–588, 2006.

[14] T. Oliphant, *Guide to NumPy*. Tregol Publishing, 2006.

[15] E. Jones, T. Oliphant, P. Peterson, *et al.*, "Scipy: Open source scientific tools for python," *http://www. scipy. org*, 2001.

[16] J. Hunter, "Matplotlib: a 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[17] F. Perez and B. Granger, "Ipython: a system for

interactive scientific computing," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 21–29, 2007.

[18] I. Young and L. van Vliet, "Recursive implementation of the gaussian filter," *Signal processing*, vol. 44, no. 2, pp. 139–151, 1995.

[19] L. Shapiro and G. Stockman, *Computer Vision*. Prentice Hall, 2001.

[20] G. Bradski and A. Kaehler, *Learning OpenCV: Computer vision with the OpenCV library*. O'Reilly Media, Inc., 2008.

[21] J. Ben-Arie, "Image processing operators and transforms generated by a set of multidimensional neural lattices that use the central limit," in *1991 IEEE International Joint Conference on Neural Networks, 1991*, pp. 987–993, 1991.

[22] H. Gao and R. Green, "A robust efficient motion segmentation algorithm," in *International conference on image and vision computing New Zealand, Auckland, New Zealand*, 2006.

[23] C. Harris and M. Stephens, "A combined corner and edge detector," in *Alvey vision conference*, vol. 15, p. 50, Manchester, UK, 1988.

[24] J. E. Solem, "Harris Corner Detector in Python." posted on http://jesolem.blogspot.com/2009/01/harris-corner-detector-in-python.html, 26 Jan. 2009.

[25] P. Ramachandran, "Performance of various python implementations for solving the 2d laplace equation."

[26] S. Behnel, R. Bradshaw, and G. Ewing, "Cython: C-extensions for python, 2008."

[27] P. Jurica and C. van Leeuwen, "Ompc: an open-source matlab®-to-python compiler," *Frontiers in Neuroinformatics*, vol. 3, 2009.

[28] C. Lejdfors and L. Ohlsson, "Implementing an embedded gpu language by combining translation and generation," in *Proceedings of the 2006 ACM symposium on Applied computing*, pp. 1610–1614, ACM New York, NY, USA, 2006.

[29] Y. Allusse, P. Horain, A. Agarwal, and C. Saipriyadarshan, "Gpucv: A gpu-accelerated framework for image processing and computer vision," in *Proceedings of the 4th International Symposium on Advances in Visual Computing, Part II*, pp. 430–439, Springer, 2008.

[30] A. Rigo, "Representation-based just-in-time specialization and the psyco prototype for python," in *Proceedings of the 2004 ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation*, pp. 15–26, ACM New York, NY, USA, 2004.

[31] P. Dubois, "A nest of pythons," *Computing in Science & Engineering*, pp. 81–84, 2005.

[32] D. Blank, D. Kumar, L. Meeden, and H. Yanco, "Pyro: A python-based versatile programming environment for teaching robotics," *Journal on Educational Resources in Computing (JERIC)*, vol. 3, no. 4, 2003.