Using Python in Computer Vision: Performance and Usability

Brian Thorne, Raphaël Grasset
HIT Lab NZ
University of Canterbury
Private Bag 4800, Christchurch
Email: brian.thorne@hitlabnz.org,raphael.grasset@hitlabnz.org

Richard Green
Computer Science and Software Engineering
University of Canterbury
Private Bag 4800, Christchurch
Email: richard.green@canterbury.ac.nz

***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 the Python Wrapper. In this paper, we present a comparative evaluation of both libraries, assessing their performance and their usability. We also describe our results regarding the performance of python wrapper vs native C implementation of OpenCV.**

***Index Terms*—Computer Vision, Python, SciPy, OpenCV**

I. 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 their attention and forged it as a popular tool for the research community.

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 the MATLAB software [2]. Although MATLAB offers an efficient high level platform for prototyping and testing algorithms, its performances doesn't compete with a well designed and optimized C/C++ implementation. More recently, we have seen the emergence of potential and valuable solutions for developing image processing and computer vision algorithms in Python.

The goal of this paper is to evaluate the performances and usability of the most common solutions 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 computer vision development that can be beneficial for any academic intrigued by the topic.

We have focused our attention on the performances of two widely used CV libraries in Python: *OpenCV* [3] and *NumPy/SciPy* [4]. 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.

After an introduction of our experimental process, we will describe successively the tests employed and the results we obtained, before finally discuss our findings and provide recommendations.

II. Experimental Process

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

Libraries

Python. Python is a general purpose dynamic programming language [5]. It is highly regarded due in no small part for its fast development time and the ease of integrating packages [6]. Python's performance makes it a viable programming language for scientific work [7], and it has also been used members in the CV community for many years [8].

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 the 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 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* [9] 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, multiple language bindings are available for OpenCV, such as OpenCVDotNet and EmguCV. Multiple bindings to OpenCV such as OpenCV Python, and PyCV [10] have been created for Python, as well as the automatically built wrapped bindings based on SWIG [11] which we tested in this paper. Complimentary, additiona tools such as GPUCV [12] 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 [13]. It's a well recognized library, offering 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 crunching and iterating.

SciPy [14] is a set of Python libraries and tools for scientific and mathematical work built on top of NumPy [4]. 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 and very useful for computer vision development: Matplotlib and IPython. Matplotlib [15] is an array and image plotting library, and IPython [16] is an improved interactive shell for Python. We will describe some of their features in the last section of this paper.

Apparatus

We conducted our test on a Intel Core 2 Duo 6600 machine, 4GB RAM, running Ubuntu 9.04 32-bit OS.

For the test we compared these different libraries (all builds were 32-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), and in Release mode with O3 compiler optimizations, MMX, fast math, SSE3. All additional packages, EXCEPT 1393, are turned on (png, jpg, gtk, gstreamer, unicap, V4L).
- OpenCV Python Wrapper (OPENCV_PY): we used the SWIG [11] 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 a prior the tests. The test environment was a large room with neon lamps at the ceiling and a low amount of ambient light.

Evaluation Protocol

For our testing, we cover different standard algorithms traditionally used in a CV applications as some major process relevant to computer vision (e.g. image acquisition). Our tests were aiming to reproduce general high level processes implied during CV applications development rather than low level function calls.

For each of the tests we describe the process, the difference in terms of syntax between the different libraries, the performance and the usability of each libraries. As we aims to provide implementation of the tests for the 3 tested libraries, some of the tests only focused comparing OPENCV_PY vs SCIPY. Reader can access the source code from the project website [1].

III. Quantitative Tests

A. Image Acquisition

Live image acquisition is largely utilized in a majority of CV applications. We thus tested, frame acquisition and frame display as a first test. Added to performance results, we describe in this section the syntax between the different libraries for implementing this test.

Acquiring and displaying an image OpenCV C Version: Algorithm 1 describes how to open up a new camera capture device, capture one frame, open a new window and display the result[2].

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

Algorithm 1 Image capture and display with OpenCV in C

Acquiring and displaying an image with OpenCV Python: Algorithm 2 show the equivalent through the python wrapper. You can observe a high level of similarity with the previous version, the main difference being in the Python code no variables types are declared.

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

Algorithm 2 Image capture and display with OpenCV in Python

Comparison: Figure 1 shows the performance results for the previous algorithms. The measurement was done over a 2 minutes period for 3 iterations, averaging the resulting frame rates. OpenCV Python and OpenCV C perform at very similar frame rates whilst carrying out an I/O bound task. OpenCV C having a marginally higher frame rate output than OpenCV Python.

The SciPy package does not currently have a direct method for image capture, so we couldn't compare live acquisition.

[1] http://code.google.com/p/pycam/wiki/PythonComputerVision

[2] 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

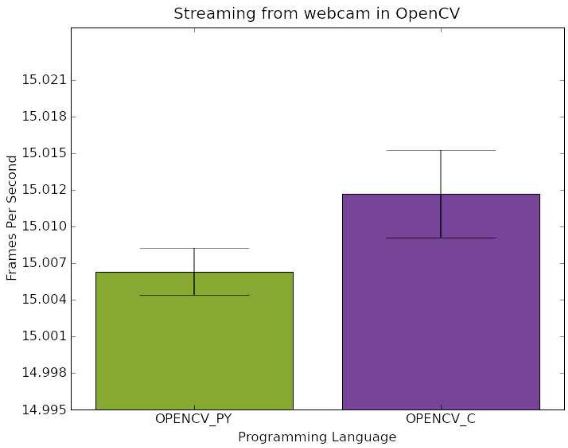


Figure 1: Comparison of capture performances between OPENCV_C, OPENCV_PY.

However, we developed a solution for converting OpenCV camera capture to 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 support NumPy image. A 640x480 RGB image takes less than 2ms to convert either way on the testing platform used throughout this report.

B. 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 to test a basic Gaussian blur. As well know, this is easily achieved by convolving the image with a Gaussian filter. Because of the separability of multidimensional Gaussian filters [17], the convolution can be applied in two ways; applying a 1 dimensional filter twice, once in each direction; or secondly the image can be convolved with a 2-dimensional Gaussian filter created by the product of two 1 dimensional filters.

Equation 1 shows the Gaussian function for obtaining the filter in one dimension and Equation 2 shows the 2 dimensional case [18].

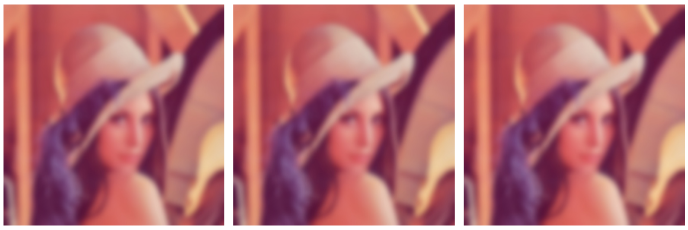
$$G(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}} \tag{1}$$

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

with *Sigma* 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 1 dimensional case, as it requires less computation.

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 [19].



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

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

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

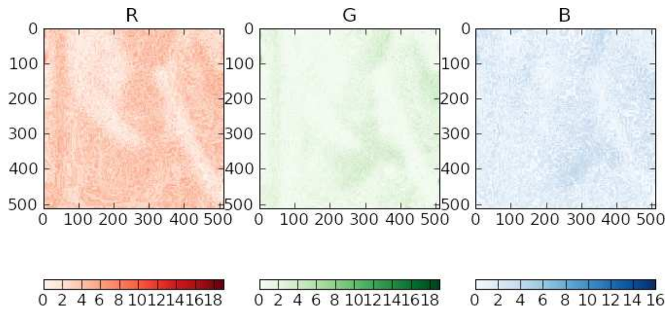


Figure 3: Channel Difference (RGB, 255 bits resolution) from Gaussian blur filter between OPENCV_PY and SCIPY.

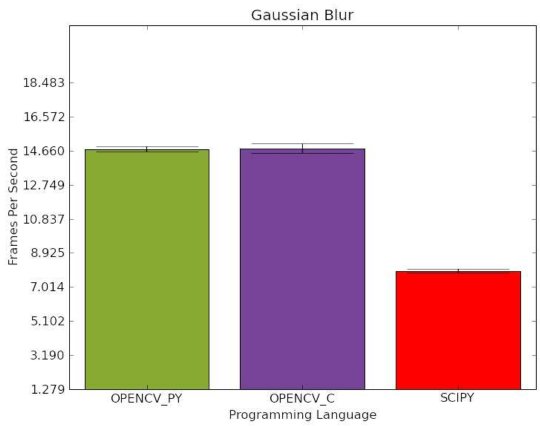


Figure 4: Comparison of gaussian blur performances between OPENCV_C, OPENCV_PY and SCIPY.

This discrepancy 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 [20]. 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 faster than SciPy.

C. 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 [21]. If the difference image is more than a certain threshold, something has changed.

An example is presented in Figure 6 after adding a cellphone to a scene for the OPENCV_PY implementation. As Figure 5 shows, the performance between Python an C are in the same order of magnitude, no significant difference were observable, where SCIPY offers inferior performances.

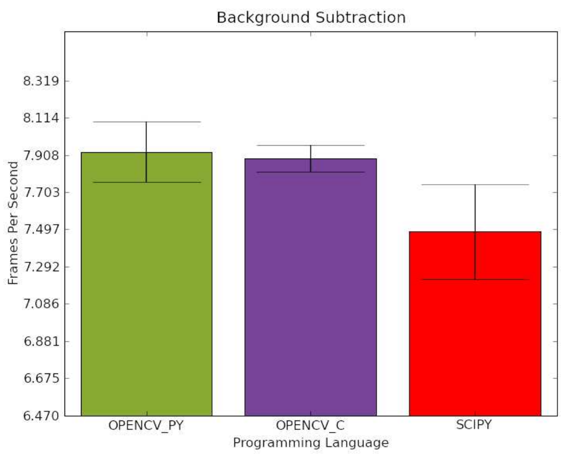
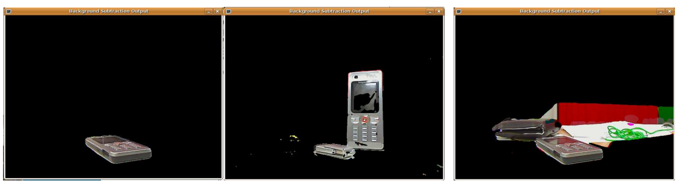


Figure 5: Comparison of background subtraction performances between OPENCV_C, OPENCV_PY and SCIPY.



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

Figure 6: Background subtraction response after adding and removing items from a scene using OPENCV_PY.

D. Feature Point Detection

Many methods in computer vision for identifying the contents of an image relies on extracting *interesting* features. Generally used feature point are corners of intersecting lines, a line ending, or any isolated point where local image regions have a high degree of variation in all directions [22]. According to [23] the Harris & Stephens 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 test to compare the different libraries.

We took and modified an existing implementation in SciPy from [23]. A filter kernel size of 3 pixels was used when computing the harris response. Results are visible 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.

(a) OPENCV_PY (b) SCIPY

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

Visual assessment of the images show a difference of the features identified also reflecting a difference in term of implementation between both libraries. Timing performances are available on the table below (measured on average on 300 iterations on the Lena dataset), OpenCV Python offering larger better performances than SciPy.

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

E. Face Detection

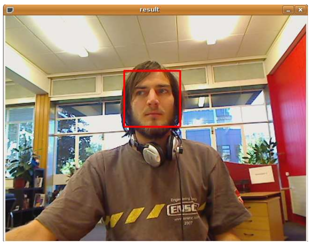
Face detection is the task of identifying the presence of any number of faces in an image, this is a specific case of general object detection.

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 framerate of 7.16 ± 0.02 Hz. The detection process itself gave very consistent timings of

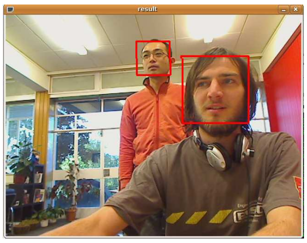
107 ± 1 ms. There is no corresponding high level functionality for Face detection in SciPy, so a performance comparison was not possible.

However, we can note that a recent project PyCV [10] improves the face detection from OpenCV utilizing SciPy.

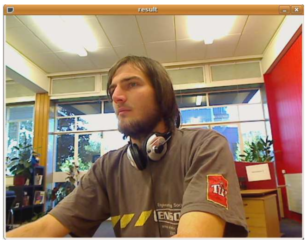


(a) Single face in frame

(b) Obscured face in frame



(c) Multiple faces



(d) Rotated face

Figure 8: Face Detection with OPENCV_PY

IV. Qualitative Comparison

Comparing OpenCV Python vs OpenCV C, the development and testing process has been observed to be shorter and easier with the Python. The value of the Python interpreter as a better rapid prototyping tool was also quite discernible regarding the development of image processing/computer vision.

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.

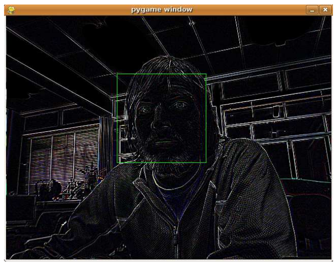
A major limitation of using Python is the portability on embedded platforms and hardware, generally requiring highly optimized C/C++ code. The stability of the actual packages is also questionable: OpenCV Python bindings are being rewritten manually to replace the SWIG produced bindings and SciPy is 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 more oriented for general scientific computing).

SciPy offers really valuable tools for the development and the monitoring of an application. Graphs can be generated easily with IPython based on the Matplotlib. In overall, IPython delivers pause execution during interpretation and give

```
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 imshow, show
In [5]: imshow(temp)
Out[5]: <AxesImage object at 0x42489d0>
In [6]: show()
```

Algorithm 3 Using IPython, the interactive shell can be used from deep inside a nested loop in a running program.

(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.

access to a live interactive shell with full timing and plotting capabilities (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.

V. Related Works

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

For example, SciPy proposes the Weave module for inlining C and C++ code that can produce code 100x faster than pure Python [24]. From a different direction a tool named OMPC has been created for compiling MATLAB code into Python[25].

For parallel programming, mixed language solutions have been shown to exhibit the same performance gains as native language solutions [7]. A different direction for parallel implementation is aimed toward the power of the graphics card (GPGPU), PyGPU and the GPUCV are two examples of projects leveraging this possibility from Python [26] [12] [27].

Another related area of research is the native performance of Python itself as demonstrated by the Psyco *just in time compiler* for Python [28] (unfortunately technically limited at the time of this writing on Python 2.6.X version for x86 machines only). We can also cite additional projects also aiming in a similar direction as: PyPy [29], 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).

To finish, we can also cite Pyro [30] a robotics simulation environment as another example of a platform including computer vision modules.

VI. Conclusion

For the CV community, Python offers a valuable platform for experimenting new algorithms very quickly. Our tests demonstrated the quantitative and qualitative value of Python especially regarding the OpenCV Python library. For beginners in computer vision development, we thus recommend Python. For advanced project development requiring real-time support and portability on embedded solution OpenCV C offers a more reliable approach.

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

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

[5] 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.

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

[7] 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.

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

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

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

[11] 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.

[12] 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.

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

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

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

[16] F. Perez and B. Granger, "IPython: a system for interactive scientific computing," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 21–29, 2007.

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

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

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

[20] 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.

[21] 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.

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

[23] 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.

[24] P. Ramachandran, "Performance of various Python implementations for solving the 2D Laplace equation."

[25] P. Jurica and C. van Leeuwen, "OMPC: an open-source MATLAB®-to-Python compiler," *Frontiers in Neuroinformatics*, vol. 3, 2009.

[26] 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.

[27] 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.

[28] 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.

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

[30] 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.