The Python Papers Monograph, Vol. 1 (2009)
Available online at http://ojs.pythonpapers.org/index.php/tppm

Introduction to Computer Vision in Python

Brian Thorne
HitLabNZ,
University of Canterbury,
brian.thorne@hitlabnz.org

Abstract

An introduction to computer vision in Python, from the general concept to its implementation with some current open-source libraries. This paper demonstrates basic computer vision examples using SciPy, OpenCV and Pygame.

Keywords: Computer Vision, OpenCV, SciPy, NumPy

1 Introduction

Image processing and computer vision (CV) has been driven by development in C/C++ and the usage of MATLAB software[1]. Although MATLAB offers an efficient high level platform for prototyping and testing algorithms, its performance does not compete with a well designed and optimised C/C++ implementation. Recently, potential and valuable solutions have emerged for developing image processing and computer vision algorithms in Python[2]. For image processing or computer vision development, two Python libraries are prominently used: *NumPy/SciPy* and *OpenCV* with a Python wrapper. This paper looks at using basic computer vision routines from OpenCV and SciPy.

2 Tools

NumPy *NumPy* gives strongly typed N-dimensional array support to Python[3]. 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 resulting in very fast and optimised raw data processing and iterating. NumPy can be found online at http://numpy.scipy.org

SciPy SciPy[4] 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, optimisation, signal processing and image processing/computer vision functions. Two major tools often used in conjunction with SciPy are very useful for computer vision development; Matplotlib and IPython. Matplotlib[6] is an array and image plotting library, and IPython[7] is an improved interactive shell for Python. SciPy can be downloaded from http://www.scipy.org

Pygame Pygame[8] is a game development framework which provides graphical, audio and input functions for creating games and multimedia applications. Pygame has recently had a camera

module added to it and with that can be used for basic image processing[9]. Pygame is based on the cross platform SDL library[10] and works on many platforms including OSX, Linux, Windows, Nokia S60[11], and the OLPC[12]. Pygame is found at http://pygame.org

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, optimised, open source implementations of, state of the art image processing and computer vision algorithms. The library is written in C[1], ensuring fast and portable code, and has been compiled for many embedded platforms.
Multiple language bindings are available for OpenCV, such as OpenCVDotNet and EmguCV. Multiple bindings to OpenCV such as OpenCV Python, and PyCV[14] have been created for Python, as well as the bindings automatically built with SWIG[15]. Additional tools such as GPUCV[16] have been made for OpenCV using graphics hardware to accelerate CV performance on the GPU. OpenCV can be downloaded from SourceForge[2] the project home page and documentation is at http://opencv.willowgarage.com

3 Computer Vision

In many CV applications such as Augmented Reality (AR) the overall process is very similar; an image frame is acquired from a video stream, undergoes an arbitrary process, and then an output image is displayed. This sequence, excluding setup, is repeated in an infinite loop. In Python there are many tools for each of these sections as shown in Table 1. Python facilitates using different tools together. Of primary concern is the intermediate stage - how does data captured in OpenCV end up being processed in SciPy and then get displayed in Pygame? Since the tools often use their own data storage format a time consuming conversion is often required. For this reason it is preferable to have a common format such as the typed arrays of NumPy or the cvMat of OpenCV.

Figure 1: Using Pygame with OpenCV to augment reality by detecting a face and overlaying a hat in real-time.

Capture	Process	Display
OpenCV	OpenCV	OpenCV
Pygame	Pygame	Pygame
V4L/VideoCapture	NumPy/SciPy	Matplotlib
C/C++	C/C++	wx/qt/gtk

Table 1: A small subset of libraries that can be used for CV with Python

[1] The library is built on top of 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 utilising the *Integrated Performance Primitives*[13] 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).

[2] at http://sourceforge.net/projects/opencvlibrary/files/

3.1 Video Acquisition

Getting and displaying a stream of images from a webcam is critical for many computer vision applications. Algorithm 1 demonstrates how to open up a new camera capture device, capture one frame, open a new window and display the result using the OpenCV library[3]. Since OpenCV is undergoing a change in official Python bindings, both the automatically SWIG wrapped version (Algorithm 1), and the new manually wrapped version (in Algorithm 2), are shown.

Algorithm 1 Image capture and display with OpenCV in Python

```
1 from opencv import highgui as hg
2 capture = hg.cvCreateCameraCapture(0)
3 hg.cvNamedWindow(Snapshot)
4 frame = hg.cvQueryFrame(capture)
5 hg.cvShowImage(Snapshot, frame)
```

Algorithm 2 Taking and displaying an image with the new Python bindings for OpenCV

```
1 >>> import cv
2 >>> cap = cv.CaptureFromCAM(0)
3 >>> frame = cv.QueryFrame(cap)
4 >>> cv.NamedWindow('capture')
5 >>> cv.ShowImage('capture', frame)
```

Algorithm 3 Image capture and display in Pygame

```
1 import pygame, pygame.camera
2 pygame.camera.init()
3 cam = pygame.camera.Camera('/dev/video0')
4 cam.start()
5 frame = cam.get_image()
6 display = pygame.display.set_mode(cam.get_size())
7 display.blit(frame,(0,0))
8 pygame.display.flip()
```

For completion the Pygame method of image capture and display is shown also (in Algorithm 3). Using this knowledge it is easy to make an object oriented version of a capture, process, and display loop for use in the next examples[4]. These programs take care of the capture and display. A Python function is expected, which when called, will carry out the image processing.

3.2 Image Filtering and Morphology

3.2.1 Gaussian Blur

One of the simplest operations in image processing is blurring an image. There are many possible reasons for blur, it is most often done to reduce noise. As this can be achieved in different ways, this paper demonstrates one method - a basic Gaussian blur. This is achieved by convolving the

[3] For presentation brevity I have omitted here the source code for error checking, cleanup and optimisation. However they are present in the source code of the tests.

[4] An example (VideoCapturePlayer) which uses OpenCV can be found at:
http://code.google.com/p/pycam/source/browse/trunk/c-vs-py/py_examples/VideoCapturePlayer.py
An example which uses Pygame can be found at:
http://code.google.com/p/pycam/source/browse/trunk/pycam/pycam/VideoCapturePlayer.py

image with a Gaussian filter. The libraries, OpenCV and SciPy, both have gaussian specific filter functionality as well as more general convolution operations.

- OpenCV includes a gaussian filter that can be applied to an image by calling the *cvSmooth* function and passing the desired window size as shown in Algorithm 4.
- SciPy has a multi-dimensional Gaussian filter that acts on a NumPy array (Algorithm 5).

Algorithm 4 Gaussian Blur with OpenCV

```
1  from pycam import VideoCapturePlayer as VCP
2  from opencv import cv
3  def gaussianBlur(image):
4      """Blur an image"""
5      result = cv.cvCreateMat(image.rows, image.cols, image.type)
6      filterSize = 43
7      # Carry out the filter operation
8      cv.cvSmooth(image, result, cv.CV_GAUSSIAN, filterSize)
9      return result

11 if __name__ == "__main__":
12     title = "Guassian Filtered Output"
13     VCP(gaussianBlur, title).main()
```

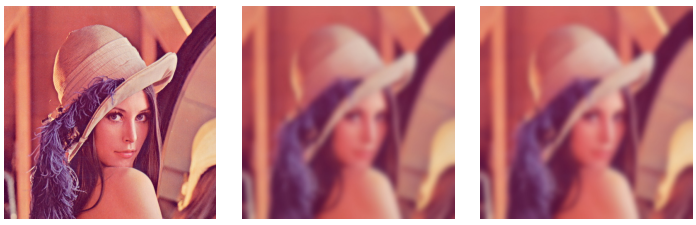
Algorithm 5 Gaussian blur filtering using SciPy (with OpenCV doing the capture)

```
1  from numpy import array, uint8
2  from scipy import signal, ndimage
3  from pycam import VideoCapturePlayer as VCP
4  from pycam import scipyFromOpenCV

6  def opencvFilt2sigma(size):
7      """OpenCV defaults to making sigma up with this formula"""
8      return (( size/2 ) - 1)*0.30 + 0.80

10 @scipyFromOpenCV
11 def gaussianBlur(np_image):
12     """Blur an image with scipy"""
13     filterSize = opencvFilt2sigma(43)
14     result = ndimage.filters.gaussian_filter(np_image, (filterSize, filterSize,
           → 1))
15     return result

17 if __name__ == "__main__":
18     title = "Guassian Filtered Output"
19     VCP(gaussianBlur,title=title).main()
```



(a) Original (b) OpenCV (c) SciPy

Figure 2: Images generated by applying the gaussian blur filter on the Lena image.

3.2.2 Dilation

Dilation and erosion are important morphological transformations often used in image processing[17]. Dilation expands bright regions and smooths out noise caused by shadows by merging components together[18]. Using the OpenCV VideoCapturePlayer introduced in section 3.1, dilation is carried out with the default kernel for 5 iterations in Algorithm 6. Figure 3 shows the dilation being carried out, note the black text on the paper being merged to white. Carrying out dilation with SciPy is shown in Algorithm 7.

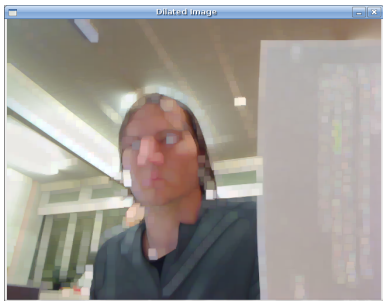


Figure 3: Carrying out dilation on webcam stream, the results of running Algorithm 6.

Algorithm 6 Image dilation in OpenCV.

```
from VideoCapturePlayer import VideoCapturePlayer as VCP
from opencv import cv

def dilateImage(image):
    cv.cvDilate(image, image, None, 5)
    return image

if __name__ == "__main__":
    VCP(dilateImage, "Dilated Image").main()
```

Algorithm 7 Image dilation in SciPy.

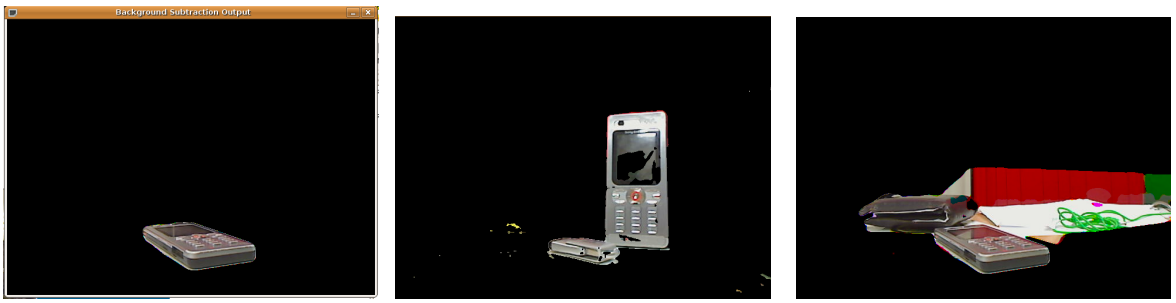
```
from pycam import VideoCapturePlayer, numpyFromSurf
from scipy.ndimage import morphology

@numpyFromSurf
def dilate(image):
    return morphology.grey_dilation(image,(10,10,1) )

if __name__ == '__main__':
    VideoCapturePlayer(processFunction=dilate).main()
```

3.3 Background subtraction

A common task in security surveillance[19], human computer interaction[20], and object tracking[21] is the detection of any visual changes in a video. In the simplest form this is done by background subtraction - a comparison of one frame to a base frame. If the intensity difference at each pixel exceeds a specified threshold, something is deemed to have changed. An example is presented in Figure 4 after adding a cell phone to a scene for an OpenCV Python implementation (Algorithm 9). To compare, the same process in SciPy is shown Algorithm 8.



(a) Adding an item (b) Minor Problems with noise (c) Addition & removal of item

Figure 4: Background subtraction response after adding and removing items from a scene using OpenCV

Algorithm 8 Background Subtraction in SciPy

```
from numpy import array, uint8, zeros
from scipy import signal, ndimage
from VideoCapturePlayer import VideoCapturePlayer as VCP
from misc import scipyFromOpenCV

@scipyFromOpenCV
def threshold_image(np_image, n=[]):
    if len(n) < 5:
        # First capture a few images - give the camera time to adjust...
        n.append(np_image[:])
        return np_image
    original = n[4]
    img = np_image[:]

    # Take the difference between the original frame and the current frame
    differenceImage = abs(np_image.astype(int) - original.astype(int)).astype( ↘
       → uint8)

    # amount of "Change" required before something will show up
    thresholdValue = 30

    # Take the N-Dimensional difference (3 channels of binary)
    differenceImage = (differenceImage >= thresholdValue)

    # Convert this to one channel binary
    differenceImage = differenceImage.mean(2).astype(bool)

    # Remove Salt & Pepper Noise
    differenceImage = ndimage.median_filter(differenceImage, size=5)

    # Create a black image of same type and shape as input
    output = zeros(img.shape).astype(img.dtype)

    # Take the original pixel at every point where the image is "different"
    output[differenceImage] = img[differenceImage]
    return output

if __name__ == "__main__":
    title = "SciPy Background Subtraction"
    VCP(threshold_image, title=title).main()
```

As figure 4a shows, the basic computer vision routine in Algorithm 9 can adequately reveal new

additions to a scene. After a time, with more objects changes, the noise starts to be a big issue (figure 4b). For example if lighting conditions change - everything is deemed to have changed. Another problem is when an object is removed, the software cannot tell the difference so it shows a *hole* where something used to be (see the effect of removing a laptop from the scene in figure 4c). With more filtering[22], and by slowly updating the original image this could easily form the basis of a security system[23].

Algorithm 9 Background Subtraction in OpenCV

```
from VideoCapturePlayer import VideoCapturePlayer as VCP
from opencv import cv

def threshold_image(image, n=[]):
    """Record the first 5 images to get a background, then diff current frame
       → with the last saved frame."""
    if len(n) < 5:
        n.append(cv.cvCloneMat(image))
        return image

    original = n[4]
    differenceImage  = cv.cvCloneMat( image )
    cv.cvAbsDiff( image, original, differenceImage )

    """The threshold value determines the amount of "Change" required
    before something will show up"""
    thresholdValue = 50
    cv.cvThreshold( differenceImage, differenceImage, thresholdValue, 255, cv.
       → CV_THRESH_BINARY )

    # Convert to one channel
    gray = cv.cvCreateImage( cv.cvGetSize(differenceImage), 8, 1 )
    cv.cvCvtColor( differenceImage, gray, cv.CV_BGR2GRAY )

    # Use median filter to remove salt and pepper noise.
    cv.cvSmooth(gray, gray, cv.CV_MEDIAN, 15)

    # Add a bit of Blur to the threshold mask
    cv.cvSmooth(gray, gray, cv.CV_GAUSSIAN, 5)

    result  = cv.cvCloneMat( image)
    cv.cvSetZero(result)

    cv.cvAnd(image,image, result, gray)
    return result

if __name__ == "__main__":
    title = "Background Subtraction Output"
    VCP(threshold_image, title).main()
```

3.4 Face Detection

Face detection is the task of identifying the presence and location of a number of faces in an image. Figure 5 shows the output from our tests running on OpenCV Python under different conditions using the face Haar-Cascade classifier that comes with OpenCV.

Before locating an object, such as a face, we must be able to describe it. Many object detection algorithms use a classifier which is a collection of feature patterns made by scanning a database of images with known content[24][25]. OpenCV comes with a selection of trained classifiers for

identifying heads, eyes, upper bodies and more. Also OpenCV comes with a tool to train a classifier. This paper just uses the classifiers included in OpenCV. As figure 5b shows, the face

(a) Singe face in a frame (b) Obscured face in frame (c) Multiple faces

Figure 5: Face Detection with OpenCV

needs to be free of obstructions for face detection to work. Simple face detection can be used to provide more user information to the computer to enable better interaction, for example, having the computer wake from screensaver when a user sits down.

3.5 Augmented Reality

Augmented reality is undergoing massive growth [26][27]. Using the face detection provided by OpenCV, an AR game can easily be made in Pygame using the webcam and face location as the interface. A very simple augmentation using Pygame and OpenCV is listed in Algorithm 10 and shown in figure 1.

Algorithm 10 Simple Augmented Reality

```
from __future__ import division
from pycam import VideoCapturePlayer, pygameFaceDetect
from pygame import image, Rect, transform
from pygame.locals import *

hat = image.load('beanie.png')

def overlayAHat(surf, face):
    # Draw an image of a hat on top of the face.
    width_factor, height_factor = 5/5, 3/5
        scaled_hat = transform.scale(hat, (int(width_factor*face.width), int( ↘
            → height_factor*face.height)))
    hat_x = int(face.left + (face.width/2) - width_factor*face.width/2)
    hat_y = int(face.top - height_factor*face.height/2)
    surf.blit(scaled_hat, (hat_x, hat_y))

def drawHatOnFaces(surf):
    faces = pygameFaceDetect.getFaces(surf)
    if faces:
        s = pygameFaceDetect.faceDetect.image_scale
        for face in faces:
            face_bounding_rect = Rect(face.x*s, face.y*s, face.width*s, face. ↘
                → height*s)
            overlayAHat(surf, face_bounding_rect)
    return surf

if __name__ == "__main__":
    VideoCapturePlayer(processFunction=drawHatOnFaces).main()
```

4 Conclusions

This paper presented a brief look at implementing basic computer vision functionality using Python. The three libraries looked at have very different goals, but can all be used for computer vision at different levels. PyGame can be used for integrating computer vision into game development, SciPy and OpenCV can be used for computer vision development and applications. For embedded platforms where speed is of utmost importance, or when computer vision functionality is the main requirement, OpenCV is the fastest and most complete tool for computer vision. Although, as shown in section 3.5, there is no constraint to use just one of these tools.

4.1 Future work

The examples for this paper are in a google code project called Pycam. At present it contains two VideoCapturePlayer classes, one for OpenCV, and one for PyGame. It would be possible to have a single class that determines what internal data type would be best suited for the given processing function, then use that library.

5 Additional Links

- Pycam - All code from this project, a small framework with examples linking OpenCV to PyGame and NumPy.
  pycam.googlecode.com

- PYCV - A Computer Vision Package for Python Incorporating Fast Training of Face Detection.
  http://www3.ntu.edu.sg/home5/pham0004/pycv/
- PYCVF - A Python Computer Vision Framework in Python currently in development.
  http://sourceforge.net/projects/pycvf/

References

[1] P. Kovesi, "MATLAB and Octave functions for computer vision and image processing," *School of Computer Science & Software Engineering, The University of Western Australia*, 2000. [Online]. Available: http://www.csse.uwa.edu.au/~pk/research/matlabfns/

[2] J. Doak and L. Prasad, "Python and Computer Vision," in *the Tenth International Python Conference*, 2002. [Online]. Available: http://www.python.org/workshops/2002-02/papers/05/index.htm

[3] T. Oliphant, *Guide to NumPy*. Tregol Publishing, 2006. [Online]. Available: http://www.tramy.us/

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

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

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

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

[8] P. Shinners *et al.*, "Pygame: Python game development." [Online]. Available: http://www.pygame.org

[9] J. Campbell, "Computer Games Programming," Letterkenny Institute of Technology Lecture Notes, 8 Jan. 2008. [Online]. Available: http://www.jgcampbell.com/bscgp1/

[10] A. Holkner, "ctypes. ctypes run!" *The Python Papers*, vol. 2, no. 2, p. 38, 2008. [Online]. Available: http://ojs.pythonpapers.org/index.php/tpp/article/viewArticle/23

[11] T. Ojala, V. Results, J. Scheible, J. Scheible, and T. Ojala, "Project MUSE Journals Leonardo Volume 42, Number 4, August 2009 MobiSpray: Mobile Phone as Virtual Spray Can for Painting BIG Anytime Anywhere on Anything," *Leonardo*, vol. 42, no. 4, 2009.

[12] L. Brown, "The $100 role model," *ITNOW*, vol. 51, no. 1, p. 8, 2009. [Online]. Available: http://itnow.oxfordjournals.org/cgi/reprint/51/1/8.pdf

[13] S. Taylor, *Intel integrated performance primitives*. Intel Press, 2004. [Online]. Available: http://software.intel.com/en-us/intel-ipp/

[14] P. Tri, "Principled Asymmetric Boosting Approaches to Rapid Training and Classification in Face Detection," Ph.D. dissertation, Nanyang Technological University, 2009.

[15] 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*. USENIX Association Berkeley, CA, USA, 1996, pp. 15–15.

[16] 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*, 2006, pp. 585–588.

[17] E. Dougherty, *Mathematical morphology in image processing*. CRC, 1992.

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

[19] R. Cucchiara, "Multimedia surveillance systems," in *Proceedings of the third ACM international workshop on Video surveillance & sensor networks*. ACM, 2005, p. 10.

[20] J. Flachsbart, D. Franklin, and K. Hammond, "Improving human computer interaction in a classroom environment using computer vision," in *Proceedings of the 5th international conference on Intelligent user interfaces*. ACM New York, NY, USA, 2000, pp. 86–93.

[21] S. McKenna, S. Jabri, Z. Duric, A. Rosenfeld, and H. Wechsler, "Tracking groups of people," *Computer Vision and Image Understanding*, vol. 80, no. 1, pp. 42–56, 2000.

[22] H. Gao and R. Green, "A robust moving object segmentation algorithm," in *Wavelet Analysis and Pattern Recognition, 2007. ICWAPR'07. International Conference on*, vol. 1, 2007. [Online]. Available: file:///home/brian/Ebooks/ComputerVision/2007-ICWAPR-RobustMovingObjectAlgorithm.pdf

[23] M. Cristani, M. Bicego, and V. Murino, "On-line adaptive background modelling for audio surveillance," in *Proceedings of the Pattern Recognition, 17th International Conference on (ICPR'04) Volume*, vol. 2, pp. 399–402.

[24] P. Viola and M. Jones, "Robust real-time face detection," *International Journal of Computer Vision*, vol. 57, no. 2, pp. 137–154, 2004.

[25] R. Lienhart and J. Maydt, "An extended set of haar-like features for rapid object detection," in *IEEE ICIP*, vol. 1, no. 2002, 2002, pp. 900–903.

[26] M. Billinghurst, R. Grasset, and J. Looser, "Designing augmented reality interfaces," *ACM SIGGRAPH Computer Graphics*, vol. 39, no. 1, p. 22, 2005.

[27] R. Azuma, Y. Baillot, R. Behringer, S. Feiner, S. Julier, and B. MacIntyre, "Recent advances in augmented reality," *IEEE Computer Graphics and Applications*, pp. 34–47, 2001.