SimpleCV Documentation

Release 1.0.0

Ingenuitas Inc

December 03, 2011

CONTENTS

a kinder, gentler machine vision python library

SimpleCV is an interface for Open Source machine vision libraries in Python. It provides a consise, readable interface for cameras, image manipulation, feature extraction, and format conversion. Our mission is to give casual users a comprehensive interface for basic machine vision functions and an elegant programming interface for advanced users.

Download from SourceForge: <http://sourceforge.net/projects/simplecv/files>

SourceForge Project Page: <http://sf.net/p/simplecv>

We like SimpleCV because:

- Even beginning programmers can write simple machine vision tests
- Cameras, video files, images, and video streams are all interoperable
- Information on image features can be extracted, sorted and filtered easily
- Manipulations are fast, with easy to remember names
- Linear algebra is strictly optional

Here is the simplecv "hello world":

```
import SimpleCV

c = SimpleCV.Camera()
c.getImage().save("picture.jpg")
```

For more code snippets, look at the cookbook, or the example scripts.

CHAPTER

ONE

INSTALLATION

You will absolutely need:

- OpenCV installed for your platform http://opencv.willowgarage.com/wiki/InstallGuide
- SciPY/Numpy installed for your platform http://www.scipy.org/Download

Once you have all the required libraries installed:

```
easy_install simplecv
```

If you need more help, look at the [installation docs](http://simplecv.sf.net/installation.html)

CHAPTER

TWO

REQUIRED LIBRARIES

- OpenCV 2.2 with Python bindings http://opencv.willowgarage.com/wiki/
- SciPY http://www.scipy.org/

CHAPTER

THREE

OPTIONAL LIBRARIES

- Python Image Library http://www.pythonware.com/products/pil/
- OpenKinect/freenect http://openkinect.org/wiki/Main_Page

Blob detection:

- CvBlob http://code.google.com/p/cvblob/
- cvblob-python https://github.com/oostendo/cvblob-python

Barcode reading:

- Zxing http://code.google.com/p/zxing/
- python-zxing https://github.com/oostendo/python-zxing

CHAPTER

FOUR

CONTENTS

4.1 SimpleCV module

4.1.1 Image

4.1.2 Features

4.1.3 Cameras

4.1.4 Streams

4.2 Installation

4.2.1 Ubuntu (10.4 or above)

Steps:

1. apt-get install dependencies
2. download, build and install the latest version of OpenCV
3. clone and install SimpleCV

Commands:

```
sudo apt-get install -y --force-yes build-essential swig gfortran cmake gcc pkg-config libjpeg62-dev
wget http://sourceforge.net/projects/opencvlibrary/files/opencv-unix/2.2/OpenCV-2.2.0.tar.bz2
bunzip2 OpenCV-2.2.0.tar.bz2
tar xf OpenCV-2.2.0.tar
mkdir build
cd build
cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local -D BUILD_PYTHON_SUPPORT=ON ..
make
sudo make install
sudo cp /usr/local/lib/python2.6/site-packages/cv.so /usr/local/lib/python2.6/dist-packages/cv.so
sudo apt-get install -y --force-yes git git python-dev python-numpy python-nose python-scipy ipython
git clone git://git.code.sf.net/p/simplecv/git.git simplecv
sudo python setup.py install
```

4.2.2 Mac OS X (10.5.8 and above)

Steps:

1. Install Xcode http://developer.apple.com/technologies/xcode.html
2. Install homebrew https://github.com/mxcl/homebrew/wiki/installation
3. Use homebrew to install opencv and git
4. Install scipy superpack, but with ipython 0.10.2 http://stronginference.com/scipy-superpack/ (download from http://ingenuitas.com/misc/superpack_10.6_2011.05.28-ipython10.sh)
5. Install python imaging library (10.6 needs ARCHFLAGS tweak)
6. clone simplecv and python setup.py install

Commands:

```
ruby -e "$(curl -fsSLk https://gist.github.com/raw/323731/install_homebrew.rb)"
wget http://ingenuitas.com/misc/superpack_10.6_2011.05.28-ipython10.sh
sh superpack_10.6_2011.05.28-ipython10.sh
brew install opencv
ln -s /usr/local/lib/python2.6/site-packages/cv.so /Library/Python/2.6/site-packages/cv.so
brew install git
ARCHFLAGS="-arch i386 -arch x86_64" brew install PIL
git clone git://git.code.sf.net/p/simplecv/git.git simplecv
cd simplecv
python setup.py install
```

4.2.3 Windows 7/Vista

Steps:

1. (OPTIONAL) Install MinGW for optional files and building openCV from source. Make sure to include C/C++ Compiler and msys package. http://sourceforge.net/projects/mingw/files/Automated%20MinGW%20Installer/
2. Install Python 2.7 http://www.python.org/getit/releases/2.7.1/
3. Install Python Setup Tools for Windows http://pypi.python.org/packages/2.7/s/setuptools/ (See: http://stackoverflow.com/questions/309412/how-to-setup-setuptools-for-python-2-6-on-windows)
4. Install the SciPy superpack: http://sourceforge.net/projects/scipy/files/scipy/0.9.0rc5/scipy-0.9.0rc5-win32-superpack-python2.7.exe/download
5. Install OpenCV: http://sourceforge.net/projects/opencvlibrary/files/opencv-win/2.2/ (See: http://luugiathuy.com/2011/02/setup-opencv-for-python/)
6. easy_install.exe simplecv (See: http://blog.sadphaeton.com/2009/01/20/python-development-windows-part-2-installing-easyinstallcould-be-easier.html)

4.3 SimpleCV Cookbook

4.3.1 Loading and Saving Images

The central class in SimpleCV is the Image() class, which wrappers OpenCV's iplImage (bitmap) and cvMat (matrix) classes and provides basic manipulation functions.

To load an image, specify the file path in the constructor:

```
my_image = Image("path/to/image.jpg")
```

To save the image, use the save method. It will automatically use the file you loaded the image from:

```
my_image.save()
```

You can also specify a new file to save to, similar to a "Save As...", and future calls to save() will save to this new file:

```
my_image.save("path/to/new_image.jpg")
#...do some stuff...
my_image.save() #saves to path/to/new_image.jpg
```

4.3.2 Image Manipulation

You can scale images using the "scale" function, so for instance to create a thumbnail. Note that this will distort perspective if you change the width and height ratios:

```
thumbnail = my_image.scale(90, 90)
```

You can also look at individual pixels:

```
r, g, b = my_image[25, 45]  #get the color trio for pixel at x = 25, y = 45
```

If you use python slices, you can extract a portion of the image. This section is returned as an Image object:

```
my_section = myimage[25:50, 45:70]    #grab a 25x25 rectangle starting at x = 25, y = 45
my_section.save("path/to/new_cropped_image.jpg")
```

You can also assign using direct pixel addressing, and draw on the image using this method:

```
black = 0.0, 0.0, 0.0    #create a black color tuple
my_image[25,45] = black  #set the pixel at x = 25, y = 45 to black
my_image[25:50, 45] = black #draw 1px wide line
my_image[25:50, 45:70] = black #create a 25x25 black rectange starting at x = 25, y = 45
```

4.3.3 Using a Camera, Kinect, or VirtualCamera

Addressing your [OpenCV supported webcam](http://opencv.willowgarage.com/wiki/Welcome/OS) is extremely easy:

```
mycam = Camera()
img = mycam.getImage()
```

If you install the [OpenKinect](http://openkinect.org/wiki/Getting_Started) library and python wrapper, you can use your Xbox Kinect to get a depth map:

```
k = Kinect()
img = k.getImage() #normal, full color webcam
depth = k.getDepth() #greyscale depth map
```

###Multiple Cameras And you can even use multiple cameras, at different resolutions:

```
mylaptopcam = Camera(0, {"width": 640, "height": 480})  #you can also control brightness, hue, gain,
myusbcam = Camera(1, {"width": 1280, "height": 720})

mylaptopcam.getImage().save("okaypicture.jpg")
myusbcam.getImage().save("nicepicture.jpg")
```

You can also initialize VirtualCameras from static data files:

```
imgcam = VirtualCamera("apples.jpg", "image")
vidcam = VirtualCamera("bananas.mpg", "video")

imgcam.getImage().save("copy_of_apples.jpg")
imgcam.getImage().save("frame_1_of_bananas.jpg")
```

4.3.4 Colors and Levels

You can also split channels, if you are interested in only processing a single color:

```
(red, green, blue) = Camera().getImage().channels()
red.save("redcam.jpg")
green.save("greencam.jpg")
blue.save("bluecam.jpg")
```

The Image class has a builtin [Histogram](http://en.wikipedia.org/wiki/Image_histogram) function, thanks to [Numpy](http://numpy.scipy.org/). Histograms can show you the distribution of brightness or color in an image:

```
hist = Camera().getImage().histogram(20)
brightpixels = 0
darkpixels = 0
i = 0

while i < length(hist):
  if (i < 10):
    darkpixels = darkpixels + hist[i]
  else:
    brightpixels = brightpixels + hist[i]
  i = i + 1

if (brightpixels > darkpixels):
  print "your room is bright"
else:
  print "your room is dark"
```

4.3.5 Features and FeatureSets

SimpleCV has advanced feature-detection functions, which can let you find different types of features. These are returned in FeatureSets which can be addressed as a group, or filtered:

```
img = Camera.getImage()

lines = img.findLines()

corners = img.findCorners()

lines.draw((255,0,0)) #outline the line segments in red
corners.draw((0,0,255)) #outline corners detected in blue

left_side_corners = corners.filter(corners.x() < img.width / 2)
#only look at corners on the left half of the image

longest_line = lines.sortLength()[0]
#get the longest line returned
```

4.3.6 Blob Detection

If you load the experimental [cvblob-python](https://github.com/oostendo/cvblob-python) library, you can also use SimpleCV to detect blobs:

```
#find the green ball
green_channel = Camera().getImage().splitChannels[1]

green_blobs = green_channel.findBlobs()
#blobs are returned in order of area, largest first

print "largest green blob at " + str(green_blobs[0].x) + ", " + str( green_blobs[0].y)
```

4.3.7 Barcode Reading

If you load the [python-zxing](https://github.com/oostendo/python-zxing) library, you can use [Zebra Crossing](http://code.google.com/p/zxing) to detect 2D and 1D barcodes in a number of various formats. Note that you will need to specify the location of the library either through the ZXING_LIBRARY %ENV variable, or as a parameter to findBarcode():

```
i = Camera().getImage()
barcode = i.findBarcode("/var/opt/zxing")

barcode.draw((0, 255, 0)) #draw the outline of the barcode in green

i.save("barcode_found.png")
print barcode.data
```

4.3.8 Haar Face Detection

You can do Haar Cascade face detection with SimpleCV, but you will need to find your own [Haar Cascade File](http://www.google.com/search?q=haarcascade_frontalface_alt.xml):

```
i = Camera().getImage()
faces = i.findHaarFeatures("/path/to/haarcascade_frontalface_alt.xml")

#print locations
for f in faces:
  print "I found a face at " + str(f.coordinates())

green = (0, 255, 0)
#outline who was drinking last night (or at least has the greenest pallor)
faces.sortColorDistance(green)[0].draw(green)
i.save("greenest_face_detected.png")
```

4.3.9 Output Streams

Rather than use GUI-based display of processed images, SimpleCV has an integrated HTTP-based JPEG streamer. It will use the old-school multipart/replace content type to continuously feed jpgs to your browser. To send the data, you just save the image to the js.framebuffer location:

```
import time
c = Camera()
js = JpegStreamer()  #starts up an http server (defaults to port 8080)
```

```
while(1)
  c.getImage().save(js.framebuffer)
  time.sleep(0.1)
```

You can also use a JpegStreamCamera to grab frames from an external source (such as an IP Cam, the "IP Webcam" android application, or another SimpleCV JpegStream.

jc = JpegStreamCamera("http://localhost:8080/") jc.getImage().save("seeyou.jpg")

CHAPTER

FIVE

VIDEOS

Nate demonstrating the Cookie Jar Alarm Example

CHAPTER

SIX

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*