Computational Imaging Tools: Concepts and Python Codes.

April 24, 2017

0.1 Background: Computational Bioimaging.

1. **What is Computational Imaging?**

   Answer: 'Biology is a highly data driven discipline and one of the primary sources of data in Biology is **imaging**.
   There are various sources of image acquisition:

   - Various forms of microscopy (Optical and electron).
   - Magnetic Resonance Imaging.
   - Other mechanisms for image acquisition.

   After acquisition images are processed to to generate data. i.e. the process to generate data is Image acquisition $\longrightarrow$ Image processing $\longrightarrow$ Data.
   This can be

   - Sometimes simply finding a suitable image demonstrating some property of interest.
   - And Other times this processing can be 'complex pipeline' designed to extract quantitative data[1].'

2. **What is an Image?**

   Answer: An image is an array, or a matrix, of square pixels (picture elements) arranged in columns and rows[2].

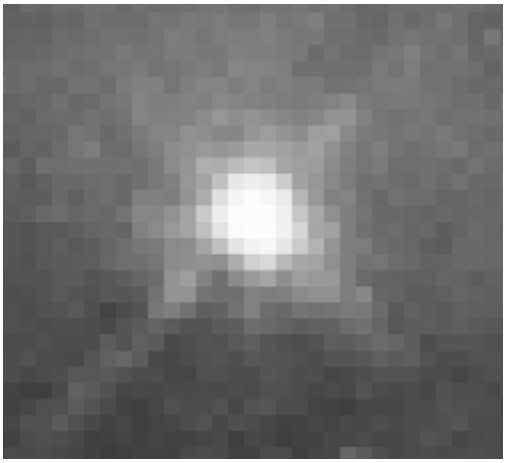

Figure 1: An image — an array or a matrix of pixels arranged in columns and rows[2].

An image is a rectangular grid of pixels. It has a definite height and a definite width counted in pixels. Each pixel is square and has a fixed size on a given display[3].
'In a (8-bit) greyscale image each picture element has an assigned intensity that ranges from 0 to 255. A greyscale image includes many shades of grey.
A normal greyscale image has 8 bit colour depth = 2^8 = 256 greyscales.
A "true colour" image has 24 bit colour depth = 8 x 8 x 8 bits = 256 x 256 x 256 colours = 16 million colours.
Some greyscale images have more greyscales, for instance 16 bit = 65536 greyscales.

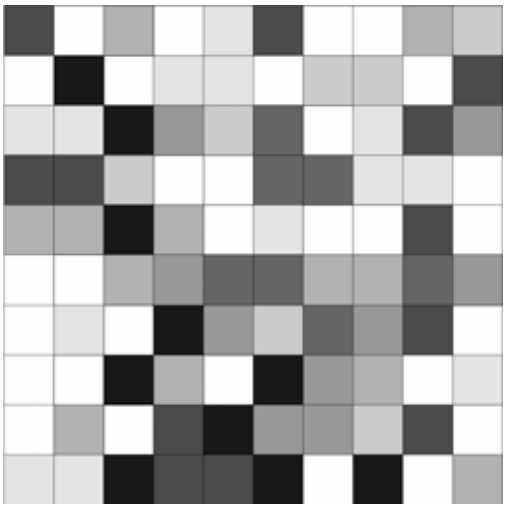

Figure 2: Each pixel has a value from 0 (black) to 255 (white). The possible range of the pixel values depend on the colour depth of the image, here 8 bit = 256 tones or greyscales[2].

In principle three greyscale images can be combined to form an image with 281,474,976,710,656 greyscales[2].'

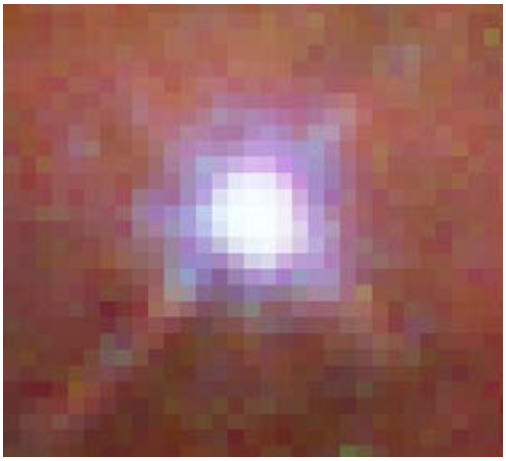

Figure 3: A true-colour image assembled from three greyscale images coloured red, green and blue. Such an image may contain up to 16 million different colours[2].

'There are two general groups of 'images': vector graphics (or line art) and bitmaps (pixel-based or 'images'). Some of the most common file formats are:

1. GIF — an 8-bit (256 colour), non-destructively compressed bitmap format. Mostly used for web. Has several sub-standards one of which is the animated GIF.
2. JPEG — a very efficient (i.e. much information per byte) destructively compressed 24 bit (16 million colours) bitmap format. Widely used, especially for web and Internet (bandwidth-limited).
3. TIFF — the standard 24 bit publication bitmap format. Compresses non-destructively with, for instance, Lempel-Ziv-Welch (LZW) compression.
4. PS — Postscript, a standard vector format. Has numerous sub-standards and can be difficult to transport across platforms and operating systems.
5. PSD – a dedicated Photoshop format that keeps all the information in an image including all the layers[2].'

3. **What is meant by pixel in image processing?**
   Answer: In digital imaging, a pixel, pel, dots, or picture element is a physical point in a raster image, or the smallest addressable element in an all points addressable display device; so it is the smallest controllable element of a picture represented on the screen(Ref reqd).

4. **What is Brightness?**
   Answer: Brightness is a relative term. It depends on your visual perception. Brightness comes into picture when we try to compare with a reference. Intensity refers to the amount of light or the numerical value of a pixel.

5. **What is contrast in image processing?**
   Answer: Contrast is the difference in luminance or colour that makes an object (or its rep-

resentation in an image or display) distinguishable. In visual perception of the real world, contrast is determined by the difference in the color and brightness of the object and other objects within the same field of view(Ref reqd).

6. **What is texture in image processing?**
   Answer: An image texture is a set of metrics calculated in image processing designed to quantify the perceived texture of an image. Image texture gives us information about the spatial arrangement of color or intensities in an image or selected region of an image(Ref reqd).

7. **What is a histogram of an image?**
   Answer: An image histogram is a type of histogram that acts as a graphical representation of the tonal distribution in a digital image. It plots the number of pixels for each tonal value. By looking at the histogram for a specific image a viewer will be able to judge the entire tonal distribution at a glance(Ref reqd).

0.2 Computational Imaging Tools : Image processing techniques using Python.

Computational Imaging Tools Tasks:

1. Make class 'ImageMaker' in python : This includes different functions to create various shapes using numpy arrays. Also, a function 'my_imshow' to plot the numpy array images. etc.
2. Create LPF, HPF, BPF, and Weiner filter.
3. Motion Correction.
4. Thresholding, Binarization and Morphological Tools.
5. Segmentation (shape based, size based, edge based, texture)

8. **Python code: Image_maker.py (numpy arrays to make different shapes.)**

```
import numpy as np
from matplotlib import pyplot as plt
from matplotlib import cm
import scipy
from scipy import signal
from scipy import ndimage
from PIL import Image
% matplotlib inline


def make_img_fft():
    arr = np.random.rand(1024,1024)
    arr = arr>0.99999
    ker = np.zeros_like(arr)
    ker[:50,:20] = 1.
    farr = np.fft.fftn(arr)
    fker = np.fft.fftn(ker)
    return np.abs(np.fft.ifftn(farr*fker))

def make_img(num_obj=100, obj_rad=5, nr=1024, nc=1024):
    arr = np.zeros((nr,nc))
    kerLen = 2*obj_rad
    (X,Y) = np.mgrid[-1:1:1j*kerLen, -1:1:1j*kerLen]
    ker = (np.sqrt(X*X+Y*Y) <= 0.9)
    rand_pos = np.random.rand(num_obj, 2)
    for (x,y) in rand_pos:
        xstart = int(x*(nc-kerLen))
        ystart = int(y*(nr-kerLen))
        arr[xstart:xstart+kerLen, ystart:ystart+kerLen] += ker
    return arr

def make_sqrimg_fft():
    arr = np.random.rand(1024,1024)
    arr = arr>0.99999
    ker = np.zeros_like(arr)
    ker[:30,:30] = 1.
    farr = np.fft.fftn(arr)
    fker = np.fft.fftn(ker)
    return np.abs(np.fft.ifftn(farr*fker))

def make_roundimg_fft():
    arr = np.random.rand(1024, 1024)
    arr = arr>0.99999
    ker = np.zeros_like(arr)
    y,x = np.ogrid[-512:511+1, -512:511+1]
    mask = x**2 + y**2 <= 15**2
```

```
    ker[mask] = 1
    farr = np.fft.fftn(arr)
    fker = np.fft.fftn(ker)
    return np.abs(np.fft.ifftn(farr*fker))

def make_trilimg_fft():
    arr = np.random.rand(1024,1024)
    arr = arr>0.99999
    #ker = np.zeros_like(arr)
    ker = np.tri(1024, 1024, 990)
    farr = np.fft.fftn(arr)
    fker = np.fft.fftn(ker)
    return np.abs(np.fft.ifftn(farr*fker))

def my_imshow(arr):
    f,ax = plt.subplots()
    ax.imshow(arr, cmap=cm.gray, interpolation='none')
    plt.show()


## To plot Kernels of different shapes.

img1 = make_img_fft()
my_imshow(img1)

img2 = make_sqrimg_fft()
my_imshow(img2)

img3 = make_roundimg_fft()
my_imshow(img3)

img4 = make_trilimg_fft()
my_imshow(img4)

my_imshow(img3 + img2 + img1 + img4)


## To make Low pass filter.

img = make_img(num_obj=4, obj_rad=6, nr=128, nc=128)
my_imshow(img)

ker = make_gaussian_kernel(lpf=.8)
my_imshow(ker)

my_imshow((convolution_filter(img, ker)))
```

Python code: Class_img_maker.py

```
class img_maker(object):
    def __init__(self):
        self.img = None
        self.ker = None

    def make_img(self, num_obj=100, obj_rad=5, nr=1024, nc=1024):
        arr = np.zeros((nr,nc))
        kerLen = 2*obj_rad
        (X,Y) = np.mgrid[-1:1:1j*kerLen, -1:1:1j*kerLen]
        ker = (np.sqrt(X*X+Y*Y) <= 0.9)
        rand_pos = np.random.rand(num_obj, 2)
        for (x,y) in rand_pos:
            xstart = int(x*(nc-kerLen))
            ystart = int(y*(nr-kerLen))
            arr[xstart:xstart+kerLen, ystart:ystart+kerLen] += ker
        self.img = arr.copy()

    def make_gaussian_kernel(self, nr=10,nc=10,lpf=2.):
        (X,Y) = 1.*np.mgrid[-1:1:1j*nc,-1:1:1j*nr]
        ker = np.exp(-(X*X+Y*Y)/(lpf*lpf) )
        self.ker = ker/ker.max()

    def apply_convolution_filter(self):
        (k_shy, k_shx) = self.ker.shape
        padded_ker = np.zeros_like(self.img)
        padded_ker[:k_shy, :k_shx] = self.ker
        farr = np.fft.fftn(self.img)
        fker = np.fft.fftn(padded_ker)
        self.img = np.abs(np.fft.ifftn(farr*fker))

    def imshow_img(self):
        f,ax = plt.subplots()
        ax.imshow(self.img, cmap=cm.gray, interpolation='none')
        plt.show()

## Using Class img_maker
my_im1 = img_maker()

my_im1.make_img()

my_im1.make_gaussian_kernel(nr=40,nc=40,lpf=.7)
my_im1.apply_convolution_filter()

my_im1.imshow_img()
```

9. **What is the purpose of filtering in image processing?**
   Answer: Filtering is a technique for modifying or enhancing an image. For example, to emphasize certain features or remove unwanted features. Filtering in image processing is implemented with **Low pass filter**, **High pass filter**, **Band pass filter** and **Wiener filter**.
   Image processing operations implemented with filtering include smoothing, sharpening, and edge enhancement.

   - **Smoothing:** Smoothing is often used to reduce noise within an image or to produce a less pixelated image. Most smoothing methods are based on low pass filters. Smoothing is also usually based on a single value representing the image such as the average value the median value of the image(Ref reqd).

   - **Sharpening:** Image sharpening is a powerful tool for emphasizing texture and drawing viewer focus. Digital camera sensors and lenses always blur an image to some degree, and this requires correction.

   - **Edge detection:** This technique is used to find the boundaries of objects within images. It works by detecting discontinuities in brightness. Edge detection is used for image segmentation and data extraction.

   1. **Low pass filter(LPF):** Low pass filter is used for smoothing noise. LPF reduces noise but accentuates blurring. It filters out high frequency noise.

   2. **High pass filter(HPF):** High pass filter is used for image sharpening and edge detection. It is just the opposite of low pass filter. It allows high frequency components of the image to pass through and filters out low frequency noise(Ref reqd). If there is no change intensity then nothing happens. But if one pixel is brighter than its immediate neighbours then it gets boosted.
      HPF reduces blurring but accentuates noise. i.e. Though HPF can often improve an image by sharpening the details, overdoing it can actually degrade the image quality significantly(Ref reqd).

   3. **Band pass filter(BPF):** Band pass filtering is a tradeoff between Blurring vs. Noise. It boosts certain midrange frequencies and partially corrects for blurring, but does not boost very high (most noise corrupted) frequencies.

   4. **Wiener filter:** Wiener filtering is the most important technique for removal of blur in images due to linear motion or unfocussed optics[7]. The Wiener filter is Mean Square Error(MSE)- optimal stationary linear filter for images degraded by additive noise and blurring[8].

   These filters are in the form of matrices and they are called as **'Kernels'** in image processing.

Python code: LPFforNoise.py (Adding noise to an image and applying low pass filter to it.)

```
## To create noisy image and applying LPF on it.

import skimage
from skimage import feature

#Creating an Image and kernel.
img = make_img(num_obj=4, obj_rad=10, nr=128, nc=128)
ker = make_gaussian_kernel(lpf=.8)
my_imshow(img)


#Add noise to the image.
noisy_img = skimage.util.random_noise(img,mode='gaussian',seed=None,clip=True)
#noisy_img = skimage.util.random_noise(img,mode='speckle',seed=None,clip=True)
#noisy_img = skimage.util.random_noise(img,mode='localvar',seed=None,clip=True)
#noisy_img = skimage.util.random_noise(img,mode='poisson',seed=None,clip=True)
#noisy_img = skimage.util.random_noise(img,mode='salt',seed=None,clip=True)
#noisy_img = skimage.util.random_noise(img,mode='pepper',seed=None,clip=True)
#noisy_img = skimage.util.random_noise(img, mode='s&p', seed=None, clip=True)

my_imshow(noisy_img)

#Apply LPF to the noisy image.
my_imshow((convolution_filter(noisy_img, ker)))
```

Python code: HPF.py (High Pass Filter)

```
## High-pass filters.

img1 = make_img_fft()
my_imshow(img1)

ker = np.array([[-1, -1, -1, -1, -1],
                [-1,  1,  2,  1, -1],
                [-1,  2,  4,  2, -1],
                [-1,  1,  2,  1, -1],
                [-1, -1, -1, -1, -1]])


ker2 = np.array([[-1, -1, -1],
                 [-1,  8, -1],
                 [-1, -1, -1]])

my_imshow(ker)
my_imshow(ndimage.convolve(img1, ker))
```

Python code: HPF2.py (High Pass Filter using different kernel.)

```
im = Image.open('lena.png')
data = np.array(im, dtype=float)
my_imshow(data)

ker2 = np.array([[-1, -1, -1],
                 [-1,  8, -1],
                 [-1, -1, -1]])
my_imshow(ker2)
my_imshow(ndimage.convolve(data, ker2))
```

Python code: HPFforNoise.py (To create noisy image and applying High Pass Filter to it.)

```
## To create noisy image and applying HPF on it.

import skimage
from skimage import feature

#Creating an Image and kernel.
img = make_img(num_obj=4, obj_rad=10, nr=128, nc=128)
my_imshow(img)

ker = np.array([[-1, -1, -1, -1, -1],
                [-1,  1,  2,  1, -1],
                [-1,  2,  4,  2, -1],
                [-1,  1,  2,  1, -1],
                [-1, -1, -1, -1, -1]])


ker2 = np.array([[-1, -1, -1],
                 [-1,  8, -1],
                 [-1, -1, -1]])


#Add noise to the image.
noisy_img = skimage.util.random_noise(img, mode='s&p', seed=None, clip=True)
my_imshow(noisy_img)

#Apply HPF to the noisy image.
my_imshow((convolution_filter(noisy_img, ker)))
```

Python code: BPF.py (Band Pass Filter)

```
## Band Pass filters with Sobel kernels.

img1 = make_img_fft()
my_imshow(img1)

img2 = make_sqrimg_fft()
#my_imshow(img2)

img3 = make_roundimg_fft()
#my_imshow(img3)

img4 = make_trilimg_fft()
#my_imshow(img4)


## Band-Pass filter with right-sobel
ker_right = np.array([[-1, 0, 1],
                      [-2, 0, 2],
                      [-1, 0, 1]])

## Band-Pass filter with left-sobel
ker_left = np.array([[1, 0, -1],
                     [2, 0, -2],
                     [1, 0, -1]])

## Band-Pass filter with top-sobel
ker_top = np.array([[1, 2, 1],
                    [0, 0, 0],
                    [-1, -2, -1]])

## Band-Pass filter with bottom-sobel
ker_bottom = np.array([[-1, -2, -1],
                       [0, 0, 0],
                       [1, 2, 1]])

my_imshow(ker_right)
my_imshow(ker_left)
my_imshow(ker_top)
my_imshow(ker_bottom)

my_imshow(ndimage.convolve(img1, ker_right))
my_imshow(ndimage.convolve(img1, ker_left))
my_imshow(ndimage.convolve(img1, ker_top))
my_imshow(ndimage.convolve(img1, ker_bottom))
```

```
#Add noise to the image.
import skimage
from skimage import feature

noisy_img = skimage.util.random_noise(img, mode='s&p', seed=None, clip=True)

my_imshow(noisy_img)

#Apply BPF to the noisy image.
my_imshow((convolution_filter(noisy_img, ker_right)))
```

Python code: WienerFilter.py (Wiener Filter)

```
#Applying Wiener filter to blurred (Low pass filtered) image.

#create an image.
img = make_img(num_obj=4, obj_rad=6, nr=128, nc=128)
my_imshow(img)

#Low pass filter kernel.
ker = make_gaussian_kernel(lpf=1)
my_imshow(ker)

#Convolution of LPF kernel and image.
blurred_img = (convolution_filter(img, ker))
my_imshow(blurred_img)

#Applying wiener filter.
WF2 = scipy.signal.wiener(blurred_img, mysize=None, noise=5)
my_imshow(WF2)

#Deconvolution for restoration of  convoluted image.
deconvolved_img_out, _ = restoration.unsupervised_wiener(blurred_img, ker)
my_imshow(deconvolved_img_out)
```

10. **What is an Image Registration?**
Answer:
Image registration is the process of aligning two or more images of the same scene. This process involves designating one image as the reference image, also called the fixed image, and applying geometric transformations or local displacements to the other images so that they align with the reference.
Images can be misaligned for a variety of reasons. Commonly, images are captured under variable conditions that can change the camera perspective or the content of the scene. Misalignment can also result from lens and sensor distortions or differences between capture devices.

Image registration is often used as a preliminary step in other image processing applications. It enables you to compare common features in different images[4].

Phase correlation or Cross-correlation.
The example code given in scikit-image processing package has 'register_translation' function defined for phase correlation.
The register_translation function uses cross-correlation in Fourier space, optionally employing an upsampled matrix-multiplication DFT to achieve arbitrary subpixel precision[5].

In signal processing cross-correlation is a measure of similarity of two series as a function of the displacement of one relative to the other. This is also known as **Inner dot product**. The cross correlation similar in nature to the convolution of two functions.

Applications : Cross-correlation has applications in pattern recognition, single particle analysis, Electron tomography, Averaging, Cryptanalysis and Neurophysiology[6].

Algorithm for phase correlation (using register_translation function from[5]) to identify the relative shift between two similar sized images. (This algorithm is based on the python code for phase-correlation given here[5].)

1. Import an image. (This is a reference image.)

2. Define a **shift** to be added. (To shift the reference image- This gives an offset-image.)

3. Obtain an **offset_image** by following steps:
   (a) Compute N-dimensional DFT of an image.
   (b) Apply a **fourier_shift** filter to N-dimensional DFT of an image shift. $\longrightarrow$ fourier_shift(np.fft.fftn(imag shift)

4. Obtain an inverse discrete fourier transform of **offset_image**.

5. Perform a **register_translation** operation on image and offset_image. This does efficient subpixel image translation registration be cross correlation.

6. Plot image and offset_image.

7. To show output of cross-correlation to show behind the scenes tasks done by algorithm.
   (a) Take 2-dimensional DFT of an image.
   (b) Take 2-dimensional DFT of conjugate of **offset_image**.
   (c) Obtain **'image_product'** by convolution of 1 and 2 above.

8. Obtain a **cc_image**(cross correlated image).
   (a) Take 2-Dimensional DFT of **'image_product'**.
   (b) Obtain a shifted array by shifting the zero frequency component to the center of the spectrum. This is done using **np.fft.fftshift** function on **'image_product'** obtained in above step. This gives the cross-correlated image.

9. For Subpixel precision perform **register_translation** operation on image and an offset_image with upsample_factor set to specific integer. This does efficient subpixel image translation registration for cross correlation.

Python code: DriftCorrection.py (Image registration using phase correlation method. This python code for phase-correlation is sourced from here[5].)

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm

import scipy
from scipy import signal
from scipy import ndimage
from scipy.ndimage import fourier_shift           # Multidimensional Fourier sh

from skimage import data
from skimage.feature import register_translation  #Efficient subpixel image tran
from skimage.feature.register_translation import _upsampled_dft

from PIL import Image

# Define an image.

def make_img(num_obj=100, obj_rad=5, nr=1024, nc=1024):
    arr = np.zeros((nr,nc))
    kerLen = 2*obj_rad
    (X,Y) = np.mgrid[-1:1:1j*kerLen, -1:1:1j*kerLen]
    ker = (np.sqrt(X*X+Y*Y) <= 0.9)
    rand_pos = np.random.rand(num_obj, 2)
    for (x,y) in rand_pos:
        xstart = int(x*(nc-kerLen))
        ystart = int(y*(nr-kerLen))
        arr[xstart:xstart+kerLen, ystart:ystart+kerLen] += ker
    return arr

##Define or import an image.

image = make_img(num_obj=8, obj_rad=8, nr=512, nc=512)
plt.imshow(image)

## The shift corresponds to the pixel offset relative to the reference image.
shift = (-22.4, 13.32)

## fourier_shift(input, shift) = Multi-dimensional fourier shift filter.
#The array is multiplied with the fourier transform of a shift operation.
#offset_image is a frequency domain shift of reference image.
offset_image = fourier_shift(np.fft.fftn(image), shift)  # Compute the N-dimens
offset_image = np.fft.ifftn(offset_image) #Computes the N-dimensional inverse d
print("Known offset (y, x): {}".format(shift))
```

```
## pixel precision first
shift, error, diffphase = register_translation(image, offset_image)
#Efficient subpixel image
                                                                    #translatio

fig = plt.figure(figsize=(8, 3))
ax1 = plt.subplot(1, 3, 1, adjustable='box-forced')
ax2 = plt.subplot(1, 3, 2, sharex=ax1, sharey=ax1, adjustable='box-forced')
ax3 = plt.subplot(1, 3, 3)

ax1.imshow(image, cmap='gray')
ax1.set_axis_off()
ax1.set_title('Reference image')

ax2.imshow(offset_image.real, cmap='gray')
ax2.set_axis_off()
ax2.set_title('Offset image')

# Show the output of a cross-correlation to show what the algorithm is
# doing behind the scenes.
image_product = np.fft.fft2(image) *  np.fft.fft2(offset_image).conj()
cc_image = np.fft.fftshift(np.fft.ifft2(image_product))
ax3.imshow(cc_image.real)
ax3.set_axis_off()
ax3.set_title("Cross-correlation")
plt.show()

print("Detected pixel offset (y, x): {}".format(shift))

# subpixel precision
shift, error, diffphase = register_translation(image, offset_image, 100)

fig = plt.figure(figsize=(8, 3))
ax1 = plt.subplot(1, 3, 1, adjustable='box-forced')
ax2 = plt.subplot(1, 3, 2, sharex=ax1, sharey=ax1, adjustable='box-forced')
ax3 = plt.subplot(1, 3, 3)

ax1.imshow(image, cmap='gray')
ax1.set_axis_off()
ax1.set_title('Reference image')

ax2.imshow(offset_image.real, cmap='gray')
ax2.set_axis_off()
ax2.set_title('Offset image')

# Calculate the upsampled DFT, again to show what the algorithm is doing
# behind the scenes.  Constants correspond to calculated values in routine.
```

```
cc_image = _upsampled_dft(image_product, 150, 100, (shift*100)+75).conj()
ax3.imshow(cc_image.real)
ax3.set_axis_off()
ax3.set_title("Supersampled XC sub-area")
plt.show()

print("Detected subpixel offset (y, x): {}".format(shift))
```

Python code: Thresholding.py

References

[1] Numpy functions.
http://training.scicomp.jic.ac.uk/docs/bioimaging_course_book/computational_imaging.html

[2] What is an Image?
http://www.cafeaulait.org/course/week9/23.html

[3] Image Processing Toolbox.
https://www.mathworks.com/help/images/approaches-to-registering-images.html

[4] Cross-Correlation(Phase Correlation).
http://scikit-image.org/docs/dev/api/skimage.feature.html#skimage.feature.register_translation

[5] Cross-correlation : Wikipedia.
https://en.wikipedia.org/wiki/Cross-correlation

[6] Wiener filter.
https://www.clear.rice.edu/elec431/projects95/lords/wiener.html

[7] Wiener filter.
http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/VELDHUIZEN/node15.html