

# 1 Python correction



```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 from mpl_toolkits.mplot3d import Axes3D
4 from skimage import io
5 from scipy import ndimage
6 from matplotlib import cm

```

## 1.1 Main function

This code is used as the main function to perform shape-from-focus reconstruction.



```

volumes = [ 'cornee' , 'vickers' ];
2 for v in volumes:
    I = io.imread( 'volume_'+v+'.tif' );
4     I = I.astype( 'float' );
    F = np.zeros( I.shape )
5     N = 11;

8     myfunctions = [ variance , tenengrad , sml ];
9     for f in myfunctions:
10         for i,im in enumerate(I):
11             F[ i ] = f(im, N);

12     # Evaluates altitudes and textures
13     Z = np.argmax(F, axis=0);
14     Z = ndimage.minimum_filter(Z, size=5);
15     T = extractTexture(I, Z);
16

```

The extraction of the texture from the altitudes (indexes) in the stack of images is performed by the following function:



```

1 def extractTexture(I, Z):
2     """
3         Extract texture from stack of images I, where Z is the index (
4             ↪ altitude)
5         I: stack of images, of shape (n, X, Y)
6         Z: index of SFF maximum (of shape (X,Y)), values are between 0 and n
7             ↪ -1
8         returns basically I[Z(i,j), i, j] for all (i,j)
9     """
10    m,n = I.shape[1:]
11    ii , jj = np . ogrid [:m,:n]
12    T = I [Z, ii , jj];
13    return T;

```

## 1.2 Sum of Modified Laplacian

Results are illustrated in Fig.1.



```

1 def sml(I, N):
2     """
3         SFF measure, SUM of modified Laplacian
4         I: image
5         N: neighborhood size
6         returns: SFF measure for each pixel
7     """
8     h = np . array ([[ -1 , 2 , -1 ]]);
9     ML = np . abs(ndimage . convolve(I, h)) + np . abs(ndimage . convolve(I, np .
10         ↪ transpose(h)));
11     S=ndimage . uniform_filter(ML, N);
12     return S;

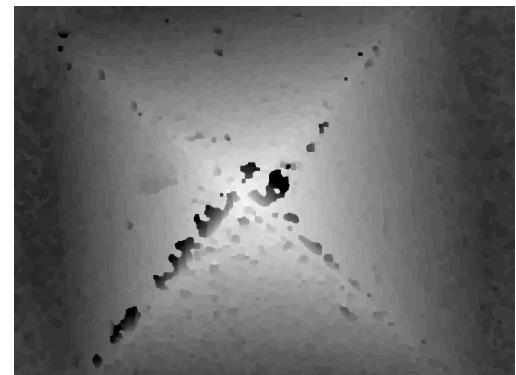
```

## 1.3 Variance

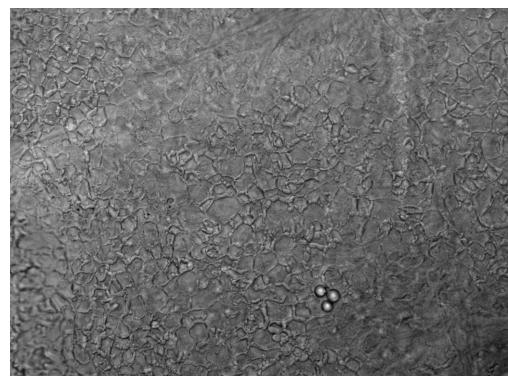
The focus measure based on the variance is a really simple method that works in most cases, see Fig.2.



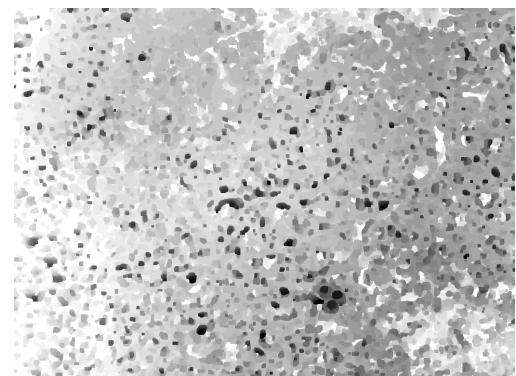
(a) Texture.



(b) Altitudes.

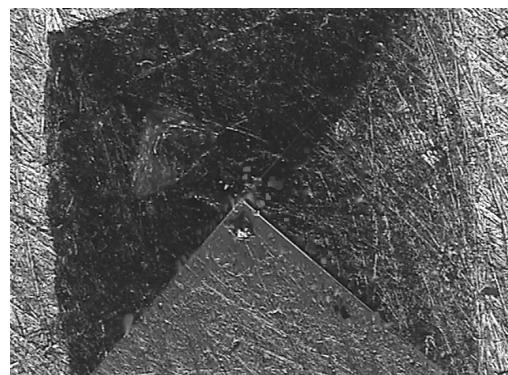


(c) Texture.

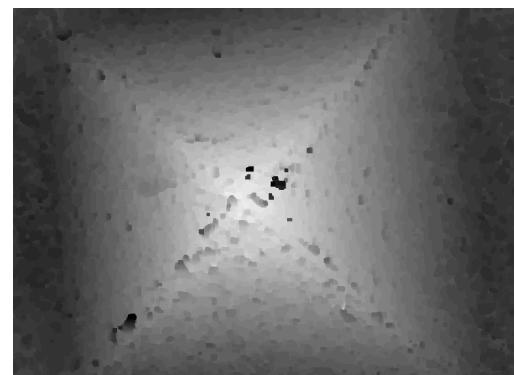


(d) Altitudes.

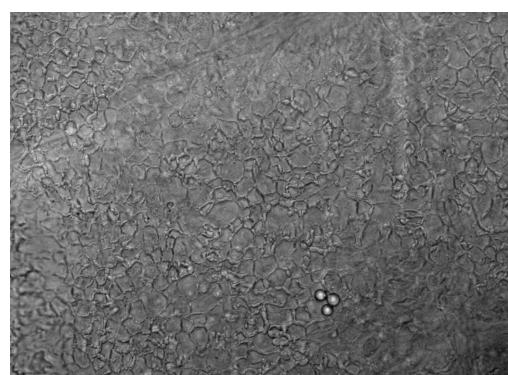
Figure 1: Texture and altitude reconstruction with the SML method.



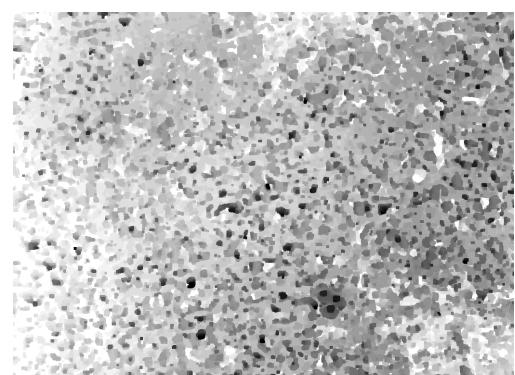
(a) Texture.



(b) Altitudes.



(c) Texture.



(d) Altitudes.

Figure 2: Texture and altitude reconstruction with the variance method.



```

1 def variance(I, N):
2     """
3         SFF measure
4         I: image
5         N: neighborhood size
6         returns: SFF measure for each pixel, results is the same shape as I
7     """
8     M = ndimage.uniform_filter(I, N);
9     D2 = (I - M) ** 2;
10    V = ndimage.uniform_filter(D2, N);
11    return V;

```

## 1.4 Tenengrad

The tenengrad method is base on a Sobel filter, see Fig.3.



```

1 def tenengrad(I, N):
2     """
3         SFF measure, Tenengrad method
4         I: image
5         N: neighborhood size
6         returns: SFF measure for each pixel, results is the same shape as I
7     """
8     Sx = ndimage.sobel(I, axis=0);
9     Sy = ndimage.sobel(I, axis=1);
10    S = np.hypot(Sx, Sy);
11    T = ndimage.uniform_filter(S, N);
12    return T;

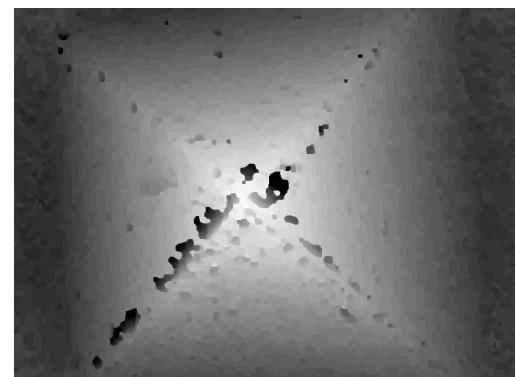
```

## 1.5 Variance of Tenengrad

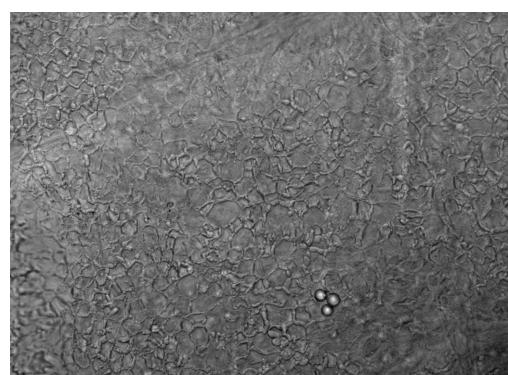
The variance of Tenengrad is an improvement of the Tenengrad method, see Fig.4.



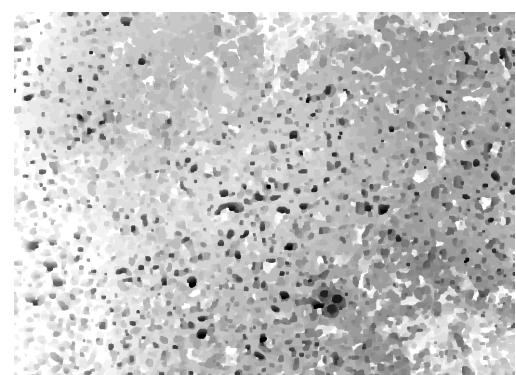
(a) Texture.



(b) Altitudes.



(c) Texture.

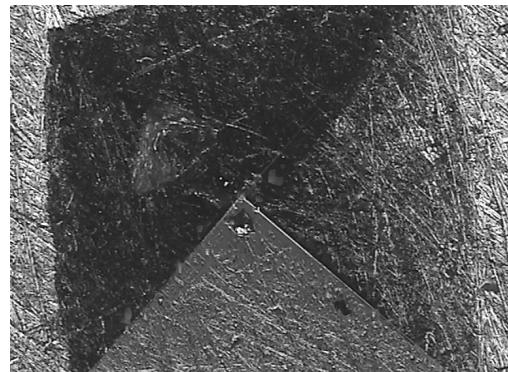


(d) Altitudes.

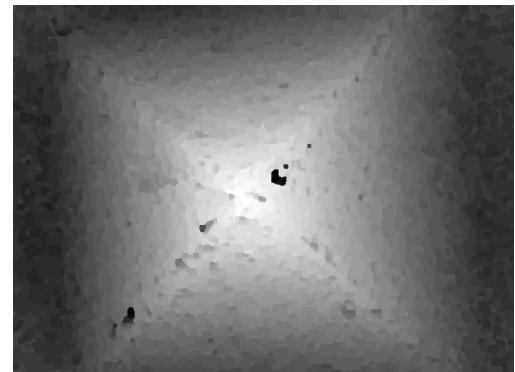
Figure 3: Texture and altitude reconstruction with the SML method.



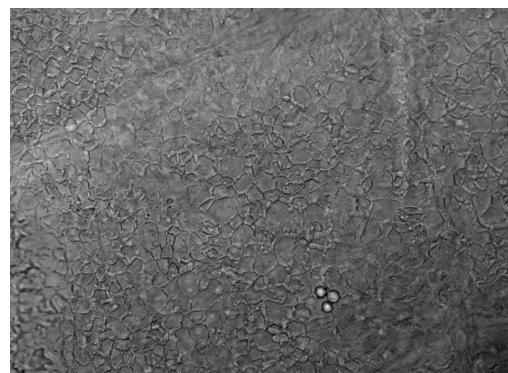
```
def varianceTenengrad(I, N):
    """
    SFF measure, variance of Tenengrad
    I: image
    N: neighborhood size
    returns: SFF measure for each pixel
    """
    Sx = ndimage.sobel(I, axis=0);
    Sy = ndimage.sobel(I, axis=1);
    S = np.hypot(Sx, Sy);
    vt = variance(S, N);
    return vt;
```



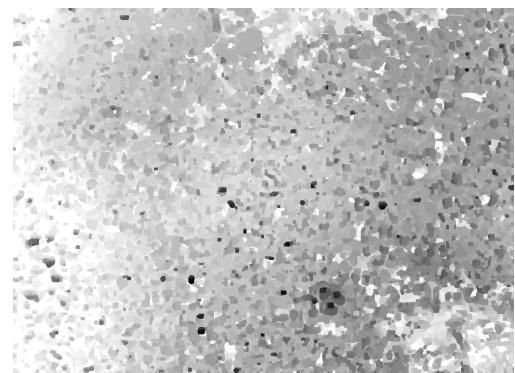
(a) Texture.



(b) Altitudes.



(c) Texture.



(d) Altitudes.

Figure 4: Texture and altitude reconstruction with the variance of Tenengrad method.