1 Python correction

The different imports used in this tutorial are:

```python
import numpy as np

import scipy.ndimage
import imageio # imread and imwrite
import matplotlib.pyplot as plt
import skimage.measure # some geometrical descriptors

# for reading files
import glob

from sklearn.cluster import KMeans
```

1.1 Geometrical functionals

The Crofton perimeter is defined by multiple projections, as well as the Feret diameter. Be careful while performing the rotation of the object (as it is a binary object, the interpolation method could introduce non integer values).

```python
def crofton_perimeter(I):
    """ Computation of crofton perimeter
    """
    inter = [];
    h = np.array([[1, -1]]);
    for i in range(4):
        II = np.copy(I);
        I2 = scipy.misc.imrotate(II, 45*i,interp='nearest');
        I3 = scipy.ndimage.convolve(I2, h);

        inter.append(np.sum(I3>100));


    crofton = np.pi/4. * (inter[0]+inter[2] + (inter[1]+inter[3])/np.sqrt
        ↪ (2));
    return crofton
```

```python
def feret_diameter(I):
    """
    Computation of the Feret diameter
    minimum: d (meso-diameter)
    maximum: D (exo-diameter)

    Input: I binary image
    """
    d = np.max(I.shape);
    D = 0;

    for a in np.arange(0, 180, 30):
        I2 = scipy.misc.imrotate(I, a, interp='nearest');
        F = np.max(I2, axis=0);
        measure = np.sum(F>100);

        if (measure<d):
            d = measure;
        if (measure>D):
            D = measure;
    return d,D;
```

The inscribed circle is just the maximum of the distance transform inside the object. The distance map for an image apple is shown in Fig.1.

```python
def inscribedRadius(I):
    """
    computes the radius of the inscribed circle
    """
    dm = scipy.ndimage.morphology.distance_transform_cdt(I>100);
    radius = np.max(dm);
    return radius;
```

The smallest enclosing circle is computed by using the code from the Project Nayuki[1] published under the GNU Lesser General Public License. The result is presented in Fig.2.

[1] https://www.nayuki.io/page/smallest-enclosing-circle

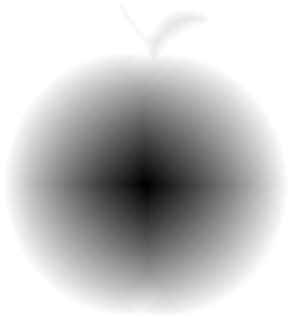

Figure 1: Distance map of an object apple. The inverse is actually displayed in order to see correctly the progression. The maximum of the distance map is the radius of the inscribed circle.

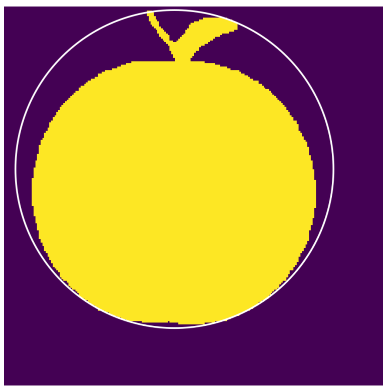

Figure 2: Circumscribed circle.

```python
def circumCircle(I):
    """
    this version uses a function provided by Project Nayuki
    under GNU Lesser General Public License
    """
    points = np.argwhere(I > 100);
    c = smallestenclosingcircle.make_circle(points);
    return c;
```

1.2 Shape diagrams

The shape diagrams are constructed by reading all the images and computing the shape descriptors.

```python
def diagrams():
    name=['apple-*.bmp', 'Bone-*.bmp', 'camel-*.bmp'];
    elongation=[];
    thinness=[];
    roundness=[];
    z=[];
    for pattern in name:
        namesList = glob.glob(pattern);
        for fichier in namesList:
            I = imageio.imread(fichier);
            radius = inscribedRadius(I);
            d,D = feret_diameter(I);
            crofton = crofton_perimeter(I);

            elongation.append(d/D);
            thinness.append(2*radius / D);
            roundness.append(4*np.sum(I>100)/(np.pi * D**2));
            z.append(crofton / (np.pi * D));


    plt.plot(elongation[0:20], thinness[0:20], "o", label='Apple')
    plt.plot(elongation[20:40], thinness[20:40], "+", label='Bone')
    plt.plot(elongation[40:60], thinness[40:60], ".", label='Camel')
    plt.legend(name)
    plt.show
    evaluateQuality(elongation, thinness);

    plt.figure();
    plt.plot(z[0:20], roundness[0:20], "o", label='Apple')
    plt.plot(z[20:40], roundness[20:40], "+", label='Bone')
    plt.plot(z[40:60], roundness[40:60], ".", label='Camel')
    plt.legend(name)
    plt.show
    evaluateQuality(z, roundness);

    plt.figure();
    plt.plot(thinness[0:20], z[0:20], "o", label='Apple')
    plt.plot(thinness[20:40], z[20:40], "+", label='Bone')
    plt.plot(thinness[40:60], z[40:60], ".", label='Camel')
    plt.legend(name)
    plt.show
    evaluateQuality(thinness, z);
```

1.3 Shape classification

The following code evaluates the quality by comparing the known class of the shape with the segmented (via the kmeans method) class. The result is illustrated in Fig.3 with an accurary of 98.3%.

```python
def evaluateQuality(x, y):
    global i
    n = 3;
    k_means = KMeans(init='k-means++', n_clusters=n)
    X = np.asarray(x);
    Y = np.asarray(y);
    pts = np.stack((X, Y));
    pts = pts.T;
    #print(pts)
    k_means.fit(pts);

    k_means_labels = k_means.labels_;
    k_means_cluster_centers = k_means.cluster_centers_;

    # plot
    fig = plt.figure()
    colors = ['#4EACC5', '#FF9C34', '#4E9A06']

    # KMeans
    for k, col in zip(range(n), colors):
        my_members = k_means_labels == k
        cluster_center = k_means_cluster_centers[k]
        plt.plot(pts[my_members, 0], pts[my_members, 1], 'o',
                 markerfacecolor=col, markersize=6)
        plt.plot(cluster_center[0], cluster_center[1], 'o',
            ↪ markerfacecolor=col,
                 markeredgecolor='k', markersize=12)
    plt.title('KMeans')
    plt.show()
    fig.savefig("kmeans"+str(i)+".pdf");
    i += 1;

    """
    Evaluation of the quality: count the number of shapes correctly
        ↪ detected
    """
    accuracy = np.sum(k_means_labels[0:20]  == scipy.stats.mode(
        ↪ k_means_labels[0:20]));
    accuracy +=np.sum(k_means_labels[20:40] == scipy.stats.mode(
        ↪ k_means_labels[20:40]));
    accuracy +=np.sum(k_means_labels[40:60] == scipy.stats.mode(
        ↪ k_means_labels[40:60]));
    accuracy = accuracy / 60 * 100;
    print('Accuracy: {0:.2f}%'.format(accuracy));
```

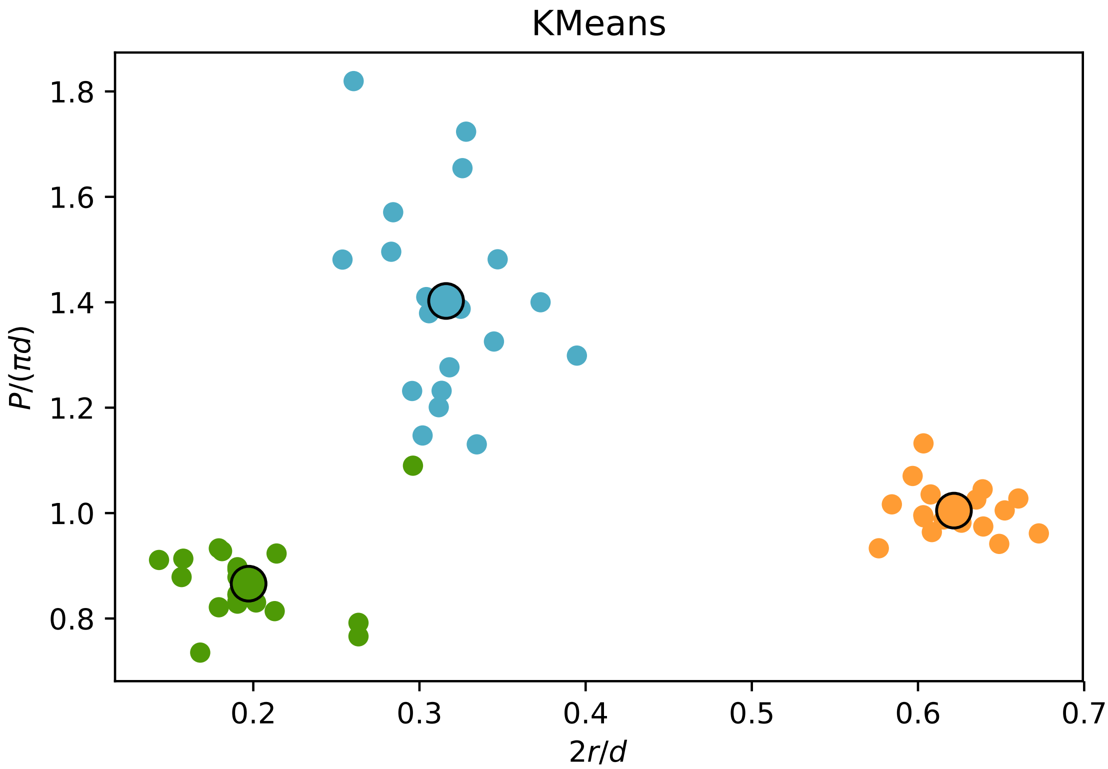

Figure 3: Illustration of the accuracy of the classification from a k-means method. The k-means method is not necessarily the adapted to these data. The measured accurary if of 98.3%.