"Differentiating Various Methods used for Image Segmentation"

Rajat Tiwari

June 2020

Abstract:

In this paper, we will deeply discuss about Edge Based,Partial Differential Equation based,Watershed Method for Image Segmentation.As we know, Image Processing and Computer Vision deals with researches in the field of Image Segmentation and Image Recognization.Image Segmentation is the process of partitioning a digital image into multiple segments.Image processing is any form of information processing for which the input is an image, such as frames of video; the output is not necessarily an image, but can be a set of features giving information about the image.

1 Edge Detection Based Image Segmentation

Edge Detection is a very well-developed field in Image Analysis. Edges typically correspond to points in the image where the gray value changes significantly from one pixel to the next. Thus, detecting Edges help in extracting useful information characteristics of the image where there are abrupt changes.Edges are those point where their is a change in pixel density.
Region boundaries and edges are closely related to each other and there is a sharp adjustment in intensity at the boundaries.As we know, we need closed boundaries to segment foreground from background but the edges detected are not connected.So we need to extend the boundaries detected. Segmentation can be applied to region detected or region acquired by spatial-taxons.

Edge detection is a process of locating an edge of an image. Detection of edges in an image is a very important step towards understanding image features. Edges consist of meaningful features and contained significant information.Since edges often occur at image locations representing object boundaries, edge detection is extensively used in image segmentation when images are divided into areas corresponding to different objects.

Categories of Edge Detection based Image Segmentation are:

- **Gradient Edge Based Segmentation**

- **Laplacian Edge Based Segmentation**

  Pfaltz in 1966 gave an efficient algorithm for doing this for 4- an 8-connected regions, termed a connected components algorithm.The algorithm operates on a raster scan, in which each pixel is visited in turn, starting at the top-left corner of the image and scanning along each row, finishing at the bottom-right corner.

  Steps for Edge Detection Method

  Step 1: Read input image
  Step 2: Apply horizontal mask Gx and vertical mask Gy to the input image
  Step 3: Apply different edge detection algorithms and find the gradient
  Step 4: Construct separate image for Gx and Gy
  Step 5: Results are combined to find the absolute magnitude of the gradient.
  Step 6: The absolute magnitude is the output slope magnitude image
  Step 7: For some slope magnitude images, the pixels values are too small or too high.

  To improve visibility of those images, scaling has to be done .For small values, it has to be scaled up by appropriate factor. For large values, it has to be scaled down by appropriate factor.

1.1 Gradient Edge Detection based Image Segmentation

The gradient method detect the edges by looking for the maximum and minimum in the first derivative of the image.
Types of Gradient Edge Detection Method are:
Canny, Prewitt, Sobel operators detect vertical and horizontal edges. Sharp edges can be separated out by appropriate thresholding.

1.1.1 Sobel Gradient Edge Detection Method -using Sobel operator

The Sobel operator performs a 2-D spatial gradient measurement on an image and so emphasizes regions of high spatial frequency that correspond to edges. Typically it is used to find the approximate absolute gradient magnitude at each point in an input grayscale image.

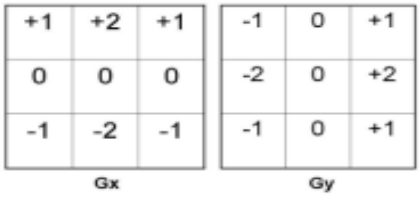


Figure 3.1 Sobel convolution kernels

This implies that the result of the Sobel operator at an image point which is in a region of constant image intensity is a zero vector and at a point on an edge is a vector which points across the edge, from darker to brighter values.
Edge detection involves mathematical methods to find points in an image where the brightness of pixel intensities changes distinctly.

1.1.2 Prewitt Gradient Edge Detection Method

Prewitt operator is method of edge detection in image processing which calculates the maximum response of a set of convolution kernels to find the local edge orientation for each pixel. The Prewitt edge detector is an appropriate way to estimate the magnitude and orientation of an edge.
Although differential gradient edge detection needs a rather time-consuming calculation to estimate the orientation from the magnitudes in the x- and y-directions. For Prewitt operator, one kernel being sensitive to edges in the vertical direction and one to the horizontal direction

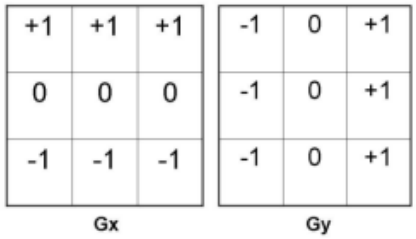


Figure 3.6 Prewitt convolution kernels (3x3)

1.1.3 Kirsch Gradient Edge Detection Method

Kirsch edge operators have been proposed for image segmentation of mammographic images.The kirsch edge detector detects edges using eight filters are applied to the image with the maximum being retained for the final image. The eight filters are a rotation of a basic compass convolution filter.

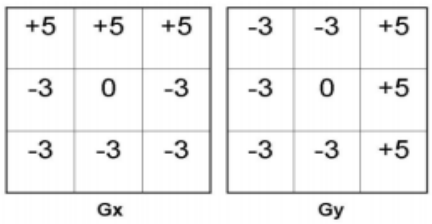


Figure 3.11 Kirsch convolution kernels (3x3)

1.1.4 Canny Gradient Edge Detection Method

Canny edge detection is a multi-step algorithm that can detect edges with noise supressed at the same time.Canny edge detection is a technique to extract useful structural information from different vision objects and dramatically reduce the amount of data to be processed.
It has been widely applied in various computer vision systems. Canny has found that the requirements for the application of edge detection on diverse vision systems are relatively similar. Thus, an edge detection solution to address these requirements can be implemented in a wide range of situations.

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

Canny edge detection algorithm is one of the most strictly defined methods that provides good and reliable detection. Owing to its optimality to meet with the three criteria for edge detection and the simplicity of process for implementation, it became one of the most popular algorithms for edge detection.

1.1.5 Robert Gradient Edge Detection Method

It uses diagonal edge gradients, susceptible to fluctuations. Gives no information about edge orientation and works best with binary images.The Roberts Cross operator performs a simple, quick to compute, 2-D spatial gradient measurement on an image. It thus highlights regions of high spatial frequency which often correspond to edges. In its most common usage, the input to the operator is a grayscale image, as is the output. Pixel values at each point in the output represent the estimated absolute magnitude of the spatial gradient of the input image at that point.

+1	0
0	-1

Gx

0	+1
-1	0

Gy

1.2 Laplacian Edge Detection based Image Segmentation

The Laplacian method searches for zero crossings in the second derivative of the image to find edges.

Types of Laplacian Edge Detection Methods are Difference of Gaussian, Laplacian of Gaussian. The Laplacian edge detector uses only one kernel. It calculates second order derivatives in a single pass.

0	-1	0
-1	4	-1
0	-1	0

The laplacian operator

-1	-1	-1
-1	8	-1
-1	-1	-1

The laplacian operator (include diagonals)

Laplacian is a derivative operator; its uses highlight gray level discontinuities in an image and try to de-emphasize regions with slowly varying gray levels. This operation in result produces such images which have grayish edge lines and other discontinuities on a dark background. This produces inward and outward edges in an image.

Comparing Various Edge Detection Methods

[16]:
```
# Importing required libraries
import cv2
import numpy as np
from matplotlib import pyplot as plt
```

[17]:
```
 # Adding Image
image = cv2.imread('img1.jpg',0)
image = cv2.resize(image,(800,800))
```

[18]:
```
 # Robertsedge operator
kernel_Roberts_x = np.array([
    [1, 0],
    [0, -1]
    ])
kernel_Roberts_y = np.array([
    [0, -1],
    [1, 0]
    ])
```

[19]:
```
# Sobel edge operator
kernel_Sobel_x = np.array([
    [-1, 0, 1],
    [-2, 0, 2],
    [-1, 0, 1]])
kernel_Sobel_y = np.array([
    [1, 2, 1],
    [0, 0, 0],
    [-1, -2, -1]])
```

[20]:
```
# Prewitt edge operator
kernel_Prewitt_x = np.array([
    [-1, 0, 1],
    [-1, 0, 1],
    [-1, 0, 1]])
kernel_Prewitt_y = np.array([
    [1, 1, 1],
    [0, 0, 0],
    [-1, -1, -1]])
```

[21]:
```
# Kirsch edge detection operator
def kirsch(image):
    m,n = image.shape
    list=[]
    kirsch = np.zeros((m,n))
    for i in range(2,m-1):
        for j in range(2,n-1):
            d1 = np.square(5 * image[i - 1, j - 1] + 5 * image[i -␣
 ↪1, j] + 5 * image[i - 1, j + 1] -
                  3 * image[i, j - 1] - 3 * image[i, j + 1] - 3 *␣
 ↪image[i + 1, j - 1] -
                  3 * image[i + 1, j] - 3 * image[i + 1, j + 1])
            d2 = np.square((-3) * image[i - 1, j - 1] + 5 *␣
 ↪image[i - 1, j] + 5 * image[i - 1, j + 1] -
                  3 * image[i, j - 1] + 5 * image[i, j + 1] - 3 *␣
 ↪image[i + 1, j - 1] -
                  3 * image[i + 1, j] - 3 * image[i + 1, j + 1])
```

```
            d3 = np.square((-3) * image[i - 1, j - 1] - 3 *␣
↪image[i - 1, j] + 5 * image[i - 1, j + 1] -
                  3 * image[i, j - 1] + 5 * image[i, j + 1] - 3 *␣
↪image[i + 1, j - 1] -
                  3 * image[i + 1, j] + 5 * image[i + 1, j + 1])
            d4 = np.square((-3) * image[i - 1, j - 1] - 3 *␣
↪image[i - 1, j] - 3 * image[i - 1, j + 1] -
                  3 * image[i, j - 1] + 5 * image[i, j + 1] - 3 *␣
↪image[i + 1, j - 1] +
                  5 * image[i + 1, j] + 5 * image[i + 1, j + 1])
            d5 = np.square((-3) * image[i - 1, j - 1] - 3 *␣
↪image[i - 1, j] - 3 * image[i - 1, j + 1] - 3
                  * image[i, j - 1] - 3 * image[i, j + 1] + 5 *␣
↪image[i + 1, j - 1] +
                  5 * image[i + 1, j] + 5 * image[i + 1, j + 1])
            d6 = np.square((-3) * image[i - 1, j - 1] - 3 *␣
↪image[i - 1, j] - 3 * image[i - 1, j + 1] +
                  5 * image[i, j - 1] - 3 * image[i, j + 1] + 5 *␣
↪image[i + 1, j - 1] +
                  5 * image[i + 1, j] - 3 * image[i + 1, j + 1])
            d7 = np.square(5 * image[i - 1, j - 1] - 3 * image[i -␣
↪1, j] - 3 * image[i - 1, j + 1] +
                  5 * image[i, j - 1] - 3 * image[i, j + 1] + 5 *␣
↪image[i + 1, j - 1] -
                  3 * image[i + 1, j] - 3 * image[i + 1, j + 1])
            d8 = np.square(5 * image[i - 1, j - 1] + 5 * image[i -␣
↪1, j] - 3 * image[i - 1, j + 1] +
                  5 * image[i, j - 1] - 3 * image[i, j + 1] - 3 *␣
↪image[i + 1, j - 1] -
                  3 * image[i + 1, j] - 3 * image[i + 1, j + 1])

                        # : Take the maximum value in each␣
↪direction, the effect is not good, use another method
            list=[d1, d2, d3, d4, d5, d6, d7, d8]
            kirsch[i,j]= int(np.sqrt(max(list)))
                        # : Rounding the die length in all␣
↪directions
```

```
            #kirsch[i, j] =int(np.sqrt(d1+d2+d3+d4+d5+d6+d7+d8))
    for i in range(m):
        for j in range(n):
            if kirsch[i,j]>127:
                kirsch[i,j]=255
            else:
                kirsch[i,j]=0
    return kirsch
```

[22]:
```
# CannyEdge Detection k is the Gaussian kernel size, t1, t2 is the␣
 ↪threshold size
def Canny(image,k,t1,t2):
    img = cv2.GaussianBlur(image, (k, k), 0)
    canny = cv2.Canny(img, t1, t2)
    return canny
```

[23]:
```
#
kernel_Laplacian_1 = np.array([
    [0, 1, 0],
    [1, -4, 1],
    [0, 1, 0]])
kernel_Laplacian_2 = np.array([
    [1, 1, 1],
    [1, -8, 1],
    [1, 1, 1]])
```

[24]:
```
 # Two convolution kernels do not have rotation invariance
kernel_Laplacian_3 = np.array([
    [2, -1, 2],
    [-1, -4, -1],
    [2, 1, 2]])
kernel_Laplacian_4 = np.array([
    [-1, 2, -1],
    [2, -4, 2],
    [-1, 2, -1]])
```

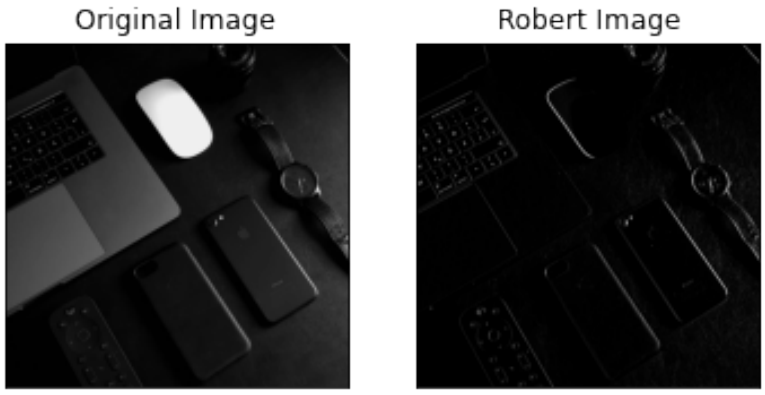
[25]:
```
 # 5*5 LoG Convolution Template
kernel_LoG = np.array([
    [0, 0, -1, 0, 0],
    [0, -1, -2, -1, 0],
    [-1, -2, 16, -2, -1],
    [0, -1, -2, -1, 0],
    [0, 0, -1, 0, 0]])
```

[26]:
```
 # convolution

output_1 = cv2.filter2D(image, -1, kernel_Prewitt_x)
output_2 = cv2.filter2D(image, -1, kernel_Sobel_x)
output_3 = cv2.filter2D(image, -1, kernel_Prewitt_x)
output_4 = cv2.filter2D(image, -1, kernel_Laplacian_1)
output_5 = Canny(image,3,50,150)
output_6 = kirsch(image)
```

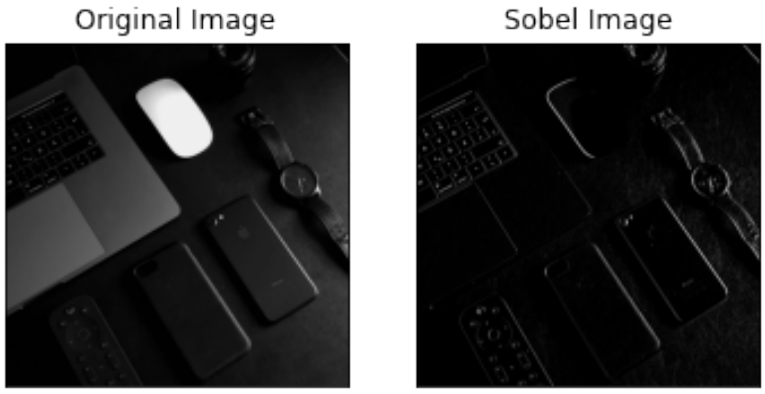
[30]:
```
# Plotting Robert Edge Detection Result
plt.subplot(121),plt.imshow(image,cmap = 'gray')
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(output_1,cmap = 'gray')
plt.title('Robert Image'), plt.xticks([]), plt.yticks([])
```

[30]:
```
(Text(0.5, 1.0, 'Robert Image'),
```



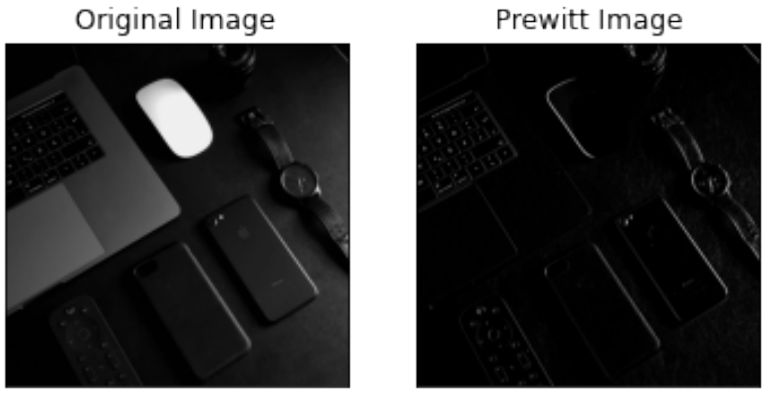
[31]:
```
plt.subplot(121),plt.imshow(image,cmap = 'gray')
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(output_2,cmap = 'gray')
plt.title('Sobel Image'), plt.xticks([]), plt.yticks([])
```

[31]:
```
(Text(0.5, 1.0, 'Sobel Image'),
```



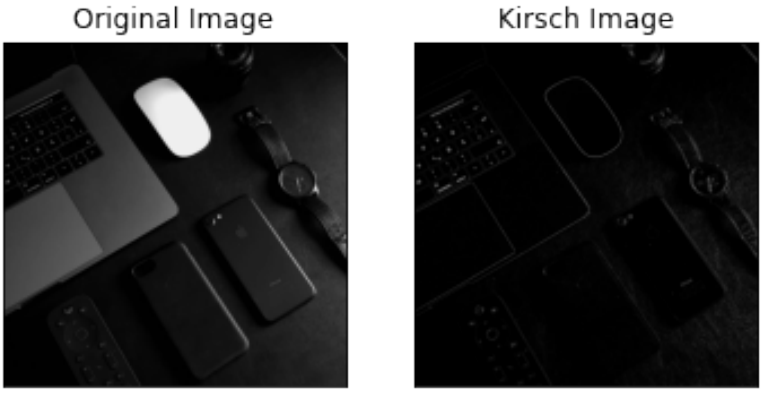
[32]:
```
plt.subplot(121),plt.imshow(image,cmap = 'gray')
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(output_3,cmap = 'gray')
plt.title('Prewitt Image'), plt.xticks([]), plt.yticks([])
```

[32]:
```
(Text(0.5, 1.0, 'Prewitt Image'),
```



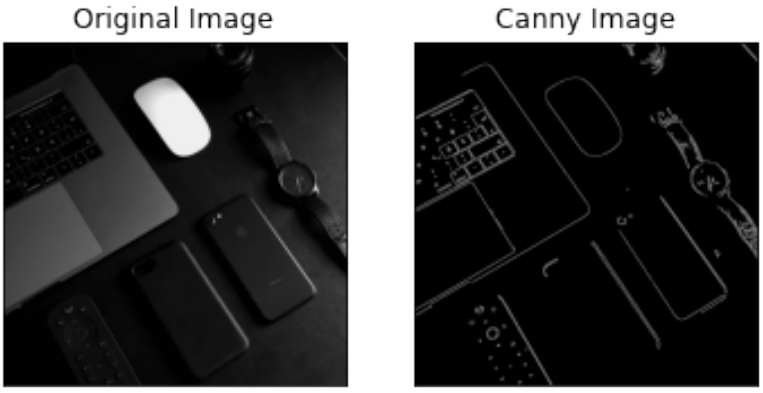
[33]:
```
plt.subplot(121),plt.imshow(image,cmap = 'gray')
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(output_4,cmap = 'gray')
plt.title('Kirsch Image'), plt.xticks([]), plt.yticks([])
```

[33]:
```
(Text(0.5, 1.0, 'Kirsch Image'),
```



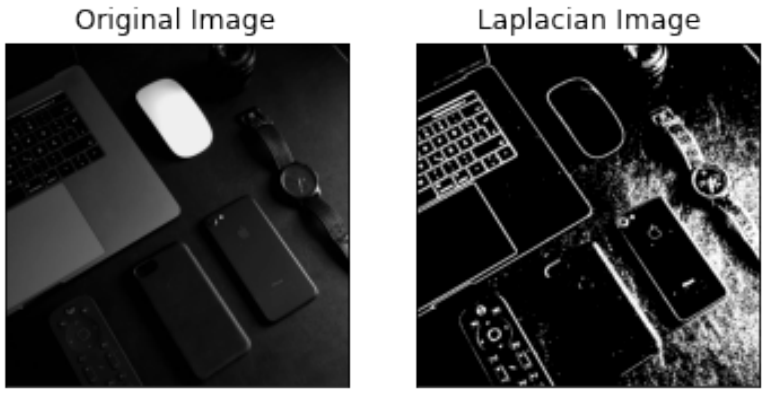
[34]:
```
plt.subplot(121),plt.imshow(image,cmap = 'gray')
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(output_5,cmap = 'gray')
plt.title('Canny Image'), plt.xticks([]), plt.yticks([])
```

[34]:
```
(Text(0.5, 1.0, 'Canny Image'),
```



[35]:
```
plt.subplot(121),plt.imshow(image,cmap = 'gray')
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(output_6,cmap = 'gray')
plt.title('Laplacian Image'), plt.xticks([]), plt.yticks([])
```

[35]:
```
(Text(0.5, 1.0, 'Laplacian Image'),
```



After comparing approx all methods of Edge Detection, we can say that the image segmented using Canny operator has best edges among all first order derivative method. Sobel is better than robert and prewitt but kirsch also done a job as compare to sobel.
Laplacian method is best for smoothing sharp images and finding their edges.

2 Watershed Method based Image Segmentation

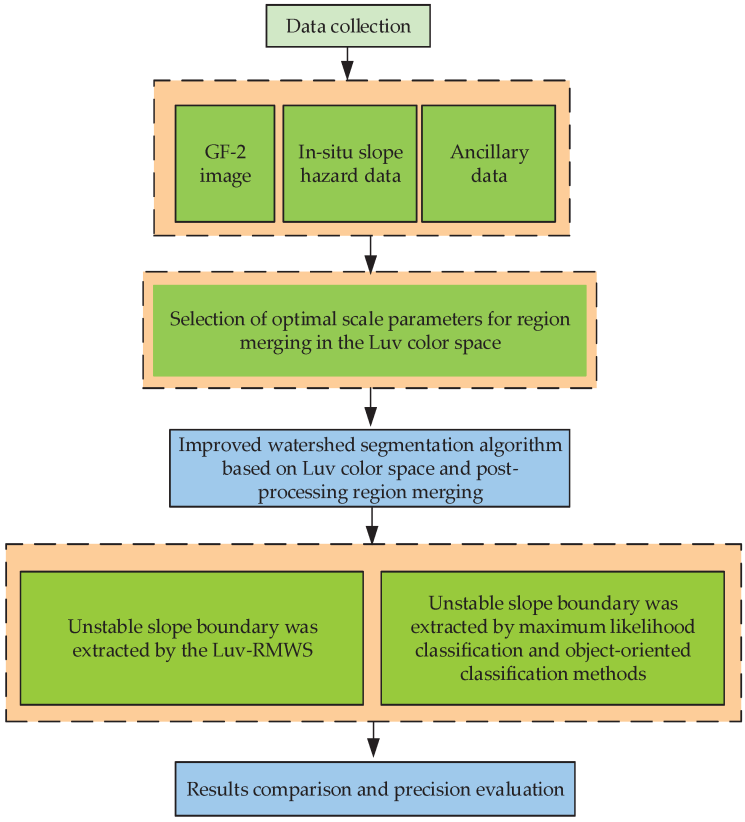
This method is considered efficient and faster.The image is considered as topographic. Pixels having the highest gradient magnitude intensities (GMIs) correspond to watershed lines, which represent the region boundaries.
Water placed on any pixel enclosed by a common watershed line flows downhill to a common local intensity minimum (LIM). Pixels draining to a common minimum form a catch basin, which represents a segment.

The algorithm introduced by Luc Vincent and Pierre Soille is based on the concept of "immersion". Each local minima of a gray-scale image I which can be regarded as a surface has a hole and the surface is immersed out into

water. Then, starting from the minima of lowest intensity value, the water will progressively fill up different catchment basins of image (surface) I.
Conceptually, the algorithm then builds a dam to avoid a situation that the water coming from two or more different local minima would be merged. At the end of this immersion process, each local minimum is totally enclosed by dams corresponding to watersheds of image.
It is appropriate to use this method to segment the high-resolution remote sensing image.

Types of Watershed Implementation method

- Distant Transform Approach
- Gradient Method
- Marker Controlled Approach



Watershed Algorithm

2.1 Distant Transform Approach

A tool used commonly in conjunction with the watershed transform for segmentation is the distance transform. It is the distance from every pixel to the nearest nonzero-valued pixel. The distance transform can be computed using toolbox function bwdist, whose calling syntax is

D=bwdist(f)

A binary image can be converted to a gray level image, which is suitable for watershed segmentation using different DT. However, different DT functions produce different effects.

[19]:
```
import numpy as np
import cv2, matplotlib.pyplot as plt
%matplotlib inline
```

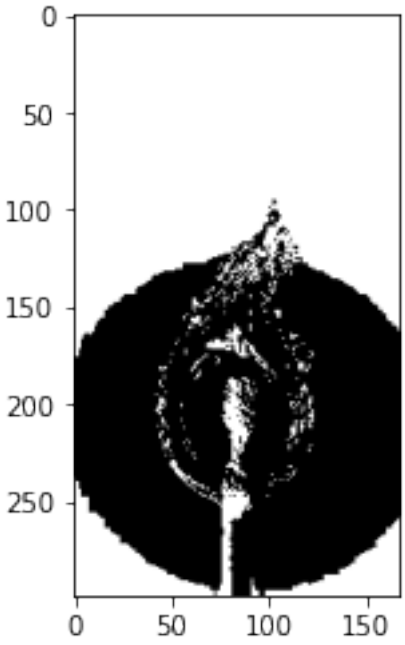
[21]:
```
# read image and show
img = cv2.imread('5d.jpg') # in BGR mode
# convert to RGB mode
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

plt.axis('off')
plt.imshow(img)
```

[23]:
```
# performing otsu's binarization
# convert to gray scale first
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
ret, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV +␣
 ↪cv2.THRESH_OTSU)
print("Threshold limit: " + str(ret))

plt.axis('on')
plt.imshow(thresh, cmap = 'gray')
```

```
Threshold limit: 61.0
```



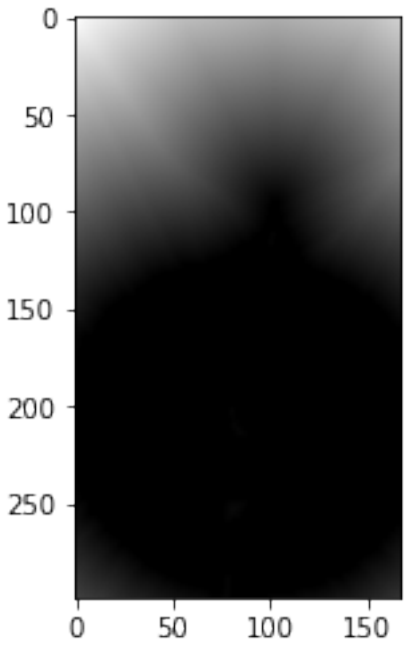
[24]:
```
# noise removal
kernel = np.ones((3, 3), np.uint8)
opening = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel,␣
 ↪iterations = 2)
```

```
# sure background area
sure_bg = cv2.dilate(opening, kernel, iterations = 3)

# sure foreground area
dist_transform = cv2.distanceTransform(opening,cv2.DIST_L2,5)
ret, sure_fg = cv2.threshold(dist_transform,0.7*dist_transform.
 ↪max(),255,0)

# Finding unknown region
sure_fg = np.uint8(sure_fg)
unknown = cv2.subtract(sure_bg,sure_fg)
```

[25]:
```
plt.imshow(dist_transform, cmap = 'gray')
```

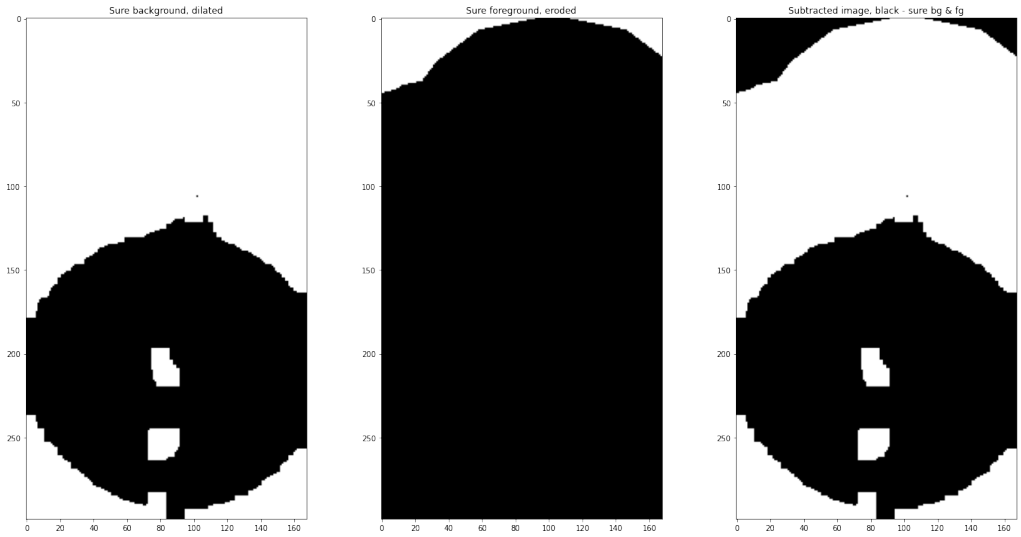


Distant Transform

[28]:
```
fig = plt.figure(figsize = (20, 10)) # to change figsize
plt.subplot(131)
plt.imshow(sure_bg, cmap = 'gray')
plt.title('Sure background, dilated')

plt.subplot(132)
plt.imshow(sure_fg, cmap = 'gray')
plt.title('Sure foreground, eroded')

plt.subplot(133)
plt.imshow(unknown, cmap = 'gray')
plt.title('Subtracted image, black - sure bg & fg')
plt.tight_layout()
# plt.subplots_adjust(wspace = 3)
# fine tuning
# f.subplots_adjust(wspace=3)
```



2.2 Gradient Method

The gradient magnitude is used to pre process a gray-scale image prior to using the watershed transform for segmentation. The gradient magnitude image has high pixel values along object edges and low pixel values everywhere else.

Watershed transform would result in watershed ridge lines along object edges. The experimental results show that the over segmentation problem, which usually appears with the watershed technique, can be attenuated, and the segmentation results can be performed using the topological gradient approach. Another advantage of this method is that it splits the segmentation process into two separate steps: first we detect the main edges of the image processed, and then we compute the watershed of the gradient detected.

[1]:
```
import numpy as np
import cv2
from matplotlib import pyplot as plt
```

[2]:
```
img = cv2.imread('5d.jpg')
b,g,r = cv2.split(img)
rgb_img = cv2.merge([r,g,b])
```

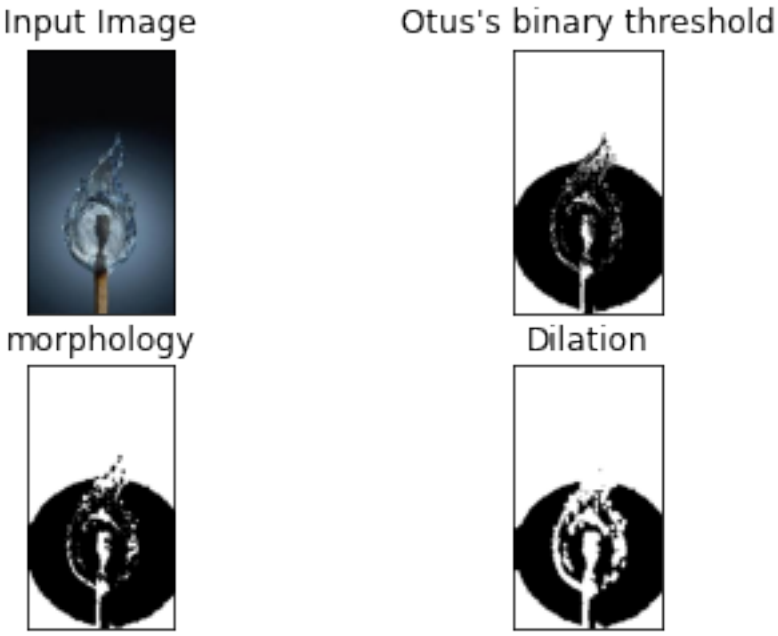
[3]:
```
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
ret, thresh = cv2.threshold(gray,0,255,cv2.THRESH_BINARY_INV+cv2.
 ↪THRESH_OTSU)
```

[4]:
```
# noise removal
kernel = np.ones((2,2),np.uint8)
#opening = cv2.morphologyEx(thresh,cv2.MORPH_OPEN,kernel,␣
 ↪iterations = 2)
closing = cv2.morphologyEx(thresh,cv2.MORPH_CLOSE,kernel,␣
 ↪iterations = 2)
```

[5]:
```
# sure background area
sure_bg = cv2.dilate(closing,kernel,iterations=3)
```

[7]:
```
plt.subplot(221),plt.imshow(rgb_img)
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(222),plt.imshow(thresh, 'gray')
```

```
plt.title("Otus's binary threshold"), plt.xticks([]), plt.
 ↪yticks([])
plt.subplot(223),plt.imshow(closing, 'gray')
plt.title("morphology"), plt.xticks([]), plt.yticks([])
plt.subplot(224),plt.imshow(sure_bg, 'gray')
plt.title("Dilation"), plt.xticks([]), plt.yticks([])
plt.show()
```



2.3 Marker Controlled Approach

As Gradient method is prone to noise and causes over segmentation. Over segmentation means a large number of segmented regions.An approach used to control over segmentation is based on the concept of markers. A marker is a connected component belonging to an image. Markers are used to modify the gradient image. Markers are of two types internal and external, internal for object and external for boundary.

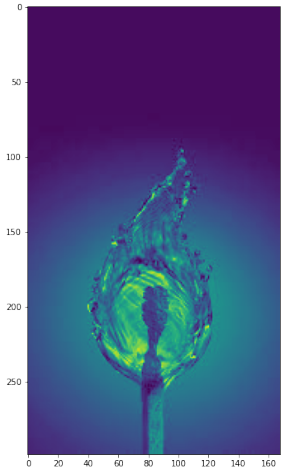
The marker-controlled watershed segmentation has been shown to be a robust and flexible method for segmentation of objects with closed contours, where the boundaries are expressed as ridges.
Markers are placed inside an object of interest; internal markers associate with objects of interest, and external markers associate with the background.
After segmentation, the boundaries of the watershed regions are arranged on the desired ridges, thus separating each object from its neighbours.

[ ]:
```
import cv2
import numpy as np
from matplotlib import pyplot as plt
```

[ ]:
```
def showImage(imageForShowing):
    plt.figure(figsize = (10, 10))
    plt.imshow(imageForShowing)
```

[ ]:
```
im = cv2.imread("5d.jpg", cv2.IMREAD_GRAYSCALE)
original = cv2.imread("5d.jpg")
print("Original Image")
showImage(im)
```

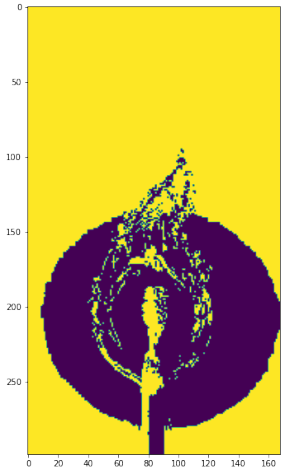
```
Original Image
```



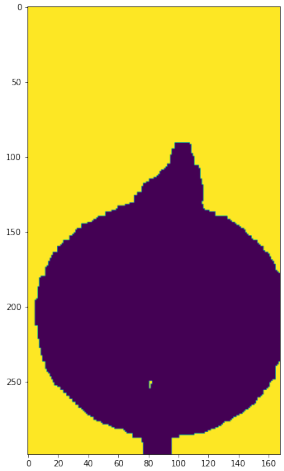
[ ]:
```
_, im = cv2.threshold(im, 72, 255, cv2.THRESH_BINARY_INV)
```

```
#ret, im = cv2.threshold(im, 100, 255,cv2.THRESH_BINARY_INV+cv2.
 ↪THRESH_OTSU)
#ret, im = cv2.threshold(im, 100, 255, cv2.THRESH_OTSU)
print("Threshold Image")
showImage(im)
```

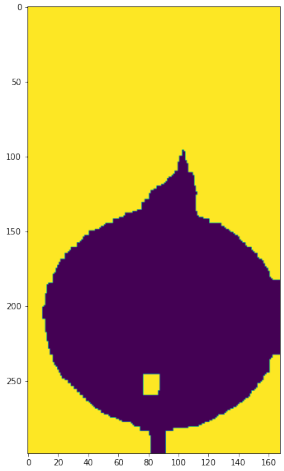
```
Threshold Image
```



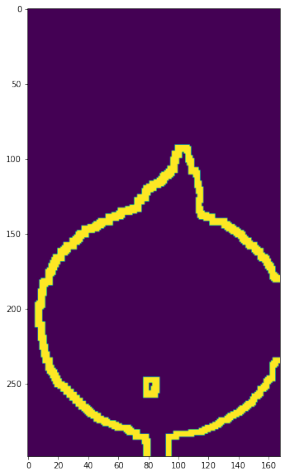
[ ]:
```
kernel = np.ones((10,10),np.uint8)
im = cv2.erode(im, kernel)
showImage(im)
```



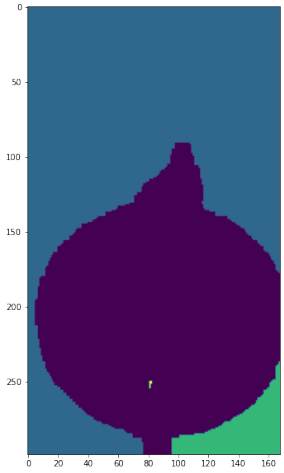
[ ]:
```
kernel = np.ones((10,10),np.uint8)
dilateImage = cv2.dilate(im, kernel, iterations = 1)
showImage(dilateImage)
```



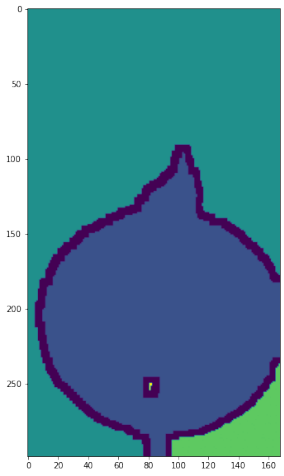
[ ]:
```
unknown = cv2.subtract(dilateImage,im)
showImage(unknown)
```



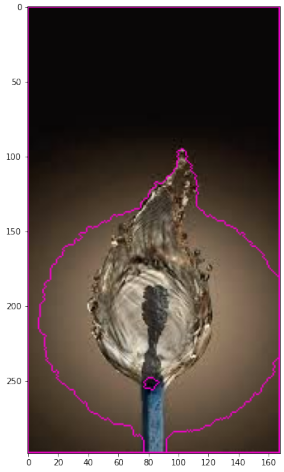
[ ]:
```
_, markers = cv2.connectedComponents(im)
showImage(markers)
```



[ ]:
```
markers = markers + 1
markers[unknown==255] = 0
showImage(markers)
```



```
[ ]: markers = cv2.watershed(original,markers)
     original[markers== -1] = [250, 0, 210]
     showImage(original)
```



After comparing all methods of Watershed Segmentation ,it can be concluded that:

- Marker Watershed Segmentation is the best method and use for high definition topographic image.
- Distant Transform is a good method but some over or less segmentation lead to error.
- Gradient method is the easiest method for segmentation hence not too much proper.

3 Partial Differential Equation Based Image Segmentation

PDE-based image segmentation techniques couple level set methods and Fast Marching Methods to quickly and accurately extract boundaries from image data. They do not require prior knowledge about the number or topology of objects in the image data.
For this purpose, the active contours and level set methods are commonly used, both exploiting the parametric curve evolution described by PDEs. The forces derived from the image cause a curve change according to the segmented objects.

Lagrangian Method are based on parameterizing the counters according to sampling strategy and evolving each element according to image.

The **Level-set method** is simple and adaptable for computing and analyzing the motion of an interface in two or three dimensions. In Level-set, curve and surfaces are represented in implicit form as zero level set of **High Definition Continuous Function**.

3.1 Methods used for PDEs Segmentation

3.1.1 The Distance Regularized Level Set Evolution and Its Application to Image Segmentation

The DRLSE utilizes the regularization of the level set function, which maintains the level set function as a signed distance function near the zero level set. This property is provided by a unique forward-and-backward (FAB) diffusion. The FAB diffusion forces the level set function to be a signed distance function only in a narrow band around zero level set.
These modifications eliminate the need for re-initialization and the related possible numerical errors. Also, the numerical implementation is much faster because a relatively large time step can be used in the finite difference scheme while sufficient numerical accuracy is assured.

3.1.2 The Re-initialization Free Level Set Evolution via Reaction Diffusion

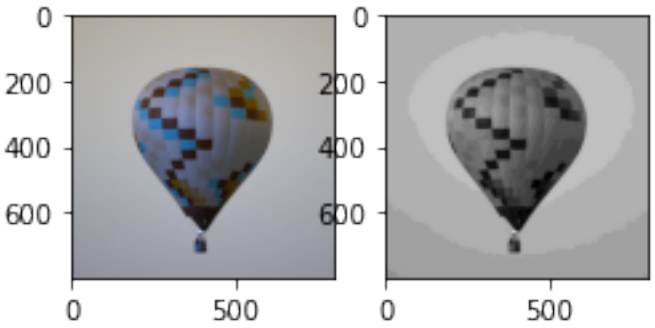
The DRLSE exhibits many advantages over re-initialization level set methods, even though there is a limited anti-leakage capability for weak boundaries and a possibility of failure in strongly noised images.These problems were solved via a method called the reaction diffusion (RD)-based levelset evolution.

[3]:
```
import cv2
import numpy as np
from matplotlib import pyplot as plt
```

[4]:
```
def viewImage(image):
    cv2.namedWindow('Display', cv2.WINDOW_NORMAL)
    cv2.imshow('Display', image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

[5]:
```
def grayscale_17_levels (image):
    high = 255
    while(1):
        low = high - 15
        col_to_be_changed_low = np.array([low])
        col_to_be_changed_high = np.array([high])
        curr_mask = cv2.inRange(gray,␣
↪col_to_be_changed_low,col_to_be_changed_high)
        gray[curr_mask > 0] = (high)
        high -= 15
        if(low == 0 ):
            break
```

[9]:
```
image = cv2.imread('ballon.jpg')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
grayscale_17_levels(gray)
plt.subplot(1,3,1),plt.imshow(image,cmap = 'gray')
plt.subplot(1,3,2),plt.imshow(gray,cmap = 'gray')
```

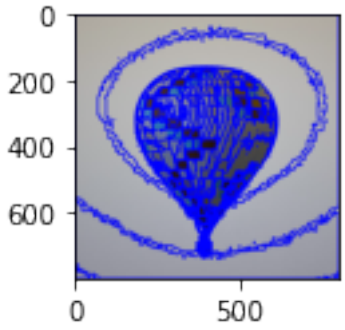


[20]:
```
def get_area_of_each_gray_level(im):
## convert image to gray scale (must br done before contouring)
    image = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
    output = []
    high = 255
    first = True
    while(1):
```

```
        low = high - 15;
        if(first == False):
            to_be_black_again_low = np.array([high])
            to_be_black_again_high = np.array([255])
            curr_mask = cv2.inRange(image, to_be_black_again_low,
            to_be_black_again_high)
            image[curr_mask > 0] = (0)
          # making values of gray level white so to calculate
        ret, threshold = cv2.threshold(image, low, 255, 0)
        contours, hirerchy = cv2.findContours(threshold,
        cv2.RETR_LIST, cv2.CHAIN_APPROX_NONE)
        if(len(contours) > 0):
          output.append([cv2.contourArea(contours[0])])
          cv2.drawContours(im, contours, -1, (0,0,255), 3)
        high -= 15
        first = False
        if(low == 0 ):
          break
    return output
```

[22]:
```
image = cv2.imread('ballon.jpg')
print(get_area_of_each_gray_level(image))
plt.subplot(1,3,1),plt.imshow(image,cmap = 'gray')
```

```
[[0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [7.0], [2.0], [0.0], [0.
↪0], [0.0],
[0.0], [0.0], [0.0], [0.0], [19.0], [0.0]]
```



3.1.3 An Efficient Algorithm for the Level Set Method Preserving the Distance Function

The algorithm is based on the split Bregman method. At first, the speed of the algorithm is 5-6 times higher than other distance-preserving level set algorithms because it is not limited by the CFL condition. Secondly, the algorithm preserves the level set function as the distance function without the need of re-initialization.

[1]:
```
import numpy as np
import scipy.ndimage
import scipy.signal
import matplotlib.pyplot as plt
from skimage import color, io
```

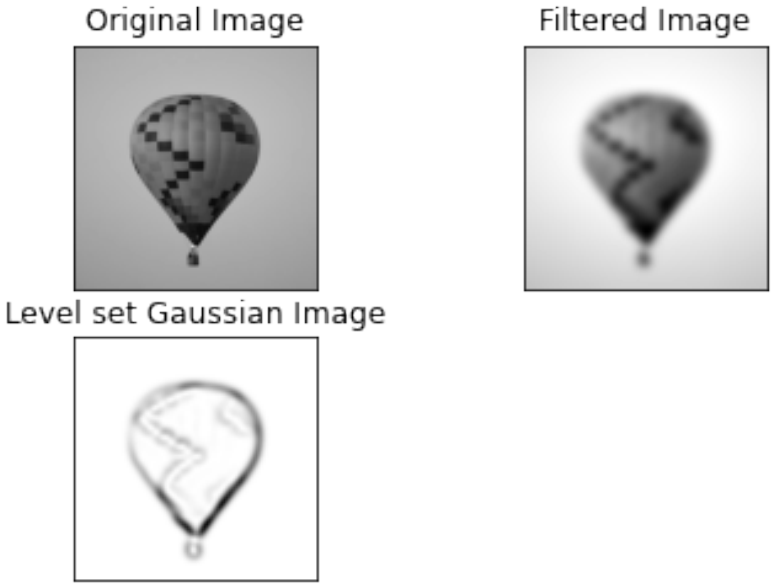
[2]:
```
def grad(x):
    return np.array(np.gradient(x))
def norm(x, axis=0):
    return np.sqrt(np.sum(np.square(x), axis=axis))
def stopping_fun(x):
    return 1. / (1. + norm(grad(x))**2)
```

[5]:
```
img = io.imread('ballon.jpg')
img = color.rgb2gray(img)
img = img - np.mean(img)

img_smooth = scipy.ndimage.filters.gaussian_filter(img, sigma=15)

F = stopping_fun(img_smooth)
```

[8]:
```
plt.subplot(221),plt.imshow(img,cmap = 'gray')
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(222),plt.imshow(img_smooth,cmap = 'gray')
plt.title('Filtered Image'), plt.xticks([]), plt.yticks([])
plt.subplot(223),plt.imshow(F,cmap = 'gray')
plt.title('Level set Gaussian Image'), plt.xticks([]), plt.
 ↪yticks([])
```



3.1.4 Harmonic Active Contours (HAC)

The HAC combines the contour and region-based criteria into one embedded functional that includes regularization terms as well. The PDE describing the level set evolution was obtained using the Euler-Lagrange method. To dispose of the CFL condition in order to speed up the implementation, the authors proposed an algorithm based on splitting the original problem to sub-optimization problems which are easy to solve and merging them together.

3.1.5 Active Contours with Selective Local or Global Segmentation: A New Formulation and Level Set Method

The method reduces the expensive reinitialization process that is commonly used in the level set to make the SBGFRLS more efficient. In principle, the SBFRLS combines the merits of the ChanVese (C-V) method and the Geodesic active contours (GAC) technique. The testing of the SBFRLS method on both synthetic and real images revealed advantages over the C-V and GAC methods.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import skimage.data as data
import skimage.segmentation as seg
import skimage.filters as filters
import skimage.draw as draw
import skimage.color as color
```

[2]:
```
def image_show(image, nrows=1, ncols=1, cmap='gray'):
    fig, ax = plt.subplots(nrows=nrows, ncols=ncols, figsize=(3,␣
 ↪3))
    ax.imshow(image, cmap='gray')
    ax.axis('off')
    return fig, ax
```

[3]:
```
# import the image
from skimage import io
image = io.imread('ballon.jpg')
image_show(image);
```

[4]:
```
image_gray = color.rgb2gray(image)
image_show(image_gray);
```

[5]:
```
def circle_points(resolution, center, radius):
    """ Generate points which define a circle on an image.Centre␣
 ↪refers to the centre of the circle     """
    radians = np.linspace(0, 2*np.pi, resolution)
    c = center[1] + radius*np.cos(radians)#polar co-ordinates
    r = center[0] + radius*np.sin(radians)
    return np.array([c, r]).T
points = circle_points(200, [400, 400], 280)[:-3]
```

[6]:
```
fig, ax = image_show(image)
ax.plot(points[:, 0], points[:, 1], '--r', lw=3)
```

[7]:
```
snake = seg.active_contour(image_gray, points)
fig, ax = image_show(image)
ax.plot(points[:, 0], points[:, 1], '--r', lw=3)
ax.plot(snake[:, 0], snake[:, 1], '-b', lw=3);
```

[8]:
```
snake = seg.active_contour(image_gray, points,alpha=0.06,beta=0.3)
fig, ax = image_show(image)
ax.plot(points[:, 0], points[:, 1], '--r', lw=3)
```

```
ax.plot(snake[:, 0], snake[:, 1], '-b', lw=3);
```

[9]:
```
image_labels = np.zeros(image_gray.shape, dtype=np.uint8)
```

[14]:
```
indices = draw.circle_perimeter(280, 440, 260)#from here
image_labels[indices] = 1
image_labels[points[:, 1].astype(np.int), points[:, 0].astype(np.
↪int)] = 2
image_show(image_labels);
```

[15]:
```
image_segmented = seg.random_walker(image_gray, image_labels)
# Check our results
fig, ax = image_show(image_gray)
ax.imshow(image_segmented == 1, alpha=0.3);
```

[16]:
```
image_segmented = seg.random_walker(image_gray, image_labels, beta␣
↪= 3000)
# Check our results
fig, ax = image_show(image_gray)
ax.imshow(image_segmented == 1, alpha=0.3);
```

After performing basic methods of PDE it is clear that active counter is best method but for cellular images. On a big image,Level set is the best method for image segmentation.

Referrences

- Efficient image segmentation using partial differential equations and morphology - Joachim Weickert at ELSEVIER journal.

- International Conference on Communication Technology and System Design Edge Based Image Segmentation Technique for Detection and Estimation of the Bladder Wall Thickness B. Padmapriya , T.Kesavamurthi, H. Wassim Ferose

- Jamil N., Soh H.C., Tengku Sembok T.M., Bakar Z.A. (2011) A Modified Edge-Based Region Growing Segmentation of Geometric Objects. In: Badioze Zaman H. et al. (eds) Visual Informatics: Sustaining Research and Innovations. IVIC 2011. Lecture Notes in Computer Science, vol 7066. Springer, Berlin, Heidelberg

- https://medium.com/@dhairya.vayada/intuitive-image-processing-watershed-segmentation-50a66ed2352e

- International Journal of Soft Computing and Engineering (IJSCE) ISSN: 2231-2307, Volume-4, Issue-1, March 2014 Image Segmentation using Watershed Transform Amandeep Kaur, Aayushi

- Bernhard Preim, Charl Botha, in Visual Computing for Medicine (Second Edition), 2014 Image Analysis for Medical Visualization

- Bin Zhou, Xiao-Lin Yang, Rui Liu and Wei Wei, 2010. Image Segmentation with Partial Differential Equations. Information Technology Journal, 9: 1049-1052.

- JIANG WEI et al: AN IMAGE SEGMENTATION METHOD BASED ON PARTIAL DIFFERENTIAL at ijssst journal

- Tara, Saikumar Bharath, R Reddy, Bhushan Ramesh, Gundelli Sandeep, Karedla Professor, Assoc Scholar, Various Image Segmentation Methods Based On Partial Differential Equation-A Survey. 3. 2248-9584.

- Suri J.S., Laxminarayan S., Gao J., Reden L. (2002) Image Segmentation Via PDEs. In: Suri J.S., Laxminarayan S. (eds) PDE and Level Sets: Algorithmic Approaches to Static and Motion Imagery. Topics in Biomedical Engineering. Springer, Boston, MA

- J. Sliž and J. Mikulka, "Advanced image segmentation methods using partial differential equations: A concise comparison," 2016 Progress in Electromagnetic Research Symposium (PIERS), Shanghai, 2016, pp. 1809-1812, doi: 10.1109/PIERS.2016.7734800.