Marker based localization

Watershed Segmentation

Team members

Niharika Jayanthi Dheeraj Kamath

Under the guidance of
Sanam Shakya

Goal

In this chapter,

- How to use marker-based water shed segmentation on images.
- We will see: **cv2.distanceTransform()**, **cv2.watershed()**

Theory

Watershed is an algorithm in image processing used for isolating objects in the image from the background. The algorithm accepts a grayscale image and a marker image. The markers is an image where you tell the watershed about the foreground objects and the background.

Working principle

Any grayscale image can be viewed as a topographic surface where high intensity denotes peaks and hill while low intensity denotes valleys. We start filling every isolated valleys(local minima) with different colored water. As water rises , depending on the peaks (gradients) nearby, water from different valleys, labeled with different colors will start to merge. To prevent this barriers are built in the locations where the water merges. We continue doing this till all peaks under water. The result will be a segmentation of the image. But this method gives a over segmented image due to noise or any other irregularities in the image. So a major enhancement in the water shed transformation consists in flooding the topographic surface from previously defined set of markers. It is an interactive image segmentation. What we do is to give different labels

for our object we know. Label the region which we are sure of being the foreground or object with one color (or intensity), label the region which we are sure of being background or non-object with another color and finally the region which we are not sure of anything, label it with 0. That is our marker. Then apply watershed algorithm. Then our marker will be updated with the labels we gave, and the boundaries of objects will have a value of -1.

Applications

1. Watershed algorithm is used to monitor traffic. It automatically segments the lanes of a road to count the number of vehicles on different lanes.
2. It can be used to detect fractures in the surface of steel.
3. Counting of objects in images can be done using watershed algorithm. An example is counting of coffee beans.

Code

1) Since segmentation needs an image in grayscale, we use the following code to convert the image to grayscale.

```
#import opencvn numpy and matplotlib
import numpy as np
import cv2
from matplotlib import pyplot as plt

#Reading the image
img = cv2.imread('water_coins.jpg')
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY) #Conversion to gray scale
```

Figure 1: Input image

2) Now, we threshold this image

```
ret, thresh = cv2.threshold(gray,0,255,cv2.THRESH_BINARY_INV
                                                    +cv2.THRESH_OTSU)
cv2.imshow("Otsu", thresh)
```

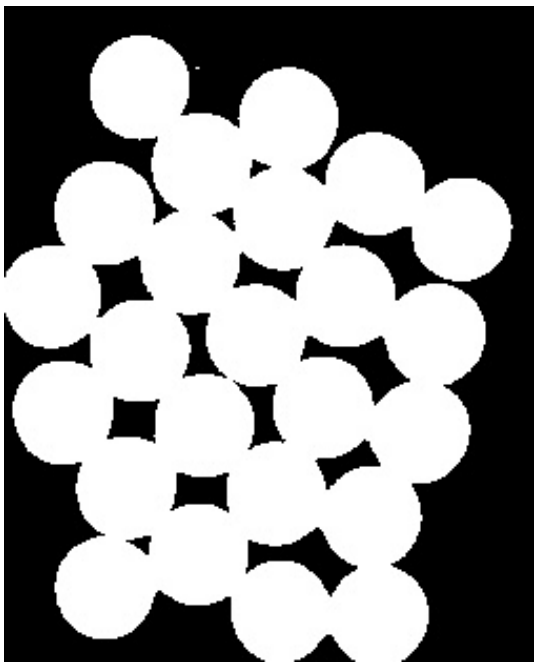

Figure 2: Otsu Threshold

3) To remove noises from the image we use morphological operations. To remove small noises we use opening. To obtain the sure background we dilate the region of the coins and then the result will be the region where we are sure that they don't contain coins.

```
kernel = np.ones((3,3),np.uint8)
opening = cv2.morphologyEx(thresh,cv2.MORPH_OPEN,kernel, iterations = 2)
sure_bg = cv2.dilate(opening,kernel,iterations=3)
```

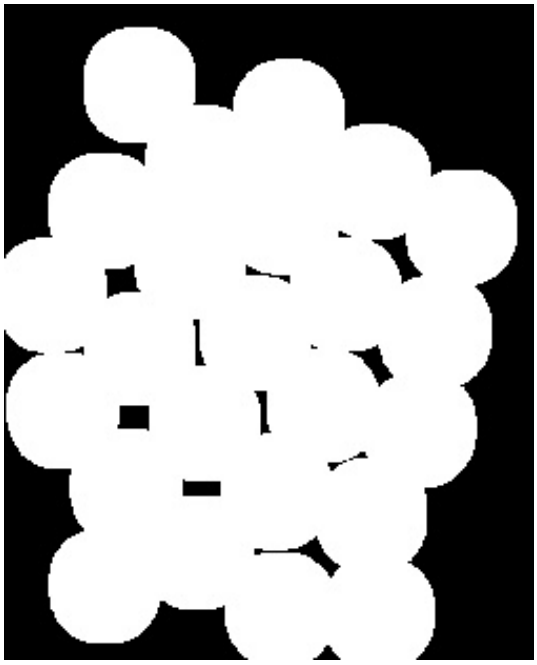

Figure 3: Sure Background

4) Now we have to find the sure foreground region. For this we apply distance transform.

```
dist_transform = cv2.distanceTransform(opening,cv2.cv.CV_DIST_L2,5)
ret, sure_fg = cv2.threshold(dist_transform,0.7*dist_transform.max(),
                                                                255,0)
```

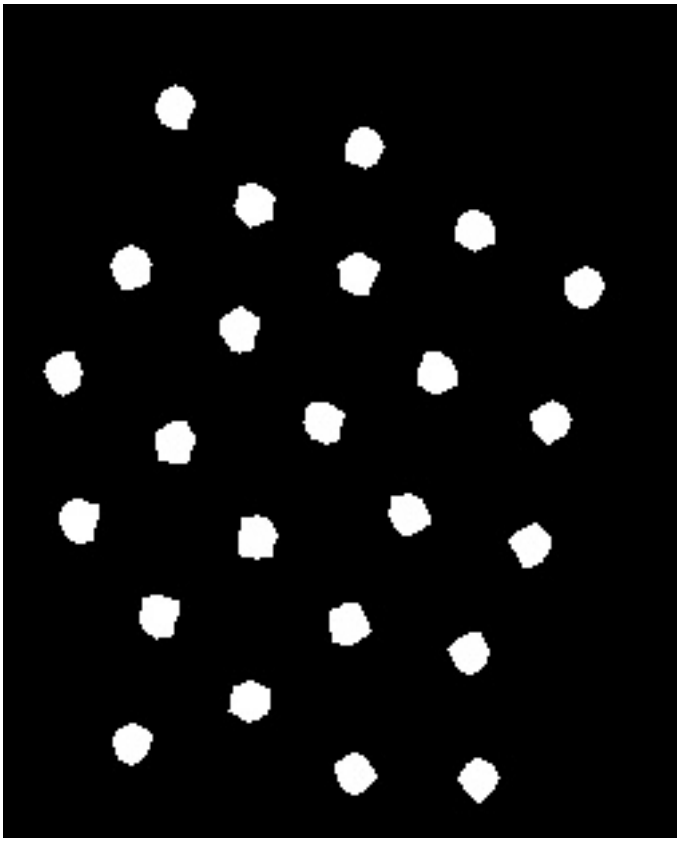

Figure 4: Sure foreground

5) The remaining regions are those which we don't have any idea, whether it is coins or background. Watershed algorithm should find it. These areas are normally around the boundaries of coins where foreground and background meet (Or even two different coins meet). We call it border. It can be obtained from subtracting sure_fg area from sure_bg area.

```
unknown = cv2.subtract(sure_bg,sure_fg)
```

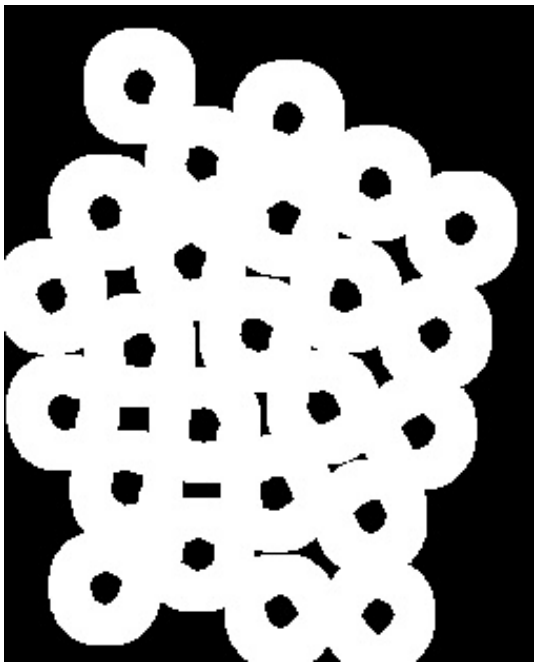

Figure 5: Intersection region

6) Now we have to create the markers. But before this we have to note that a marker is same size as that of a image but of int32 data type. We will be using sure_fg image for the marker.

```
sure_fg = np.uint8(sure_fg)
```

7) Now we know for sure which are region of coins, which are background and all. So we create markers and label the regions inside it. The regions we know for sure (whether foreground or background) are labelled with any positive integers, but different integers, and the area we don't know for sure are just left as zero. Now to do this there are two ways:

- For those using python version 3.0 and above :

  ```
  ret, markers = cv2.connectedComponents(sure_fg)
  markers = markers + 1
  markers[unknown==255] = 0
  ```

- For those using python version below 3.0:

```
    contours, hierarchy = cv2.findContours(sure_fg,cv2.RETR_TREE,
                                                    cv2.CHAIN_APPROX_SIMPLE)

    #Creating a numpy array for markers and converting the image
    # to 32 bit using dtype paramter
    marker = np.zeros((gray.shape[0], gray.shape[1]),dtype = np.int32)

    marker = np.int32(sure_fg) + np.int32(sure_bg)

    for id in range(len(contours)):
    cv2.drawContours(marker,contours,id,id+2, -1)

    marker = marker + 1
    marker[unknown==255] = 0

unknown = cv2.subtract(sure_bg,sure_fg)
```

8) Applying water shed segmentation on the image.

```
    cv2.watershed(img, marker)
    img[marker==-1]=(0,0,255) # to draw the contours in red
```

Figure 6: Post watershed image

9)To observe the image in a colormap we use the matplotlib library.

```
imgplt = plt.imshow(marker)
plt.colorbar()
plt.show()
```

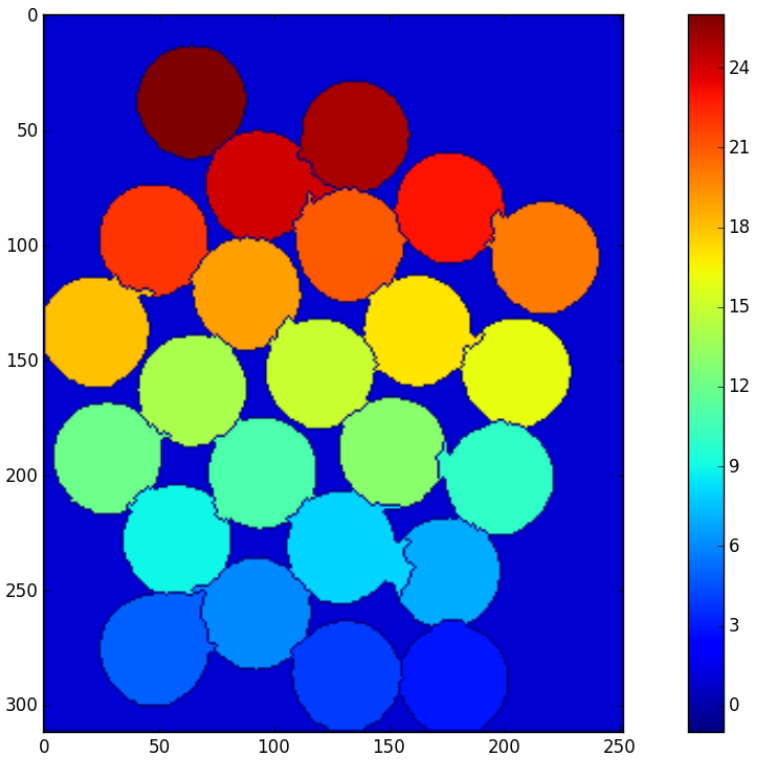

Figure 7: Post watershed image

```
#import opencvn numpy and matplotlib
import numpy as np
import cv2
from matplotlib import pyplot as plt

#Reading the image
img = cv2.imread('water_coins.jpg')
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)        #Conversion to gray sc
ret, thresh = cv2.threshold(gray,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_O

cv2.imshow("Otsu", thresh)

# 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)


# Finding sure foreground area
dist_transform = cv2.distanceTransform(opening,
                                       cv2.cv.CV_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)

cv2.imshow("Sure bg", sure_bg)
cv2.imshow("Sure fg",sure_fg)
cv2.imshow("unknown",unknown)

#Marker Labelling (for those with python version below 3.0)
contours, hierarchy = cv2.findContours(sure_fg,cv2.RETR_TREE
                                       ,cv2.CHAIN_APPROX_SIMPLE)

# Creating a numpy array for markers and converting the image to
# 32 bit using dtype paramter
marker = np.zeros((gray.shape[0], gray.shape[1]),dtype = np.int32)

marker = np.int32(sure_fg) + np.int32(sure_bg)

for id in range(len(contours)):
cv2.drawContours(marker,contours,id,id+2, -1)

marker = marker + 1
marker[unknown==255] = 0

"""
# Marker labelling (For those with python version 3.0 and above)
ret, markers = cv2.connectedComponents(sure_fg)

# Add one to all labels so that sure background is not 0, but 1
markers = markers+1
```

```
# Now, mark the region of unknown with zero
markers[unknown==255] = 0
"""

cv2.watershed(img, marker)
img[marker==-1]=(0,0,255)

cv2.imshow('watershed', img)

#To display using colormap
imgplt = plt.imshow(marker)
plt.colorbar()                #Creates a bar of colors
plt.show()                    #Displays the windows

cv2.waitKey(0)
cv2.destroyAllWindows()       #Destroys all windows
```

Additional Resources

1) Code - Github link
2) http://cmm.ensmp.fr/~beucher/wtshed.html#examples
3) About watershed segmentation
4) Image processing lectures
5) Info on defining markers