ITMO University

Image Processing:
Lab4

Prepared by

Bassel Alshawareb

April 7, 2023

1. Introduction

We are applying series of morphological operations on image containing balls which are in contact with each other. We perform an algorithm to split the balls into distinct objects with clear spaces between them.

2. Object splitting

The problem concerns splitting balls in an image and defining clear boundaries between them.

Figure 1 Original image.

The algorithm works as follow:

1) Converting the image into gray scale. Then thresholding using the usual Opencv threshold function. Here we define a high threshold value since the photo is in general bright, and the background clearly has high intensity values.

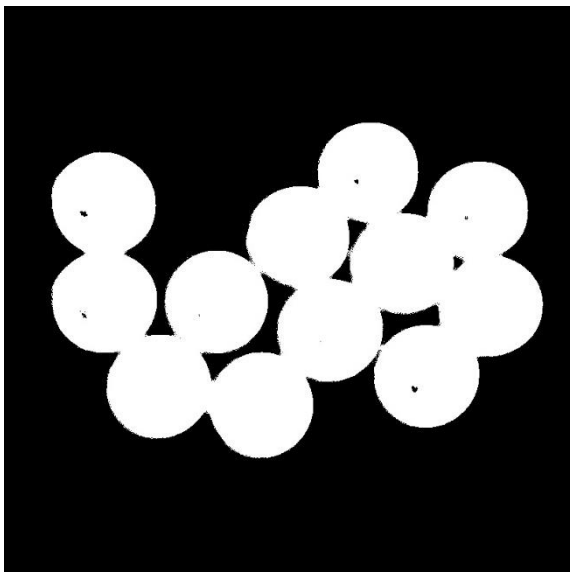

Figure 2 Thresholded image.

2) The resulting binary image contains gaps, so in order to close them we apply closing morphology.

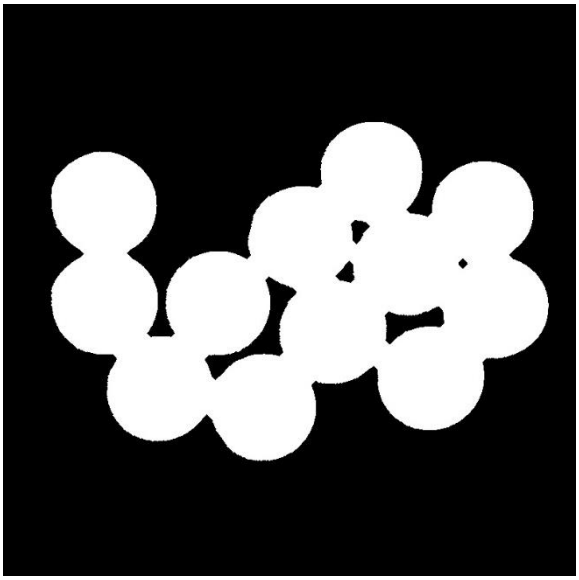

Figure 3 Binary image after closing.

3) Now we apply series of erosions until the circles shrink very smally and are about to vanish. The number of iterations of applying erosions was set manually to make suer that no circle will vanish. We are interested in shrinking the circles so that when we will apply the dilation will we make sure that they will converge right in the place of connection, where the circles were merging already.

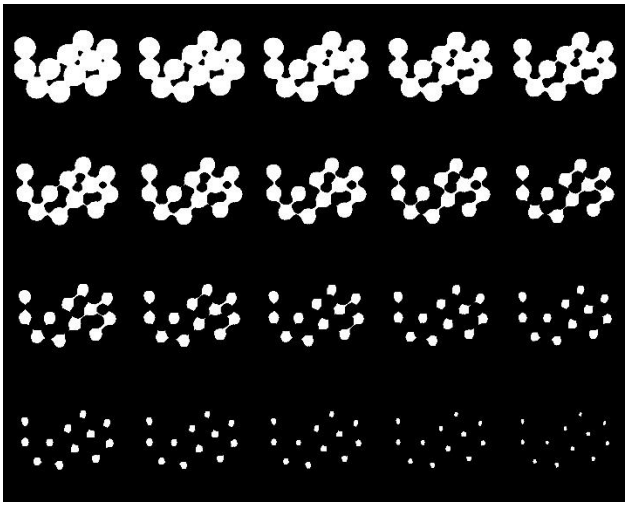

Figure 4 Image after applying series of erosions.

4) We keep applying dilation to the image followed by opening, and during every iteration we calculate the error, which define the change in the image after applying the opening, with respect to the image after applying the dilation. This error will appear only when part of different circles are meeting. In other words, the pixels where the error shows positive values are the boundary between the circles.

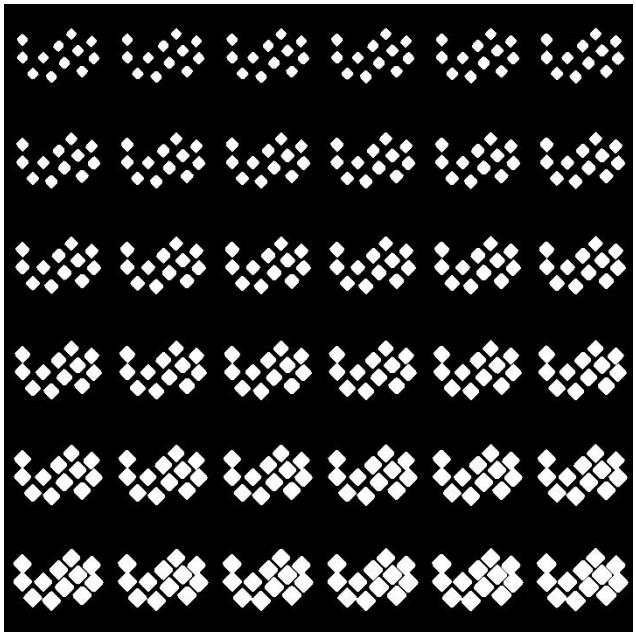

Figure 5 Image after applying series of dilation.

The resulting error function after applying all the opening and dilation operations:

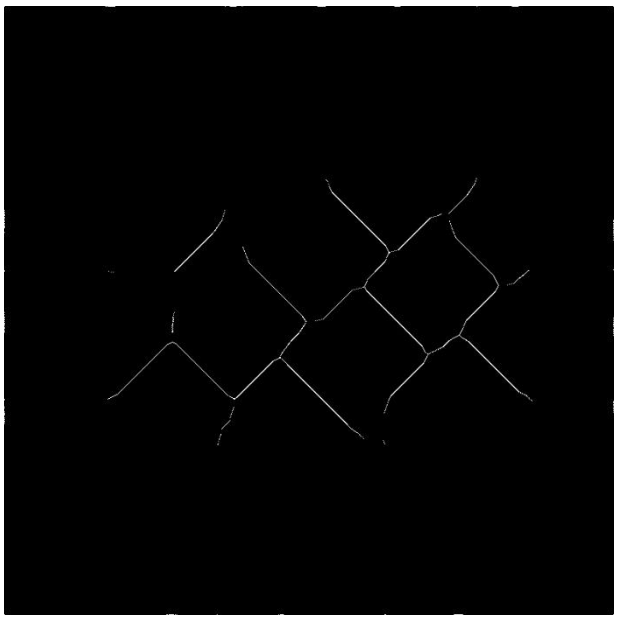

Figure 6 The borders between the ball. The borders are calculated whenever the expanding balls hit each other.

5) We subtract the image from the error

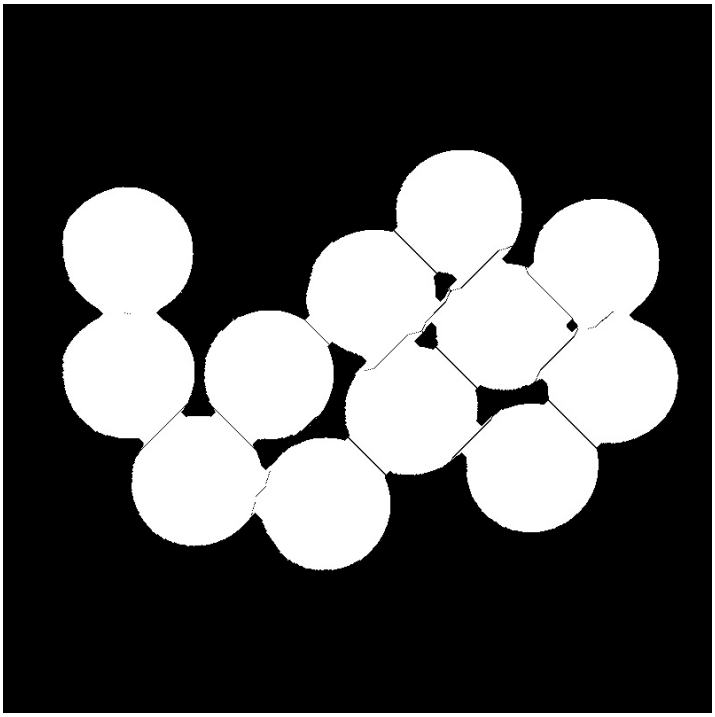

Figure 7 Substracting the binary image from the calculated borders.

6) We apply series of opening and erosion to make sure that the boundaries will be clear and that there will be a remarkable distance between the objects.

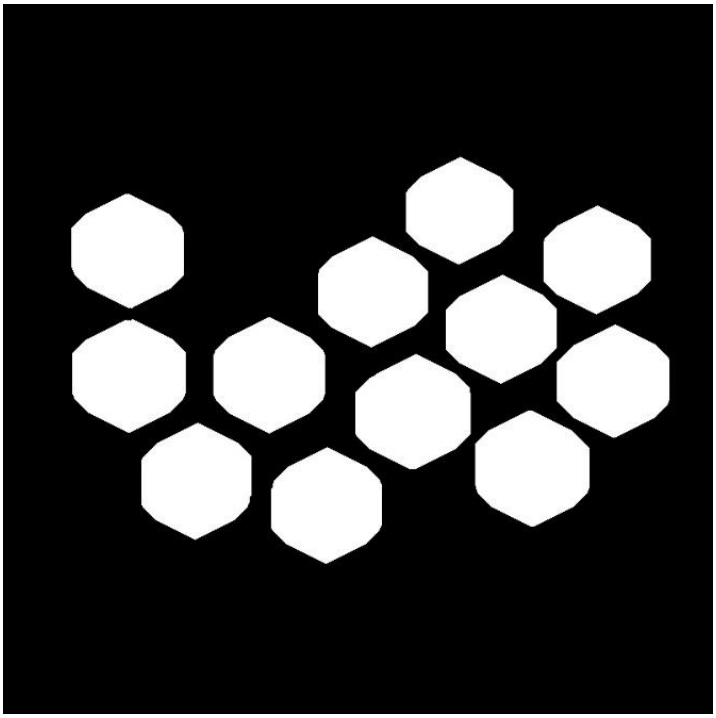

Figure 8 The segmented balls.

Questions:

1. **Does the opening result include the closing result?**
   No. Closing contains the opening result.
2. **What morphological filter should be applied to remove ledges from an object?**
   Opening.
3. **How can you find the object edges using morphological operations?**
   Subtract the image from the image after applying erosion on it.
4. **What is morphology?**
   morphology is a branch of mathematical morphology that deals with the analysis and processing of geometric structures in images. Morphological operations are based on set theory, and are used to extract image components that are useful in the representation and description of objects in an image.
   Morphological operations work by applying a structuring element, which is a small binary image, to the input image. The structuring element is used to probe the input image in a sliding window fashion, and the output image is produced by applying a logical operation to the pixel values within the window.

Code:

```
import cv2
import numpy as np
import os
import matplotlib.pyplot as plt


def decorator(fun):
    name=fun.__name__+'.jpg'
    par_dir=os.getcwd()
    path=os.path.join(par_dir,"outputs")
    try:
        os.mkdir(path)
    except OSError as error:
        pass
    def wrapper(I,size=3, *outnames):
        rows , cols = I. shape [0:2]
        out= fun(I,size)
        cv2.imshow("org", I)
        cv2.imshow("out", out)
        cv2.waitKey(0)
        cv2.destroyAllWindows()
        if outnames is not tuple():
            for i in range(len(outnames)):
                outpath = pth(outnames)
                cv2.imwrite(outpath, out)
        else:
            outpath=pth(name)
            cv2.imwrite(outpath, out)
        return out
    return wrapper

@decorator
def conv2bn(I , size=0):
    gray= cv2.cvtColor(I, cv2.COLOR_BGR2GRAY)
    t,bw=cv2.threshold(gray, 225,  255, cv2.THRESH_BINARY_INV)
    return bw


def morph_er(I, size=3, it=1):
    B=cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (size,size))
    out=cv2.morphologyEx(I, cv2.MORPH_ERODE, B, iterations=it)
    return out

def pth(name):
    par_dir=os.getcwd()
    return os.path.join(os.path.join(par_dir, "outputs"), name)

def morph_di(I,size=3):
    cl=cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (size,size))
```

```
    return cv2.morphologyEx(I, cv2.MORPH_DILATE, cl)


def morph_close(I,size=3, it=1):
    cl=cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (size,size))
    return cv2.morphologyEx(I, cv2.MORPH_CLOSE, cl, iterations=it)

def morph_open(I,size=3, it=1):
    cl=cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (size,size))
    return cv2.morphologyEx(I, cv2.MORPH_OPEN, cl, iterations=it)

def make_concat(I, fun, size=3):
    rows, cols= I.shape[0], I.shape[1]
    erosions=np.zeros((4*rows, 5*cols) , dtype=np.uint8) # This
matrix concatenates all the updates on the photo
    n=20
    j=0
    k=0
    for i in range(n):
        I=fun(I, size)
        if i%5==0 and i!=0:
            j=j+1
            k=0
        erosions[j*rows:rows*(j+1), k*cols: cols*(k+1)]= I
        k=k+1
    return [I, erosions]



I=cv2.imread("inputs/balls.jpg")
rows, cols= I.shape[0], I.shape[1]
bw=conv2bn(I)
closed=morph_close(bw,it=5)
out, erosions = make_concat(closed, morph_er, 7 )
E=np.zeros_like(out)
n=6
closes=np.zeros((n*rows, n*cols) , dtype=np.uint8)
i=0
k=0
j=0
marg=30
while not out.all():
    out_ref = morph_close(out)
    if (out_ref==out).all():
        out_ref=morph_di(out, 3)
        outclose= morph_close(out_ref)
        T=outclose-out_ref
    else:
        outclose=out_ref
        T=outclose-out
```

```
    out = outclose
    E=np.bitwise_or(E, T)
    k=k+1
    if k<marg:
        continue
    if (k-marg)%n==0 and (k-marg)!=0:
        i=0
        j=j+1
    if k <n**2+marg:
        closes[j*rows:rows*(j+1), i*cols:cols*(i+1)]=out

    i=i+1
res1=E*255+closed

res=morph_open(res1,size=7,it=22)
res=morph_er(res, size=3,it=7)
```