Assignment #4

Johnathan Spinelli (spinellj)

February 19, 2022

1 Theory

1.1 Morphology Operations

1.1.1 Erosion

Using the given structuring element, the eroded image would be:

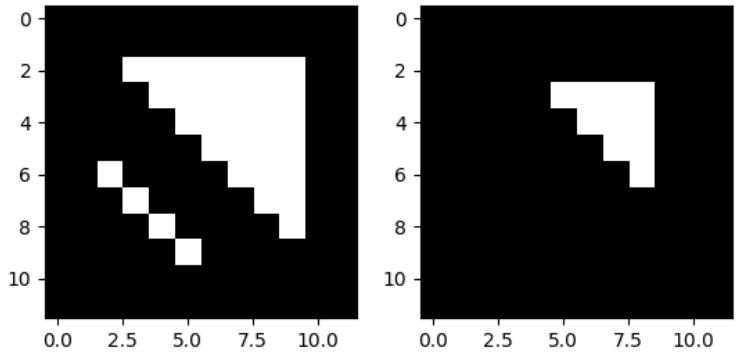

Figure 1: Original Image and Eroded Image

$$ErodedImage = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

1.1.2 Dilation

Using the given structuring element, the dilated image would be:

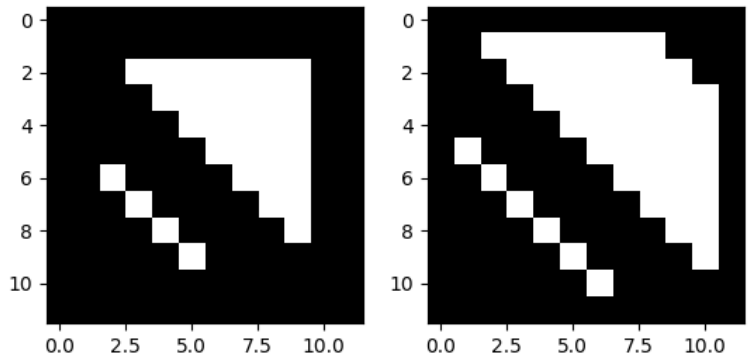

Figure 2: Original Image and Dilated Image

$$DilatedImage = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

1.1.3 Opening

Opening is a two-step process, where the image is dilated, then eroded. After the dilation, the image would be:

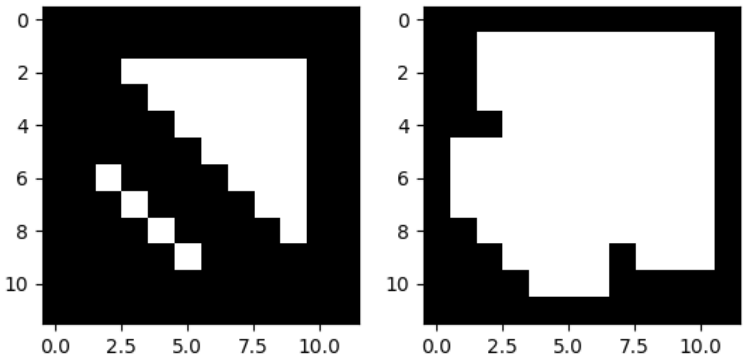

Figure 3: Original Image and Dilated Image (Closing 1/2)

$$OpenedImage1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Eroding the new image would result in:

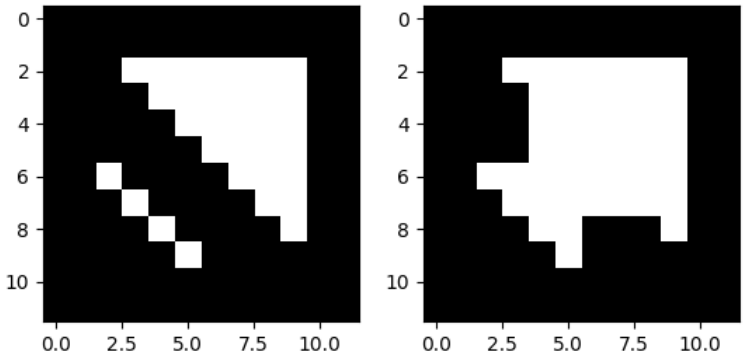

Figure 4: Original Image and Opened Image (Closing 1/2)

$$OpenedImage2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

1.2 Distance and Boundary

1.2.1 4-Neighbours and 8-Neighbours

In the sense of 4-neighbour boundaries, a pixel is considered a boundary if at least one of the pixel's horizontal or vertical neighbours are not within the region. In the sense of 8-neighbour boundaries, the same pixels would be included as before, but now pixels with diagonal neighbours not included in the region would also be considered as a boundary. The following images highlight the boundary pixels for each scenario:

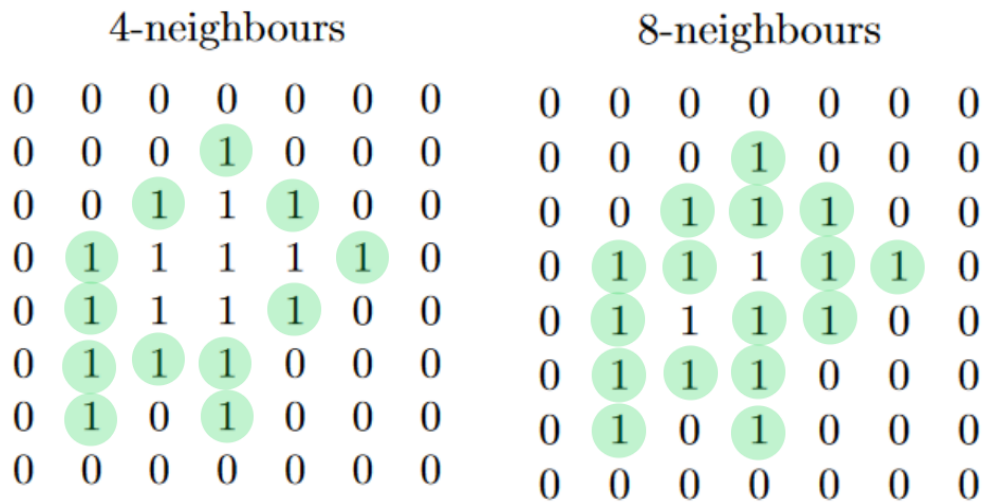

Figure 5: Boundary pixels in the sense of 4-neighbours and 8-neighbours

1.2.2 4-Neighbour Distances

Using the 4-neighbours sense boundary, the D_8 and Dm distances were calculated. In the D_8 scenario, the distance can be calculated using the equation $D_8(p,q) = max|x-s|, |y-t|$ Since point p is 5 pixels directly above point q, the D_8 value is 5. Using the D_m method, the path was drawn out, showing a 5 unit path mainly following diagonal neighbours. The distances are both 5

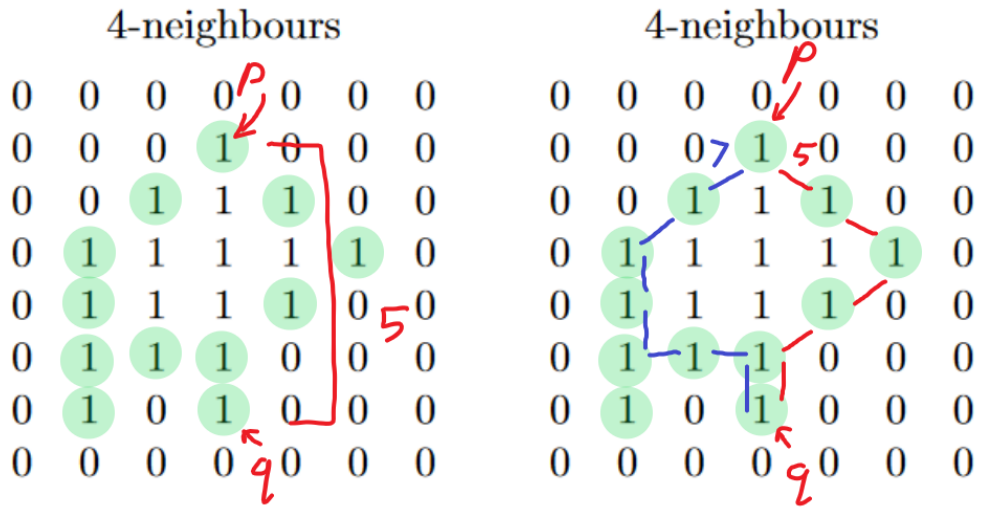

Figure 6: D_8 and D_m distances plotted on the 4-neighbour boundary

1.2.3 8-Neighbour Distances

Using the 8-neighbours sense boundary, the D_4 and Dm distances were calculated. In the D_4 scenario, The x distance and y distances were computed. The difference in x was 0, as pixel p is directly above pixel q. The difference in y was 5. The D_4 distance is the sum of these two values, so the result is 5. The D_m value was calculated by identifying the shortest path between q and p. In this scenario, the path only consists of lateral jumps and no diagonal jumps. The total length of the path, therefore the D_m value, is 7.

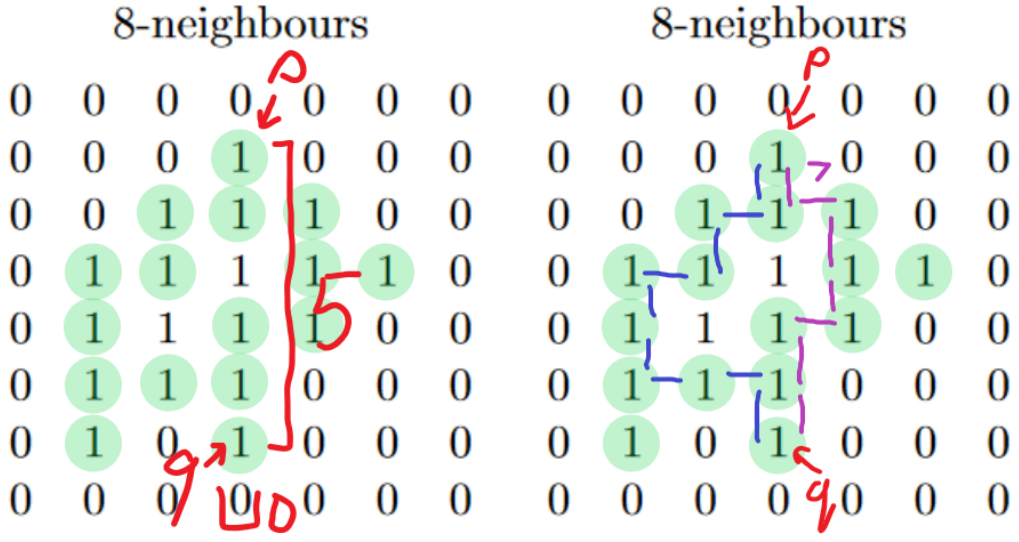

Figure 7: D_4 and D_m distances plotted on the 4-neighbour boundary

1.3 Extracting License Plate

Extracting the license plate will require a set of steps to prepare the image, examine the image, detect the contours, and optimize the selection. The first step is preprocessing. Before looking for the license plate in the image, the algorithm will use a Gaussian blur to filter out noise in the image (generally good practice). To identify edges and

outlines of shapes, the algorithm will use the Canny edge detector. These edges will then be turned into contours, which can provide details about the shape and size of the edges.

With a set of contours, the algorithm will search for the contour that best matches the license plate in the image. From the previous lab, the Canny edge detector identified the entire perimeter of the license plate, with some other smaller edges in the background. The contour that matches the license plate perimeter should be the largest rectangular contour. Once the rectangular contour is found, the algorithm will create a mask using the contour to extract the license plate from the original image.

2 Implementation

2.1 Extracting License Plate

The algorithm from the previous question was implemented, and works very well, as seen by the results below. This algorithm doesn't expect the license plate to be a perfect rectangle; it works with quadrilateral shapes in the case that the picture is taken from an angle, just like the given image. The scale of the image should not affect the results, as the image is extracted based on the edges of the image. The same applies to illumination conditions - if the image produces visible edges on the outer edge of the license plate, the algorithm should be able to extract them and produce a contour out of them.

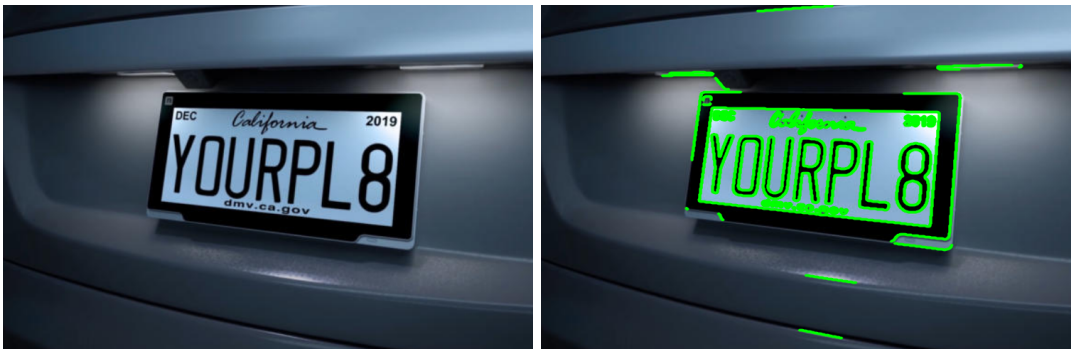

Figure 8: Original Image and all detected contours displayed

Figure 9: Optimal contour detected, extracted license plate from inside of contour

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

# Read image, make gray copy
img = cv2.imread('3.png')
img_g = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Blur and canny for contour detection
img_blur = cv2.GaussianBlur(img_g, (3,3), 0)
img_c = cv2.Canny(img_blur, 200, 250)
```

```
# Get contours
all_contours = img.copy()
contours, hierarchy = cv2.findContours(img_c,cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
cv2.drawContours(all_contours, contours, contourIdx=-1, color=(0,255,0),
                 thickness=2, lineType = cv2.LINE_AA)
cv2.imshow('All Contours', all_contours)
cv2.imwrite('all_contours.png', all_contours)

# Detected parameters
max_p = None
max_cont = None

for cont in contours:
    # First contour
    if max_p == None:
        max_p = cv2.arcLength(cont, True)
        max_cont = cont

    elif cv2.arcLength(cont,True) > max_p:
        # Longer contour than before
        max_p = cv2.arcLength(cont, True)
        max_cont = cont

        # If first fully enclosed rectangle, stop
        epsilon = 0.1*cv2.arcLength(cont, True)
        approx = cv2.approxPolyDP(cont, epsilon, True)
        if len(approx) >= 4:
            break

# If contour detected:
if max_cont.any():
    # Display contour on image
    good_contour = img.copy()
    cv2.drawContours(good_contour, [max_cont], contourIdx=-1, color=(0,255,0),
                     thickness=2, lineType = cv2.LINE_AA)
    cv2.imshow('Good Contour', good_contour)
    cv2.imwrite('good_contour.png', good_contour)

    # Create a mask, extract the image with mask
    mask = np.zeros(img_g.shape, np.uint8)
    cv2.drawContours(mask,[max_cont],0,255,-1,)
    cv2.imshow('mask',mask)
    extracted = cv2.bitwise_and(img,img,mask=mask)
    cv2.imshow('Extracted License',extracted)
    cv2.imwrite('extracted_license.png',extracted)

else:
    print("No contours found, sorry :(")

cv2.waitKey(0)
cv2.destroyAllWindows()
```