Όραση Υπολογιστών
Εργαστήριο 1

Αναστάσιος Στέφανος Αναγνώστου 03119051
Σπυρίδων Παπαδόπουλος 03119058

7 Απριλίου 2023

1 Μέρος 1ο

1.1 Δημιουργία Εικόνων Εισόδου

Το παρακάτω τμήμα κώδικα διαβάζει την εικόνα, μετατρέποντάς την σε γκρίζα, και την κανονικοποιεί. Στην συνέχεια, την εκτυπώνει, για να φανεί ότι διαβάστηκε επιτυχώς, και προστίθενται σε αυτήν θόρυβοι διαφορετικής εντάσεως, χρήσει των συναρτήσεων getstd.

```
def getstd(image, psnr):
    return (np.max(image)-np.min(image))/(10**(
        psnr/20))

# read the image, convert to gray scale and
    normalize it
image = cv2.imread("cv23_lab1_part12_material/
    edgetest_23.png", cv2.IMREAD_GRAYSCALE)
image = image.astype(np.float64)/image.max()

fig, axs = plt.subplots(1,1)
axs.imshow(image, cmap='gray')
axs.set_title("Original Image")
plt.show(block=False)
plt.pause(0.01)

# add noise to the images. 10db and 20db
# in the psnr context, less dBs equals more noise
image10db = image + np.random.normal(0, getstd(
    image, 10), image.shape)
image20db = image + np.random.normal(0, getstd(
    image, 20), image.shape)
```

1.2 Υλοποίηση Αλγορίθμων Ανίχνευσης Ακμών

Αρχικά, υλοποιείται κατάλληλη συνάρτηση για δημιουργία των δύο επιθυμητών γραμμικών φίλτρων, τα οποία προσεγγίζουν τα συνεχή φίλτρα με τις αποκρίσεις

1. 2D Gaussian $G_\sigma(x, y)$
2. Laplacian-of-Gaussian (LoG) $h(x, y) = \nabla^2 G_\sigma(x, y)$

Την δημιουργία των φίλτρων υλοποιεί η συνάρτηση `myfilter` η οποία χρησιμοποιεί την βοηθητική `my2dconv`.

```
def my2dconv(image, kernel):
    ix, iy = image.shape
    nx, ny = kernel.shape
    result = np.zeros((ix + nx - 1, iy + ny - 1))
    padded = np.pad(image, [(nx//2, nx//2),(ny//2,ny
        //2)], mode='constant')
    for i in range(nx//2, ix + nx//2):
        for j in range(ny//2, iy + ny//2):
            result[i, j] = np.sum(padded[i-nx//2: i+nx
                //2+1, j-ny//2: j+ny//2+1] * kernel)
    return result[nx//2:ix+nx//2, ny//2:iy+ny//2]

def myfilter(sigma, method):
    if not (method == "gaussian" or method == "log"):
        print("Error: method has to be either \"
            gaussian\" or \"log\"")
        exit(2)
    n = int(2*np.ceil(3*sigma)+1)
    gauss1D = cv2.getGaussianKernel(n, sigma)
    gauss2D = gauss1D @ gauss1D.T
    if (method == "gaussian"):
        return gauss2D
    laplacian = np.array([[0,1,0],
        [1,-4,1],
        [0,1,0]])
    logkernel = my2dconv(gauss2D, laplacian)
    return logkernel
```

Αφού δημιουργηθούν τα φίλτρα, επιχειρείται η προσέγγιση των σημείων μηδενισμού της λαπλασιανής με την περιγραφόμενη μέθοδο. Στην συνέχεια, απορρίπτονται τα σημεία αυτά στα οποία η εικόνα βρίσκεται κάτω από ένα κατώφλι. Τα αντίστοιχα κομμάτια κώδικα είναι ως εξής:

```
X = (L > 0).astype(np.uint8)
Y = (cv2.dilate(X, cross)) - (cv2.erode(X, cross))

gradx, grady = np.gradient(smooth)
```

```
grad = np.abs(gradx + 1j * grady)
D = ((Y == 1) & (grad > (theta * np.max(grad))))
```

Η ολοκληρωμένη συνάρτηση ανίχνευσης ακμών είναι τελικά:

```
def EdgeDetect(image, sigma, theta, method):
    if (not (method == "linear" or method == "
       nonlinear")):
        print("Error: method has to be either \"linear
           \" or \"nonlinear\"")
        exit(2)

    gaussf = myfilter(sigma, "gaussian")
    smooth = cv2.filter2D(image, -1, gaussf)
    cross = cv2.getStructuringElement(cv2.MORPH_CROSS,
        (3,3))
    if (method == "linear"):
        # construct the laplacian of gaussian kernel
        # and use it to filter the image
        logfilter = myfilter(sigma, "log")
        imgloged = cv2.filter2D(image, -1, logfilter)
    elif (method == "nonlinear"):
        # Perform morphological operations using
        # the cross structuring element
        imgloged = cv2.dilate(smooth, cross) + cv2.
           erode(smooth, cross) - 2*smooth

    # the imgloged variable is visible only if one of
       the if blocks
    # get executed. hopefully, this always happens
    L = imgloged

    # type uin8 is needed for compatibility with the
    # dilate and erode functions. otherwise, the
       matrix's
    # elements would have boolean type.
    X = (L > 0).astype(np.uint8)
    Y = (cv2.dilate(X, cross)) - (cv2.erode(X, cross))

    gradx, grady = np.gradient(smooth)
    grad = np.abs(gradx + 1j * grady)
    D = ((Y == 1) & (grad > (theta * np.max(grad))))
    return D
```

1.3 Αξιολόγηση των Αποτελεσμάτων Ανίσχνευσης Ακμών

Για την αξιολόγηση των αποτελεσμάτων, πρέπει να υπολογισθούν οι πραγματικές ακμές, χρήσει της μη θορυβημένης εικόνας, και να συγκριθούν με τις προκύπτουσες από την παραπάνω συνάρτηση βάσει ενός κριτηρίου ποιότητας. Η υλοποίηση του κριτηρίου είναι η εξής:

```
def QualityMetric(real, computed):
    # use the following names for compatibility
    # with the project's guide.
    T = real
    D = computed
    DT = (D & T)
    # the matrices are supposed to be boolean
    # therefore the sum() functions counts the
    # elements that are true / 1.
    cardT = T.sum()
    cardD = D.sum()
    cardDT = DT.sum()

    prTD = cardDT/cardT
    prDT = cardDT/cardD

    C = (prDT + prTD)/2
    return
```

Οι δε πραγματικές ακμές βρίσκονται από την μη θορυβημένη εικόνα ως εξής:

```
cross = cv2.getStructuringElement(cv2.MORPH_CROSS,
   (3,3))
M = cv2.dilate(image, cross) - cv2.erode(image, cross)
T = ( M > thetareal ).astype(np.uint8)
```

Τώρα, μπορούν να δοκιμαστούν διάφορες τιμές παραμέτρων εξομάλυνσης και κατωφλιού ώστε να ληφθούν τα βέλτιστα αποτελέσματα σε κάθε περίπτωση. Ο ολοκληρωμένος κώδικας είναι:

```
def edgedetectintro():
    # read the image, convert to gray scale and
       normalize it
    image = cv2.imread("cv23_lab1_part12_material/
       edgetest_23.png", cv2.IMREAD_GRAYSCALE)
    image = image.astype(np.float64)/image.max()

    fig, axs = plt.subplots(1,1)
    axs.imshow(image, cmap='gray')
```

```
axs.set_title("Original Image")
plt.show(block=False)
plt.pause(0.01)

# add noise to the images. 10db and 20db
# in the psnr context, less dBs equals more noise
image10db = image + np.random.normal(0, getstd(
    image, 10), image.shape)
image20db = image + np.random.normal(0, getstd(
    image, 20), image.shape)

# Play around with sigma and theta in order to
# obtain the best results.
noised_images = [image10db, image20db]
sigma = [1.5, 3]
theta = [0.2, 0.2]
thetareal = 0.01

for index, img in enumerate(noised_images):
    N1 = EdgeDetect(img,sigma[index],theta[index],
        "linear")
    N2 = EdgeDetect(img,sigma[index],theta[index],
        "nonlinear")

    # the non linear method gives the best results
    # therefore we name it D and continue our
        evaluation
    D = N2
    cross = cv2.getStructuringElement(cv2.
        MORPH_CROSS, (3,3))
    M = cv2.dilate(image, cross) - cv2.erode(image
        , cross)
    T = ( M > thetareal ).astype(np.uint8)
    print(T.shape)

    fig, axs = plt.subplots(2,2)
    axs[0,0].imshow(img, cmap='gray')
    axs[0,0].set_title("Noised Image")
    axs[0,1].imshow(N1, cmap='gray')
    axs[0,1].set_title("Linear method")
    axs[1,0].imshow(N2, cmap='gray')
    axs[1,0].set_title("Non linear method")
    axs[1,1].imshow(T, cmap='gray')
    axs[1,1].set_title("Actual Edges")
    plt.show(block=False)
    plt.pause(0.01)
```

```
        plt.savefig(f"image-plots/edges-intro{index}.
            jpg")

        C = QualityMetric(T, D)
        print(f"The quality criterion is C[{index}] =
            {C}")
```

1.4 Εφαρμογή των Αλγορίθμων Ανίχνευσης Ακμών σε Πραγματικές Εικόνες

2 Μέρος 2ο