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`. Στο σημείο αυτό σημειώνεται, ότι ο θόρυβος 10dB είναι μεγαλύτερης ένταση του 20dB.

```
    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

Για την ανίχνευση ακμών, 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

    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 = [3, 1.5]
    theta = [0.2, 0.2]
    thetareal = 0.08

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

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

        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, N1)
        print(f"Linear method: C[{index}] = {C}")
        C = QualityMetric(T, N2)
        print(f"Non linear method: C[{index}] = {C}")
```

Κατόπιν δοκιμών, οι παρακάτω παράμετροι έδιναν τα βέλτιστα αποτελέσματα.

$$\begin{aligned} \sigma = \begin{bmatrix} 3 & 1.5 \end{bmatrix}, \theta = \begin{bmatrix} 0.2 & 0.2 \end{bmatrix}, \theta_{real} = 0,08 \\ C_{10db-lin} = 0.6182010279466628 \\ C_{10db-non} = 0.7362110886293316 \\ C_{20db-lin} = 0.9375887192342924 \\ C_{20db-non} = 0.9671201094187654 \end{aligned} \tag{1.1}$$

1.4 Εφαρμογή των Αλγορίθμων Ανίχνευσης Ακμών σε Πραγματικές Εικόνες

Η παραπάνω αναπτυχθείσα μέθοδος δοκιμάζεται τώρα σε πραγματική, μη θορυβημένη εικόνα. Η εικόνα είναι ένα στιγμιότυπο της πόλης Κιότο της Ιαπωνίας και περιέχει αρκετές λεπτές ακμές. Για τον λόγο αυτόν, ήταν αναγκαία η χρήση μικρής παραμέτρου εξομάλυνσης, ώστε να μπορέσουν να εντοπιστούν οι ακμές, αλλά όχι τόσο μικρής ώστε να αναγνωριστούν ψεύτικες ακμές.

```
def edgedetectreal():
    # read image and normalize it
    kyoto = cv2.imread("cv23_lab1_part12_material/
       kyoto_edges.jpg", cv2.IMREAD_GRAYSCALE)
    kyoto = kyoto.astype(np.float64)/kyoto.max()

    # play around with sigma and theta
    # big sigma => much smoothing => not fine details
    # big theta => less edges
    # small theta => many edges

    sigma = 0.3
    theta = 0.2
    thetareal = 0.23

    N1 = EdgeDetect(kyoto, sigma, theta, "linear")
    N2 = EdgeDetect(kyoto, sigma, theta, "nonlinear")

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

    fig, axs = plt.subplots(2,2)
    axs[0,0].imshow(kyoto, cmap='gray')
    axs[0,0].set_title("Gray Image")
    axs[0,1].imshow(T, cmap='gray')
    axs[0,1].set_title("Actual Edges")
```

```
    axs[1,0].imshow(N1, cmap='gray')
    axs[1,0].set_title("Linear edge detection")
    axs[1,1].imshow(N2, cmap='gray')
    axs[1,1].set_title("Non Linear edge detection")
    plt.show(block=False)
    plt.pause(0.01)
    plt.savefig("image-plots/edges-real.jpg")

    C = QualityMetric(T, N1)
    print(f"Linear edge detection is C = {C}")
    C = QualityMetric(T, N2)
    print(f"Non linear edge detection is C = {C}")
```

Τελικά, οι επιλεχθείσες παράμετροι και τα αντίστοιχα αποτελέσματα ήταν:

$$\begin{gathered} \sigma = 0.3, \theta = 0.2, \theta_{real} = 0.23 \\ Linearedgedetection is C = 0.818604135617401 \\ Nonlinearedgedetection is C = 0.8188562865181751 \end{gathered} \tag{1.2}$$

Παρατηρείται, ότι τόσο η γραμμική όσο και η μη γραμμική μέθοδος δίνουν παραπλήσια και ικανοποιητικά αποτελέσματα, με την μη γραμμική μέθοδο να είναι ελαφρώς πιο αποτελεσματική.

2 Μέρος 2ο

Στο μέρος αυτό επιχειρείται ο γενικότερος εντοπισμός σημείων ενδιαφέροντως. Για αυτόν τον λόγο, αναπτύσσονται διάφορες τεχνικές, οι οποίες αναζητούν διαφόρων ειδών σημεία, σε μία ή πολλαπλές κλίμακες.

2.1 Βοηθητικές Συναρτήσεις

Κατά την διάρκεια εκπόνησης της εργασίας, παρατηρήθηκε ότι ο ίδιος κώδικας επαναλαμβανόταν σε σχεδόν όλες τις μεθόδους, απλώς με διαφορετικές παραμέτρους. Προς αποφυγή, λοιπόν, κουραστικής επανάληψης, αναπτύχθηκαν οι ακόλουθες συναρτήσεις, οι οποίες θα χρησιμοποιούνται στις παρακάτω μεθόδους με σκοπό να είναι πιο εύληπτη η διατύπωσή τους.

Η συνάρτηση `InterestPointCoord` επιστρέφει τις συντεταγμένες των σημείων ενδιαφέροντος, βάσει του εκάστοτε κριτηρίου.

```
def InterestPointCoord(r, sigma, theta):
    # r is a previously evaluated criterion
    # sigma is used for the size of the structure
    # theta is a threshold

    # evaluate the following 2 conditions
    # condition 1
    ns = 2*np.ceil(3*sigma) + 1
```

```
    bsq = disk_strel(ns)
    cond1 = ( r == cv2.dilate(r, bsq) )
    # condition 2
    maxr = np.max(r)
    cond2 = ( r > theta * maxr )
    # choose the pixels that satisfy both of them
    # return their coordinates and their scale
    x, y = np.where(cond1 & cond2)
    # for compatibility with the utility function
    # provided by the lab staff, the y coordinate
    # has to come before the x coordinate
    indices = np.column_stack((y,x))
    return indices
```

Η συνάρτηση LogMetric εφαρμόζει την LoG μετρική ώστε να επιλέξει ως σημεία ενδιαφέροντος αυτά τα οποία μεγιστοποιούν την μετρική σε γειτονιά εύρους 3 κλιμάκων. Επιστρέφει τις συντεταγμένες των σημείων ενδιαφέροντος και την κλίμακα του καθενός.

```
def LogMetric(logs, itemsperscale, N):
    # log((x,y), s) = (s^2)|Lxx((x,y),s) + Lyy((x,y),s
       )|
    # returns the coordinates of the points that
       maximize
    # the log metric in a neighborhood of 3 scales
    # (prev scale), (curr scale), (next scale)
    final = []
    for index, items in enumerate(itemsperscale):
        logp = logs[max(index-1,0)]
        logc = logs[index]
        logn = logs[min(index+1,N-1)]
        for triplet in items:
            x = int(triplet[1])
            y = int(triplet[0])
            prev = logp[x,y]
            curr = logc[x,y]
            next = logn[x,y]
            if (curr >= prev) and (curr >= next):
                final.append(triplet)
    return np.array(final)
```

Τέλος, η συνάρτηση smooth_gradient υπολογίζει τις παραγώγους της εξομαλυμένης εικόνας. Αναλόγως της παραμέτρου deg επιστρέφει πρώτου, δευτέρου ή και των δύο βαθμών τις παραγώγους.

```
def smooth_gradient(image, sigma, deg):
    # define the filters according to the arguments
    Gs = myfilter(sigma, "gaussian")
```

```
    # smoothen the image
    smooth = cv2.filter2D(image, -1, Gs)
    # calculate the gradient on both directions
    gradx, grady = np.gradient(smooth)
    if (deg==1):
        return (gradx, grady)
    elif (deg==2):
        gradxx, gradxy = np.gradient(gradx)
        gradxy, gradyy = np.gradient(grady)
        return (gradxx, gradxy, gradyy)
    elif (deg==3):
        gradxx, gradxy = np.gradient(gradx)
        gradxy, gradyy = np.gradient(grady)
        return (gradx, grady, gradxx, gradxy, gradyy)
    print("deg = 1 for (gradx, grady)")
    print("deg = 2 for (gradxx, gradxy, gradyy)")
    print("deg = 3 for (gradx, grady, gradxx, gradxy,
       gradyy)")
    exit(2)
```

2.2 Ανίχνευση Γωνιών

2.3 Πολυκλιμακωτή Ανίχνευση Γωνιών

2.4 Ανίχνευση Blobs

2.5 Πολυκλιμακωτή Ανίχνευση Blobs

2.6 Επιτάχυνση Ανίχνευσης Blobs

2.7 Επιτάχυνση Πολυκλιμακωτή Ανίχνευσης Blobs