

Ψηφιακή Επεξεργασία Εικόνας

Καϊλίδης Κύριλλος ΑΜ: 4680

Εαρινό Εξάμηνο.2023-2024

Ζήτημα 1: Patches

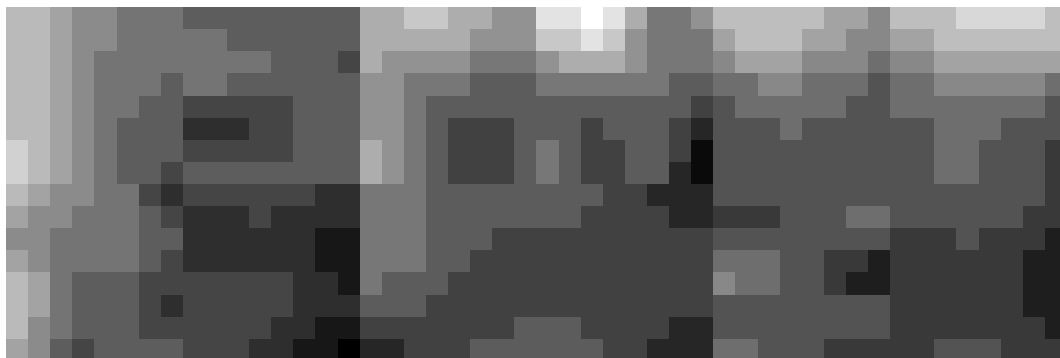
(a) Υλοποίηση της Συνάρτησης `image_patches()`

Η συνάρτηση `image_patches` έχει υλοποιηθεί στο αρχείο `filters.py`. Η συνάρτηση αυτή διαιρεί μια εικόνα σε ασπρόμαυρη κλίμακα σε μη επικαλυπτόμενα τμήματα εικόνας διαστάσεων `16x16 pixels`. Κάθε τμήμα κανονικοποιείται ώστε να έχει μέση τιμή μηδέν και διακύμανση ίση με τη μονάδα.

Ο κώδικας της συνάρτησης `image_patches` είναι ο εξής:

```
def image_patches(image, patch_size=(16, 16)):
    patches = []
    H, W = image.shape
    M, N = patch_size
    for i in range(0, H, M):
        for j in range(0, W, N):
            patch = image[i:i+M, j:j+N]
            if patch.shape == (M, N): # Ensure patch is of correct size
                # Normalize patch to have zero mean and unit variance
                patch = (patch - np.mean(patch)) / np.std(patch)
            patches.append(patch)
    return patches
```

Τα τρία δείγματα τμημάτων από την εικόνα `grace_hopper.png` εμφανίζονται παρακάτω:



(b) Κανονικοποίηση Τμημάτων με Μηδενική Μέση Τιμή

Η κανονικοποίηση των τμημάτων ώστε να έχουν μηδενική μέση τιμή και διακύμανση ίση με τη μονάδα είναι σημαντική για την ανάλυση ομοιότητας μεταξύ των τμημάτων. Η μέση τιμή αφαιρεί τη συνολική ένταση φωτεινότητας, ενώ η διακύμανση ομαλοποιεί την κατανομή των τιμών των pixel.

Συζήτηση:

Εάν θέλουμε να μετρήσουμε την ομοιότητα μεταξύ τμημάτων χρησιμοποιώντας εσωτερικά γινόμενα, η μηδενική μέση τιμή βοηθά στην εξάλειψη της επιρροής από τις διαφορετικές φωτεινότητες των εικόνων. Έτσι, οι διαφορές στη φωτεινότητα ή τον φωτισμό δεν επηρεάζουν τη σύγκριση των τμημάτων. Για παράδειγμα, με μια τιμή όπου το σκοτεινό αντιστοιχεί στο 0 και το φωτεινό στο 1, υπάρχει απώλεια πληροφορίας σχετικά με τη μέση φωτεινότητα, η οποία μπορεί να επηρεάσει τη σύγκριση. Αντίθετα, με τιμές που κυμαίνονται από -1 έως 1, η κατανομή είναι συμμετρική γύρω από το μηδέν, διευκολύνοντας την ανίχνευση ομοιότητας ανεξάρτητα από τον συνολικό φωτισμό.

(c) Εφαρμογή των Τμημάτων σε Προβλήματα Ευθυγράμμισης και Αναγνώρισης Αντικειμένων

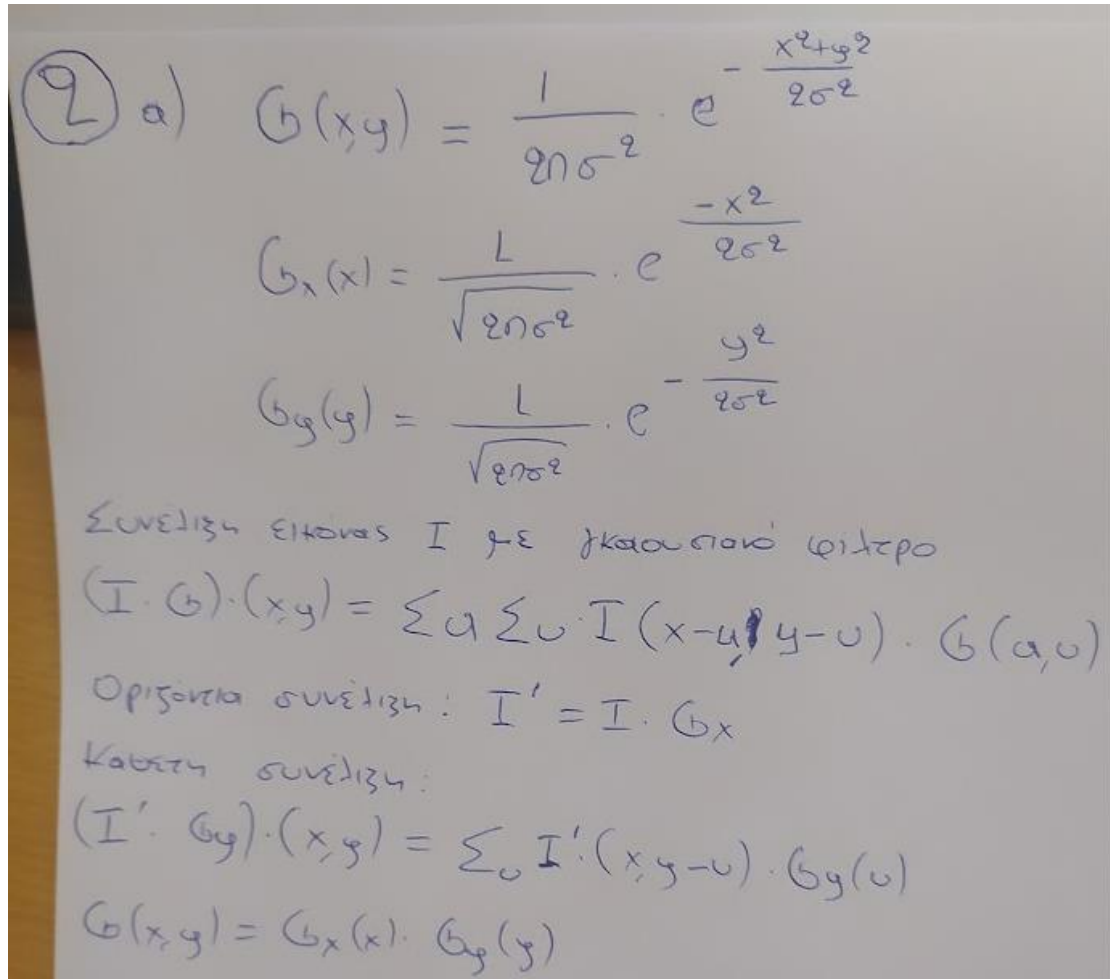
Η χρήση των τμημάτων με μηδενική μέση τιμή είναι χρήσιμη για προβλήματα όπως η αντιστοίχιση (matching) ή η αναγνώριση αντικειμένων, καθώς εξασφαλίζει ότι οι διαφορές στη φωτεινότητα δεν επηρεάζουν την ανάλυση. Ωστόσο, αυτά τα τμήματα μπορεί να μην είναι ιδανικά για όλα τα προβλήματα, εξαιτίας:

1. Της θέσης: Τα τμήματα είναι τοπικά και μπορεί να μην περιέχουν ολόκληρο το αντικείμενο, επηρεάζοντας την ευθυγράμμιση αν το αντικείμενο μετακινηθεί ή περιστραφεί.
2. Της κλίμακα: Αν αλλάξει η κλίμακα του αντικειμένου, τα τμήματα ενδέχεται να μην ταιριάζουν πλέον με τα αρχικά, καθιστώντας δυσκολότερη την αναγνώριση.
3. Της έντασης φωτεινότητας: Αν και η κανονικοποίηση βοηθά, ενδέχεται να μην ανιχνεύει όλες τις λεπτές διαφορές στη δομή που προκαλούνται από διαφορετικά φωτιστικά περιβάλλοντα.

Συνεπώς, ενώ τα τμήματα με μηδενική μέση τιμή είναι χρήσιμα για την αντιμετώπιση διαφορών στη φωτεινότητα, άλλες παράμετροι όπως η θέση, η κλίμακα, και οι λεπτομέρειες της έντασης μπορεί να απαιτούν περαιτέρω επεξεργασία ή άλλες τεχνικές για να επιτευχθεί ακρίβεια στην αντιστοίχιση και αναγνώριση αντικειμένων.

Ζήτημα 2: Συνέλιξη και Γκαουσιανό Φίλτρο

(a) Απόδειξη Ισοδυναμίας 2Δ και 1Δ Γκαουσιανών Φίλτρων



② a) $G(x,y) = \frac{1}{2\pi\sigma^2} \cdot e^{-\frac{x^2+y^2}{2\sigma^2}}$

$G_x(x) = \frac{L}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{x^2}{2\sigma^2}}$

$G_y(y) = \frac{L}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{y^2}{2\sigma^2}}$

Συνέλιξη εικόνας I με γκαουσιανό φίλτρο

$(I \cdot G)(x,y) = \sum_u \sum_v I(x-u, y-v) \cdot G(u,v)$

Οριζόντια συνέλιξη: $I' = I \cdot G_x$

Κατακόρυφη συνέλιξη:

$(I' \cdot G_y)(x,y) = \sum_v I'(x, y-v) \cdot G_y(v)$

$G(x,y) = G_x(x) \cdot G_y(y)$

Αυτό σημαίνει ότι η συνέλιξη με το 2Δ Γκαουσιανό φίλτρο είναι ισοδύναμη με την εφαρμογή δύο 1Δ Γκαουσιανών φίλτρων διαδοχικά, και οι διακυμάνσεις (σ^2) των 1Δ και 2Δ φίλτρων είναι ίδιες. Δηλαδή, η τυπική απόκλιση σ που χρησιμοποιούμε για τα φίλτρα είναι κοινή και στις δύο περιπτώσεις.

(b) Συνάρτηση `convolve()`

Η συνάρτηση `convolve` έχει υλοποιηθεί στο αρχείο `filters.py` για να εκτελεί συνέλιξη με συμπλήρωση μηδενικών. Ακολουθεί ο κώδικας της συνάρτησης:

```
def convolve(image, kernel):
    """
    --- Zitima 2.b ---
    Return the convolution result: image * kernel.
    Reminder to implement convolution and not cross-correlation!
    Caution: Please use zero-padding.

    Input- image: H x W
         |   kernel: h x w
    Output- convolve: H x W
    """
    H, W = image.shape
    h, w = kernel.shape
    pad_height = h // 2
    pad_width = w // 2

    # Pad image with zeros
    padded_image = np.pad(image, ((pad_height, pad_height), (pad_width, pad_width)), mode='constant')
    output = np.zeros_like(image)

    # Flip the kernel for convolution
    kernel_flipped = np.flip(kernel)

    # Perform convolution
    for i in range(H):
        for j in range(W):
            region = padded_image[i:i+h, j:j+w]
            output[i, j] = np.sum(region * kernel_flipped)

    return output
```

(c) Εφαρμογή Gaussian Φίλτρου και Αποτελέσματα

Εφαρμόσαμε ένα Gaussian φίλτρο 3x3 με τυπική απόκλιση $\sigma=0.572$ στην εικόνα `grace_hopper.png`. Το αποτέλεσμα παρουσιάζεται παρακάτω:



(d) Σημασία του Αθροίσματος Συντελεστών ίσου με 1

Είναι σημαντικό για ένα φίλτρο εξομάλυνσης να έχει άθροισμα συντελεστών ίσο με 1 για τους εξής λόγους:

- **Διατήρηση Έντασης:** Ένα φίλτρο με άθροισμα 1 διατηρεί την ένταση της εικόνας, καθώς η συνολική φωτεινότητα της εικόνας παραμένει αμετάβλητη.
- **Αποφυγή Παραμορφώσεων:** Εξασφαλίζει ότι το φιλτράρισμα δεν προσθέτει ή αφαιρεί συνολική φωτεινότητα, αποφεύγοντας την παραμόρφωση των τιμών της εικόνας.

(e) Πυρήνες Συνέλιξης για Παράγωγους

Οι πυρήνες συνέλιξης για τον υπολογισμό των παραγώγων είναι:

$$K_x = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

$$K_y = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

(f) Ρουτίνα edge_detection()

Η συνάρτηση edge_detection() υπολογίζει το μέτρο της κλίσης της εικόνας. Ακολουθεί ο κώδικας της συνάρτησης:

```
def edge_detection(image):  
    """  
    --- Zitima 2.f ---  
    Return Ix, Iy and the gradient magnitude of the input image.  
  
    Input- image: H x W  
    Output- Ix, Iy, grad_magnitude: H x W  
    """  
  
    # Sobel kernels for edge detection  
    kx = np.array([-1, 0, 1]).reshape((1, 3)) # 1 x 3  
    ky = np.array([-1, 0, 1]).reshape((3, 1)) # 3 x 1  
  
    Ix = convolve(image, kx)  
    Iy = convolve(image, ky)  
  
    # Gradient magnitude  
    grad_magnitude = np.sqrt(Ix**2 + Iy**2)  
  
    return Ix, Iy, grad_magnitude
```

(g) Σύγκριση Ακμών Πριν και Μετά την Εφαρμογή Γκαουσιανού Φίλτρου

Χρησιμοποιήσαμε την αρχική εικόνα και την εικόνα που έχει φιλτραριστεί με το Γκαουσιανό φίλτρο ως είσοδο στην `edge_detection()`. Παρακάτω παρουσιάζονται οι έξοδοι:

q3_edge:



q3_edge_gaussian:



Διαφορά: Η εικόνα που έχει φιλτραριστεί με το Γκαουσιανό φίλτρο έχει πιο λείες ακμές και λιγότερο θόρυβο σε σύγκριση με την αρχική εικόνα. Το φίλτρο μειώνει τις απότομες μεταβολές έντασης, κάνοντας τις ακμές λιγότερο ευδιάκριτες αλλά πιο καθαρές.

Ζήτημα 3: Τελεστής Sobel

a)

Υπολογισμός της Συνέλιξης

Για την συνέλιξη του 3x3 Gaussian φίλτρου GS με τον οριζόντιο πυρήνα kx πολλαπλασιάζουμε τους 2 πίνακες οπότε έχουμε:

$$S_x = G_s * k_x = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} * \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Χρησιμοποιούμε την ιδιότητα της συνέλιξης: $d/dx(I * G_s) = I * (d/dx)G_s$ και μόλις δείξαμε ότι $(d/dx)G_s = S_x$.

Από τα παραπάνω προκύπτει $d/dx(I * G_s) = I * S_x$

b) Υλοποίηση της Συνάρτησης `sobel_operator()`

Η συνάρτηση `sobel_operator()` έχει υλοποιηθεί στο αρχείο `filters.py`. Ακολουθεί ο κώδικας της συνάρτησης:

```

def sobel_operator(image):
    """
    --- ZitiMa 3.b ---
    Return Gx, Gy, and the gradient magnitude using the Sobel operator.

    Input- image: H x W
    Output- Gx, Gy, grad_magnitude: H x W
    """

    # Sobel filters
    Sx = np.array([[1, 0, -1],
                   [2, 0, -2],
                   [1, 0, -1]]) # Horizontal
    Sy = np.array([[1, 2, 1],
                   [0, 0, 0],
                   [-1, -2, -1]]) # Vertical

    Gx = convolve(image, Sx)
    Gy = convolve(image, Sy)

    # Gradient magnitude
    grad_magnitude = np.sqrt(Gx**2 + Gy**2)

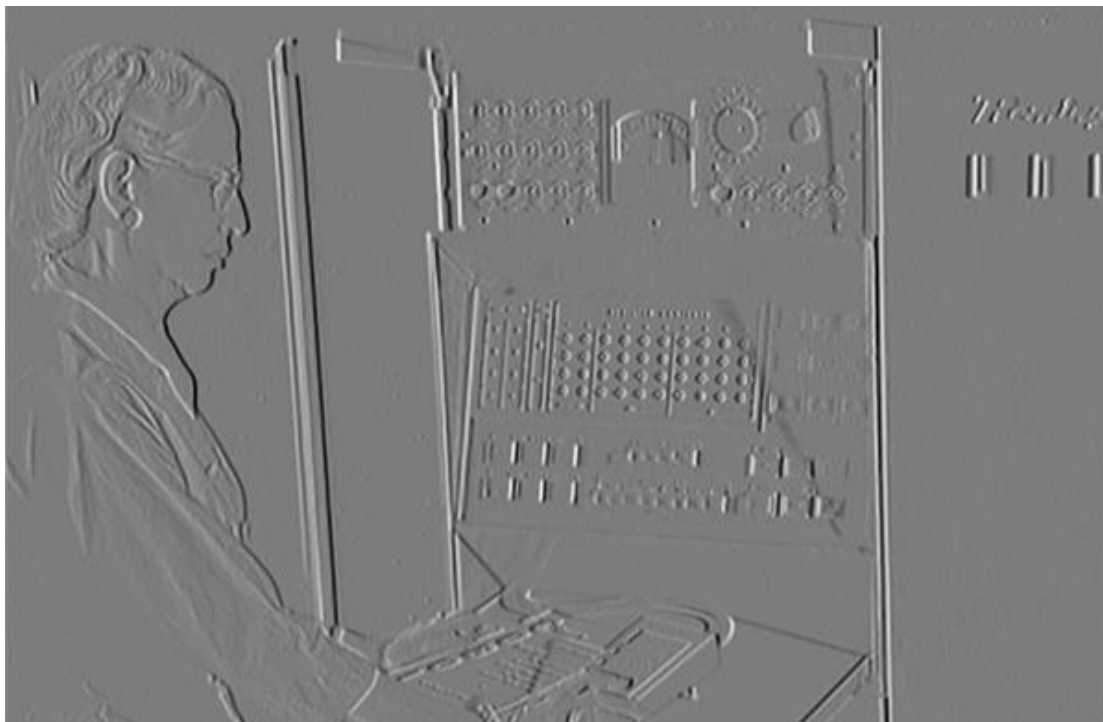
    return Gx, Gy, grad_magnitude

```

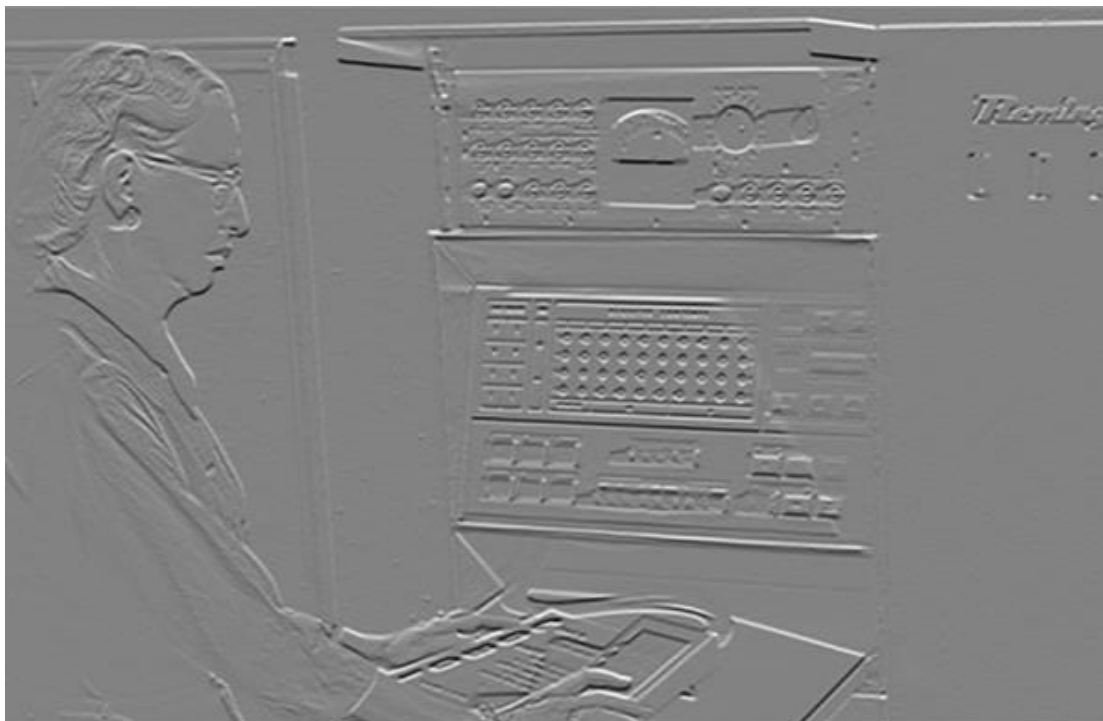
c) Αποτελέσματα Εφαρμογής του Τελεστή Sobel

Εφαρμόσαμε τον τελεστή Sobel στην εικόνα `grace_hopper.png` και παρακάτω παρουσιάζουμε τα αποτελέσματα:

1. Οριζόντια Συνέλιξη I*Sx:



2. Κατακόρυφη Συνέλιξη I*Sy:



3) Μέτρο Κλίσης:

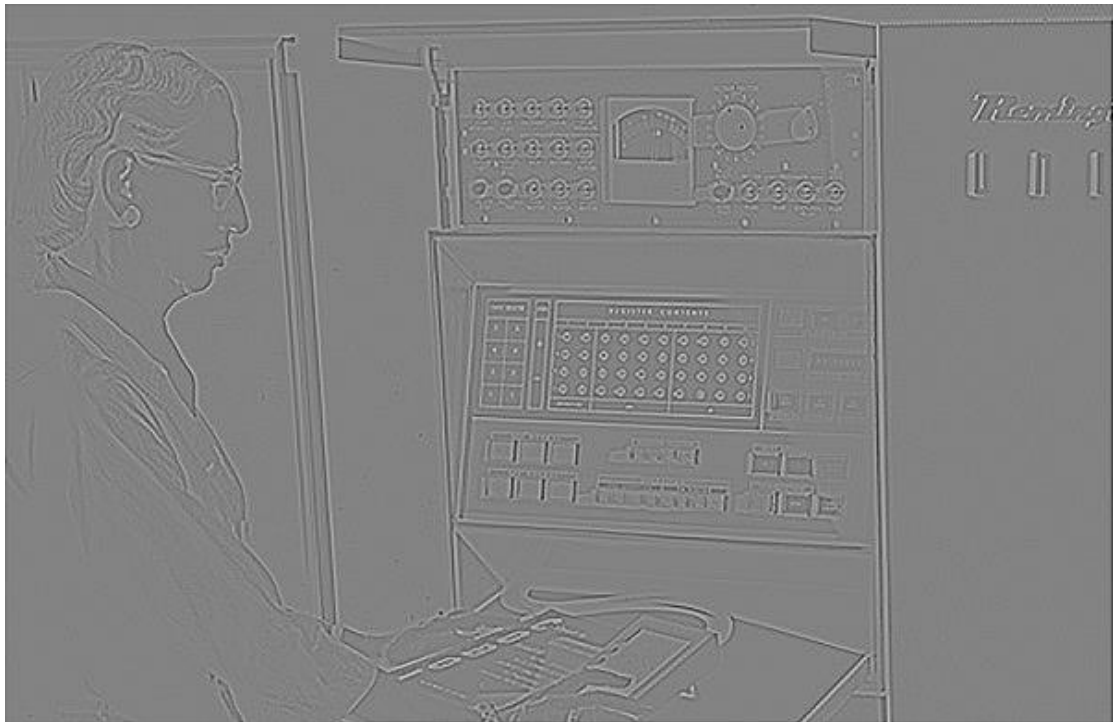


Ζήτημα 4: Λαπλασιανή της Γκαουσιανής της εικόνας (LoG)

ι) Εξόδοι των φίλτρων LoG

Εφαρμόσαμε τα φίλτρα LoG1 και LoG2 στην εικόνα `grace_hopper.png`. Τα αποτελέσματα παρουσιάζονται παρακάτω:

Log1:



Log2:



ii) Διαφορές Μεταξύ των Φίλτρων:

Η κεντρική τιμή στο φίλτρο LoG1 είναι θετική, ενώ στο φίλτρο LoG2 είναι αρνητική λόγω της αναστροφής του σήματος.

Και τα δύο φίλτρα είναι συμμετρικά, αλλά το φίλτρο LoG2 έχει έναν αρνητικό πυρήνα γύρω από το κέντρο που είναι πιο εκτεταμένος σε σύγκριση με το φίλτρο LoG1.

Το φίλτρο LoG2 είναι κλιμακωμένο για να μειώσει την ένταση της απόκρισης του φίλτρου. Αυτό επηρεάζει την ένταση των ανιχνευμένων ακμών.

Ικανότητα Ανίχνευσης Ακμών:

Και τα δύο φίλτρα LoG μπορούν να ανιχνεύσουν ακμές καθώς τονίζουν τις περιοχές όπου η δεύτερη παράγωγος της έντασης της εικόνας αλλάζει γρήγορα. Τα σημεία με υψηλή απόκριση κατά απόλυτη τιμή αντιπροσωπεύουν περιοχές με έντονες μεταβολές στην ένταση της εικόνας, που συνήθως αντιστοιχούν σε ακμές.

Εκτός από την ανίχνευση ακμών, τα φίλτρα LoG μπορούν επίσης να ανιχνεύσουν περιοχές με σημεία υψηλής καμπυλότητας.