MYE046 – Υπολογιστική Όραση: Άνοιξη 2023

2η Σειρά Ασκήσεων: 25% του συνολικού βαθμού

Διδάσκων: Άγγελος Γιώτης

- ΠΑΡΑΔΟΣΗ: **Σάββατο, 13 Μαΐου, 2023 23:59**

Γενικές Οδηγίες

Απαντήστε στα παρακάτω ζητήματα χρησιμοποιώντας Python στο συνημμένο σημειωματάριο Jupyter και ακολουθήστε τις παρακάτω οδηγίες:

- Οι ασκήσεις είναι **ατομικές** - δεν επιτρέπεται η μεταξύ σας συνεργασία για την υλοποίηση/παράδοσή τους.
- Δεν επιτρέπεται να χρησιμοποιήσετε κώδικα που τυχόν θα βρείτε στο web (είτε αυτούσιο, είτε παραγόμενο από AI). Η χρήση κώδικα τρίτων θα έχει σαν αποτέλεσμα τον αυτόματο μηδενισμό σας.
- Όλες οι λύσεις πρέπει να είναι γραμμένες σε αυτό το σημειωματάριο `Jupyter notebook`.
- Ο κώδικάς σας πρέπει να σχολιαστεί εκτενώς.
- Αφού ολοκληρώσετε (υλοποιήσετε και εκτελέσετε) τις απαντήσεις σας στο σημειωματάριο (notebook), εξαγάγετε το notebook ως PDF και υποβάλετε, τόσο το σημειωματάριο όσο και το PDF (δηλαδή τα αρχεία `.ipynb` και `.pdf`) στο `turnin` του μαθήματος, μαζί με ένα συνοδευτικό αρχείο `onoma.txt` που θα περιέχει το ον/μο σας και τον Α.Μ. σας.
- Οι απαντήσεις θα παραδοθούν με την εντολή: **turnin assignment_2@mye046 onoma.txt assignment2.ipynb assignment2.pdf**
- Μπορείτε να χρησιμοποιήσετε βασικά πακέτα γραμμικής άλγεβρας (π.χ. `NumPy`, `SciPy` κ.λπ.), αλλά δεν επιτρέπεται να χρησιμοποιείτε τα πακέτα/βιβλιοθήκες που επιλύουν άμεσα τα προβλήματα. Μη διστάσετε να ρωτήσετε τον διδάσκοντα εάν δεν είστε σίγουροι για τα πακέτα που θα χρησιμοποιήσετε.
- Συνιστάται ιδιαίτερα να αρχίσετε να εργάζεστε στις ασκήσεις σας το συντομότερο δυνατό!

Late Policy: Εργασίες που υποβάλλονται καθυστερημένα θα λαμβάνουν μείωση βαθμού 10% για κάθε 24 ώρες καθυστέρησης. Οι εργασίες δεν θα γίνονται δεκτές 72 ώρες (3

ημέρες) μετά την προθεσμία παράδοσης. Για παράδειγμα, παράδοση της εργασίας 2 ημέρες μετά την προθεσμία βαθμολογείται με άριστα το 20 (από 25).

Άσκηση 1: Φιλτράρισμα Εικόνας (image filtering) [10 μονάδες]

Ζήτημα 1.1 Υλοποίηση συνέλιξης[6 μονάδες]

Σε αυτό το πρόβλημα, θα υλοποιήσετε τη λειτουργία φιλτραρίσματος συνέλιξης χρησιμοποιώντας συναρτήσεις της βιβλιοθήκης NumPy, αλλά χωρίς να χρησιμοποιήσετε συναρτήσεις που λύνουν απευθείας το πρόβλημα, όπως η συνάρτηση συνέλιξης "numpy.convolve".

Όπως έχουμε δει και στο μάθημα, η συνέλιξη μπορεί να θεωρηθεί ως ένα κυλιόμενο παράθυρο που υπολογίζει ένα άθροισμα των τιμών των pixel που σταθμίζονται από τον αναποδογυρισμένο πυρήνα (a sum of pixel values weighted by the flipped kenrel).

Η έκδοσή σας θα πρέπει: i) να συμπληρώσει μια εικόνα με μηδενικά στα άκρα της εικόνας - zero-padding (επάνω-κάτω, δεξιά-αριστερά), ii) να αναστρέψει (flip) τον πυρήνα της συνέλιξης οριζόντια και κάθετα, και iii) να υπολογίσει ένα σταθμισμένο άθροισμα της γειτονιάς σε κάθε pixel.

Ζήτημα 1.1.1 [1 μονάδα]

Πρώτα θα χρειαστεί να υλοποιήσετε τη συνάρτηση **zero_pad**.

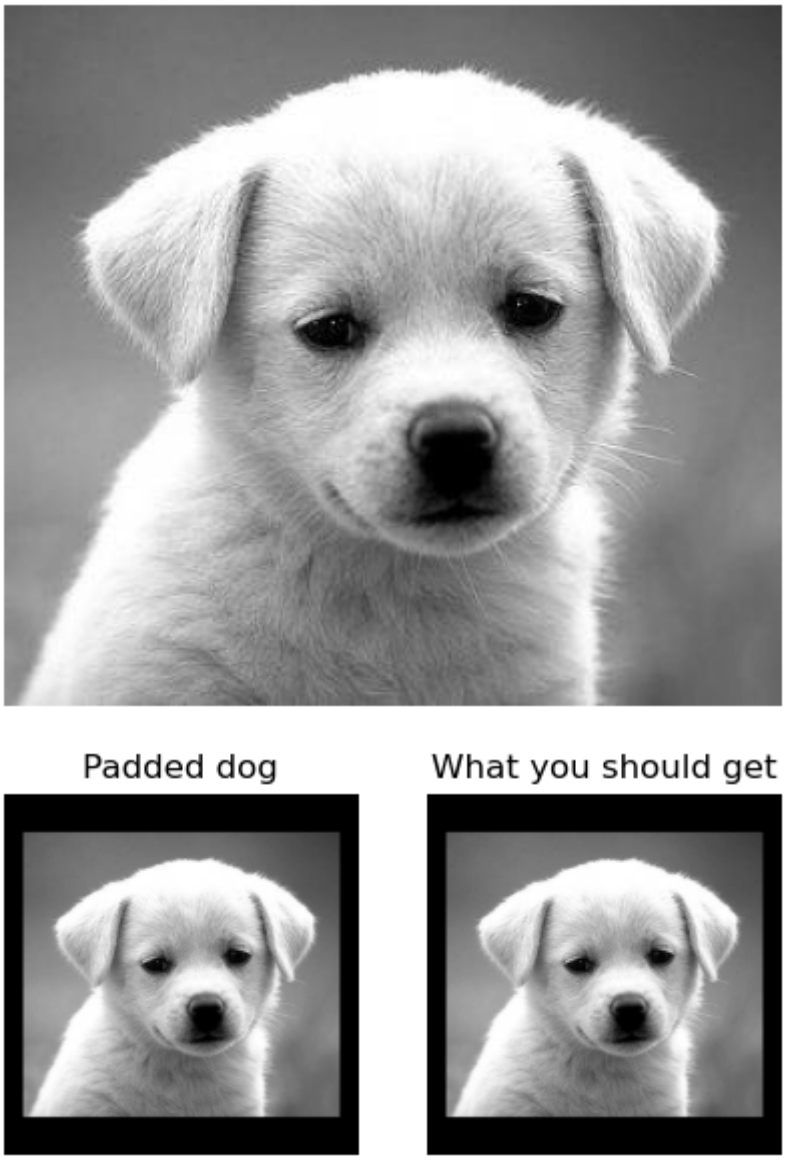
In [524…

```
import numpy as np
from time import time
from skimage import io
%matplotlib inline
import matplotlib.pyplot as plt
```

In [541…

```
def zero_pad(image, pad_top, pad_down, pad_left, pad_right):
    """ Zero-pad an image.

    Ex: a 1x1 image [[1]] with pad_top = 1, pad_down = 1, pad_left = 2, pad_right =

        [[0, 0, 0, 0, 0],
         [0, 0, 1, 0, 0],
         [0, 0, 0, 0, 0]]         of shape (3, 5)

    Args:
        image: numpy array of shape (H, W)
        pad_left: width of the zero padding to the left of the first column
        pad_right: width of the zero padding to the right of the last column
        pad_top: height of the zero padding above the first row
        pad_down: height of the zero padding below the last row

    Returns:
        out: numpy array of shape (H + pad_top + pad_down, W + pad_left + pad_right
    """
    """ ==========
    YOUR CODE HERE
    ========== """
    # We are creating an array with zero values with the output shape
```

```
    # (out: numpy array of shape (H + pad_top + pad_down,
    # W + pad_left + pad_right))
    # so we can have all zeros we need and then put
    # inside this array the values
    # from image in the correct indexes
    out = np.zeros((image.shape[0] + pad_top + pad_down,\
                    image.shape[1] + pad_left + pad_right))

    # Inserting the values from image to the result array
    # We should let zeros be as they are until the correct index
    # (pad_top and pad_left in our example)
    # and then start inserting the values from the image
    # We dont have to worry about the zeros in the right
    # and bottom of the image
    # they will be correct because image is in the correct possition
    out[pad_top:image.shape[0] + pad_top, \
        pad_left:image.shape[1] + pad_left] = image

    return out

# Open image as grayscale
img = io.imread('images/dog.jpg', as_gray=True)

# Show image
plt.imshow(img,cmap='gray')
plt.axis('off')
plt.show()

pad_width = 20 # width of the padding on the left and right
pad_height = 40 # height of the padding on the top and bottom

padded_img = zero_pad(img, pad_height, pad_height, pad_width, pad_width)

# Plot your padded dog
plt.subplot(1,2,1)
plt.imshow(padded_img,cmap='gray')
plt.title('Padded dog')
plt.axis('off')

# Plot what you should get
solution_img = io.imread('images/padded_dog.jpg', as_gray=True)
plt.subplot(1,2,2)
plt.imshow(solution_img,cmap='gray')
plt.title('What you should get')
plt.axis('off')

plt.show()
```



Ζήτημα 1.1.2 [3 μονάδες]

Τώρα υλοποιήστε τη συνάρτηση `conv`, **χρησιμοποιώντας το πολύ 2 βρόχους επανάληψης**. Αυτή η συνάρτηση θα πρέπει να δέχεται μια εικόνα f και έναν πυρήνα/φίλτρο h ως εισόδους και να εξάγει το αποτέλεσμα της συνέλιξης (προκύπτουσα εικόνα) $(f*h)$ που έχει το **ίδιο** σχήμα (διαστάσεις) με την εικόνα εισόδου (χρησιμοποιήστε συμπλήρωση μηδενικών - zero padding, για να το πετύχετε). Θα θεωρήσουμε πως χρησιμοποιούμε μόνο πυρήνες με περιττό πλάτος και περιττό ύψος. Ανάλογα με τον υπολογιστή, η υλοποίησή σας θα χρειαστεί περίπου ένα δευτερόλεπτο ή λιγότερο για να εκτελεστεί.

Υπόδειξη: Για να έχει το αποτέλεσμα της συνέλιξης $g(x,y) = h(x,y) * f(x,y)$ το **ίδιο σχήμα** με την εικόνα εισόδου f, θα πρέπει οι διαστάσεις της συμπληρωμένης (με μηδενικά) εικόνας "padded_f" να είναι $P = A + C - 1$ και $Q = B + D - 1$, όπου $A, B:height, width$ της εικόνας f, ενώ $C, D:height, width$, του πυρήνα h.

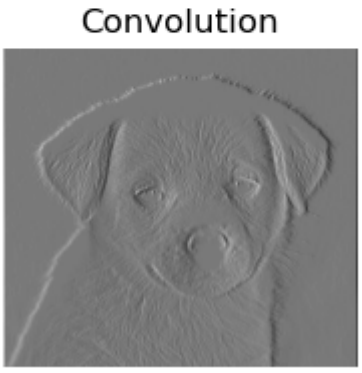
In [542…

```
def conv(image, kernel):
    """ An efficient implementation of a convolution filter.

    This function uses element-wise multiplication and np.sum()
    to efficiently compute a weighted sum of the neighborhood at each
```

```
    pixel.

    Hints:
        - Use the zero_pad function you implemented above
        - You should need at most two nested for-loops
        - You may find np.flip() and np.sum() useful
        - You need to handle both odd and even kernel size

    Args:
        image: numpy array of shape (Hi, Wi)
        kernel: numpy array of shape (Hk, Wk)

    Returns:
        out: numpy array of shape (Hi, Wi)
    """
    Hi, Wi = image.shape
    Hk, Wk = kernel.shape
    out = np.zeros((Hi, Wi))

    """ ==========
    YOUR CODE HERE
    ========== """
    # We flip the kernel because we need the columns to be
    # in reverse order
    kernel = np.flip(kernel)
    #print(kernel)
    # We need to find the padded image coordinates
    # Padded image coordinates will be
    # for x -> paddindx + Hi
    # for y -> paddingy + Wi
    paddingx = Hk - 1
    paddingy = Wk - 1
    # Create the padded image with the function we built above
    #padded_img = zero_pad(image, int(np.ceil(paddingy/2)),
    # int(np.floor(paddingy/2)), int(np.ceil(paddingx/2)), int(np.floor(paddingx/2)
    # We use half the value for each x and y because they will be
    # added together in the function
    padded_img = zero_pad(image, paddingx//2, paddingx//2,\
                          paddingy//2, paddingy//2)

    # Iterating through the image with two for loops
    # Each for loop for rows and columns

    for x in range(image.shape[0]):
        for y in range(image.shape[1]):
            # Convolve for each element
            # (using the sum function to sum the elemnts)
            # and putting it in the out array
            out[x, y] = (kernel * padded_img[x: x + Hk, y: y + Wk]).sum()

    return out

# Simple convolution kernel.
kernel = np.array(
[
    [1,0,-1],
    [2,0,-2],
    [1,0,-1]
])

t1 = time()
out = conv(img, kernel)
t2 = time()
print("took %f seconds." % (t2 - t1))
```

```
# Plot original image
plt.subplot(2,2,1)
plt.imshow(img,cmap='gray')
plt.title('Original')
plt.axis('off')

# Plot your convolved image
plt.subplot(2,2,3)
plt.imshow(out,cmap='gray')

plt.title('Convolution')
plt.axis('off')

# Plot what you should get
solution_img = io.imread('images/convolved_dog.jpg', as_gray=True)
plt.subplot(2,2,4)
plt.imshow(solution_img,cmap='gray')
plt.title('What you should get')
plt.axis('off')

plt.show()
```

```
took 0.353182 seconds.
```



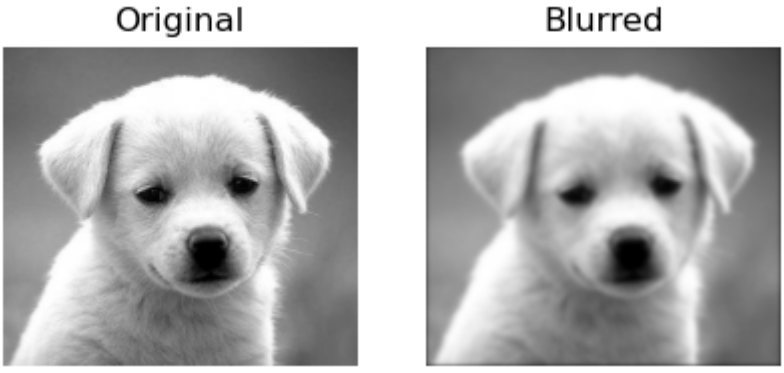
Ζήτημα 1.1.3 [1 μονάδα]

Τώρα ας φιλτράρουμε μερικές εικόνες! Σε αυτό το ζήτημα, θα εφαρμόσετε τη συνάρτηση συνέλιξης που μόλις υλοποιήσατε για να δημιουργήσετε μερικά ενδιαφέροντα εφέ εικόνας. Πιο συγκεκριμένα, θα χρησιμοποιήσετε συνέλιξη για να "θολώσετε" (blur) και να "οξύνετε" (sharpen) την εικόνα.

Αρχικά, θα εφαρμόσετε συνέλιξη για θόλωση εικόνας. Για να το πετύχετε αυτό, πραγματοποιήστε συνέλιξη της εικόνας του σκύλου με ένα Γκαουσιανό φίλτρο 13x13 για $\sigma = 2,0$. Μπορείτε να χρησιμοποιήσετε τη συνάρτηση που σας δίνετε για να πάρετε τον Γκαουσιανό πυρήνα της συνέλιξης.

In [543…

```python
def gaussian2d(sig):
    """
    Creates 2D Gaussian kernel with a sigma of `sig`.
    """
    filter_size = int(sig * 6)
    if filter_size % 2 == 0:
        filter_size += 1

    ax = np.arange(-filter_size // 2 + 1., filter_size // 2 + 1.)
    xx, yy = np.meshgrid(ax, ax)
    kernel = np.exp(-0.5 * (np.square(xx) + np.square(yy)) / np.square(sig))
    return kernel / np.sum(kernel)

def blur_image(img):
    """Blur the image by convolving with a Gaussian filter."""
    blurred_img = np.zeros_like(img)
    """ ==========
    YOUR CODE HERE
    ========== """
    # Creating the Gaussian kernel with sigma equal to 2.0
    kernel = gaussian2d(2.0)
    # Using convulation with the function we built above
    # Parameters are the image (img) and the Gaussian kernel
    blurred_img = conv(img,kernel)


    return blurred_img

# Plot original image
plt.subplot(2,2,1)
plt.imshow(img,cmap='gray')
plt.title('Original')
plt.axis('off')

# Plot blurred image
plt.subplot(2,2,2)
plt.imshow(blur_image(img),cmap='gray')
plt.title('Blurred')
plt.axis('off')

plt.show()
```

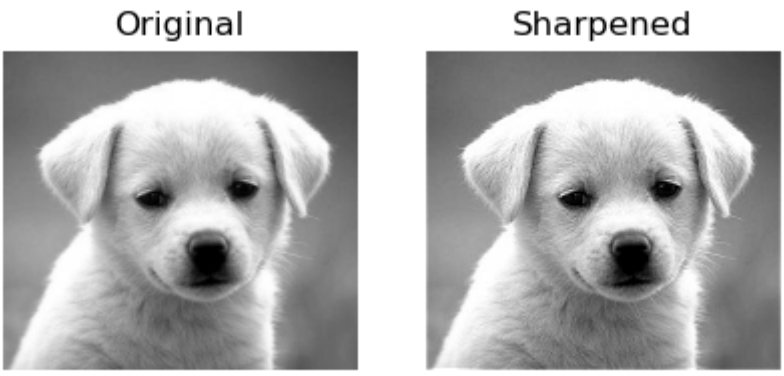


Ζήτημα 1.1.4 [1 μονάδα]

Στη συνέχεια, θα χρησιμοποιήσουμε τη συνέλιξη για την όξυνση (αύξηση ευκρίνειας) των εικόνων. Πραγματοποιήστε συνέλιξη της εικόνας με το ακόλουθο φίλτρο για να δημιουργήσετε ένα πιο ευκρινές αποτέλεσμα. Για ευκολία, σας δίνετε και το φίλτρο όξυνσης:

In [544…
```
sharpening_kernel = np.array([
    [1, 4,     6,  4, 1],
    [4, 16,   24, 16, 4],
    [6, 24, -476, 24, 6],
    [4, 16,   24, 16, 4],
    [1,  4,    6,  4, 1],
]) * -1.0 / 256.0
```

In [545…
```
def sharpen_image(img):
    """Sharpen the image by convolving with a sharpening filter."""
    sharpened_img = np.zeros_like(img)
    """ ==========
    YOUR CODE HERE
    ========== """
    # Using convulation with the function we built above
    # Parameters are the image (img) and the sharpening kernel
    sharpened_img = conv(img,sharpening_kernel)

    return sharpened_img

# Plot original image
plt.subplot(2,2,1)
plt.imshow(img, vmin=0.0, vmax=1.0,cmap='gray')
plt.title('Original')
plt.axis('off')

# Plot sharpened image
plt.subplot(2,2,2)
plt.imshow(sharpen_image(img), vmin=0.0, vmax=1.0,cmap='gray')
plt.title('Sharpened')
plt.axis('off')

plt.show()
```



Ζήτημα 1.2 Αντιστοίχιση/Ταίριασμα Προτύπου (Template Matching) [4 μονάδες]

Υποθέτουμε το παρακάτω πρόβλημα. Έστω ένας υπάλληλος κάποιου καταστήματος super market είναι υπεύθυνος για τον περιοδικό έλεχγο των ραφιών, με σκοπό την αναπλήρωσή τους με προϊόντα που έχουν εξαντληθεί/πωληθεί (restocking sold-out items). Σε αυτή την περίπτωση, η ανάπτυξη μιας εφαρμογής υπολογιστικής όρασης, η οποία θα "βλέπει" και θα καταγράφει σε πραγματικό χρόνο τα προϊόντα στα ράφια θα μπορούσε να αυτοματοποιήσει τη δουλειά του υπαλλήλου.

Ευτυχώς, κάτι τέτοιο μπορεί να επιλυθεί ακόμη και με πρωταρχικές τεχνικές ψηφιακής επεξεργασίας εικόνας που βασίζονται στη συνέλιξη, η οποία μπορεί να αξιοποιηθεί για την αντιστοίχιση μιας εικόνας με κάποιο πρότυπο (template matching):

- Ένα αναποδογυρισμένο (flipped) πρότυπο *t* πολλαπλασιάζεται με τις περιοχές μιας μεγαλύτερης εικόνας *f* για να υπολογιστεί πόσο παρόμοια είναι κάθε περιοχή με το πρότυπο (πόσο μοιάζει κάθε περιοχή με την εικόνα προτύπου). Σημειώστε, ότι θα πρέπει να αναστρέψετε το φίλτρο πριν το δώσετε στη συνάρτηση συνέλιξης, έτσι ώστε συνολικά να μην είναι αναποδογυρισμένο όταν κάνετε συγκρίσεις.

- Επίσης, Θα χρειαστεί να αφαιρέσετε τη μέση τιμή της εικόνας ή του προτύπου (όποια και αν επιλέξετε, αφαιρέστε την ίδια τιμή, τόσο από την εικόνα όσο και από το πρότυπο) έτσι ώστε η λύση σας να μην είναι ευαίσθητη προς τις περιοχές υψηλότερης έντασης (λευκές).

- Δοκιμάστε να εκτελέσετε αρχικά τη συνέλιξη του ανεστραμμένου πυρήνα (προτύπου) με την εικόνα, χωρίς να αφαιρέσετε τη μέση τιμή και δείτε την ευαισθησία του αποτελέσματος σε περιοχές υψηλότερης έντασης. Εξηγείστε (σε σχόλια) γιατί η αφαίρεση της μέσης τιμής (και από τις 2 εικόνες) αντιμετωπίζει το πρόβλημα, κάνοντας τη λύση σας ανθεκτική σε περιοχές υψηλής έντασης.

- Παρέχεται το πρότυπο ενός προϊόντος (template.jpg) και η εικόνα του ραφιού (shelf.jpg). Θα χρησιμοποιήσετε συνέλιξη για να βρείτε το προϊόν στο ράφι.

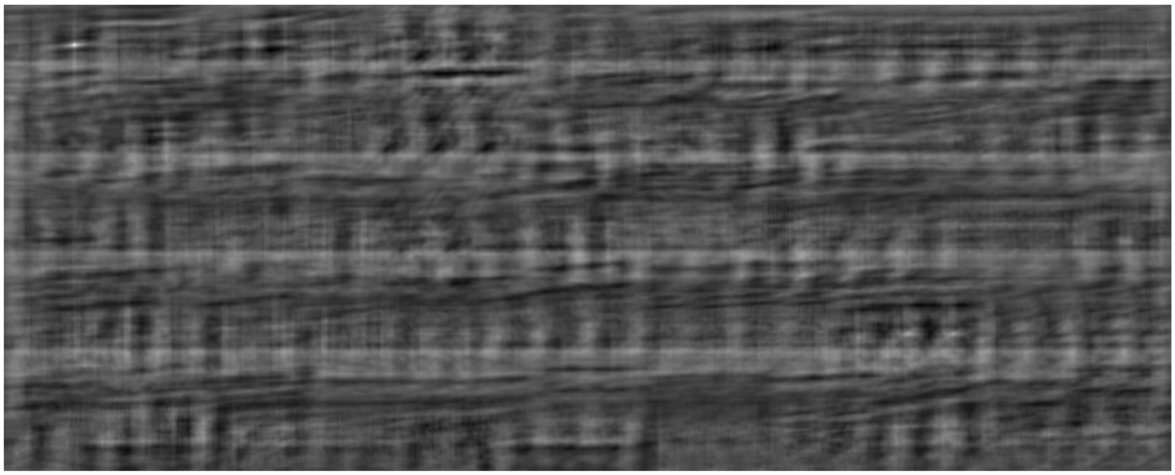
In [546…

```
# Load template and image in grayscale
img = io.imread('images/shelf.jpg')
img_gray = io.imread('images/shelf.jpg', as_gray=True)
temp = io.imread('images/template.jpg')
temp_gray = io.imread('images/template.jpg', as_gray=True)

# Perform a convolution between the image (grayscale) and the template (grayscale)
# the result in the out variable
""" ==========
YOUR CODE HERE
========== """
# We are flipping the image because when we will convolve it with
# the shelf we want it to be the way it is (not flipped)
fliped_temp = np.flip(temp_gray)
# Finding the mean value from the template array
# and subtracting it from each element, both from
# shelf image and template image
mean = np.mean(temp_gray)
fliped_temp[:] -= mean
img_gray -= mean
# Convolving the gray scale images after the computations above
out = conv(img_gray,fliped_temp)

# We subtract the mean from images values because we want to avoid
# pixels(cells) with large values (white). The value of a white pixel
# is 255, by using this technique we dont let any cells have big values
# like this. The picture this way is normalized.

# Find the (x, y) coordinates of the maximum value in the out variable
""" ==========
YOUR CODE HERE
========== """
```

```
# From the convolution image we find the maximum value
# and store the coordinates of that value in the results
results = np.where(out == np.amax(out))
y = results[0]
x = results[1]

# Display product template
plt.figure(figsize=(20,16))
plt.subplot(3, 1, 1)
plt.imshow(temp_gray, cmap="gray")
plt.title('Template')
plt.axis('off')

# Display convolution output
plt.subplot(3, 1, 2)
plt.imshow(out, cmap="gray")
plt.title('Convolution output (white means more correlated)')
plt.axis('off')

# Display image
plt.subplot(3, 1, 3)
plt.imshow(img, cmap="gray")
plt.title('Result (blue marker on the detected location)')
plt.axis('off')

# Draw marker at detected location
plt.plot(x, y, 'bx', ms=35, mew=5)
plt.show()
```

Template

Convolution output (white means more correlated)



Result (blue marker on the detected location)

Άσκηση 2: Ανίχνευση Ακμών (Edge detection) [15 μονάδες]

Σε αυτό το πρόβλημα, θα υλοποιήσετε τα βήματα του ανιχνευτή ακμών "Canny". Πρέπει να ακολουθήσετε τα βήματα με τη σειρά που σας δίνετε.

Ζήτημα 2.1 Εξομάλυνση (Smoothing) [1 μονάδα]

Αρχικά, πρέπει να εξομαλύνουμε τις εικόνες για να αποτρέψουμε τον θόρυβο να θεωρηθεί ως ακμές. Για αυτήν την άσκηση, χρησιμοποιήστε ένα φίλτρο Γκαουσιανού πυρήνα (Gaussian) 9x9 με $\sigma = 1,5$ για να εξομαλύνετε τις εικόνες.

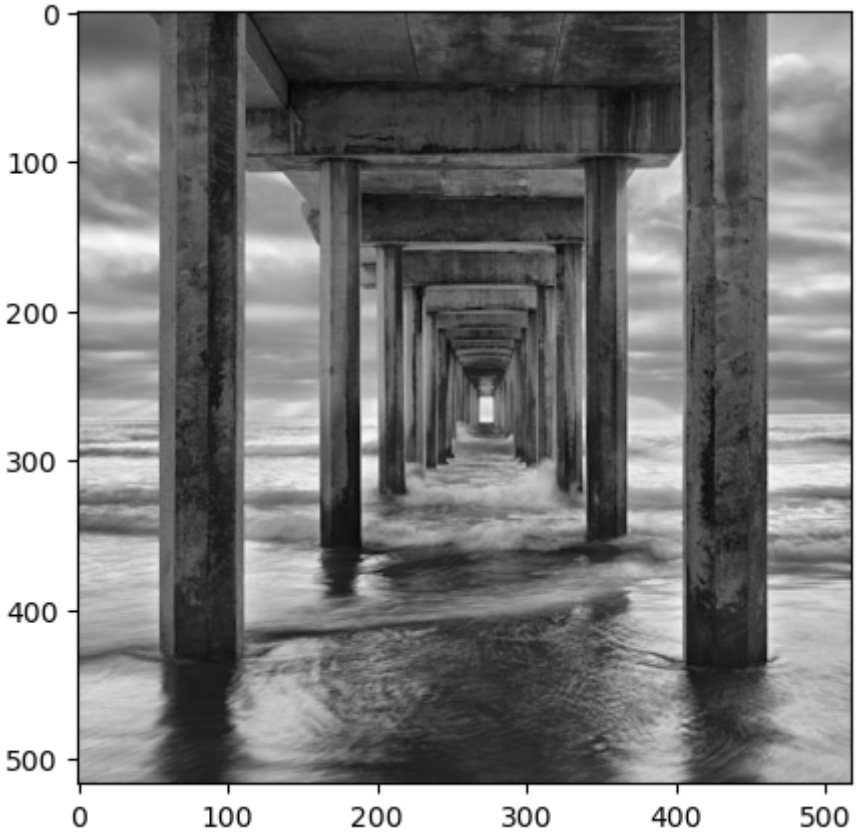
In [547…
```
import numpy as np
from skimage import io
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from scipy.signal import convolve
%matplotlib inline

import matplotlib
matplotlib.rcParams['figure.figsize'] = [5, 5]
```

In [548…
```
def gaussian2d(sig=None):
    """Creates a 2D Gaussian kernel with
    side length `filter_size` and a sigma of `sig`."""
    filter_size = int(sig * 6)
    if filter_size % 2 == 0:
        filter_size += 1

    ax = np.arange(-filter_size // 2 + 1., filter_size // 2 + 1.)
    xx, yy = np.meshgrid(ax, ax)
    kernel = np.exp(-0.5 * (np.square(xx) + np.square(yy)) / np.square(sig))
    return kernel / np.sum(kernel)
```

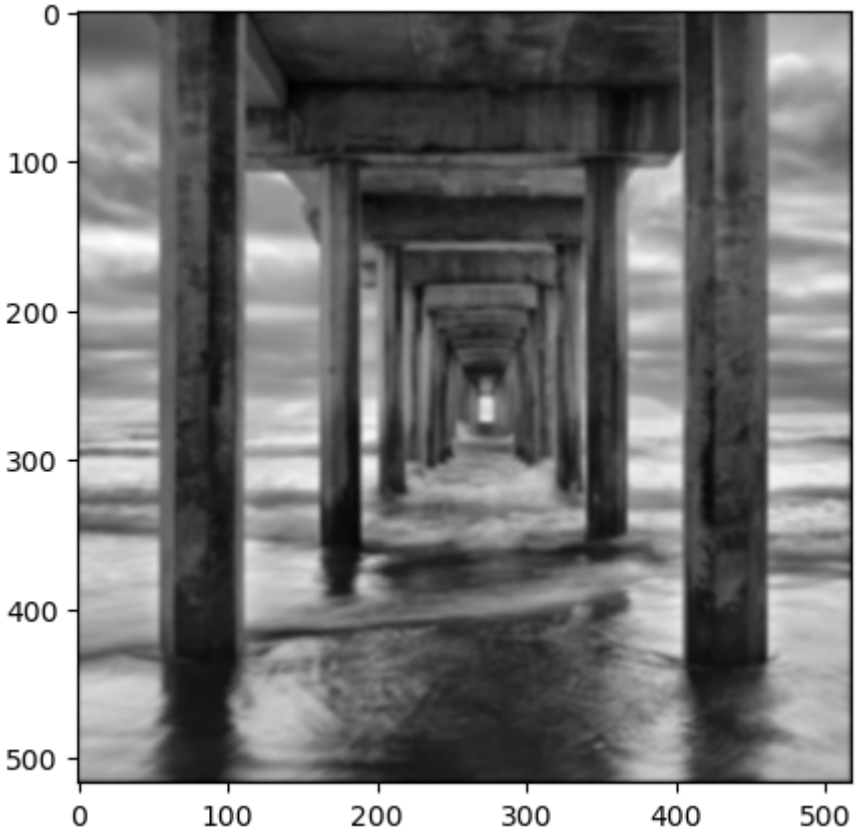
In [549…
```
def smooth(image):
    """ ==========
    YOUR CODE HERE
    ========== """
    # Creating the Gaussian kernel with sigma equal to 2.0
    # Using convulation with the function we built above
    # Parameters are the image and the Gaussian kernel
    # It seems like its making the picture worse but it is
    # usefull for finding edges
    kernel = gaussian2d(1.5)
    out = conv(image,kernel)
    return out
```

In [550…
```
# Load image in grayscale
image = io.imread('images/canny.jpg', as_gray=True)
assert len(image.shape) == 2, 'image should be grayscale; check your Python/skimage
smoothed = smooth(image)
print('Original:')
plt.imshow(image, cmap=cm.gray)
plt.show()

print('Smoothed:')
plt.imshow(smoothed, cmap=cm.gray)
plt.show()
```

```
Original:
```



Smoothed:



Ζήτημα 2.2 Υπολογισμός Παραγώγου (Gradient Computation [4 μονάδες]

Αφού ολοκληρώσετε την εξομάλυνση, βρείτε την παράγωγο/κλίση της εικόνας στην οριζόντια και κάθετη κατεύθυνση. Υπολογίστε την εικόνα του μέτρου (μεγέθους) κλίσης

(gradient magnitude) ως $|G| = \sqrt{G_x^2 + G_y^2}$. Η κατεύθυνση της ακμής για κάθε pixel δίνεται από την εξίσωση $G_\theta = \tan^{-1}\left(\frac{G_y}{G_x}\right)$.

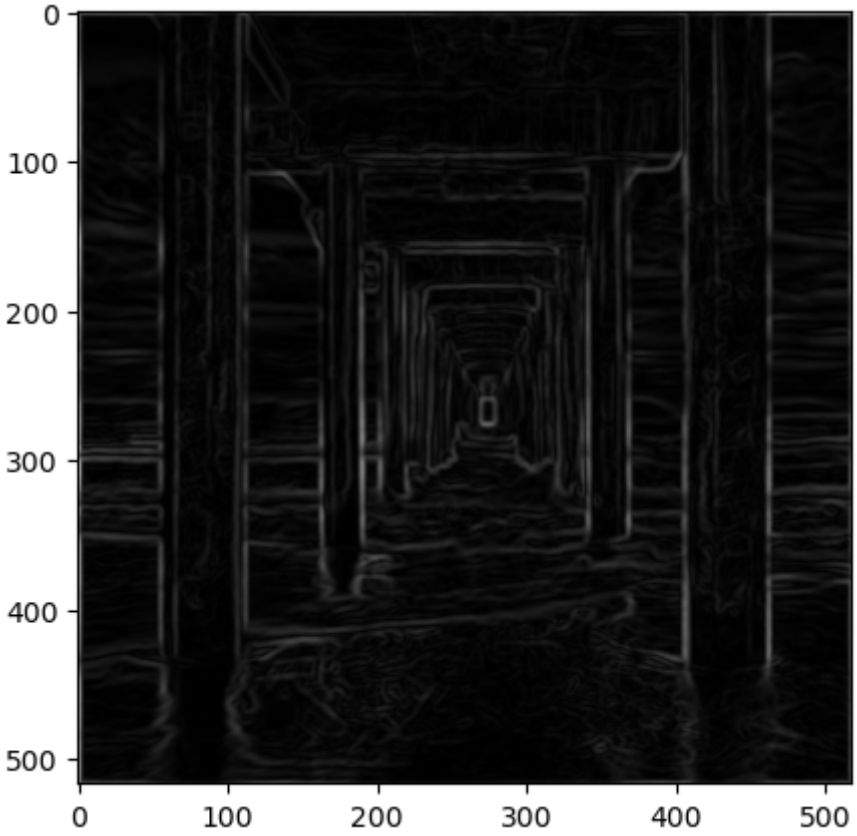
In [551…

```
def gradient(image):
    """ ==========
    YOUR CODE HERE
    ========== """
    # Creating the magnitude and theta arrays, firstly with zeros
    g_mag = np.zeros((image.shape[0],image.shape[1]))
    g_theta = np.zeros((image.shape[0],image.shape[1]))
    # Padding the image with 1 column/row in each side
    # because we want to make calculations to the cells
    # being on the edges in the image
    padded_img = zero_pad(image, 1, 1, 1, 1)
    # We compute the changes in gradient in both the x and y direction
    for x in range(image.shape[0]):
        for y in range(image.shape[1]):
            # We compute the horizontal change by taking
            # the difference between the east and west pixels
            gx = padded_img[x+1][y] - padded_img[x-1][y]
            # The vertical change by taking the difference
            # between the south and north pixels
            gy = padded_img[x][y+1] - padded_img[x][y-1]
            # Pythagorean theorem
            g_mag[x,y] = np.sqrt((gx*gx) + (gy*gy))
            # If we dont use this if, python shows a
            # warning about gx being 0
            # and we cant devide with 0
            if gx != 0:
                # We are using arctan because its equal to tan^-1
                g_theta[x,y] = np.arctan((gy/gx))

    return g_mag, g_theta
```

In [552…

```
g_mag, g_theta = gradient(smoothed)
print('Gradient magnitude:')
plt.imshow(g_mag, cmap=cm.gray)
plt.show()
```

```
Gradient magnitude:
```



Ζήτημα 2.3 Καταστολή μη-μεγίστων (Non-Maximum Suppression) [5 μονάδες]

Θα θέλαμε οι ακμές μας να είναι ευκρινείς (sharp), σε αντίθεση με αυτές στην εικόνα ντεγκραντέ (gradient image). Χρησιμοποιήστε καταστολή μη-μεγίστων για να διατηρήσετε όλα τα τοπικά μέγιστα και απορρίψτε τα υπόλοιπα. Μπορείτε να χρησιμοποιήσετε την ακόλουθη μέθοδο για να το κάνετε:

- Για κάθε εικονοστοιχείο στην εικόνα του μέτρου (μεγέθους) της κλίσης (gradient magnitude image):
  - Στρογγυλοποιήστε την κατεύθυνση της κλίσης θ στο πλησιέστερο πολλαπλάσιο των 45° (το οποίο θα αναφέρουμε ως ve).
  - Συγκρίνετε την ισχύ της ακμής (edge strength) στο τρέχον εικονοστοιχείο (δηλαδή το μέτρο της κλίσης) με τα εικονοστοιχεία κατά μήκος της κατεύθυνσης κλίσης $+ve$ και $-ve$ στην 8-γειτονιά του (8-connected pixel neighborhood).
  - Εάν το εικονοστοιχείο δεν έχει μεγαλύτερη τιμή από τους δύο γείτονές του στις κατευθύνσεις κλίσης $+ve$ και $-ve$, καταργήστε (suppress) την τιμή του εικονοστοιχείου (ορίστε το σε 0). Ακολουθώντας αυτή τη διαδικασία, διατηρούμε τις τιμές μόνο εκείνων των pixel που έχουν μέγιστα μεγέθη κλίσης στη γειτονιά κατά μήκος των κατευθύνσεων κλίσης $+ve$ και $-ve$.
- Επιστρέψτε το αποτέλεσμα ως την εικόνα-απόκριση της καταστολής μη-μεγίστων (NMS).

In [553…

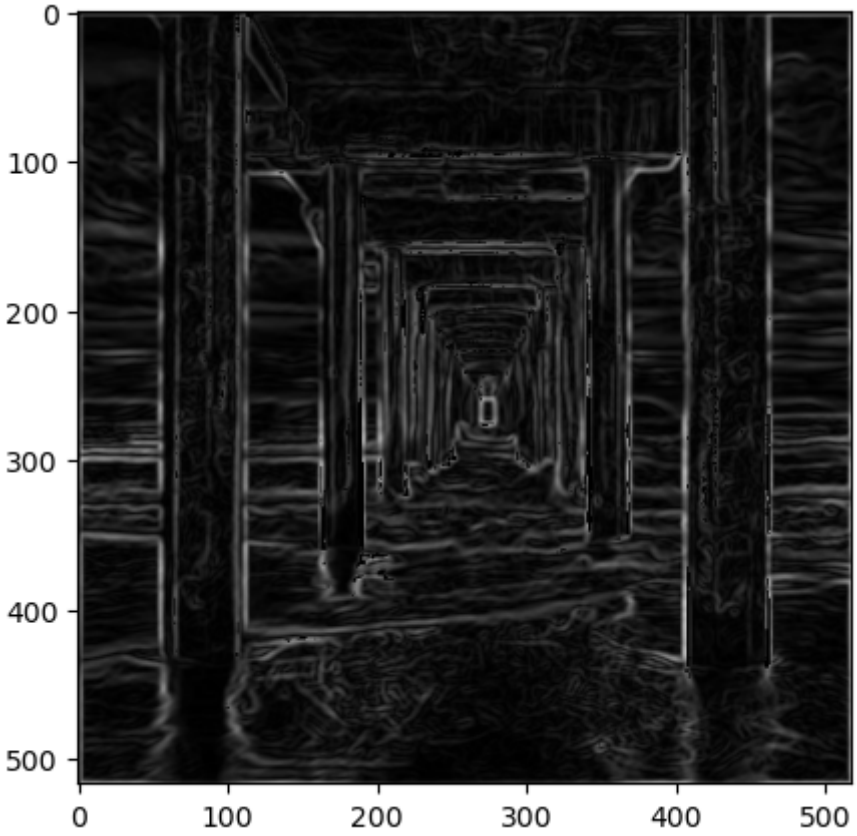
```
def nms(g_mag, g_theta):
    """ ==========
    YOUR CODE HERE
    ========== """
    # Creating the nmsresponse array, firstly with zeros
```

```
    nms_response = np.zeros((g_mag.shape[0],g_mag.shape[1]))
    # Padding the image with 1 column/row in each side
    # because we want to make calculations to the cells
    # being on the edges in the image
    padded_g_mag = zero_pad(image, 1, 1, 1, 1)
    for x in range(g_mag.shape[0]):
        for y in range(g_mag.shape[1]):
            # The values in theta array are in radians so
            # we multiply each value with (180/pi) to convert
            # it to degrees
            ve = ((g_theta[x][y]) * (180/np.pi)) #// 45
            # Because we are using arctan function the values will
            # be representet in the east side of the current cell.
            # So we split the degrees range
            # such as we group them in
            # 3 groups. We do this because we want
            # to find out in wich 2
            # neighborhoods we will check
            # (the neighborhoods are in opposite direction)
            if ve >= -22.499 and ve <= 22.499:
                # East and West
                if g_mag[x][y] > padded_g_mag[x+1][y] or \
                g_mag[x][y] > padded_g_mag[x-1][y]:
                    nms_response[x][y] = 0
                else:
                    nms_response[x][y] = g_mag[x][y]
            elif ve >= -67.499 and ve <= 67.499:
                # SE and SW and NE and NW
                if g_mag[x][y] > padded_g_mag[x+1][y+1] or \
                g_mag[x][y] > padded_g_mag[x-1][y-1] \
                or g_mag[x][y] > padded_g_mag[x+1][y-1] or \
                g_mag[x][y] > padded_g_mag[x-1][y+1]:
                    nms_response[x][y] = 0
                else:
                    nms_response[x][y] = g_mag[x][y]
            else:
                # South and North
                if g_mag[x][y] > padded_g_mag[x][y+1] or \
                g_mag[x][y] > padded_g_mag[x][y-1]:
                    nms_response[x][y] = 0
                else:
                    nms_response[x][y] = g_mag[x][y]


    return nms_response
```

In [554…

```
nms_image = nms(g_mag, g_theta)
print('NMS:')
plt.imshow(nms_image, cmap=cm.gray)
plt.show()
```

```
NMS:
```



Ζήτημα 2.4 Κατωφλίωση Υστέρησης (Hysteresis Thresholding) [5 μονάδες]

Επιλέξτε κατάλληλες τιμές κατωφλίων και χρησιμοποιήστε την προσέγγιση κατωφλίου που περιγράφεται στη διάλεξη 5. Αυτό θα αφαιρέσει τις ακμές που προκαλούνται από το θόρυβο και τις χρωματικές διαφοροποιήσεις. Μπορείτε να ανατρέξετε και σε άλλες πηγές (βιβλιογραφία, διαδίκτυο) για περισσότερες πληροφορίες στην προσέγγιση κατωφλίου.

- Ορίστε δύο κατώφλια `t_min` και `t_max`.
- Εάν το `nms > t_max`, τότε επιλέγουμε αυτό το pixel ως ακμή.
- Εάν `nms < t_min`, απορρίπτουμε αυτό το pixel.
- Αν `t_min < nms < t_max`, επιλέγουμε το pixel μόνο αν υπάρχει διαδρομή από/προς άλλο pixel με `nms > t_max`. (Υπόδειξη: Σκεφτείτε όλα τα pixel με `nms > t_max` ως σημεία έναρξης/εκκίνησης και εκτελέστε αναζήτηση BFS/DFS από αυτά τα σημεία εκκίνησης).
- Η επιλογή της τιμής των χαμηλών και υψηλών κατωφλίων εξαρτάται από το εύρος των τιμών στην εικόνα μεγέθους κλίσης (gradient magnitude image). Μπορείτε να ξεκινήσετε ορίζοντας το υψηλό κατώφλι σε κάποιο ποσοστό της μέγιστης τιμής στην εικόνα μεγέθους ντεγκραντέ (gradient magnitude image), π.χ. `thres_high = 0,2 * image.max()`, και το χαμηλό όριο σε κάποιο ποσοστό του υψηλού ορίου, π.χ. `thres_low = 0,85 * thres_high`. Έπειτα, μπορείτε να συντονίσετε/τροποποιήσετε (tune) αυτές τις τιμές όπως θέλετε.

In [555…

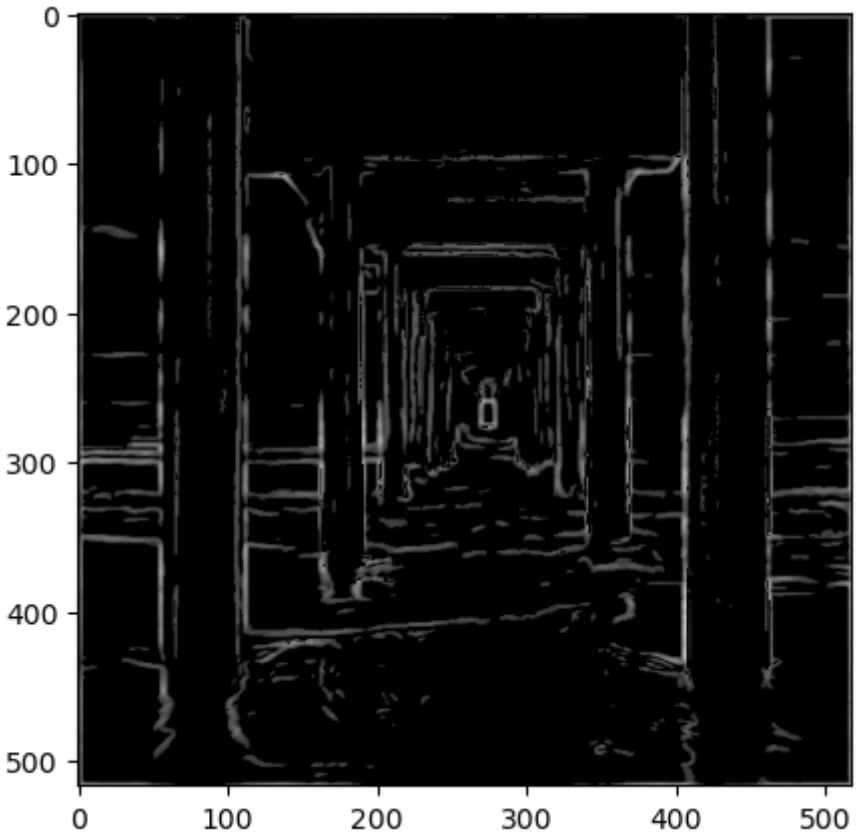
```python
def hysteresis_threshold(image, g_theta, use_g_theta=False):
    """ ==========
    YOUR CODE HERE
    ========== """
    # By changing the numbers I think this ones fits better
```

```
    t_max = np.amax(image) * 0.22
    t_min = t_max * 0.80
    # Creating the result array, firstly with zeros
    result = np.zeros((image.shape[0],image.shape[1]))
    # Padding the image with 1 column/row in each side
    # because we want to make calculations to the cells
    # being on the edges in the image
    padded_img = zero_pad(image, 1, 1, 1, 1)
    for x in range(image.shape[0]):
        for y in range(image.shape[1]):
            # The first case when the value of
            # the cell is bigger than t_max
            if (image[x][y] > t_max):
                result[x][y] = image[x][y]
            # The case when the value is between t_min and t_max
            # We check every neighborhood cell and
            # if only one of them (or more)
            # has his value bigger than t_max we add
            # the current cell to the result array
            elif (image[x][y] < t_max and image[x][y] > t_min):
                if (padded_img[x-1][y-1] > t_max or padded_img[x-1][y] > \
                    t_max or padded_img[x-1][y+1] > t_max or \
                   padded_img[x][y-1] > t_max or padded_img[x][y+1] > \
                    t_max or padded_img[x+1][y-1] > t_max or \
                   padded_img[x+1][y] > t_max or padded_img[x+1][y+1] > t_max):
                    result[x][y] = image[x][y]

    return result
```

In [556…

```
thresholded = hysteresis_threshold(nms_image, g_theta)
print('Thresholded:')
plt.imshow(thresholded, cmap=cm.gray)
plt.show()
```

```
Thresholded:
```



Οδηγίες υποβολής

Μην ξεχάσετε να κάνετε turnin **τόσο** το αρχείο Jupyter notebook όσο και το PDF αρχείο αυτού του notebook μαζί με το συνοδευτικό αρχείο `onoma.txt` : **turnin assignment_2@mye046 onoma.txt assignment2.ipynb assignment2.pdf**

Βεβαιωθείτε ότι το περιεχόμενο σε **κάθε κελί εμφανίζεται** καθαρά στο τελικό σας αρχείο PDF. Για να μετατρέψετε το σημειωματάριο σε PDF, μπορείτε να επιλέξετε **έναν** από τους παρακάτω τρόπους:

1. Google Collab (Συνιστάται): You can `print` the web page and save as PDF (e.g. Chrome: Right click the web page $\rightarrow$ Print... $\rightarrow$ Choose "Destination: Save as PDF" and click "Save"). Προσοχή στην περίπτωση όπου κώδικας/σχόλια εμφανίζονται εκτός των ορίων της σελίδας. Μια λύση είναι η αλλαγή γραμμής π.χ. σε σχόλια που υπερβαίνουν το πλάτος της σελίδας.

2. Local Jupyter/JupyterLab(Συνιστάται): You can `print` the web page and save as PDF (File $\rightarrow$ Print... $\rightarrow$ Choose "Destination: Save as PDF" and click "Save"). Προσοχή στην περίπτωση όπου κώδικας/σχόλια εμφανίζονται εκτός των ορίων της σελίδας. Μια λύση είναι η αλλαγή γραμμής π.χ. σε σχόλια που υπερβαίνουν το πλάτος της σελίδας.

3. Local Jupyter/JupyterLab(Συνιστάται!): You can `export` and save as HTML (File $\rightarrow$ Save & Export Notebook as... $\rightarrow$ HTML). Στη συνέχεια μπορείτε να μετατρέψεται το HTML αρχείο αποθηκεύοντάς το ως PDF μέσω ενός browser.