1 Written Part

1(a) K^2HW

If the filter is not separable, then the process of performing a convolution requires $K \cdot K$ operations per pixel, and since the image has $H \cdot W$ pixels in total, so the number of operations would be K^2HW.

1(b) $2KHW$

If the filter is separable, this operation could be speed up by first performing a $1D$ horizontal convolution followed by a $1D$ vertical convolution, then the process of performing a convolution requires $K + K$ operations per pixel, and since the image has $H \cdot W$ pixels in total, so the number of operations would be $2KHW$.

2(a) A vertical derivative, $\frac{\partial G(x,y)}{\partial y}$, of a Gaussian filter G is a separable filter for both isotropic and anisotropic cases.

2(a).01 isotropic Gaussian filter

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

$\frac{\partial G(x,y)}{\partial y} = -\frac{y}{2\pi\sigma^4} e^{-\frac{x^2+y^2}{2\sigma^2}} = [-\frac{1}{\sigma^2\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}}] \cdot [\frac{y}{\sigma^2\sqrt{2\pi}} e^{-\frac{y^2}{2\sigma^2}}]$

Since the above function could be converted into $f(x) \cdot f(y)$, so it's a separable filter.

2(a).02 anisotropic Gaussian filter

$G(x,y) = \frac{1}{2\pi\sigma_x\sigma_y} e^{-(\frac{x^2}{2\sigma_x^2}+\frac{y^2}{2\sigma_y^2})}$

$\frac{\partial G(x,y)}{\partial y} = -\frac{y}{2\pi\sigma_x\sigma_y^3} e^{-(\frac{x^2}{2\sigma_x^2}+\frac{y^2}{2\sigma_y^2})} = [-\frac{1}{\sigma_x\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma_x^2}}] \cdot [\frac{y}{\sigma_y^3\sqrt{2\pi}} e^{-\frac{y^2}{2\sigma_y^2}}]$

Since the above function could be converted into $f(x) \cdot f(y)$, so it's a separable filter.

2(b) A Laplacian of Gaussians (LoG) is NOT a separable filter.

The LoG is the sum of the second derivatives of the Gussian, $LoG = \frac{\partial^2 G}{\partial x^2} + \frac{\partial^2 G}{\partial y^2}$, which could not be directly separated into two 1D kernels, thus it is not a separable filter.

2(c) We could apply the singular value decomposition (SVD) on this 3×3 filter and get $F = U\Sigma V^\intercal$, which as a sum of matrix multiplications, each of a single row/column from the $U \in \mathbb{R}^{3\times3}$ and $V \in \mathbb{R}^{3\times3}$, multiplied by the diagonal entry from $\Sigma \in \mathbb{R}^{3\times3}$. The matrix Σ contains entries, which called "singular values", on the diagonal only and they are sorted from the highest to the lowest value. If only one singular value is non-zero, which means that it is a rank 1 matrix and then this filter is separable.

3 Cross correlation is NOT commutative.

Proof:

By the definition of cross correlation, we have:

$Corr[f, g]_t = Corr[g, f]_{-t}$

which means $f(t) * g(t) = [g\bar{(}t) * f\bar{(}t)](-t)$

but it needs to satisfy $f(t) * g(t) = g(t) * f(t)$ to be commutative,

so that the cross correlation is not commutative.

4 Convolve the image with a Gaussian filter with $\sigma = \sqrt{5}$

$\sigma = \sqrt{\sigma_1^2 + \sigma_2^2} = \sqrt{1^2 + 2^2} = \sqrt{5}$

Proof:

Want to show that $G(x, y|\sigma) = G(x, y|\sqrt{\sigma_1^2 + \sigma_2^2}) = G(x, y|\sigma_1) \cdot G(x, y|\sigma_2)$

Note that if X_1, X_2 are two independent random variables and $X_1 \sim N(0, \sigma_1^2 I)$ and $X_2 \sim N(0, \sigma_2^2 I)$ with two probability density functions G_1 and G_2 respectively, then $X = X_1 + X_2$ would have a density function $G = G_1 \cdot G_2$. And given that $X \sim N(0, (\sigma_1^2 + \sigma_2^2)I)$, we have the probability density function (PDF) of variable X:

$$\begin{aligned} f_X(X) &= \frac{1}{2\pi\sqrt{|\Sigma|}} exp\{-\frac{1}{2} X^\intercal \Sigma^{-1} X\} \\ &= \frac{1}{2\pi(\sigma_1^2 + \sigma_2^2)} exp\{-\frac{X^\intercal X}{2(\sigma_1^2 + \sigma_2^2)}\} \\ &= \frac{1}{2\pi(\sigma_1^2 + \sigma_2^2)} exp\{-\frac{x^2 + y^2}{2(\sigma_1^2 + \sigma_2^2)}\} \\ &= G(x, y|\sqrt{\sigma_1^2 + \sigma_2^2}) \\ &= G(x, y|\sigma) \end{aligned}$$

where $|\Sigma| = |(\sigma_1^2 + \sigma_2^2)I| = (\sigma_1^2 + \sigma_2^2)^2, X = (x, y)$

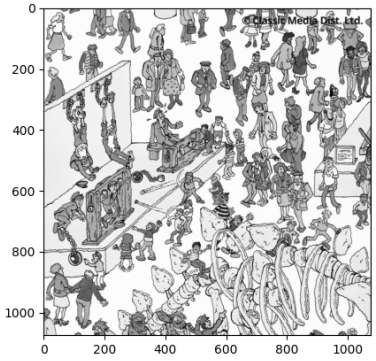
Therefore, we've proved that $\sigma = \sqrt{\sigma_1^2 + \sigma_2^2}$.

2 Coding Part

1(a)

the original image

the image after convolution

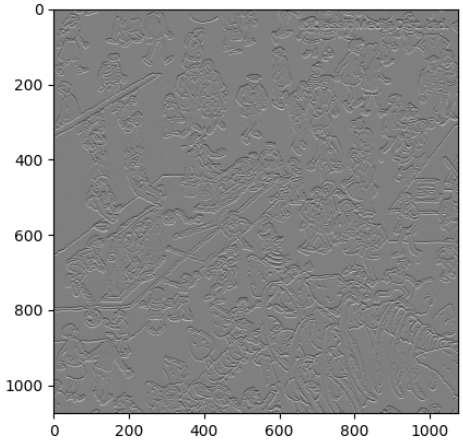


1(b)

Function implement refers to the script appendix.
Found that the above given filter is NOT separable.

1(c) Since the given filter is not separable, so I chose another separable filter.

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$



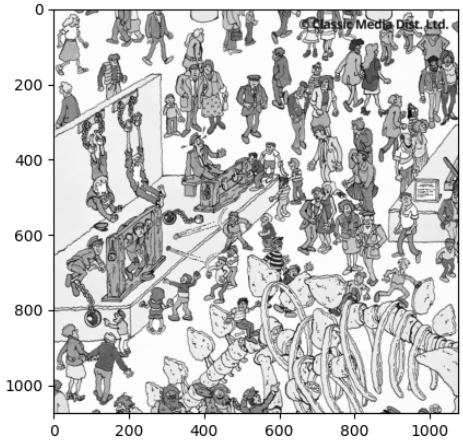
time comparison:

```
--- convolution takes 2.8190128803253174 seconds ---
--- faster convolution takes 0.12801790237426758 seconds ---
```

Thus, the faster convolution with the separable filter cost less time.

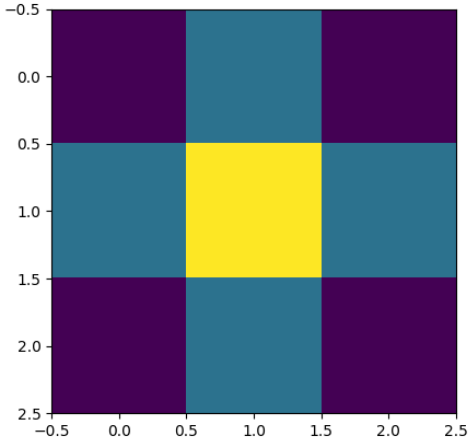
3 Coding Part Continued

1(d)

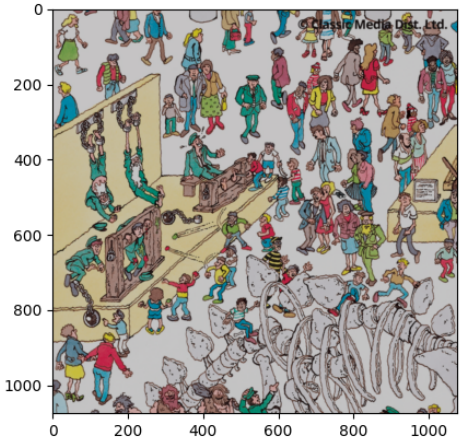


2 Convolve the attached waldo.png with a 2D Gaussian filter with sigma of 1 and kernel size of 3.

display the filter



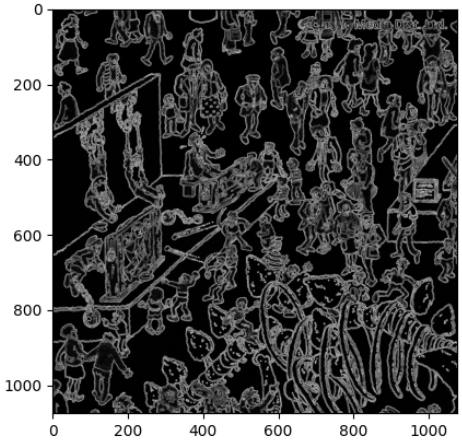
display the convolution



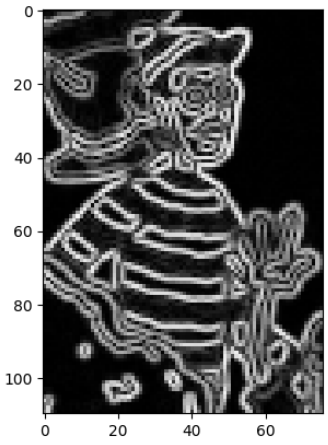
3 Coding Part Continued

3(a) Compute magnitude of gradients for the attached images

waldo.png

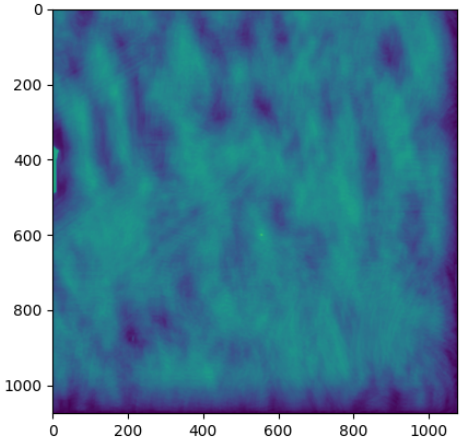


template.png

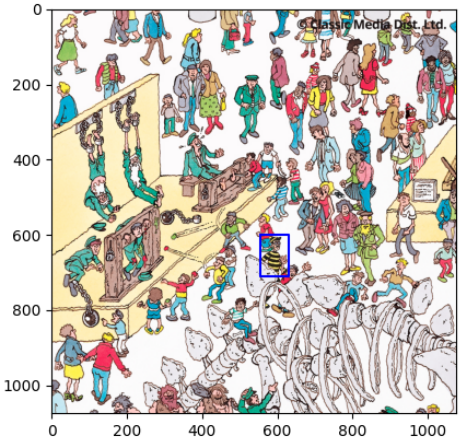


3(b)

Result of normalized cross-correlation

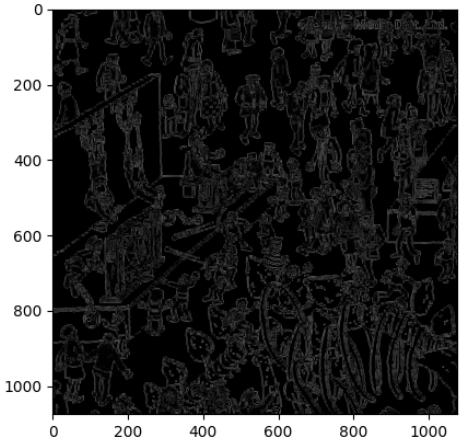


Result of localizing the template



3 Coding Part Continued

4. Visualize the Canny edge detector results on waldo.png



File - /Users/mac/Desktop/22Winter/CSC420/A1/Assignment1/1a.py

```python
import numpy as np
from skimage import io
from matplotlib import pyplot as plt
import time


def convolution(img, filter_matrix):
    """
    Write a function for computing convolution of the 2D (grayscale) image
    and a 2D filter.
    """

    # flip the filter vertically and horizontally
    filter_matrix = filter_matrix[::-1, ::-1]
    filter_shape = filter_matrix.shape[0]
    padding = filter_shape - 1

    # zero padding
    padded = np.zeros((img.shape[0] + padding, img.shape[1] + padding))

    # copy in the image matrix between the padding in the matrix
    for row in range(padding//2, padded.shape[0] - (padding//2)):
        for col in range(padding//2, padded.shape[1] - (padding//2)):
            padded[row, col] = img[row - (padding//2), col - (padding//2)]

    # make the output matrix be the same size as the input image
    result = np.zeros(img.shape)

    for row in range(result.shape[0]):
        for col in range(result.shape[1]):
            img_section = padded[row: (row + filter_shape),
                                 col: (col + filter_shape)]
            # convolution calculation
            result[row, col] = img_section.flatten().dot(filter_matrix.flatten())

    return result


if __name__ == '__main__':
    start_time = time.time()

    input_filter = np.array([[0, 0.125, 0], [0.5, 0.5, 0.125], [0, 0.5, 0]])
    input_img = io.imread('waldo.png', as_gray=True)
    output = convolution(input_img, input_filter)

    print("--- convolution takes %s seconds ---" % (time.time() - start_time))

    plt.imshow(output, cmap='gray')
    plt.show()

```

File - /Users/mac/Desktop/22Winter/CSC420/A1/Assignment1/1b.py

```python
import numpy as np


def separable(input_filter):
    """
    Write a function to verify the input filter is separable or not.
    """

    # apply SVD
    u, sigma, v = np.linalg.svd(input_filter)
    print(sigma)

    # Check if the 2nd diagonal element of the Sigma matrix is zero,
    # considering rounding errors, choose to round to 14 digits
    if round(sigma[1], 14) == 0:
        print("This is a separable filter")
        return True
    print("This is NOT a separable filter")
    return False


if __name__ == '__main__':
    filter_input = np.array([[0, 0.125, 0], [0.5, 0.5, 0.125], [0, 0.5, 0]])
    separable(filter_input)

```

File - /Users/mac/Desktop/22Winter/CSC420/A1/Assignment1/1c.py

```python
import numpy as np
from scipy import ndimage
from skimage import io
from matplotlib import pyplot as plt
import time


def faster_convolution(img, filter_matrix):
    """
    Write a faster convolution function leveraging the fact that
    the filter is separable.
    """

    # apply SVD to the filter
    u, sigma, v = np.linalg.svd(filter_matrix)

    # The separated horizontal vertical filters
    horizontal = np.sqrt(sigma[0]) * np.asmatrix(u[:, 0])
    vert = np.sqrt(sigma[0]) * np.asmatrix(v[0])

    # convolve image by the horizontal filter
    horizontal_output = ndimage.convolve(img, horizontal)
    # convolve image by the vertical filter.
    result = ndimage.convolve(horizontal_output, vert.T)

    return result


if __name__ == '__main__':
    start_time = time.time()

    # choose a separable filter
    input_filter = np.array([[1, 0, -1], [2, 0, -2], [1, 0, -1]])
    input_img = io.imread('waldo.png', as_gray=True)
    output = faster_convolution(input_img, input_filter)

    print("--- faster convolution takes %s seconds ---" %
          (time.time() - start_time))

    plt.imshow(output, cmap='gray')
    plt.show()

```

```python
import numpy as np
from skimage import io
from matplotlib import pyplot as plt


def cross_correlation(image, filter_matrix):
    """
    Implement cross-correlation.
    """

    height, width = image.shape
    frame, col_pad, row_pad = zero_pad(image, filter_matrix)

    result = np.empty_like(frame)
    # traverse all pixels and calculate the correlation
    for i in range(col_pad, col_pad + height):
        for j in range(row_pad, row_pad + width):
            result[i, j] = correlation(frame, filter_matrix, i, j)

    return result[col_pad: col_pad + height, row_pad: row_pad + width]


def correlation(image, filter_matrix, i, j):
    """
    Implement correlation.
    """
    height, width = filter_matrix.shape

    m = int((height - 1) / 2)
    n = int((width - 1) / 2)

    img = image[i - m: i + m + 1, j - n: j + n + 1].flatten()
    filter_matrix = filter_matrix.flatten()

    return np.dot(img, filter_matrix)


def zero_pad(img, fil):
    """
    Helper function to apply zero padding
    """
    i, j = fil.shape

    pad_axis_1 = (i, i)
    pad_axis_2 = (j, j)

    padding = (pad_axis_1, pad_axis_2)
    frame = np.pad(img, padding, mode='constant', constant_values=0)

    return frame, i, j


if __name__ == '__main__':
    input_filter = np.array([[0, 0.125, 0], [0.5, 0.5, 0.125], [0, 0.5, 0]])
    input_img = io.imread('waldo.png', as_gray=True)
    output = cross_correlation(input_img, input_filter)

    plt.imshow(output, cmap='gray')
    plt.show()

```

File - /Users/mac/Desktop/22Winter/CSC420/A1/Assignment1/2.py

```python
import numpy as np
from skimage import io
from scipy import ndimage
from matplotlib import pyplot as plt


def gaussian_filter(size, std):
    """
    Generate Gaussian filter with kernel size and std (sigma) as input.
    """

    # template
    gauss = np.zeros((size, size))
    # index of mean
    mu = (size - 1) / 2

    for row in range(size):
        for col in range(size):
            power = np.exp(-((row - mu) ** 2 + (col - mu) ** 2)
                           / (2 * (std ** 2)))
            gauss[row, col] = power / (2 * np.pi * (std ** 2))

    return gauss


if __name__ == '__main__':
    img_matrix = io.imread('waldo.png')

    # use a gaussian filter with kernel size=3 and std=1
    gauss_filter = gaussian_filter(3, 1)
    plt.imshow(gauss_filter)
    plt.show()

    # Convolve each RGB dimension
    img_matrix[:, :, 0] = ndimage.convolve(img_matrix[:, :, 0], gauss_filter)
    img_matrix[:, :, 1] = ndimage.convolve(img_matrix[:, :, 1], gauss_filter)
    img_matrix[:, :, 2] = ndimage.convolve(img_matrix[:, :, 2], gauss_filter)

    plt.imshow(img_matrix)
    plt.show()

```

File - /Users/mac/Desktop/22Winter/CSC420/A1/Assignment1/3.py

```python
import numpy as np
from skimage import io
from scipy import ndimage
from matplotlib import pyplot as plt


def magnitude_gradient(img):
    """
    Compute magnitude of gradients for the input image.
    """
    img_matrix = io.imread(img, as_gray=True)

    # choose the sobel filter
    sobel_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
    sobel_y = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])

    # use the sobel filter for the horizontal gradient
    horizontal = ndimage.convolve(img_matrix, sobel_x)
    # use the sobel filter for the vertical gradient
    vertical = ndimage.convolve(img_matrix, sobel_y)

    # get square root of the sum of squares of vertical and horizontal gradients
    result = np.sqrt(horizontal ** 2 + vertical ** 2)

    return result


def grid_localize(filter_img, img):
    """
    Write a function that localizes the input filter image in the input image
    based on the magnitude of gradients.
    """

    image = io.imread(img)  # colored image
    # using function from part 3a to compute the gradient magnitudes
    img_gradient = magnitude_gradient(img)
    filter_gradient = magnitude_gradient(filter_img)

    frame, col_pad, row_pad = zero_pad(img_gradient, filter_gradient)

    height, width = img_gradient.shape
    result = np.empty_like(frame)

    for i in range(col_pad, col_pad + height):
        for j in range(row_pad, row_pad + width):
            result[i, j] = normalized_correlation(frame, filter_gradient, i, j)

    similarity = result[col_pad: col_pad + height, row_pad: row_pad + width]
    max_index = similarity.argmax()

    i, j = np.unravel_index(max_index, similarity.shape)
    corners = np.array([[i, j],
                        [i + col_pad - 1, j],
                        [i + col_pad - 1, j + row_pad - 1],
                        [i, j + row_pad - 1],
                        [i, j]])

    # visualize the normalized cross-correlation
    plt.imshow(similarity)
    plt.show()
```

```
61
62      # visualize the template matching
63      plt.plot(corners[:, 1], corners[:, 0], 'b')
64      plt.imshow(image)
65      plt.show()
66
67
68  def normalized_correlation(img, filter_matrix, i, j):
69      """
70      helper function to conduct normalized correlation
71      """
72      height, width = filter_matrix.shape
73
74      # flatten images
75      img = img[i: i+height, j: j+width].flatten()
76      filter_matrix = filter_matrix.flatten()
77
78      dot = np.dot(img, filter_matrix)
79      norm_img = np.linalg.norm(img)
80      norm_filter = np.linalg.norm(filter_matrix)
81
82      if (norm_img * norm_filter) == 0:
83          result = 0
84      else:
85          result = dot/(norm_img * norm_filter)
86
87      return result
88
89
90  def zero_pad(img, fil):
91      """
92      Helper function to apply zero padding
93      """
94      i, j = fil.shape
95
96      pad_axis_1 = (i, i)
97      pad_axis_2 = (j, j)
98
99      padding = (pad_axis_1, pad_axis_2)
100     frame = np.pad(img, padding, mode='constant', constant_values=0)
101
102     return frame, i, j
103
104
105 if __name__ == '__main__':
106
107     # 3a compute and visualize the magnitude of gradients
108     plt.imshow(magnitude_gradient('waldo.png'), cmap='gray')
109     plt.show()
110
111     plt.imshow(magnitude_gradient('template.png'), cmap='gray')
112     plt.show()
113
114     # 3b localize the template in the image
115     grid_localize('template.png', 'waldo.png')
116
```

File - /Users/mac/Desktop/22Winter/CSC420/A1/Assignment1/4.py

```python
import numpy as np
from skimage import io
from scipy import ndimage
from matplotlib import pyplot as plt


def canny_edge_detector(img):
    """
    Implement the Canny edge detector, performing non-maxima suppression.
    """
    # load the image as matrix
    img_matrix = io.imread(img, as_gray=True)

    # Apply Gaussian filter
    img_matrix = ndimage.gaussian_filter(img_matrix, sigma=1, order=0)

    # choose the sobel filter
    sobel_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
    sobel_y = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])

    horizontal = ndimage.convolve(img_matrix, sobel_x)
    vertical = ndimage.convolve(img_matrix, sobel_y)

    # get square root of the sum of squares of vertical and horizontal gradients
    gradient = np.sqrt(horizontal ** 2 + vertical ** 2)

    # compute angles for each pixel
    angles = np.zeros(img_matrix.shape)

    for row in range(angles.shape[0]):
        for col in range(angles.shape[1]):
            current_angle = np.arctan2(vertical[row, col], horizontal[row, col])
            # convert to degrees
            current_angle = current_angle * 180 / np.pi
            if current_angle < 0:
                current_angle += 180
            angles[row, col] = current_angle

    # apply non-maximum suppression
    non_max = np.zeros(img_matrix.shape)
    for row in range(1, non_max.shape[0]-1):
        for col in range(1, non_max.shape[1]-1):
            # edge direction
            direction = 45 * round(angles[row, col] / 45)
            # save the neighbor edge strengths
            if direction == 0 or direction == 180:
                left = gradient[row, col - 1]
                right = gradient[row, col + 1]
            elif direction == 45:
                left = gradient[row + 1, col - 1]
                right = gradient[row - 1, col + 1]
            elif direction == 90:
                left = gradient[row - 1, col]
                right = gradient[row + 1, col]
            else:
                left = gradient[row - 1, col - 1]
                right = gradient[row + 1, col + 1]

            # compare current pixel's edge strengths with neighbors
            if left < gradient[row, col] and right < gradient[row, col]:
```

```
61                 non_max[row, col] = gradient[row, col]
62             else:
63                 non_max[row, col] = 0
64 
65     return non_max
66 
67 
68 if __name__ == '__main__':
69     canny_edge = canny_edge_detector('waldo.png')
70     plt.imshow(canny_edge, cmap='gray')
71     plt.show()
72 
```