COMPUTER VISION
PROF. GUIDO GERIG
SPRING 2018

SUBMITTED BY:

ACHYUT TIBREWALLA
Net ID: at3617

ASSIGNMENT 2

INTRODUCTION:

In this project, I investigated the use of spatial filters for image smoothing, edge detection, and template matching. The images used were provided by the professor. All code and figures for this project were created using Python, OpenCV and Matplotlib.

The first part of this project used a symmetric filter of size k*k to smooth the images. I applied linear filter to the image. The size of the kernel used was 3*3 and 5*5 to show the effects of the blurring of the image using linear filter.

The next portion of the project deals with edge detection. The image was smoothed using the linear filter. Edge detection is accomplished by estimating the partial derivative of image intensity at each pixel in the x and y directions to form a gradient. The norm of the resulting vector at each pixel can be used to create an edge map. The orientation map was calculated using the partial derivatives. The orientation map values were scaled to display the images. As an extra thing, I also applied Laplacian operator to the smoothed images to detect the edges.

The final part of this project looks at template matching which is the process of locating occurrences of a small "template" image in a larger image. This is done using spatial filtering and determining the maximum correlation found when applying the template mask to the image undergoing testing. For optimally detecting the peaks i.e., the detection of the template in the image, I used Laplacian operator.

1. Image Smoothing

Algorithm:

1. Read the image
2. Input the filter size, k
3. Initialize filter mask with 1, for linear filtering
4. Slide the filter over the image and calculate the new intensity value for all pixels, except the boundary pixel. We normalize the filter.
5. Display the new intensity values in the matrix as the filtered image.

Results of the implementation:

Original Image:

3X3 Filtered Image:

With 3x3 mask, we see that the edges in the images are starting to get blurred.

5*5 Filtered Image:

We know that linear filter blurs the sharp edges, and we can see that in our filtered image as we increase the filter mask, the filtered image edges shows an increased blur. The 5x5 linear filter shows more blurring than 3x3 filter.

2. EDGE DETECTION

Algorithm:

1. Read the image
2. Smooth the image by filtering
3. Calculate the partial derivative(Ix) along x- direction using 1X3 mask [- ½, 0, ½]
4. Calculate the partial derivative(Iy) along y-direction using 3X1 mask, transpose of the above mask
5. Calculate the vector norm sqrt(Ix*Ix + Iy*Iy). This is the edge map of the image.
6. Calculate the orientation angle as $\tan^{-1}$(Iy/Ix). Scale the value between [0,255] for display purposes.

7. Optional: I applied Laplacian operator to detect the edges too.

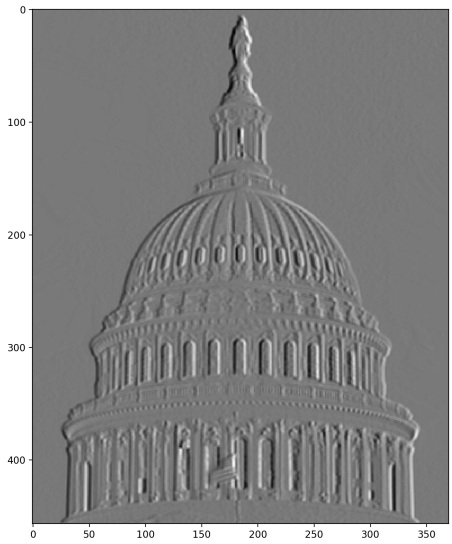
Results of the implementation:

Original Image:

3X3 Filtered Image:

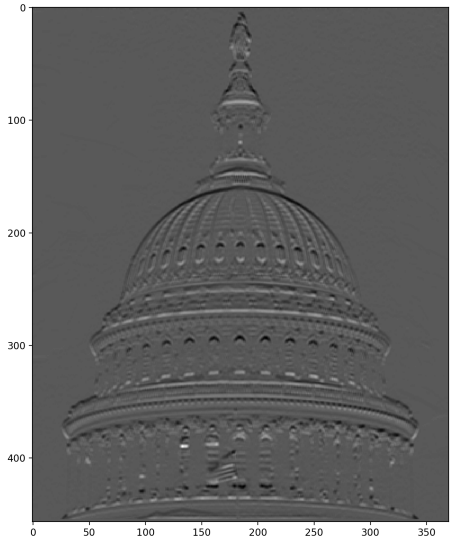
The smoothed image using 3x3 linear mask, before applying first derivatives.

Derivative along x-direction (dx) image:



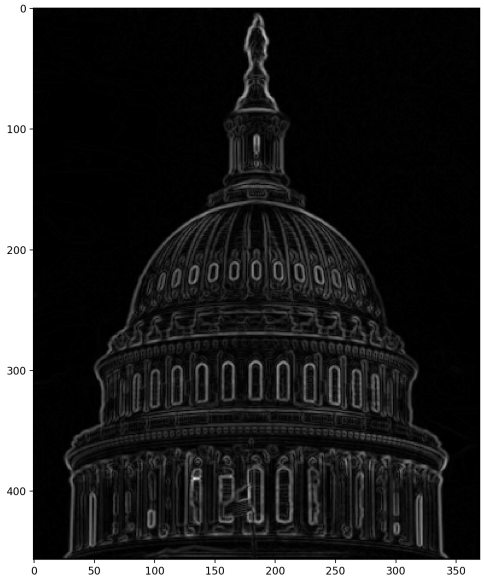
The first derivative image along x direction was obtained by convolving the image with a 1x3 mask [- ½, 0, ½]. The edges in the x direction are more prominent.

Derivative along y-direction (dy) image:



The first derivative image along y direction was obtained by convolving the image with a 3x1 mask [- ½, 0, ½]. The edges in the y direction are more prominent in this image.

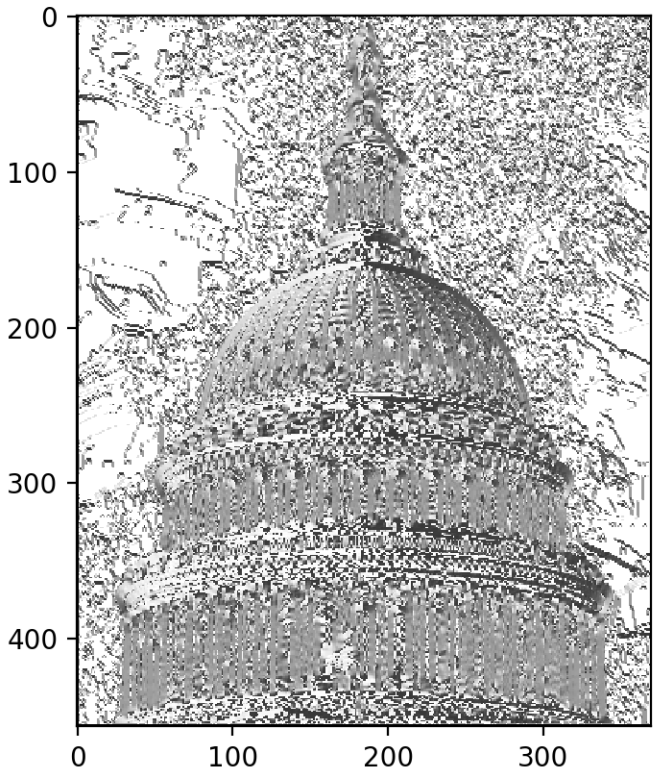
Edge map Image:



The Ix (first derivative along x- direction) and Iy (first derivative along y- direction) maps pick up different edges better. Ix detects vertical edges very well, whereas Iy is better at determining horizontal edges.
When we construct the edge map, we use the vector norm which uses both values and properly extracts the correct edges.

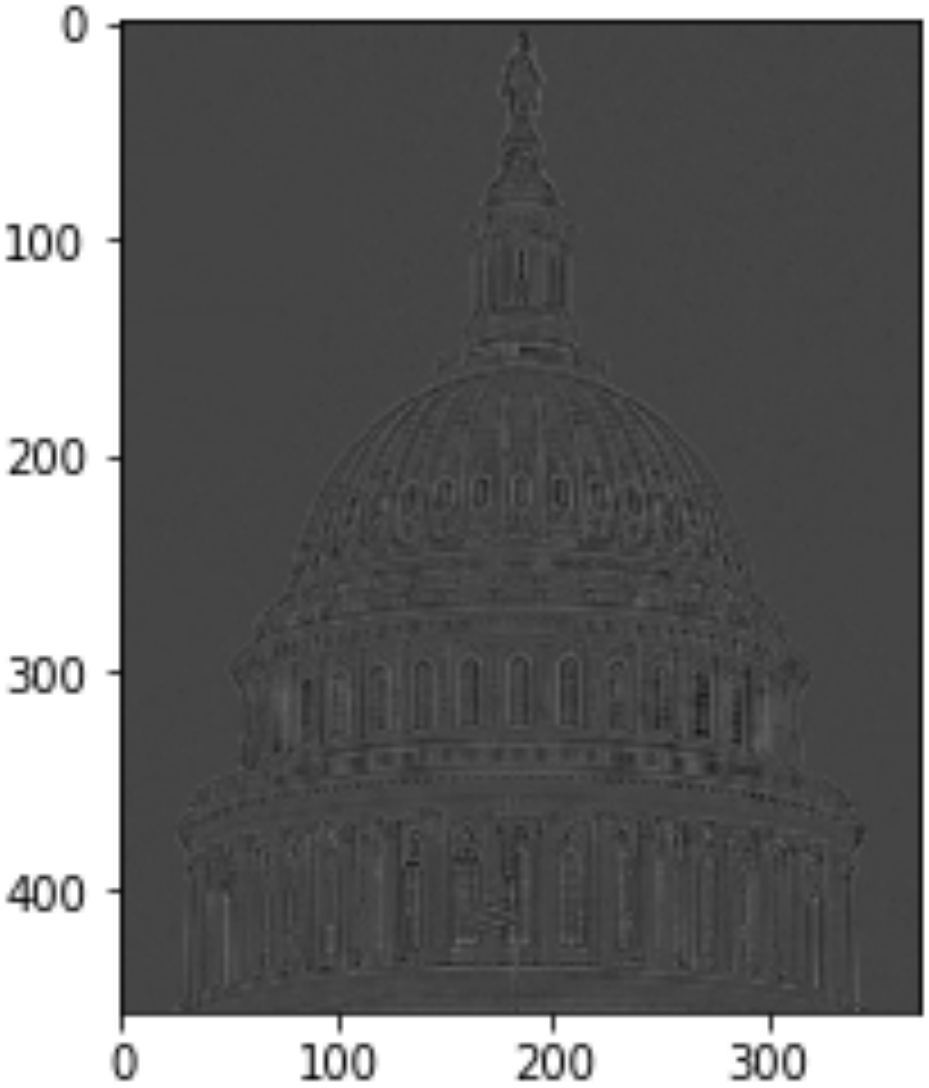
The vector norm is calculated using the formula= Square root (Ix * Ix) + (Iy * Iy)

Orientation map Image:



The orientation angle map preserves the overall structure of the image. We calculate the orientation map using the formulae = $\tan^{-1}(Iy/Ix)$.
We get the orientation map matrix values in radians. I scaled the values in the range [0..255] and displayed the image.

Edge Detection using Laplacian:



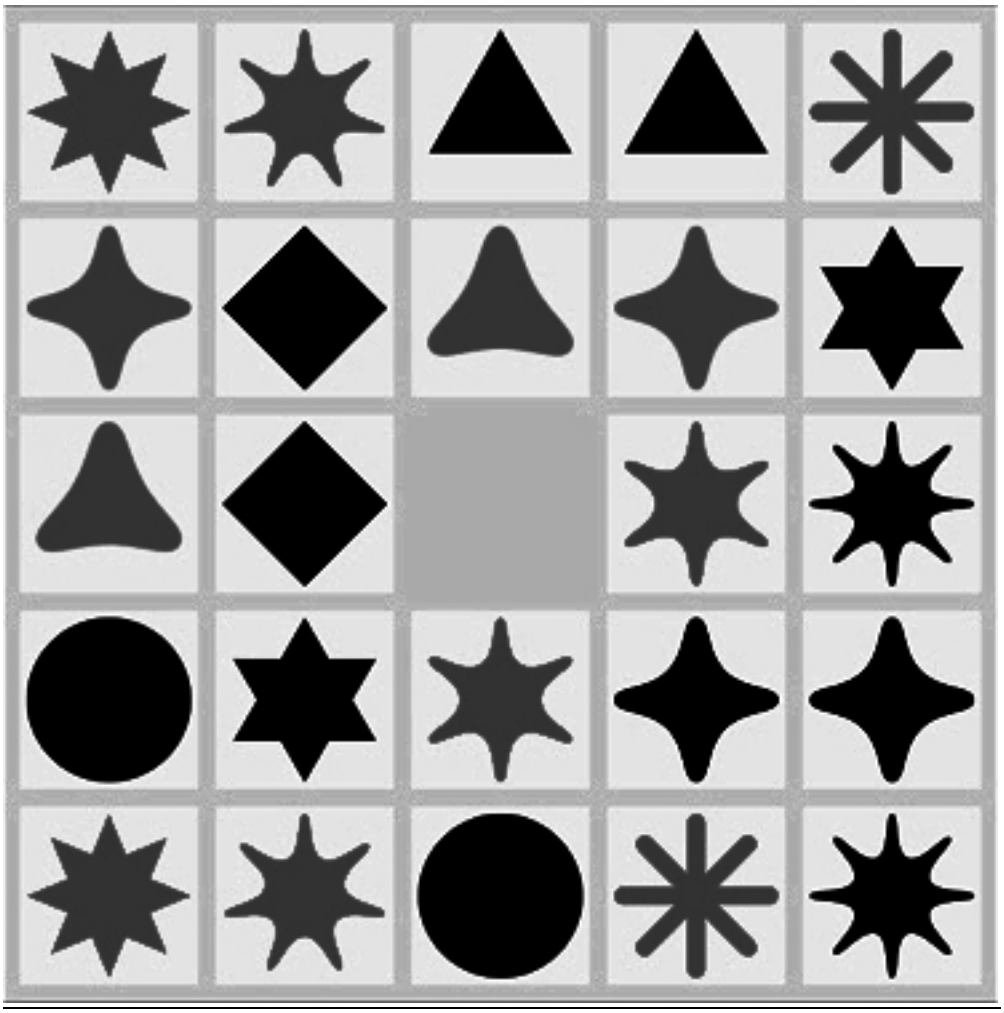
I applied Laplacian operator to see the edges in the images. The Laplacian operator was applied on smoothed image. It detects the edges very well.

3. TEMPLATE MATCHING

ALGORITHM:

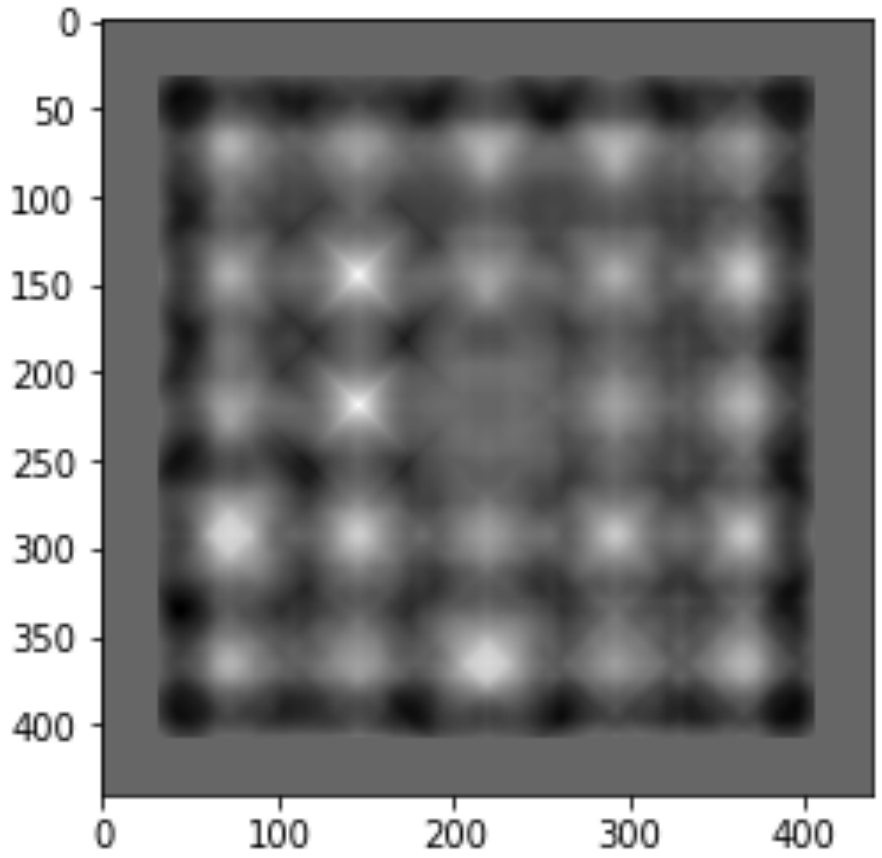
1. Read the image.
2. Read the template to be matched.
3. Convert the template to a zero-mean template.
4. Move the filter over the image array. Convert the respective region of the image, which is being processed to zero-mean region.
5. Convolve the image with the filter.
6. The matched region in the images will be visible where we get a peak.
7. To assert that the correlation values lies between -1 and 1, we calculate the normalized cross correlation, which is the original correlation divided by the magnitudes(norms) of the two zero mean vectors, i.e., filter and the image.
8. Apply thresholding to the above obtained correlation image to get the correlation peaks which indicates the location of the matched template. Display the threshold image.
9. For optimally detecting peak, we can apply Laplacian to the above obtained correlation image and enhance peaks of best correlating regions.
10. Apply thresholding to detect peaks. Display the result.

ORIGINAL IMAGE:



TEMPLATE:

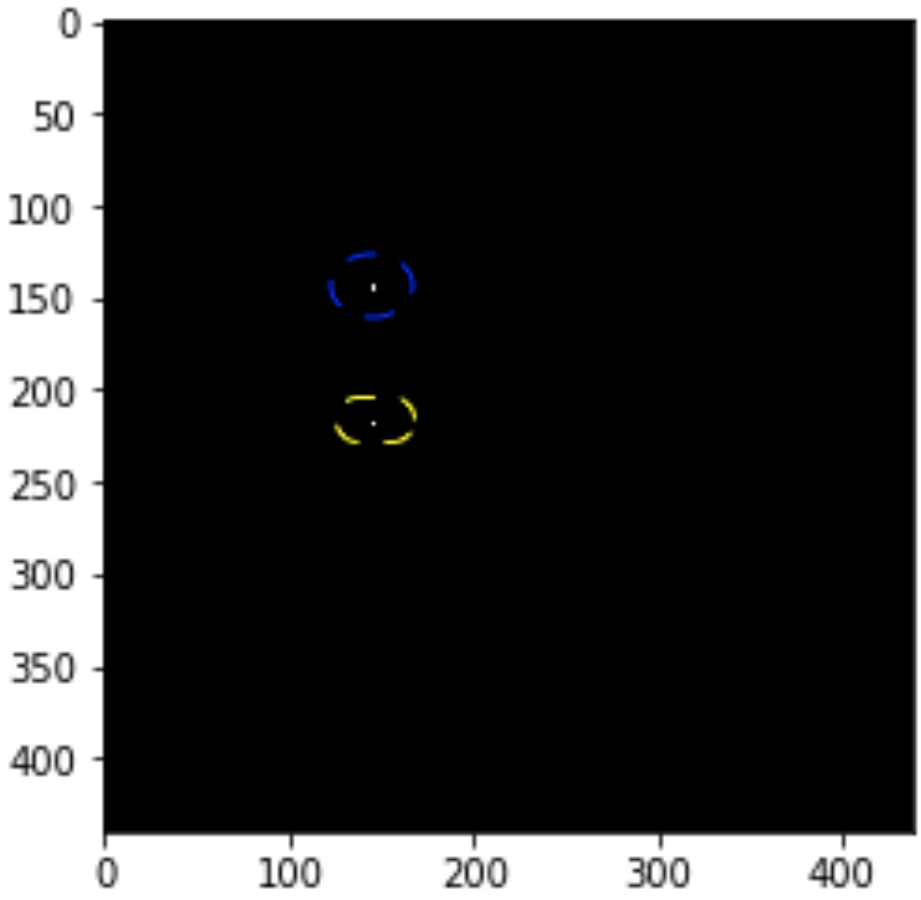
CORRELATION IMAGE:



I added padding to the original image so that the boundaries can be filtered too. The gray image around is the padding and not the part of the image.

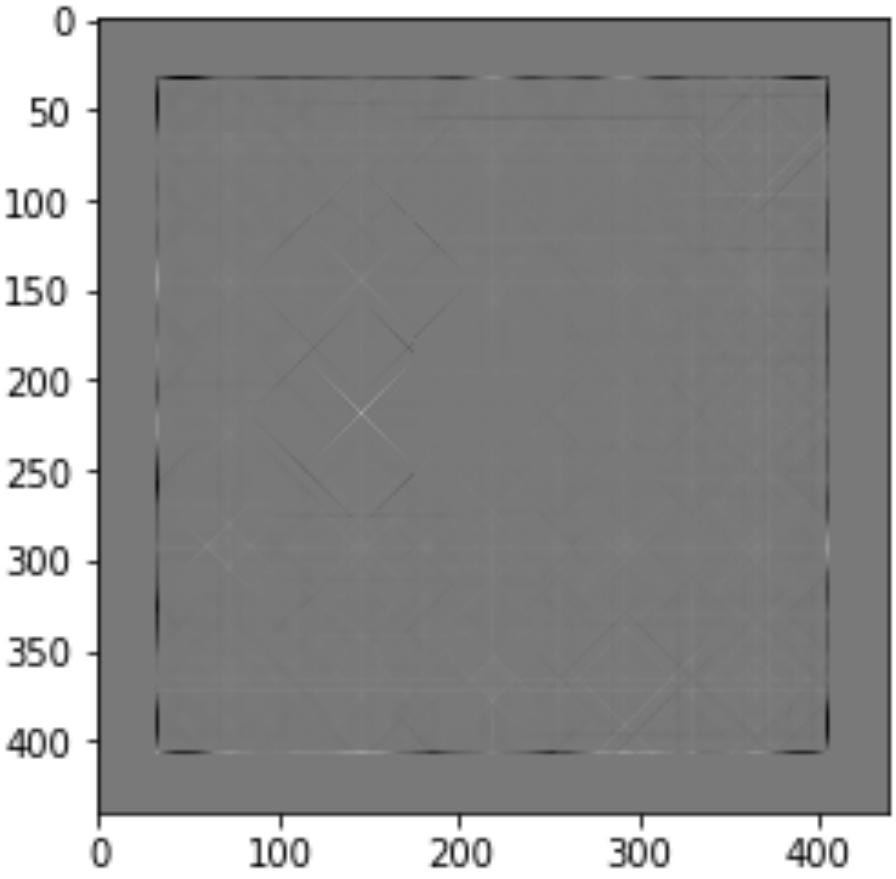
We first make our filter i.e., template a zero-mean filter. Before convolving the filter with the image, we make our image sub region also a zero-mean region.
The correlation image was obtained by moving the filter i.e., the template over the image. From the above image, I can see two peaks and that is where my template was found in the image.

Threshold Image:



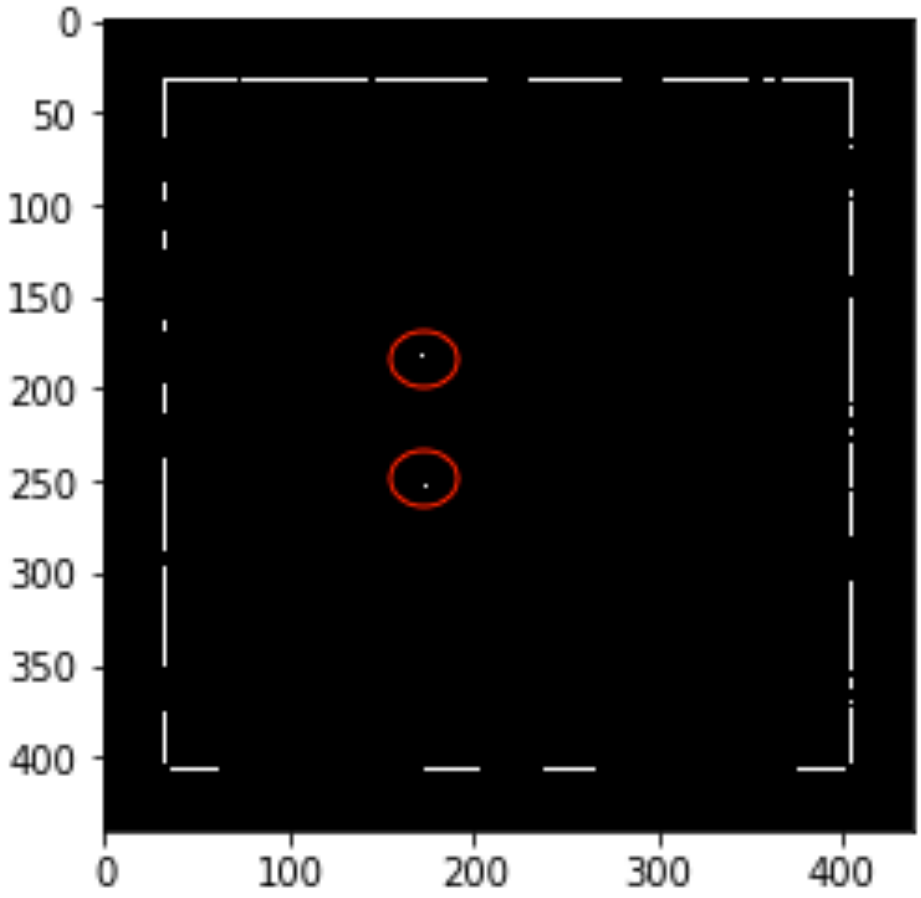
I applied a thresholding on the correlated image to get the peaks i.e., where exactly my template was found on the original image.

IMAGE AFTER APPLYING LAPLACIAN OPERATOR:



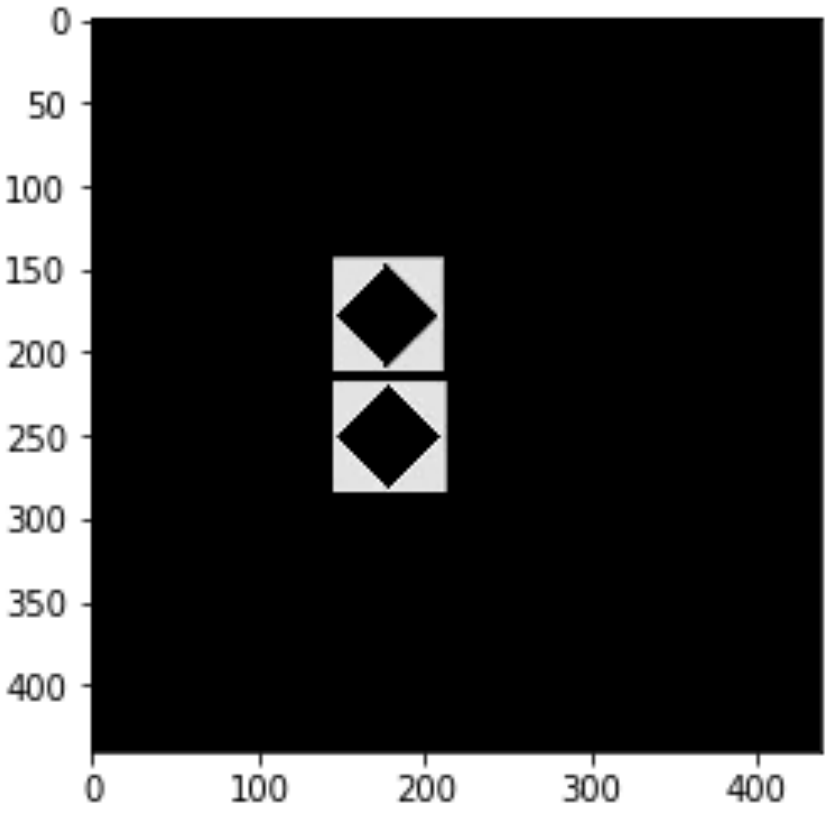
The above image was obtained by applying Laplacian operator to the correlation image. The Laplacian operator is the second derivative of the image. The mask for the Laplacian operator is ([[0,-1,0], [-1,4,-1], [0,-1,0]]). For my image, the Laplacian operator detects the templates very well.

Peak detected correlation image:



We again apply threshold to the image obtained after applying the Laplacian operator to the correlation image. For this problem, thresholding the image obtained after correlation with the template and applying Laplacian to the correlated image, both works very well.

TEMPLATE LOCATION ON THE IMAGE:



The template was pasted on the thresholded image, where we detected peaks. The position of the peaks that is the x and y location of the peaks is where I pasted the template.