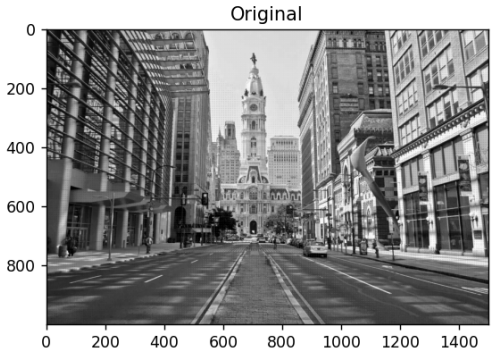
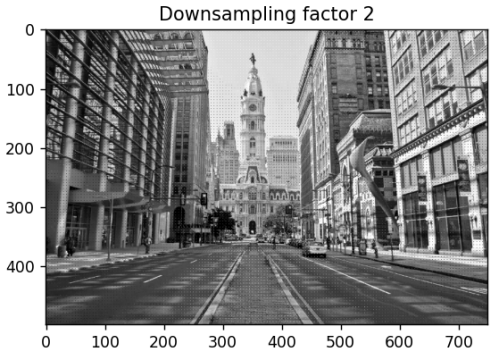
Name - Ajeet Kumar Yadav
Program - M.tech
Stream- Signal Processing
SR no - 21117
Assignment - 4

Note:

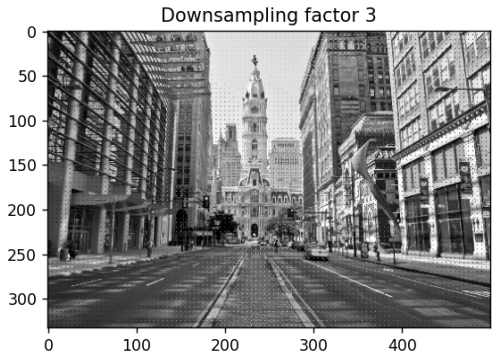
1. To run the code first extract the zip file and then open wrapper.py. After running wrapper.py hit the Enter key and see the output. After analyzing the output close all picture windows and hit Enter again for the next program.
2. I have used Visual studio code to write code, and the codes are working properly on VS code and may cause some errors on different IDEs having different versions of prerequisites.

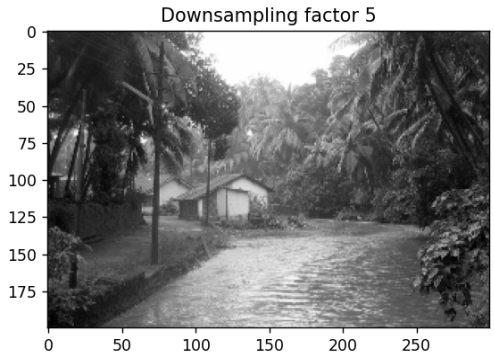
Q.1 (a). Output: (i) Downsample of the image city.png by factor 2.



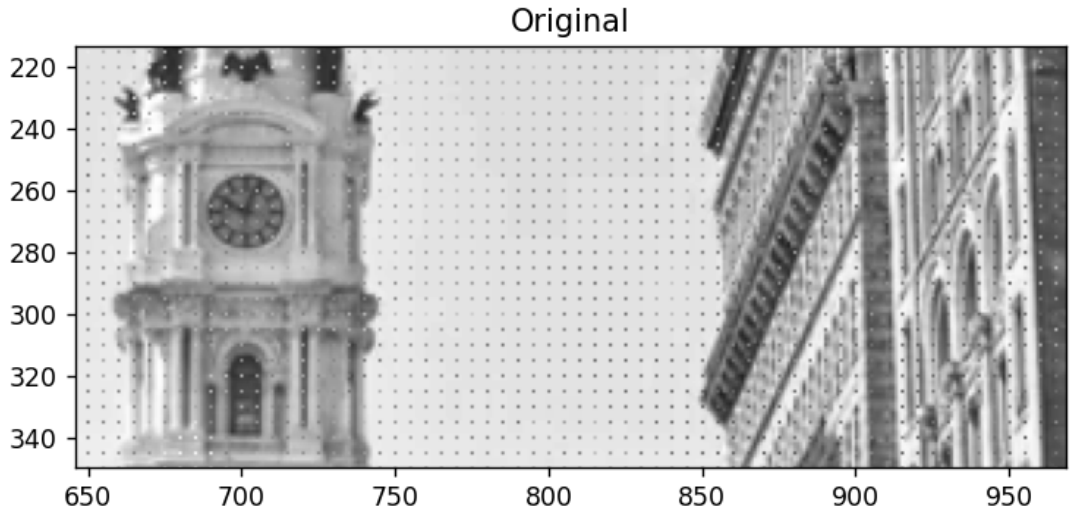


Output: (ii) Downsampled version of the image by factors 4 and 5 are



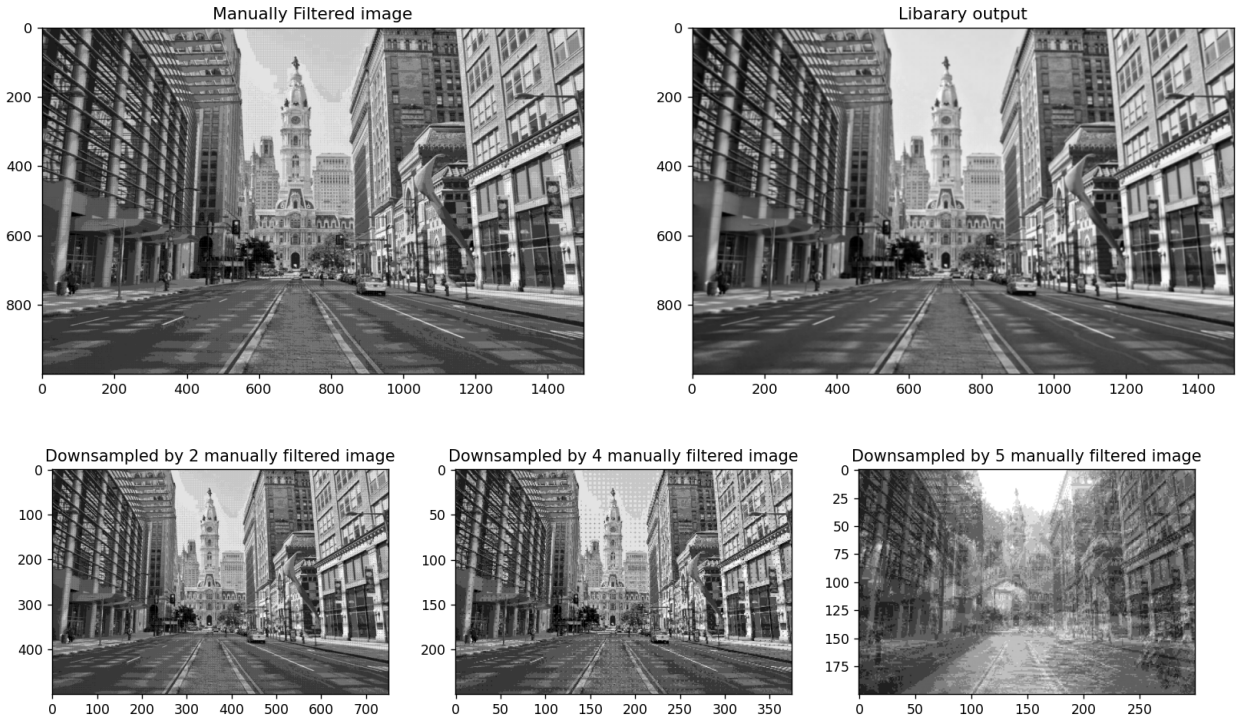


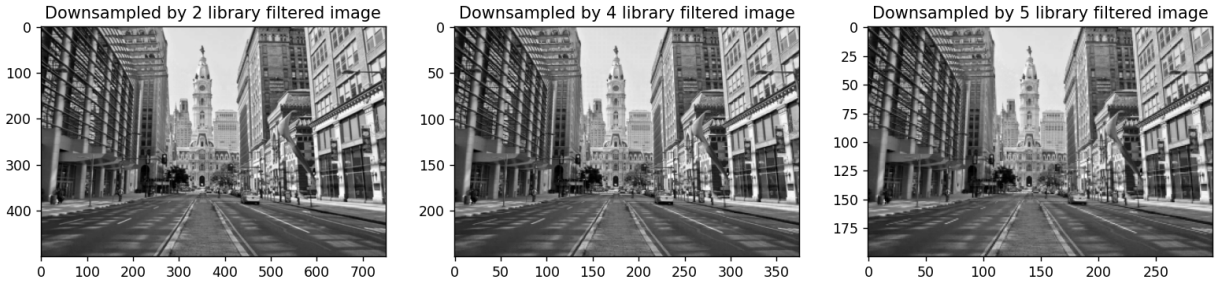
Observations:

i)	Original image contains some equidistance separated pixels of a different image. Although these pixels are not visible in the original image if we zoom into the original image we can see some periodic repetition of particular types of pixels. These pixels are  comparatively far from each other hence they do not make any sense. When we downsample the image by factor 2 alternatively two of one pixel are selected in both directions and the distance between those pixels decreases and these pixels become more visible. 
iii)	When we downsample it by a factor of 5 original picture vanishes and a new picture appears. It happens because in the original picture pixel values of another smaller size picture are spread uniformly. If we insert pixels of one image into the second image by keeping a pixel gap of 5, then by downsampling the second image by a factor of 5 we will get the first image. This was a very interesting task done by the assignment designer

This technique can be used in secret information transmission. Hidden information can be extracted by applying the same procedure through which it is upsampled.

Q.1 (b) Output(s):





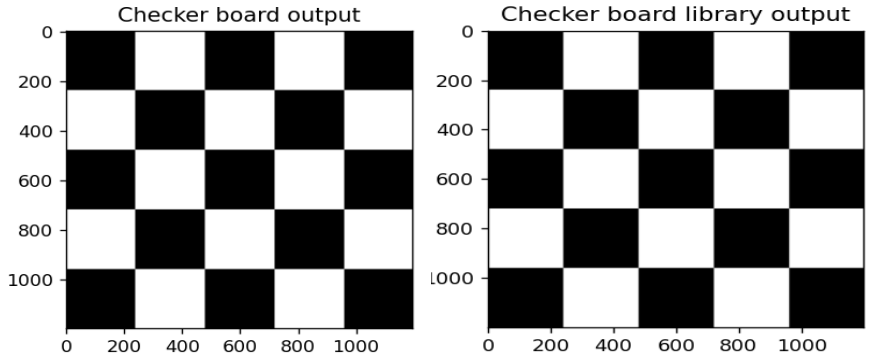
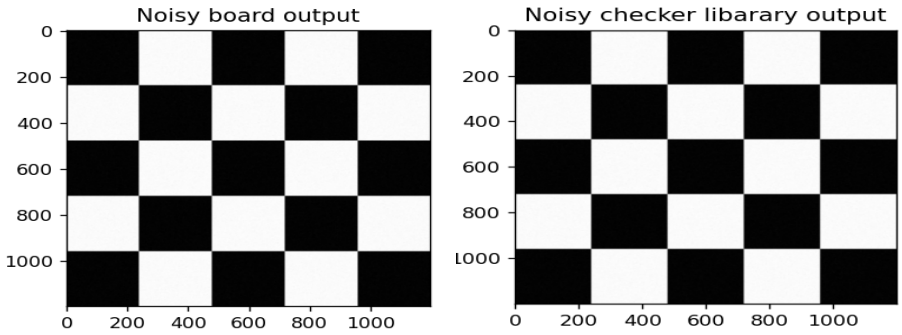
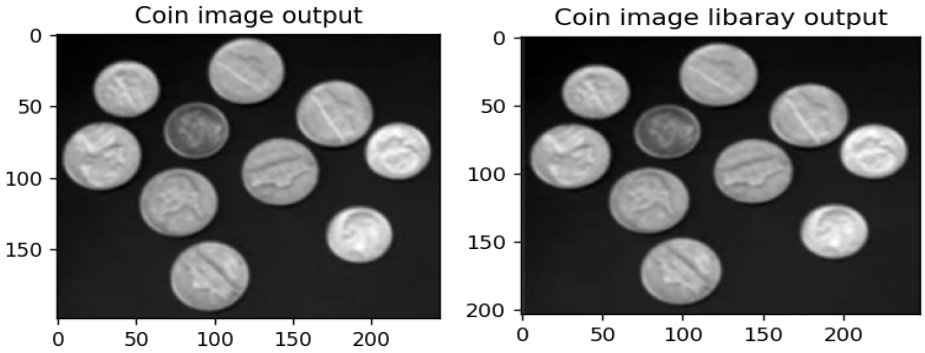
Observations:

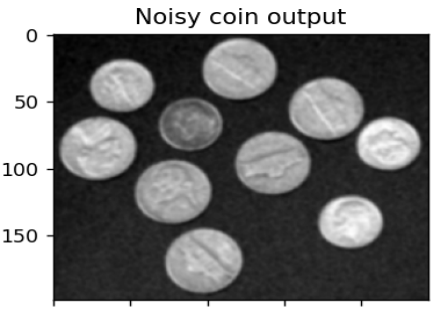
i)	Manually filtered image has some edges artifacts and also it contains less smoothen small patches in the image, while in library output the image is more smooth than the previous one. After zooming the manually coded output we can see some small irregularities in the image. While in the case of library function output there is less visibility of these artifacts  And

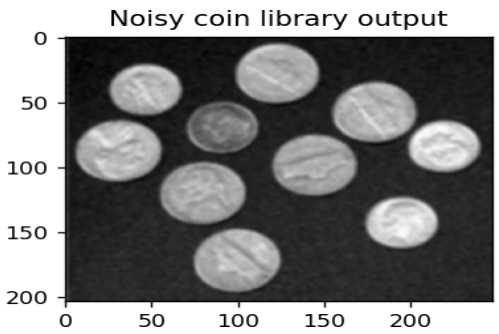
	 By zooming these images we can see the difference. This may be happening because of the explicit multiplication of the gaussian filter, and also we are calculating the filter manually which may have less precision than inbuilt.
ii)	When first we filter the image it becomes smooth. Smoothing is done by taking the pixel value of neighbor pixels hence after smoothing the correlation between pixels increases. The earlier image was containing the information of two images but after smoothing now it is only one image and if you downsample that you will not get the hidden image. Manually coded gaussian smoothing does not do much better smoothing hence in the downsampled image we can see some vibe of both images. 

Q. 2 (a) Observations and results are in table

i)	Smoothing in Checkerboard.png does not give much visible difference through the manual code as well as library code. Because it is an image which has not much noise and irregularities.

	
ii)	In the case of NoisyCheckerboard.png image has some amount of noise after passing through the gaussian filter that noise is almost removed and we get a smooth image.  And the library output and manual code output are the same.
iii)	In the case of Coins.png we are also getting the same image as the library function output image 
iv)	NoisyCoin.png contains some amount of noise, and after passing with filter noise

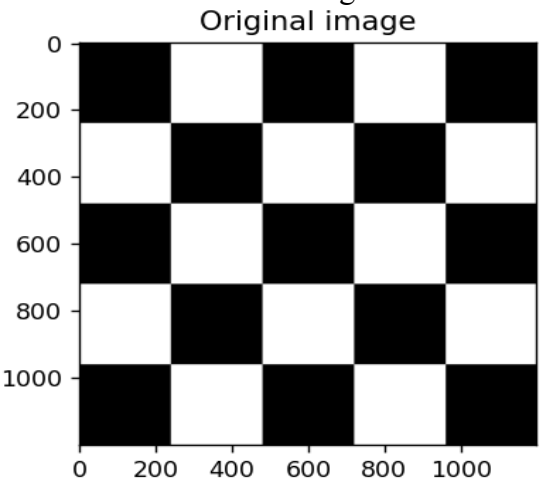


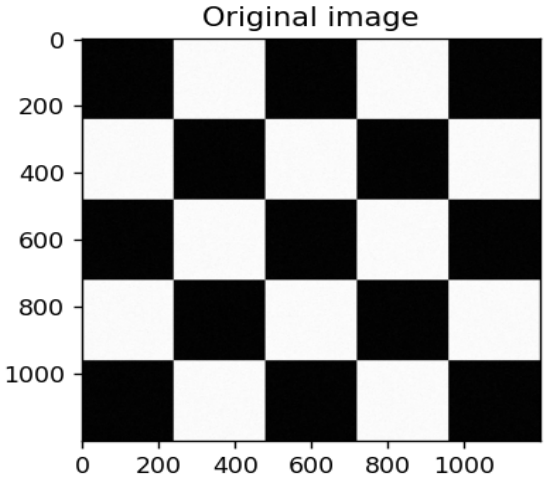


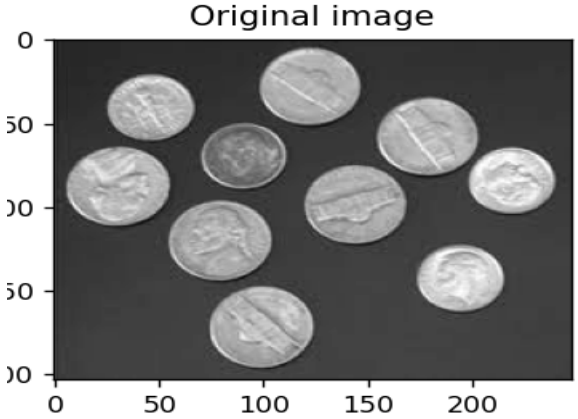
reduces

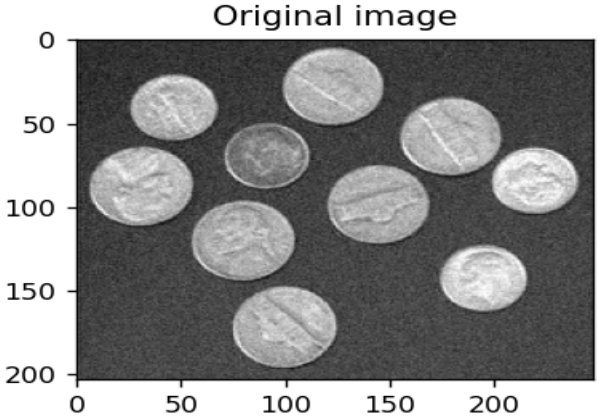
Q. 2 (b) Observations and Outputs are listed in the below table

i) There is the total of four images ordered here.

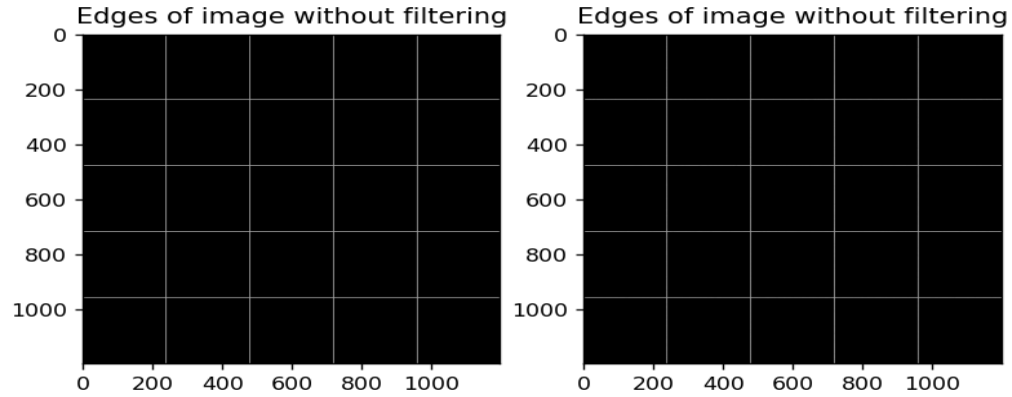


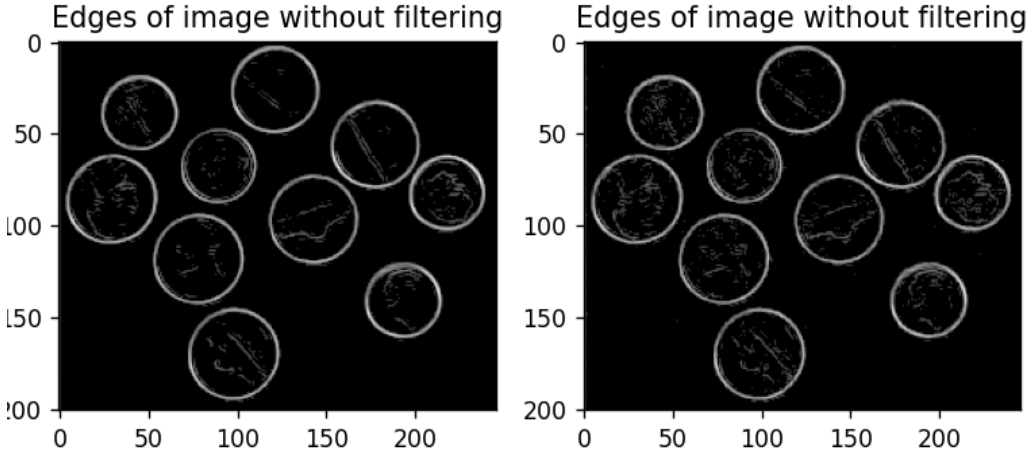






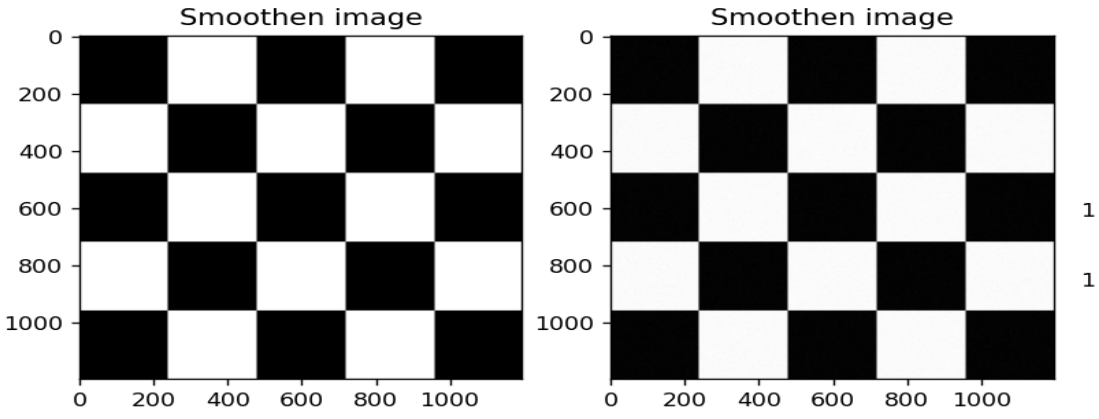
ii) And the edges of these four images are respectively

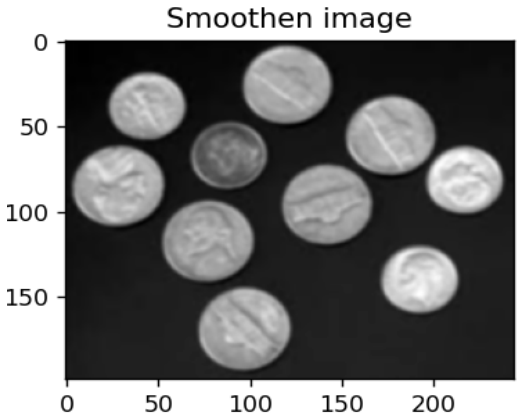


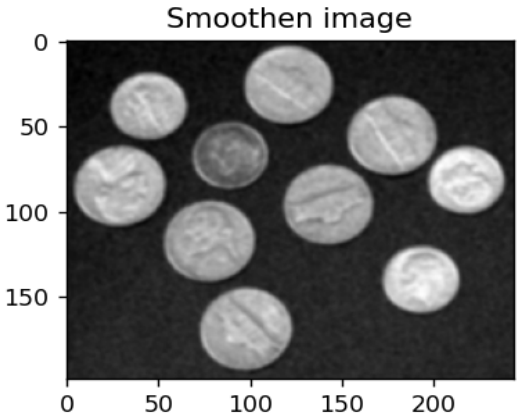


And these outputs we are getting at the threshold value 100. In the coin image, some of the surfaces are very rough and the code is also detecting them as edges but after thresholding we are getting fair output.

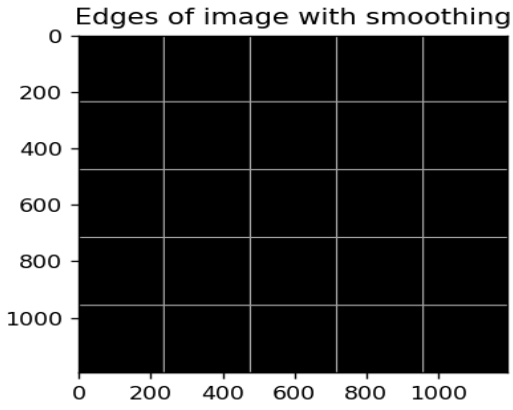
iii) Output of filtered image is as

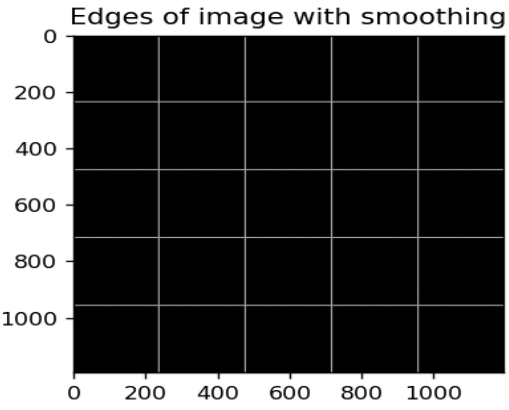


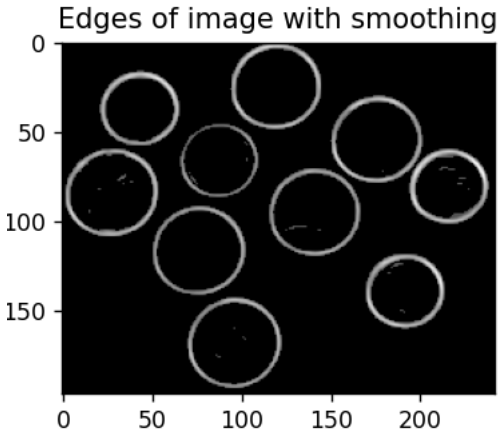


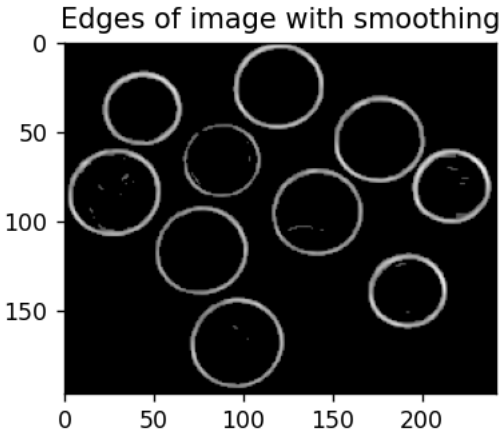


These images are more smooth than non-filtered, hence we should not get the rough surfaces in coin as edge.





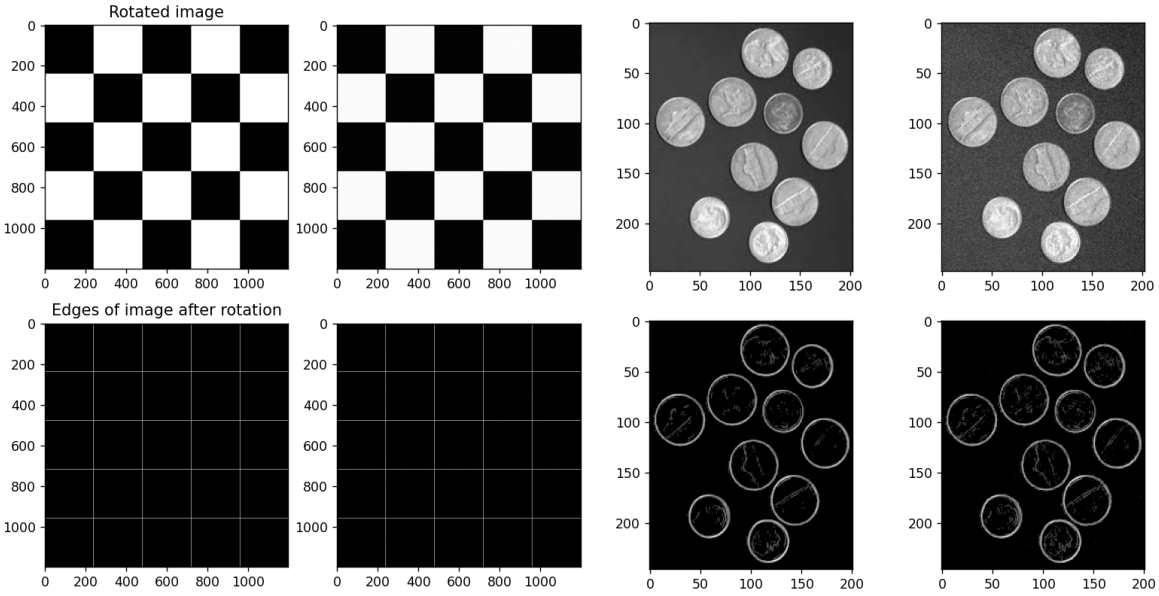




And yes we can see the smoothened image edges are the only relevant edges.

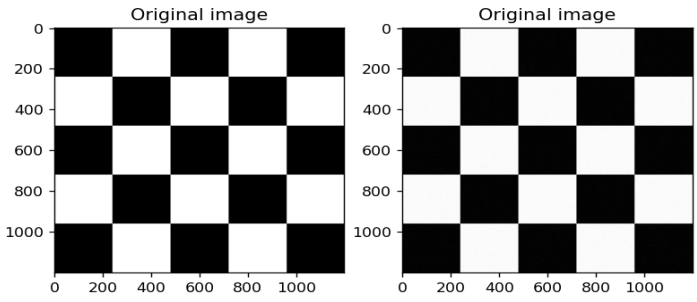
Q. 2 (c) Observation(s) and output:

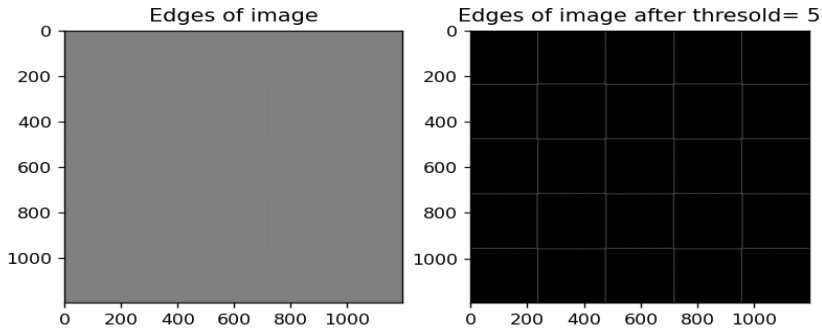
It will work properly on the rotated image as well because we are applying differentiation in both directions. This is the output of edge detection on rotated images.



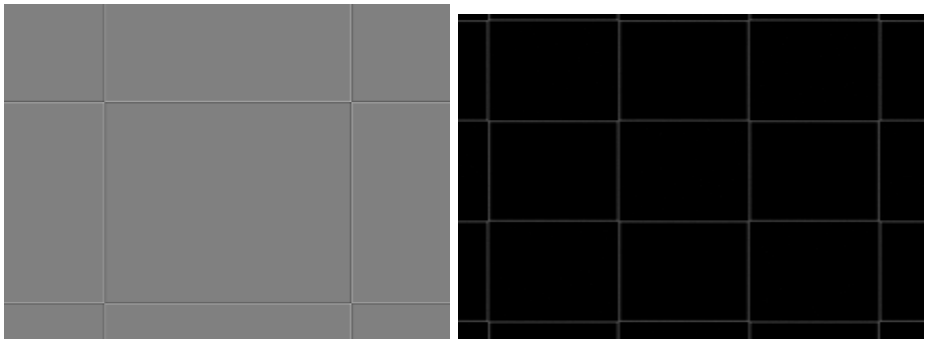
Q. 2 (d)

Output and observations:

i)	 First two images are. On the right side, image noise is present hence we need to do smoothing before applying the laplacian operator to detect the edges.
ii)	Corresponding laplacian outputs are given below.

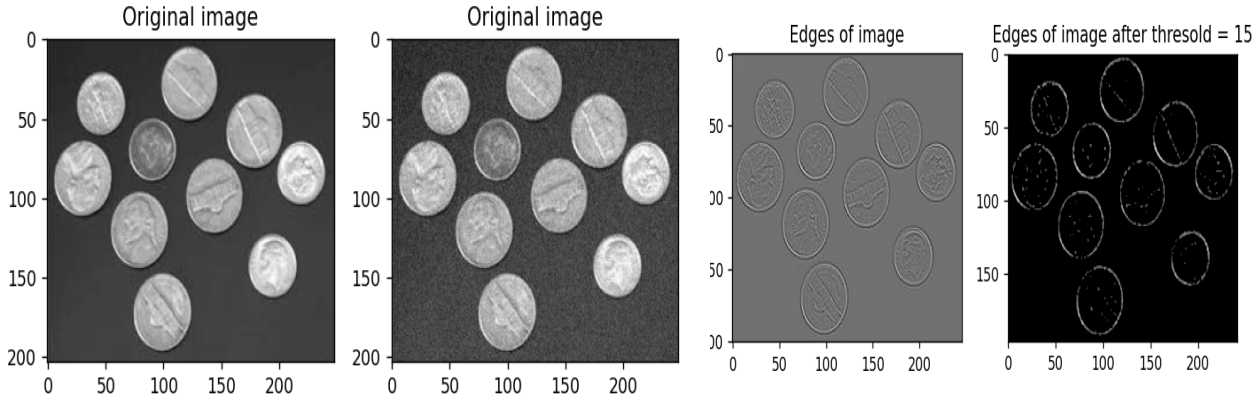


The first image is a clean non-smoothen image but the second image is smoothened before the application of the laplacian operator. Laplacian gives a second-order derivative hence the output in the first case look like this. If we zoom the image then we can clearly see the edges.



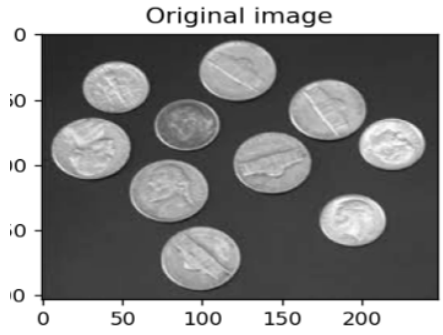
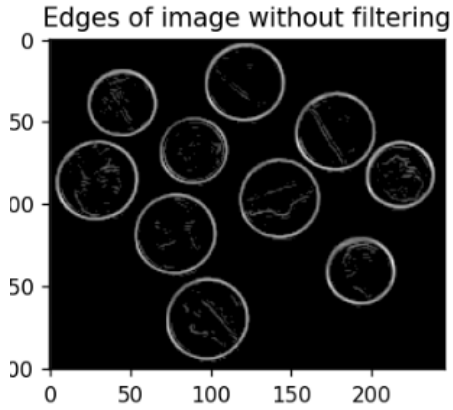
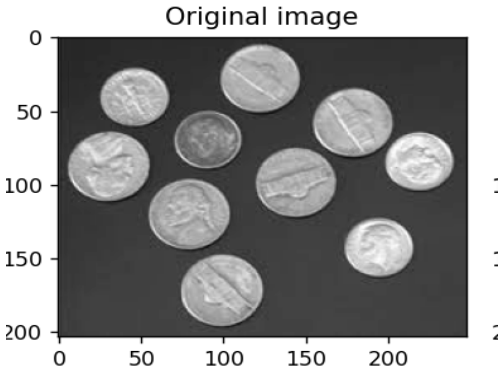
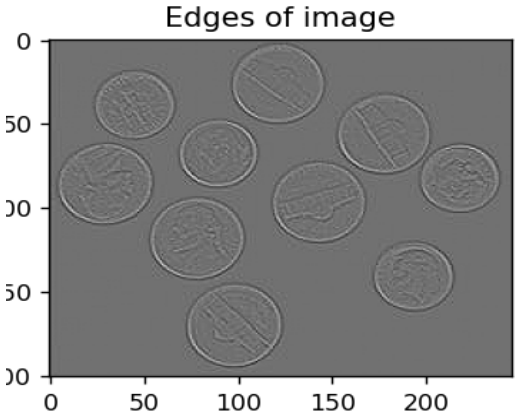
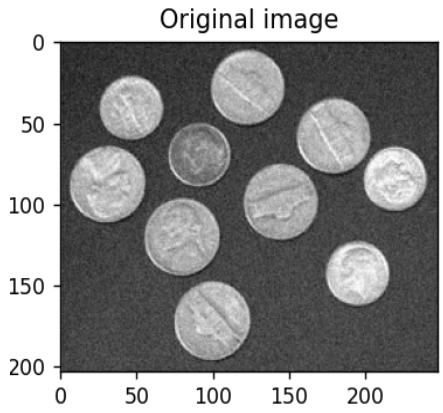
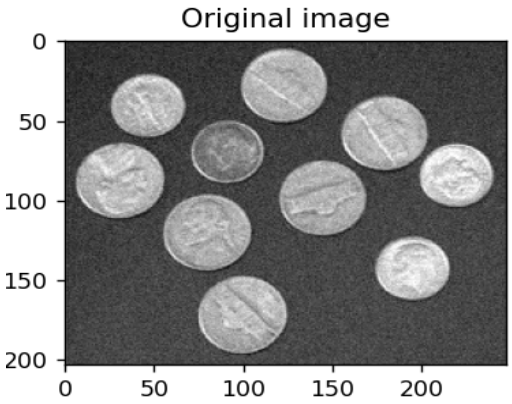
While if we first smooth the image and then apply the Laplacian operator on the image then it gives better output, because in the smooth there are less variation of pixels happens.

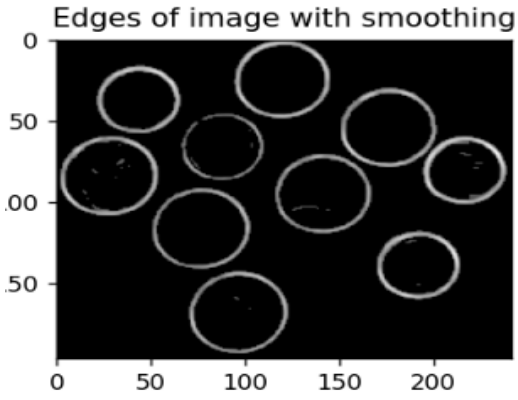
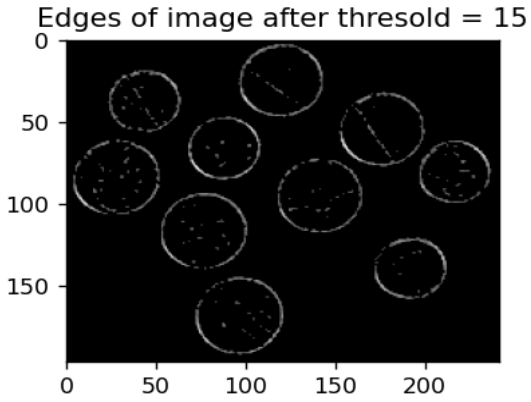
iii) And output corresponding to these images are as follows.



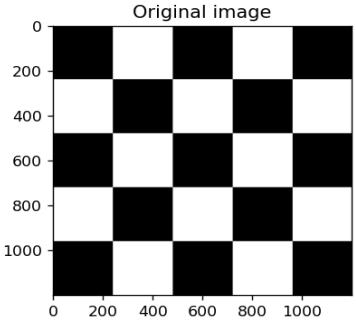
The second image contains noise and after smoothing we are performing edge detection at threshold value 15.

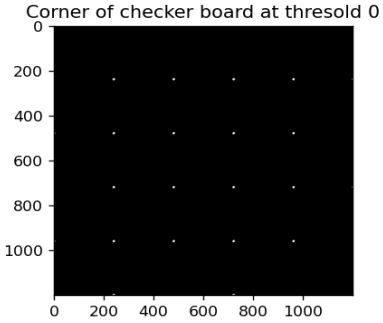
Comparison of first-order and second-order gradient-based edge detectors.

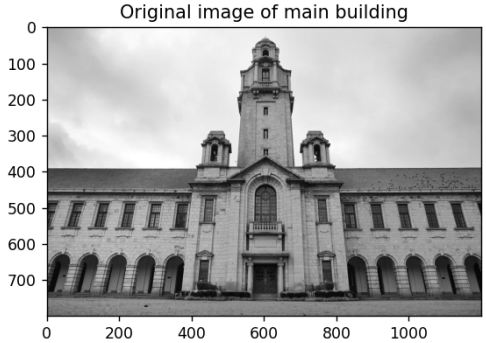
First order detector	Second order detector
Clean image  And Output corresponding to this.  Output is clean and some rough surfaces on the coins are also present.	 And output corresponding to this.  Output edges are not much clear after the application of the second order of derivative.
 Noisy input image	 Noisy input image.

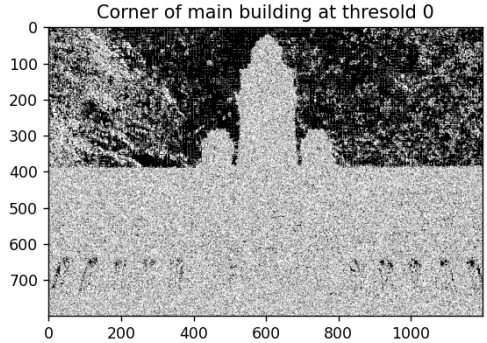
Corresponding first-order output is:  In this case, the first-order operator is giving a better output.	Output of second order  Second order operator output is not much more accurate than first-order output.

Q. 3 (a) Output(s)

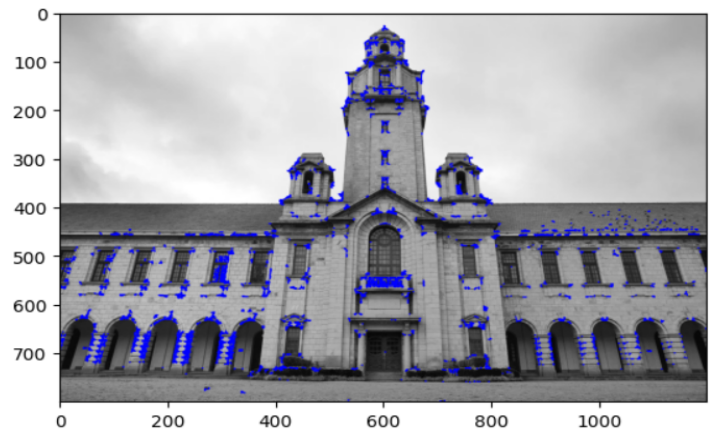
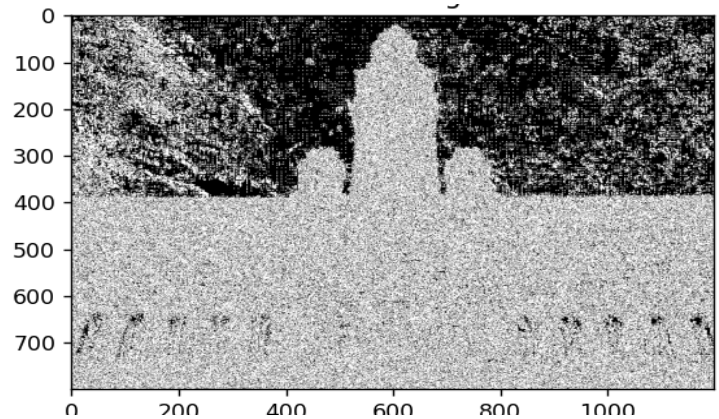
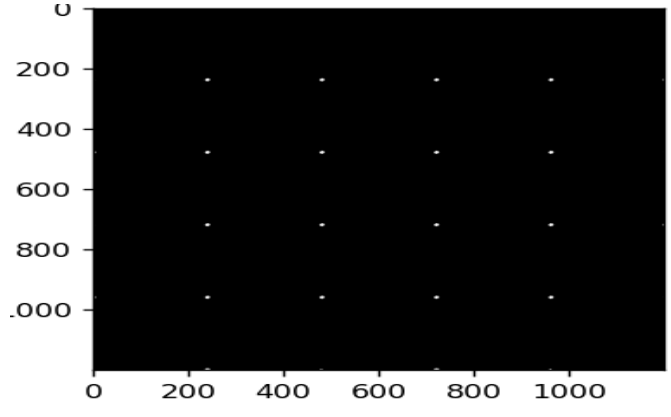


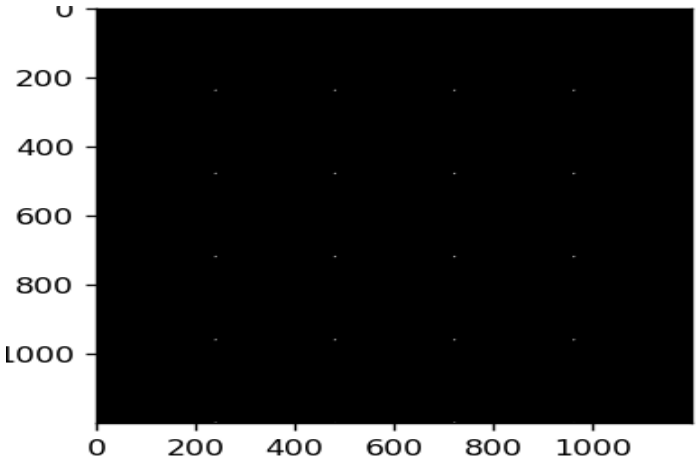






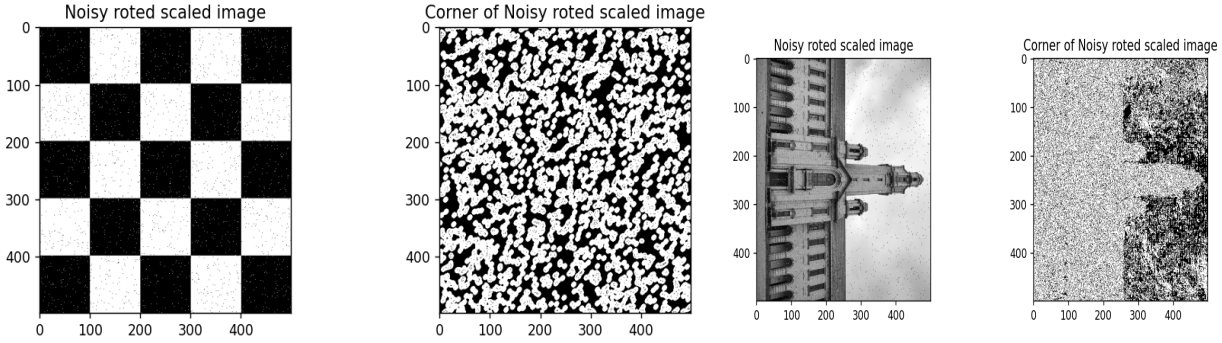
The above corner detection is performed as threshold value zero. Further, we can get different outputs at different thresholds. Some of them are mentioned below. And also the output of MainBuilding.png is not much appreciable but if we Bilate it with the original image then we can get a very appreciable output.

T = 0.01	
T = 0	
T = 1	

T = 10	

Q. 3 (b)

This method will work on rotated and scaled images but may not work on noisy images. Some light amount of noise can give output. To see the limit of the algorithm I am using typical salt and pepper noise. It is not giving good results on salt and pepper noise.

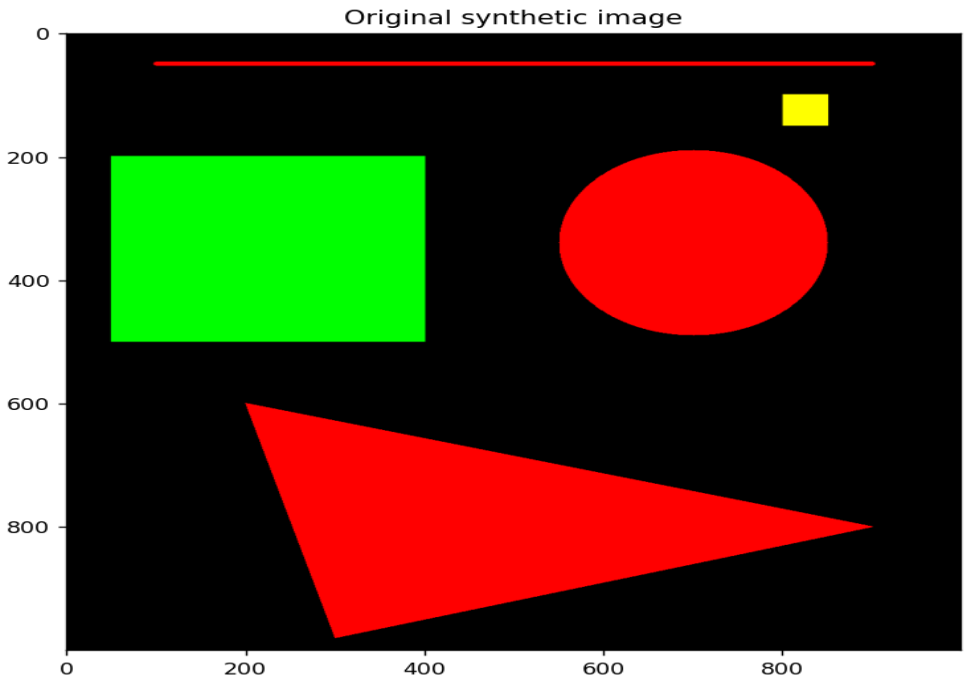


Salt and pepper noise has impulse of intensity and the algorithm may treat the corner. Hence we are seeing the many corners in the output image. But it may work on gaussian noised images till a limit.

Q. 3 (c)

My Synthetic image is given below and it contains blob, corners, rotated corners, line etc.

If we apply the Harris corner detection on this image then we are able to detect corners of the line, corners of the rectangle, and some part of Sharpe edges of the circle but, we are not getting the corner of the triangle even after changing threshold value.



And the output is: