Name : Deepshikhar Bhardwaj (M20MA004)
Roll Number : M20MA004
M.Tech DCS
IIT Jodhpur

Computer Vision
CSL7360

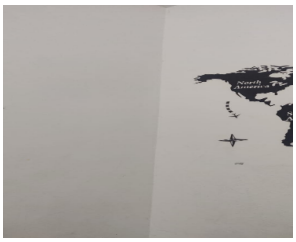
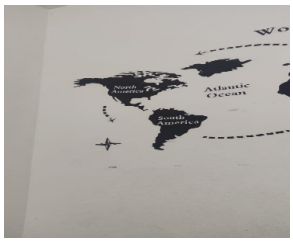
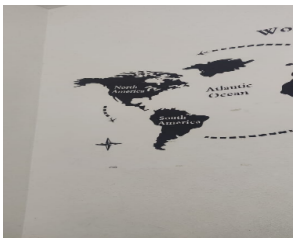
Assignment 2

1] Panorama Creation

For creating Panorama using Image stitching, as an input image I clicked 5 photos through my camera in such a fashion that there lied some level of intersection and this overlapped region helped the algorithm of inbuilt function stitch_create to implicitly create the panorama based on the images in the list in the python code.

Images selected for the panorama creation are given below:

Input Images

Image (i)	Image (i+1)		Combined Image(i, i+1)
		=	
		=	
		=	
		=	

Now stitching the results of the combined results

Observation :

Stitch method implicitly performs the blending operation.

The image stitching method is capable of producing more aesthetically pleasing output panorama images through the use of gain compensation and **image blending**.

2] Corner Detection using Harris Algorithm

Using in-built function:
window size = 3,
Harris Parameter ‘k’ = 0.04 and
threshold = 0.01 * max score of result image

In built output

```
threshold 83013710.0 * 0.01
Number of corners : 16564
```

```
threshold 41408350.0 * 0.01
Number of corners 8452
```

threshold 48073616.0 * 0.01
Number of corners : 2093

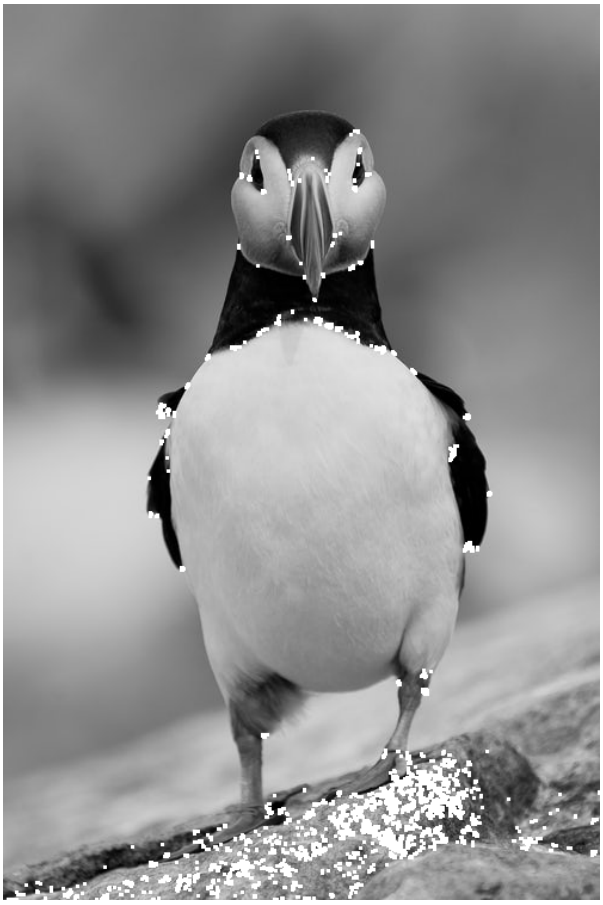
threshold 129765280.0 * 0.01
Number of corners: 1120

threshold 19312298.0 * 0.01
Number of corners : 1218

threshold 33503594.0 * 0.01
Number of corners : 5618

threshold 9803336.0 * 0.01
Number of corners : 2482

threshold 107388870.0 * 0.01
Number of corners   2840



threshold 29200912.0 * 0.01
Number of corners 9816

threshold 22471114.0 * 0.01
Number of corners 2138

Implementing from scratch:

Taking window size = 3, k = 0.04 and threshold = 0.01 * maximum score of the resulted image

In built output	Scratch Output
 threshold 83013710.0 * 0.01 Number of corners 16564	 Number of corners detected 39885
 threshold 41408350.0 * 0.01 Number of corners 8452	 Number of corners detected 12113
 threshold 48073616.0 * 0.01 Number of corners 2093	 Number of corners detected 3187

threshold 129765280.0 * 0.01
Number of corners 1120

Number of corners detected 2752

threshold 19312298.0 * 0.01
Number of corners 1218

Number of corners detected 1155

threshold 33503594.0 * 0.01
Number of corners 5618

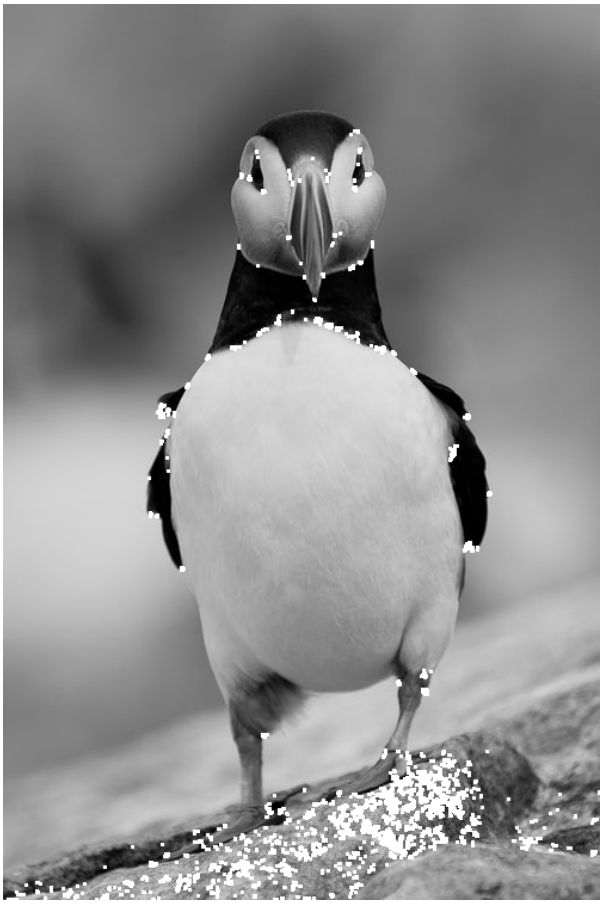
Number of corners detected 5942

threshold 9803336.0 * 0.01
Number of corners 2482

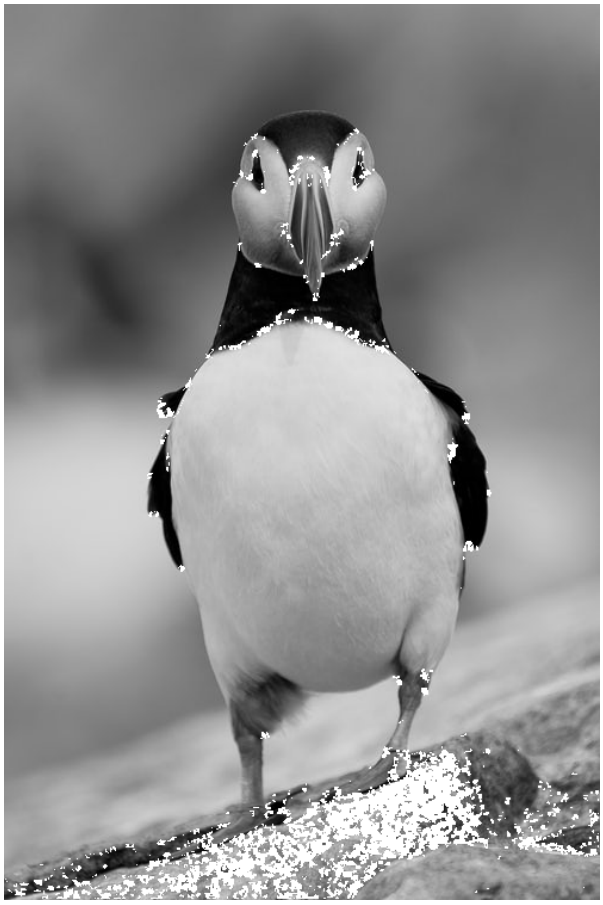
Number of corners detected 1850

threshold 107388870.0 * 0.01
Number of corners 2840

Number of corners detected 5432



threshold 29200912.0 * 0.01
Number of corners 9816



Number of corners detected 14952

threshold 22471114.0 * 0.01
Number of corners 2138

Number of corners detected 2498

Observations:
For the same threshold value, the inbuilt harris corner algorithm is able to produce better results as in the scratch algorithm I have used a simple EigenValues enhanced version to determine the corners based on R value.

Window size variation

```
threshold = 83013710.0 * 0.01
```

```
window_size = 1
Number of corners detected 0
```

window_size = 2

Number of corners detected 39885

window_size = 3

Number of corners detected 39885

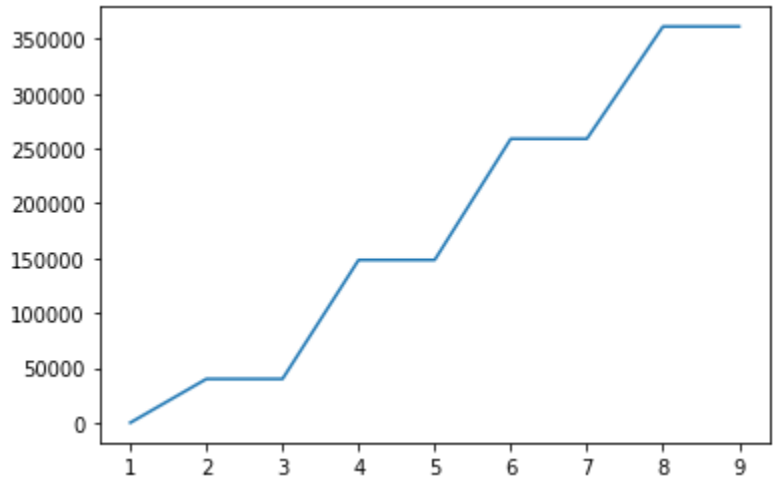
window_size = 4

Number of corners detected 148361

window_size = 5

Number of corners detected 148361

Observations:



Threshold Value variation

```
window_size = 3
```

```
Threshold = 83013710.0*0.01 (=0.01* max score of result image)
Number of corners   8265
```

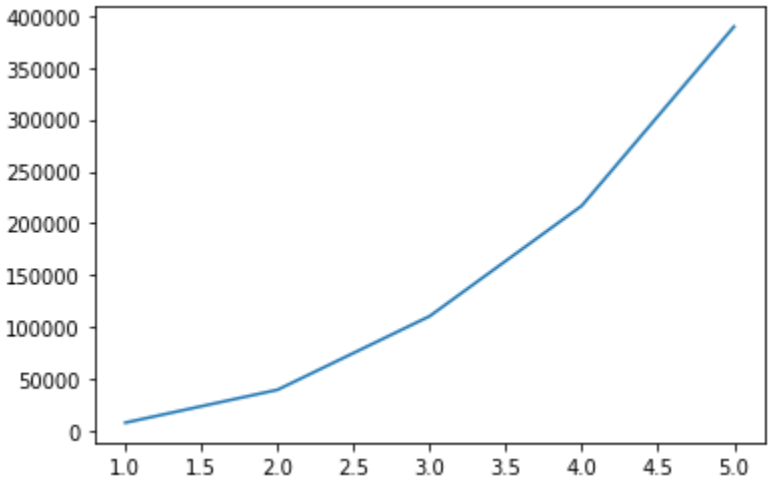
```
threshold = 0.1 * threshold
old threshold 83013710.0 * 0.01
new threshold 83013710.0 * 0.001
Number of corners detected 110747
```

```
threshold = 0.01 * threshold
old threshold 83013710.0 * 0.001
```

```
new threshold 83013710.0 * 0.0001
Number of corners detected 217072
```

```
threshold =0.1 * threshold
old threshold 83013710.0 * 0.0001
new threshold 83013710.0 * 0.00001
Number of corners detected 389399
```

Observations:

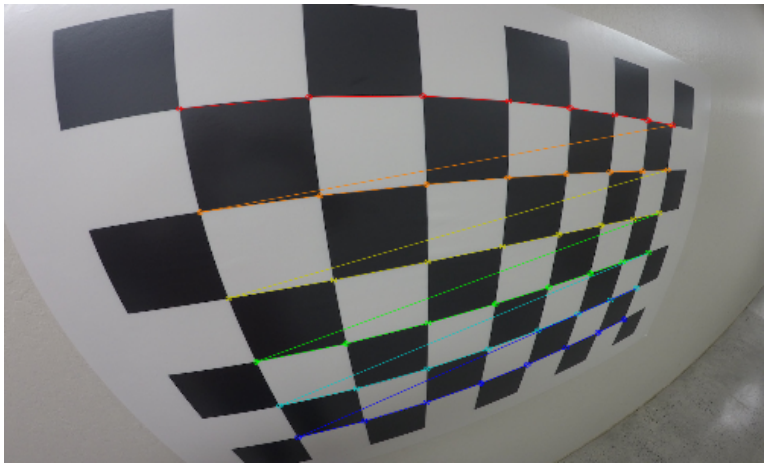


Y-axis : number of corners

X-axis : ‘i’ abstractly denoting to, 1 as [dst.max()/100] , 2 as [dst.max()/101] and so on respectively.

As there is decreased in the threshold value, the number of corners detected are increased exponentially.

3] Camera Calibration



.

.

Intrinsic and Distortion Matrix

```
Intrinsic Matrix
[[559.66086702   0.         651.12251224]
 [  0.         560.74158563 498.77835168]
 [  0.           0.           1.        ]]

 Distortion Matrix
```

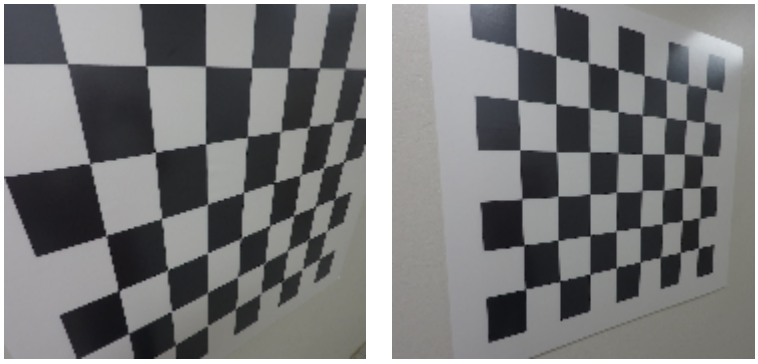
```
[[-2.32230232e-01  6.13449663e-02  4.37961599e-06  5.48089827e-05
  -7.48458246e-03]]
```

fx = 559.66
fy = 560.74

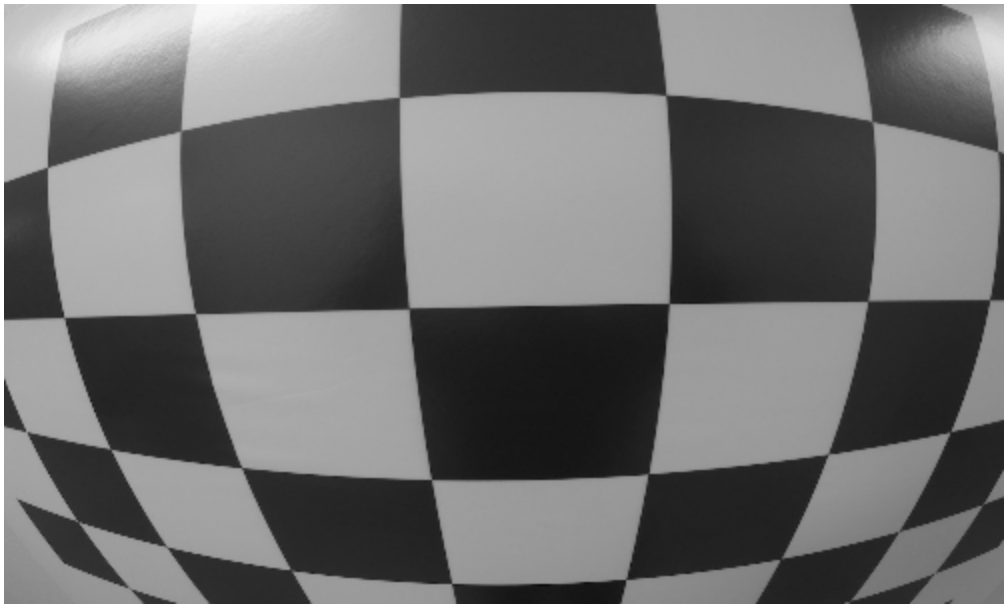
Radial Distortion Coefficients： `-2.32230232e-01  6.13449663e-02 -7.48458246e-03`

Tangential Distortion Coefficient : `4.37961599e-06  5.48089827e-05`

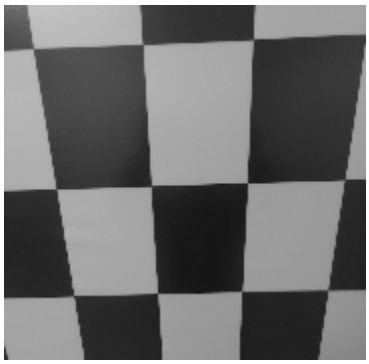
Undistort + Crop



Test image



Optimal matrix -> undistorting -> cropping -> Output

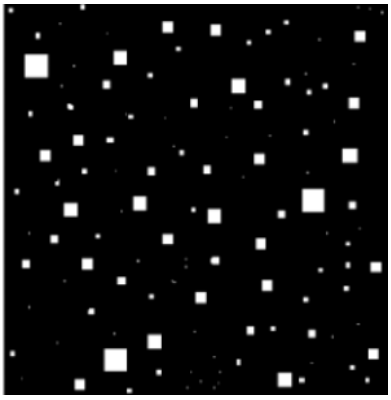
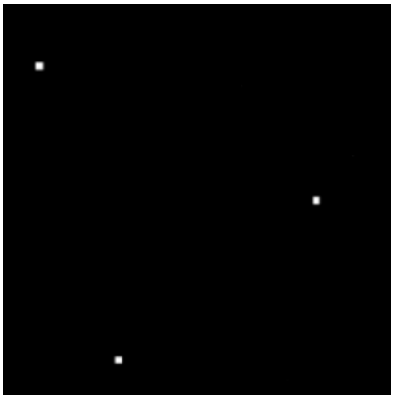
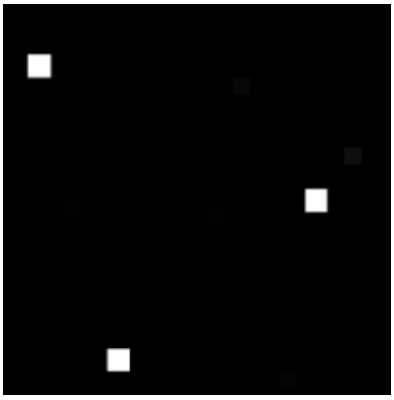


Observations: As per the observed results the radial distortion has been removed but there still lies some tangential distortion.

4] Morphology

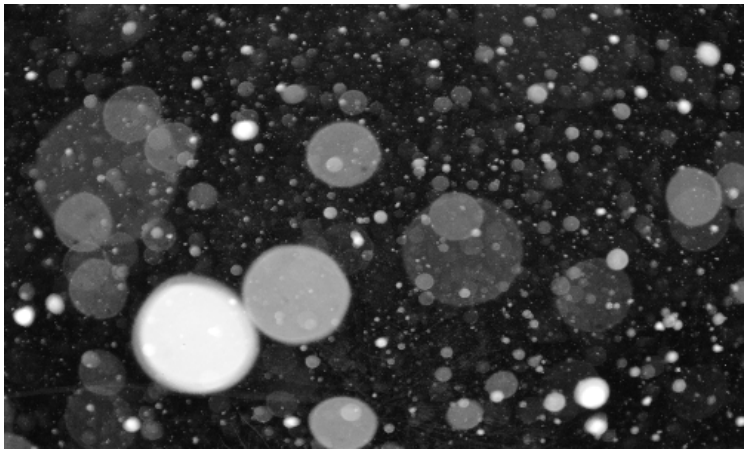
Applying Logic as explained in class for detecting top 3 squares

Input data

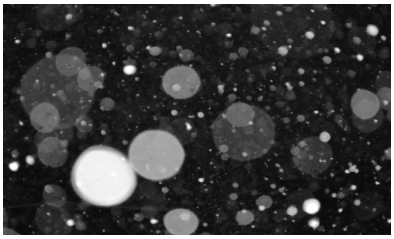
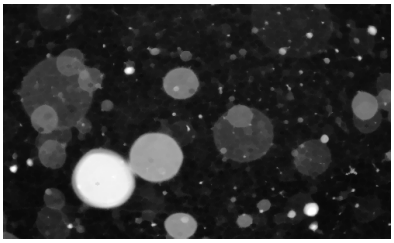
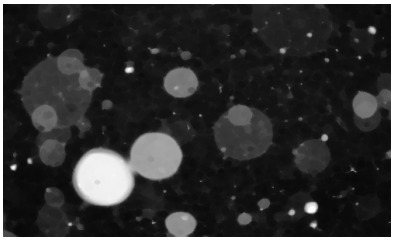
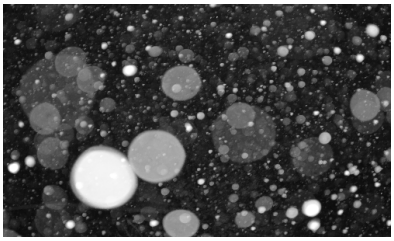
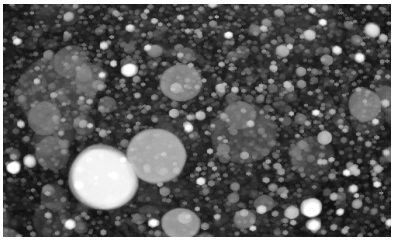
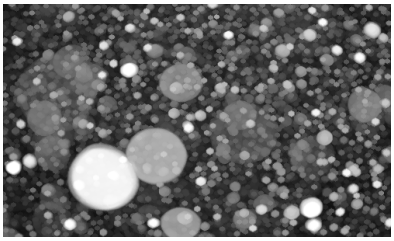
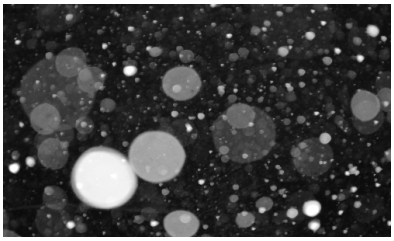
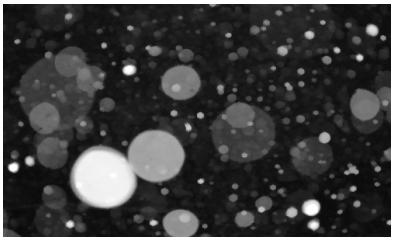
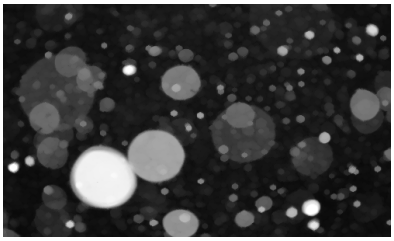
Image	Erosion	Dilation
		

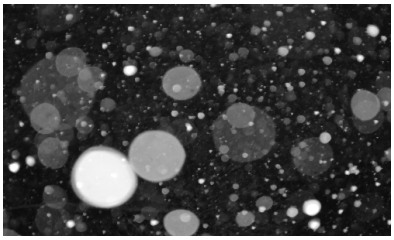
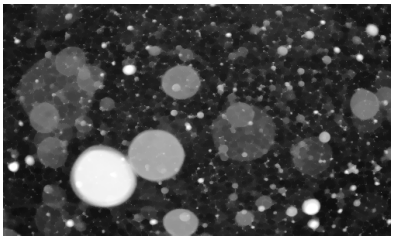
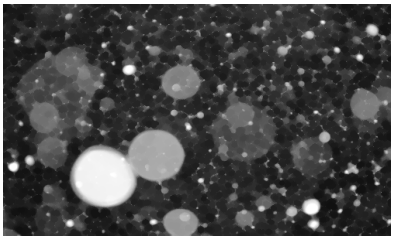
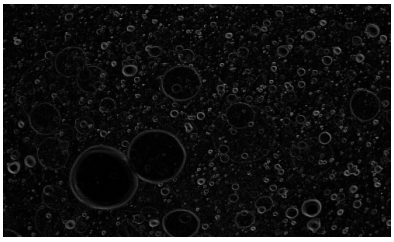
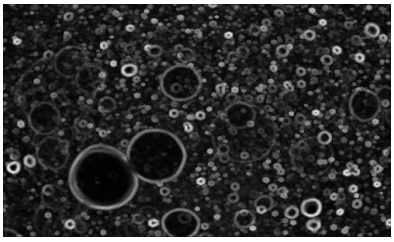
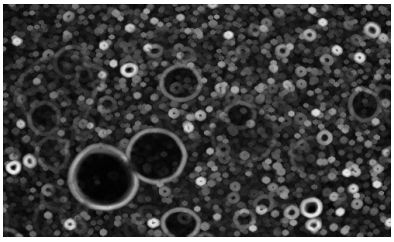
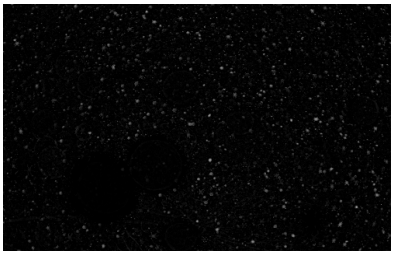
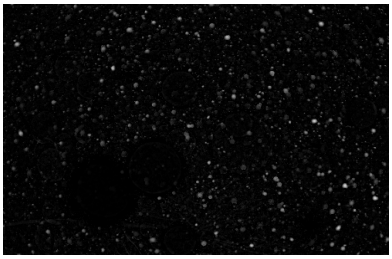
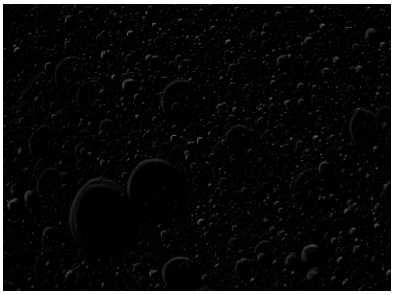
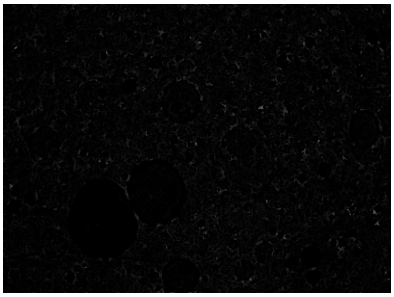
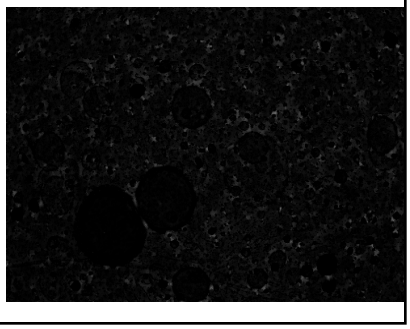
Observation : Detection of top 3 squares.
Original Image had max square pixels of size 15x15 so I took a 13x13 square kernel and applied erosion operation followed by the dilation operation.

Data 1



Taking Structuring element as Elliptical kernel

For 2x2 Elliptical	For 5x5 Elliptical	For 7x7 Elliptical
EROSION 		
DILATION 		
OPENING 		
CLOSING		

		
GRADIENT 		
TOPHAT 		
BLACKHAT 		

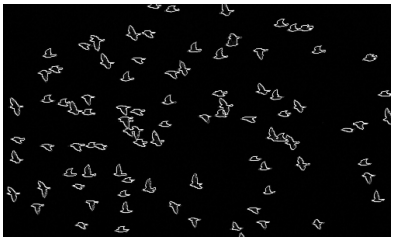
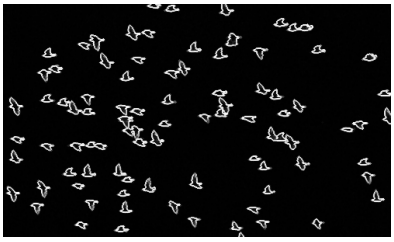
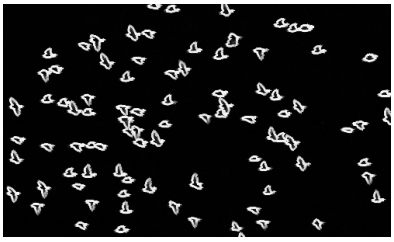
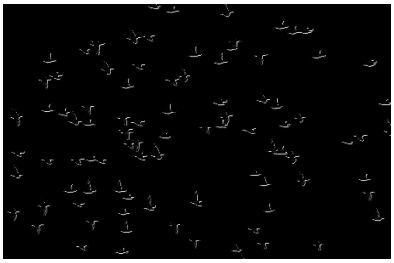
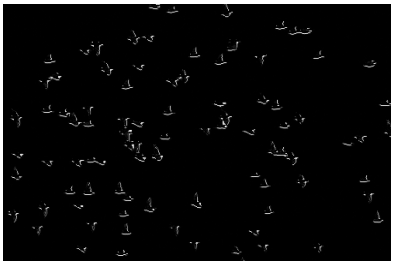
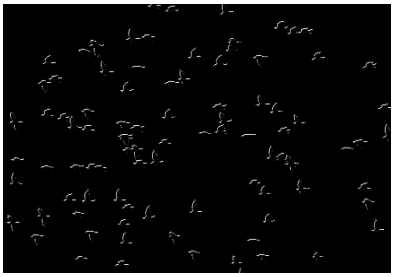
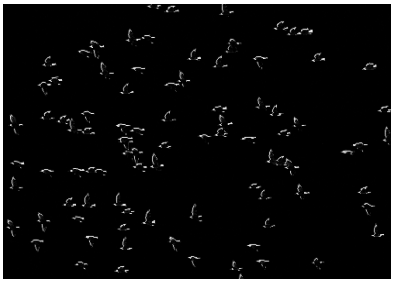
Observations:

After Experimenting with different types of Structural Elements, I found Elliptical S.E. working best for the Image 1.

Reason : Concluding all structuring elements at the end of Image 5.

Data 2

Elliptical 2x2	Cross 3x3	Cross 4x4
EROSION 		
DILATION 		
OPENING 		

CLOSING 		
GRADIENT 		
TOPHAT 		
BLACKHAT 		

Observations:

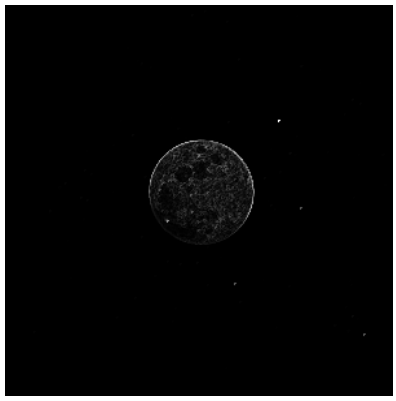
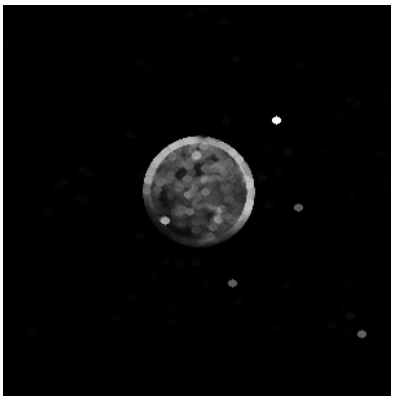
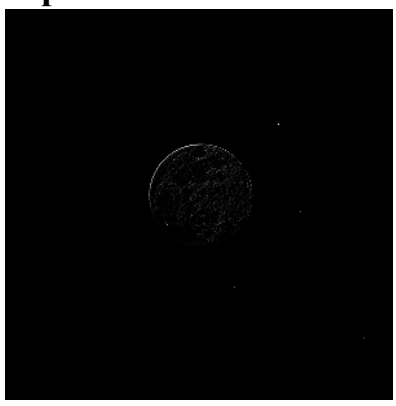
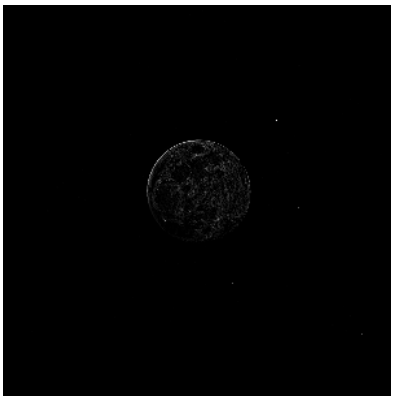
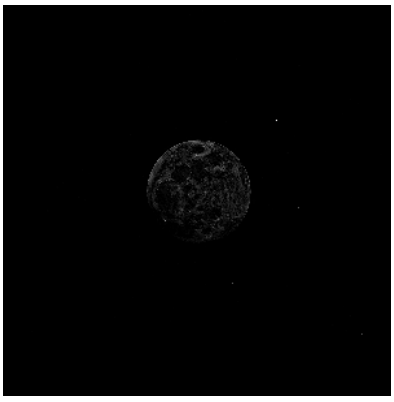
Interesting Observation : As I performed erosion on the above image, the result output was more like dilation (i.e. boundary enhanced) and opposite when operated with dilation, the image objects boundary got shrinked.

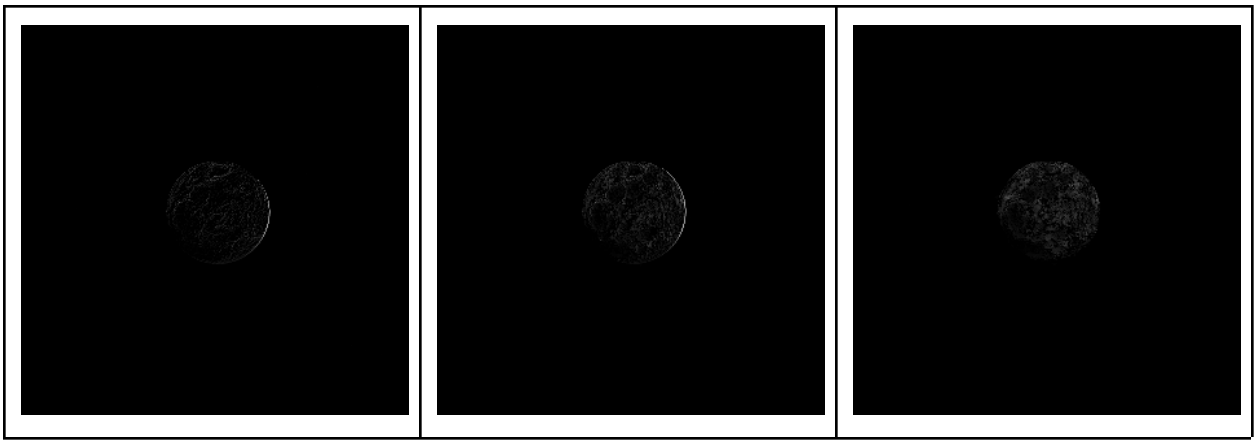
Reason : Due to Complement Nature of the given Input Image

Data 3

Elliptical 2x2	**Elliptical 4x4**	**Elliptical 7x7**
Erosion		

		
Dilation 		
Opening 		

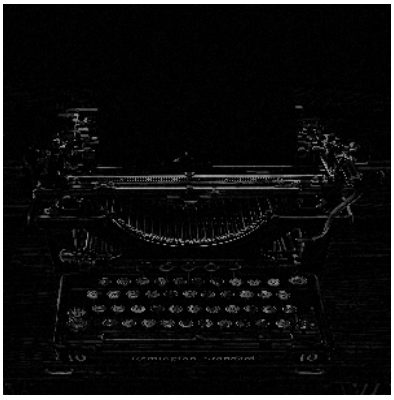
Closing 		
Gradient 		
Tophat 		
Blackhat		



Observations:
After experimenting with other S.E. I found Elliptical Structural Element produced good results for Image 3.

Data 4

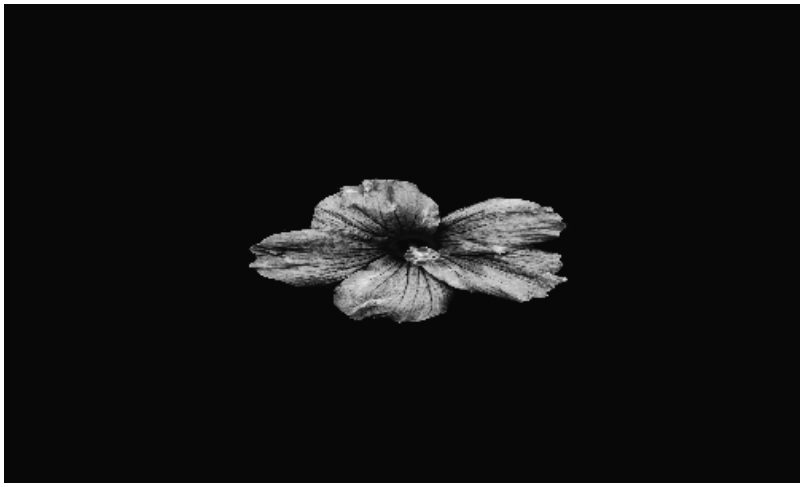
RECT 2X2	RECT 3X3	RECT 5X5
EROSION 		
DILATION 		
OPENING 		

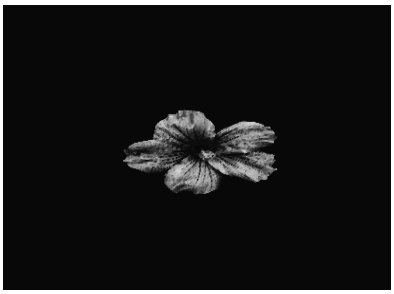
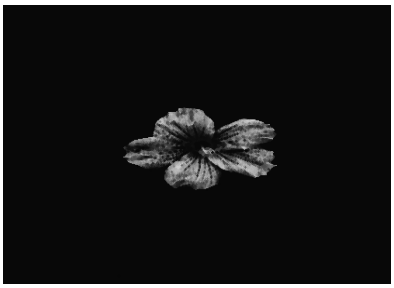
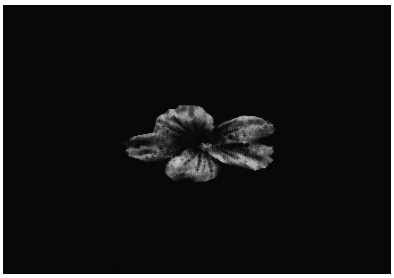
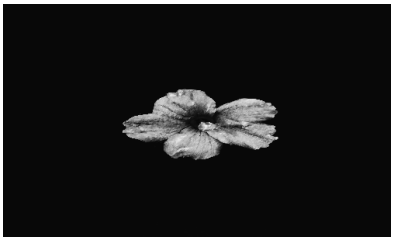
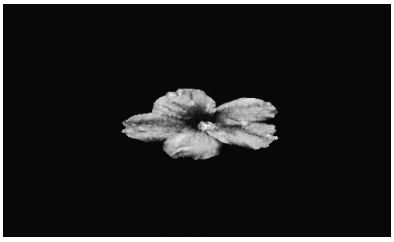
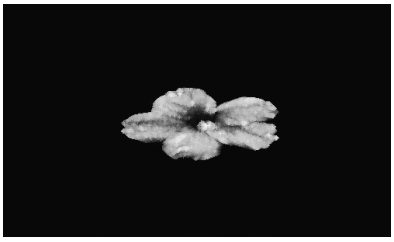
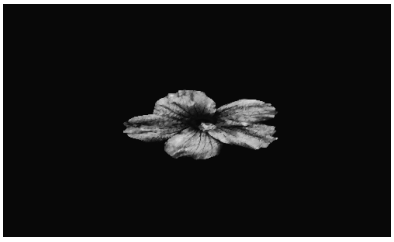
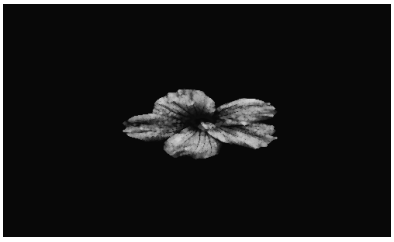
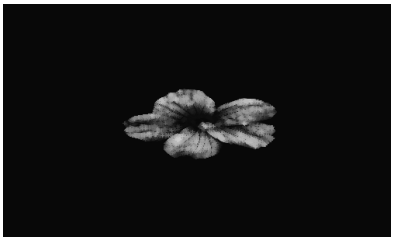
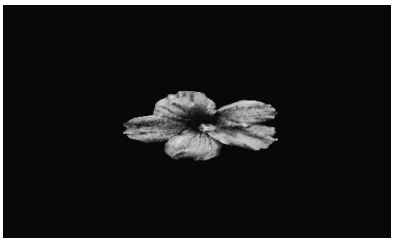
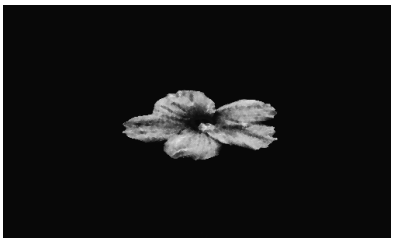
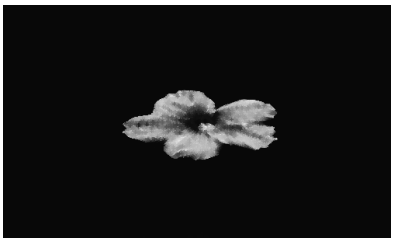
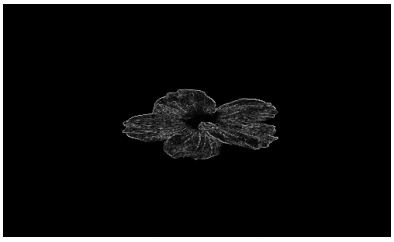
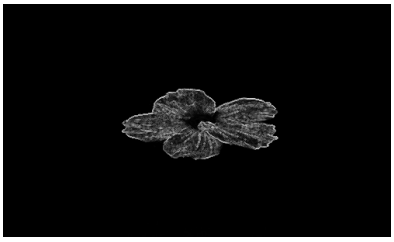
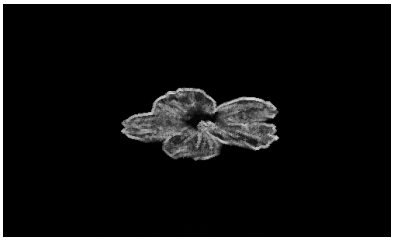
CLOSING 		
GRADIENT 		
TOPHAT 		

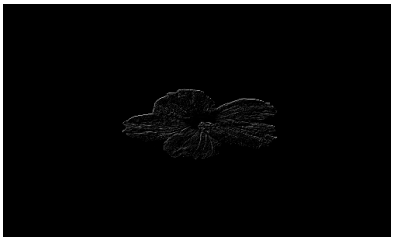
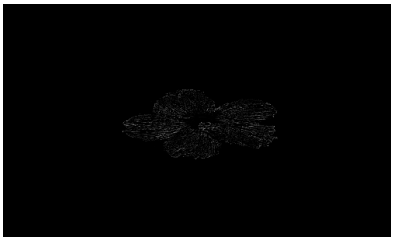
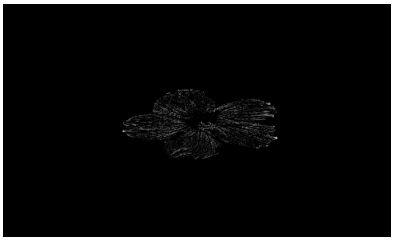
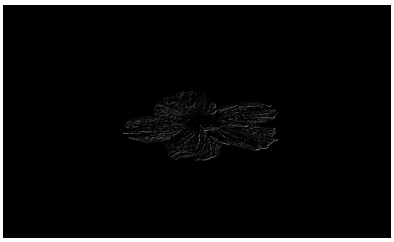
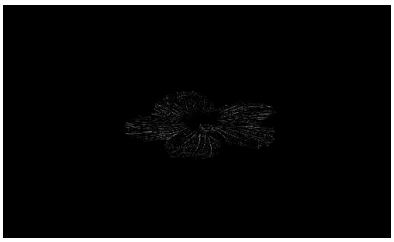
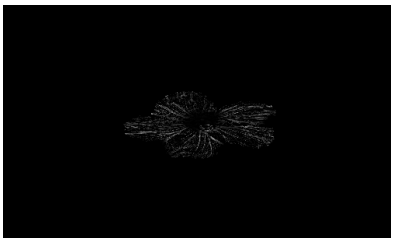
BLACKHAT 		

Observations:
After experimenting with other S.E. I found Rectangular Structural Element produced good results for Image 4.

Data 5



CROSS 2X2	CROSS 3X3	CROSS 5X5
EROSION 		
DILATION 		
OPENING 		
CLOSING 		
GRADIENT 		

TOPHAT 		
BLACKHAT 		

Observations:

After experimenting with other S.E. I found Cross Structural Element produced good results for Image 4.

Overall Conclusion:

Choosing the right structural Element plays a vital role in morphological operations.
After so many experiments, I found if we want some operation on circular object use of circular or elliptical S.E. produces good results.
If image objects are square or rectangular shaped, the choice of rectangular S.E. will be a good choice.

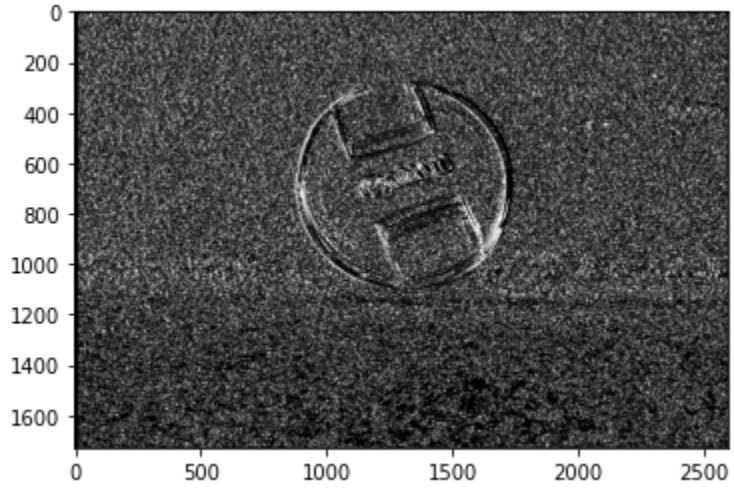
Choice over size of S.E. will also be another hyper parameter which plays another important role. It depends upon the business logic of which size we should opt for S.E.

Example : As per image example taken in class, size of 13x13 will be a good choice.

5] Stereo Vision

Left Image **Right Image**

Using inbuilt function
StereoBM_create()



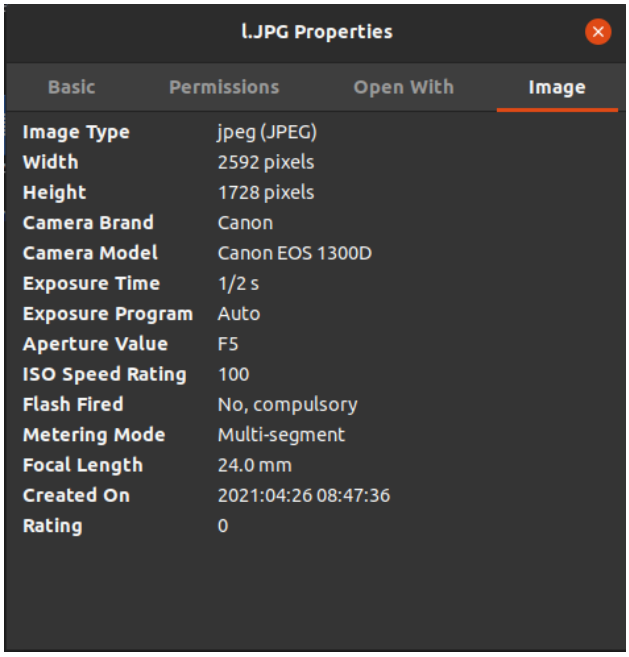
Disparity Map of the Object

Device configurations and Images configurations

Baseline = 1mm

Focal length = 24 mm

Depth = 1 * 24/(disparity)



Disparity Map

```
[[-16 -16 -16 ... -16 -16 -16]
 [-16 -16 -16 ... -16 -16 -16]
 [-16 -16 -16 ...   0 -16 -16]
 ...
 [-16 -16 -16 ... -16 -16 -16]
 [-16 -16 -16 ... -16 -16 -16]
 [-16 -16 -16 ... -16 -16 -16]]
```

Depth map

```
array([[-1.5       , -1.5       , -1.5       , ..., -1.5       ,
        -1.5       , -1.5       ],
       [-1.5       , -1.5       , -1.5       , ..., -1.5       ,
        -1.5       , -1.5       ],
       [-1.5       , -1.5       , -1.5       , ...,  0.32876712,
        -1.5       , -1.5       ],
       ...,
       [-1.5       , -1.5       , -1.5       , ..., -1.5       ,
        -1.5       , -1.5       ],
       [-1.5       , -1.5       , -1.5       , ..., -1.5       ,
        -1.5       , -1.5       ],
       [-1.5       , -1.5       , -1.5       , ..., -1.5       ,
        -1.5       , -1.5       ]])
```

Results:
StereoBM_create() takes two parameters :
i) NumberofDisparity : It increases the search space from default as 0 to NumberofDisparity

ii) BlockSize : Smaller the block size , accurate is the result while larger the block size , less accurate result.

6] Hough Circle

Using Canny Detector for edge detection

v/steps is a hyper parameter obtained the step size for the hough space to define the centers and radii of the circles calculated (visually through cone with various values of 'r')
x, y = center coordinates of circle
r = radius of circle

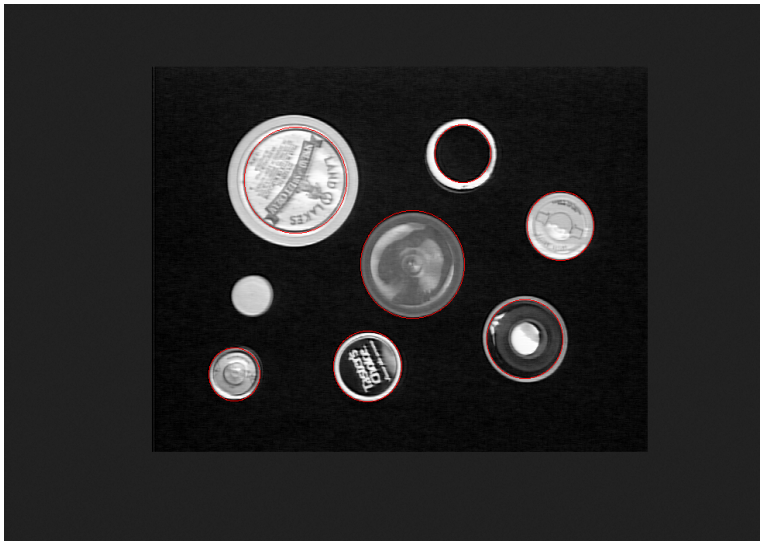
Input Data 1

min radius=30,
max radius=120,
steps=100

Output Format:
v/steps, x, y, r

```
0.69 592 186 36
0.68 718 276 43
0.63 469 451 44
0.62 297 461 33
0.62 376 219 66
0.62 672 417 49
```

0.61 527 324 67



Input Data 2

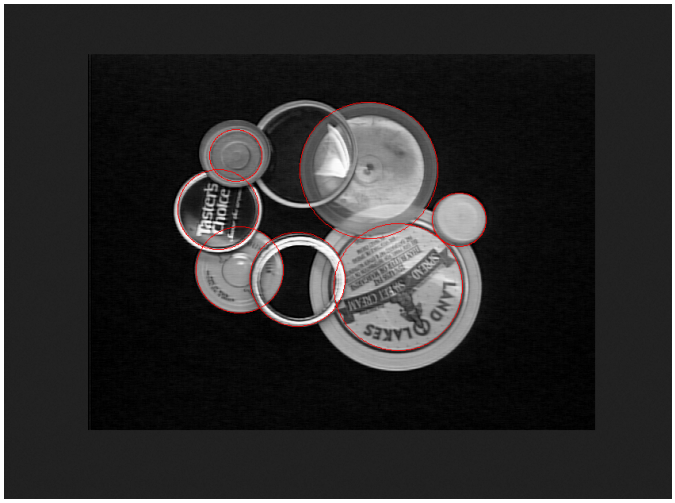
v/steps, x, y, r

0.63 567 190 68
0.59 172 443 39
0.58 319 360 33
0.57 432 311 43
0.53 128 283 45
0.52 376 151 53
0.44 556 415 57
0.43 233 214 31
0.42 285 427 30
0.42 301 168 46
0.41 186 212 30
0.41 242 410 37
0.41 440 220 41

Input Data 3

v/steps, x, y, r

```
0.58 270 262 51
0.57 297 339 55
0.55 370 351 60
0.55 497 361 81
0.54 292 193 33
0.54 574 275 34
0.48 460 212 87
```



References

https://www.pyimagesearch.com/2018/12/17/image-stitching-with-opencv-and-python/
Theory Reference : https://www.youtube.com/watch?v=Ltqt24SQQoI
Theory Reference : https://www.youtube.com/watch?v=8aNOzgIbaeA&t=120s
Other Reference : OpenCV documentations