NOTE: USE OPENCV FOR ALL THE QUESTIONS and GIVE EXPLANATIONS FOR WHAT METHOD/TECHNIQUE/APPROACH USED

ATTACH THE CODE AND THE OUTPUT IMAGE FOR ALL THE TASKS AND THEN SUBMIT

1. Given a random image 'img' complete the following task

   Find the centroid of this figure by writing the Python OpenCV code.

   Size of image:
   0< width < 1000 ...in pixels
   0< height < 1000 ...in pixels

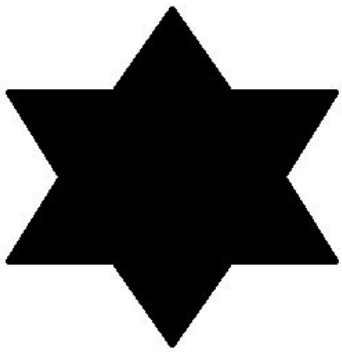
   Input:
   Input image 'img'

   Output:
   The centroid coordinates of the object in the terminal.

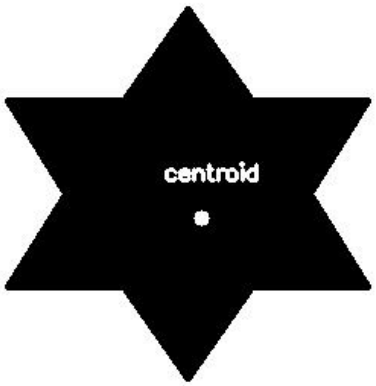
NOTE: The image for this problems statement is given as an attachment

A1.:

- I read the image, Converted it to grayscale and then used threshold function to convert it to binary so i can get a mask for the object
- I then used the opencv moments function to calculate the moment of the blob
- Then i use the formula to calculate the centroid(given here:https://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=moments#moments)
- The formula calculates moments and then finds the mass centre by taking the ratio of the moments.
- We then print the centroid
- As a sanity check and for visualization i plotted the centroid onto the image(this is commented. As the output of problem did not state to plot it but just give centroid coordinates)



Original Image

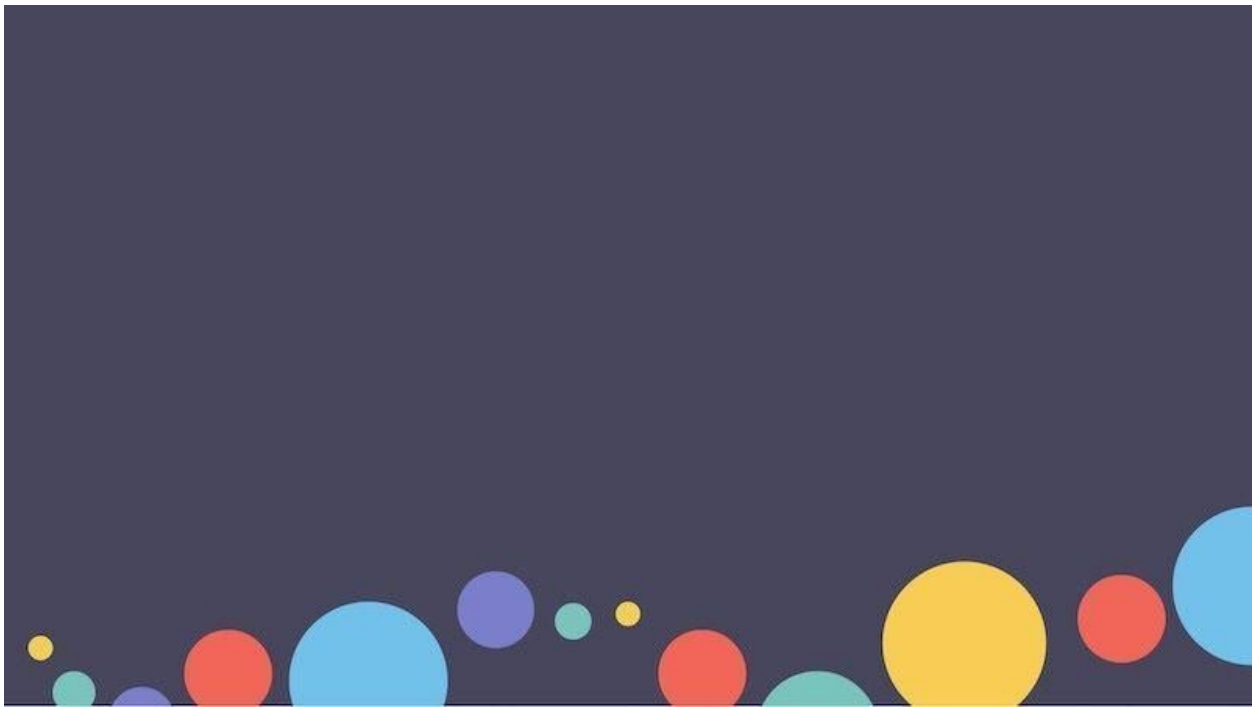


Centroid Output

Code for 1st SubTask:(Formatting of all the code is janky as it cant fit in one page.Refer to .py for better formatted code)

```
import cv2
import numpy as np

def main():
    img=cv2.imread("Task 1.png",-1)
    # cv2.imshow("Original Image",img)
#display original image(uncomment to view)
    # cv2.waitKey(0) & 0xFF
    grey_img=cv2.imread("Task 1.png",0)                                    #load
image in grayscale
    # img=np.zeros([256,256,3],np.uint8)                                   #for
testing with a circle
    # img=cv2.circle(img,(128,128),100,(255,255,255),-1)
    _,th=cv2.threshold(grey_img,127,255,0)
#convert to binary so we can detect shape
    M=cv2.moments(th)

    x = int(M["m10"]/M["m00"])                                             #find
centroid by formula of moments
    y = int(M["m01"]/M["m00"])
    print("Centroid: ")
    print("X = "+str(x)+", Y = "+str(y))

    cv2.circle(img, (x, y), 5, (255, 255, 255), -1)                       #label
for visualisation. please uncomment to view
    cv2.putText(img, "centroid", (x - 25, y - 25),cv2.FONT_HERSHEY_SIMPLEX,
0.5, (255, 255, 255), 2)

    # cv2.circle(img, (x, y), 5, (0, 0, 0), -1)                            #for
circle(as circle is white so label should be black)
    # cv2.putText(img, "centroid", (x - 25, y - 25),cv2.FONT_HERSHEY_SIMPLEX,
0.5, (0, 0, 0), 2)

    # cv2.imshow("Centroid Image", img)                                     #to
show image(for visualisation) please uncomemnt to see
    # cv2.waitKey(0) & 0xFF
    # cv2.destroyAllWindows()

if __name__ == "__main__":
    main()
```

2. Convert the given images to grayscale and binary and state the significance of converting the images(what are the advantages of converting it in computer vision)?



Original image

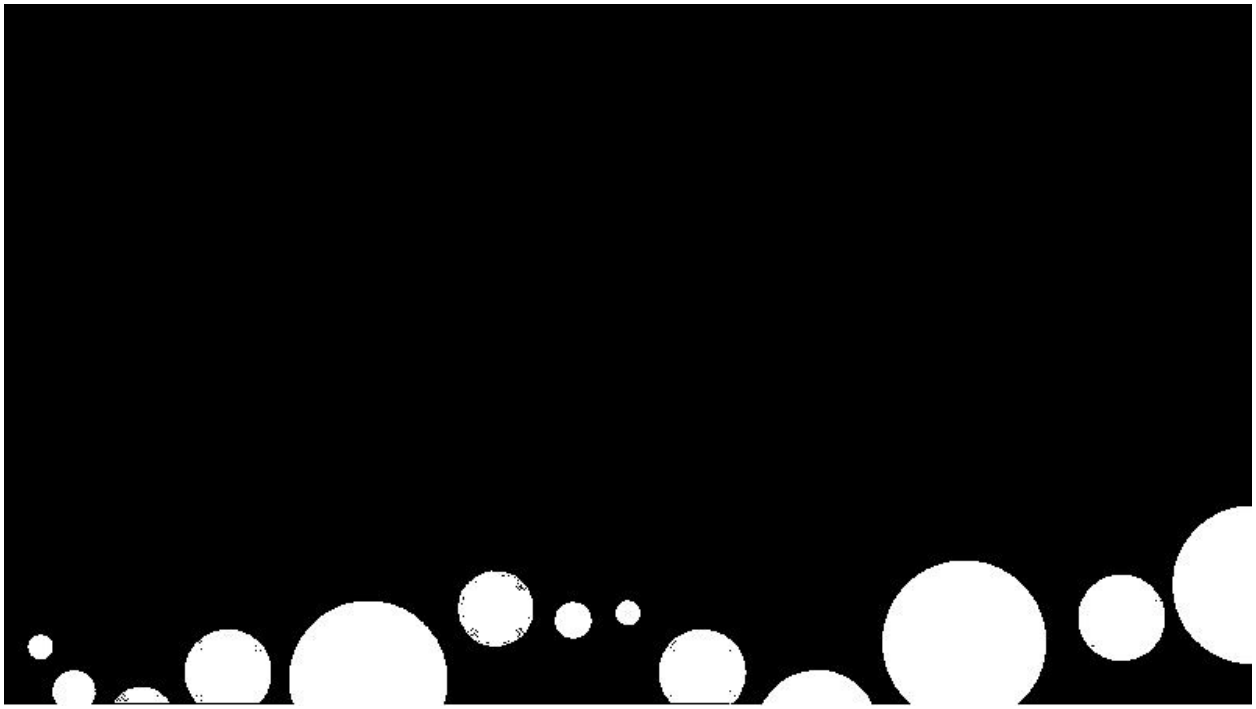
A2.:

- I read the image as it is unchanged
- Then i use opencv cvtColor function to change the color map from BGR to Grayscale
- Then use opencv threshold function to threshold below 127 to white and above 127 to black making it binary.

Advantages of converting to grayscale and Binary:

- When we convert from BGR to grayscale we now have only 1 channel to work with and so thresholding it is easy. Also as the luminosity method of converting that opencv uses to convert gives the circles with vibrant colours higher brightness and so thresholding becomes very easy to extract the circles
- When we threshold it to binary all the dark areas turn to black and the bright areas to white. This gives us a map for the circles and now we can apply the map to the original image and extract only the circles. This is especially useful in the 4th subtask as you will see.

Greyscale Image



Binary Image

Code for 2nd SubTask:

```
import cv2
import numpy as np

def main():
    img=cv2.imread("Task 2.jpg",-1)
#read and show original image
    cv2.imshow("Original image",img)
    cv2.waitKey(0) & 0xFF
    grey=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
#convert to greyscale and show
    cv2.imshow("GreyScale Image",grey)
    cv2.waitKey(0) & 0xFF
    _,binary=cv2.threshold(grey,127,255,0)
#convert to binary and show
    cv2.imshow("Binary Image",binary)
    cv2.waitKey(0) & 0xFF
    cv2.destroyAllWindows()
if __name__=="__main__":
    main()
```

3. Using OpenCV methods find out the type of noise in the given image and try your best to denoise it using various denoising techniques. Write about which methods you tried and which method was the best?

Original Image

A3.: Type of noise:

- I read the image
- On going through the Image processing book by Gonzales and Wintz(http://web.ipac.caltech.edu/staff/fmasci/home/astro_refs/Digital_Image_Processing_2ndEd.pdf)- The chapter on image filtering, I found that to see what type of noise is dominant in an image. You have to cut out a small uniformly coloured part of the image and draw its histogram. On analysing the histogram it can be seen that the PDF of the histogram shows the PDF of the added noise.
- So I did just that. And the PDF of the resulting histogram was Gaussian in nature.

- As a sanity check I found another strip and repeated the process. That too returned a Gaussian PDF. So I concluded that the dominant noise present in the image was Gaussian noise.

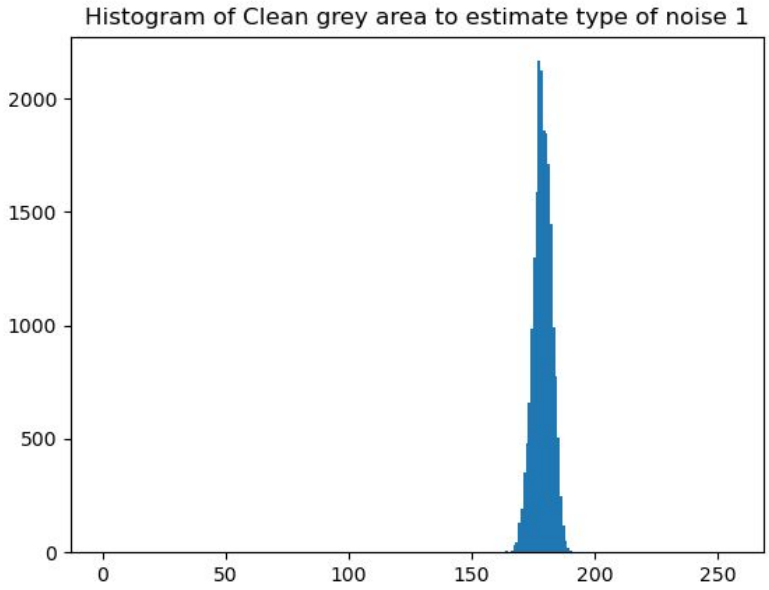
  Types of filters used:

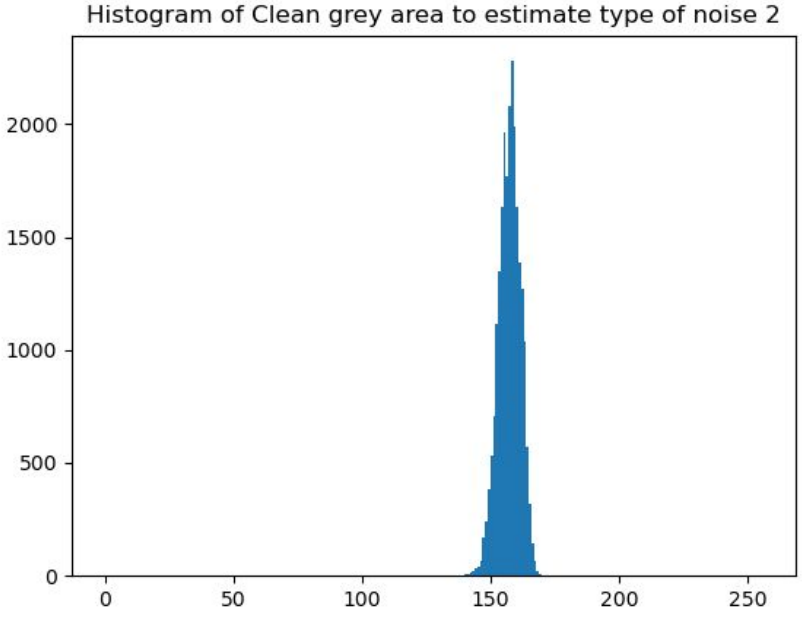
- I used 7 fiters to denoise the image: Median, Bilateral, Gaussian, Non Local, Arithmetic Mean, Geometric Mean and Harmonic Mean Filter
- I used their standard opencv definitions(Harmonic one had to be inverse of the image box filtered's inverse) and used the general definition of PSNR and RMSE to calculate the scores.
- I then cut out a small corner out of each image and stitched them together for visually comparing each filter's effect on the noise
- On analysing the scores, Bilateral Filter was the best as it removed maximum noise while retaining most of the detail.
- But on visually analysing, Non local Filter removed the maximum noise. On analysing it's output, I noticed that there was a massive loss in detail and the picture was cartoonish.
- As we all know that while denoising the balance between the amount of noise and detail needs to be maintained as if low detail picture is fed into detection algorithms or edge detection algorithm, the result will be fuzzy and inaccurate and our application might not work as intended,
- **Hence in terms of pure denoising: Non Local Filter is the best but there is a significant loss in detail**
- **But keeping most of the detail, maximum noise removal was performed by Bilateral Filter.**

(The images uploaded in pdf are squished as I had to show the comparison on the same page. If you want a full size image then feel free to add an imwrite function to the program and run it to save the images.)

Output by Bilateral Filter

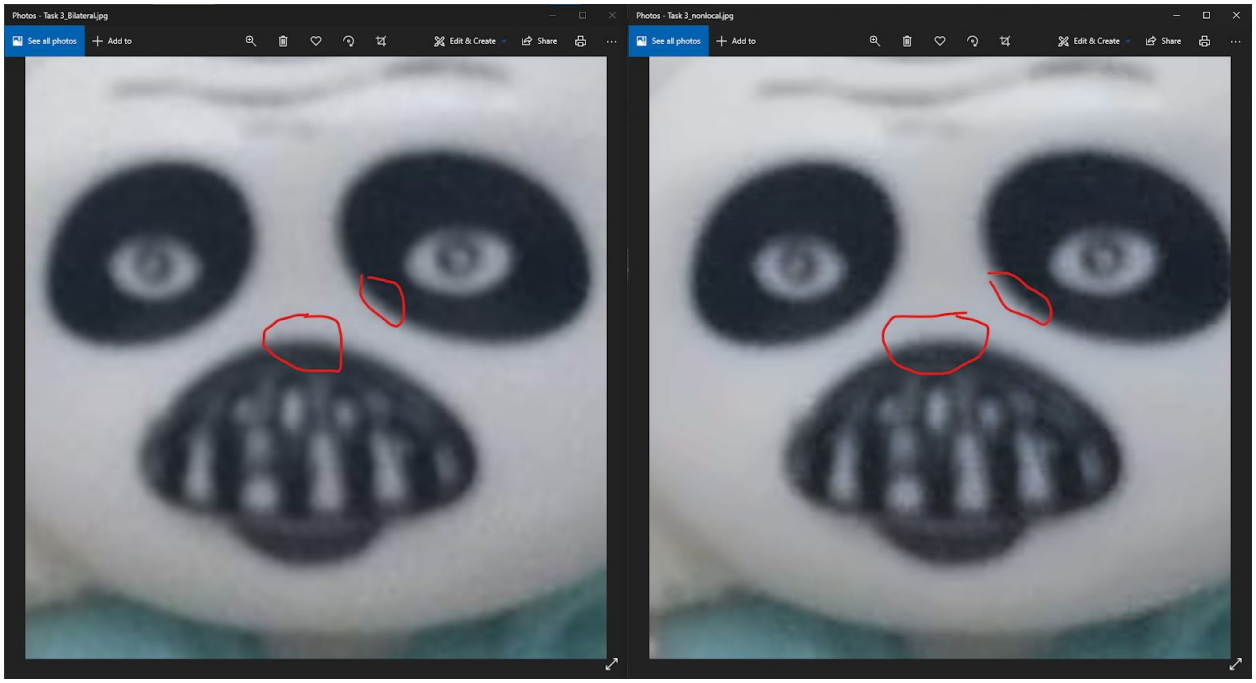
Output by Non Local Means Filter





Both Histograms of uniform area give gaussian distribution

Comparison of All Images(Might be unclear, save from .py file using an imwrite function)



On zooming into the face it can be seen that the white pixels and black pixels in the toy's face have been mashed together in non-local image(right). While in Bilateral(left), they are still separable.
If the task was to use face recognition after denoising, then the algorithm will perform poorly on non local due to less detail

(It is not visible in this image as the image got squashed in google docs. Please extract full size image to view properly)

Code for 3rd SubTask:

```
import cv2
from scipy.ndimage.filters import uniform_filter
from scipy.ndimage.measurements import variance
from math import log10, sqrt
import numpy as np
from matplotlib import pyplot as plt

def median_filter(img):
#median filter
    return cv2.medianBlur(img, 3)

def PSNR(original, compressed):
#calculate RMSE and PSNR
    mse=np.mean((original-compressed)**2)
    if(mse==0):
        return 100
    max_pixel=255.0
    psnr=20*log10(max_pixel/sqrt(mse))
    return (sqrt(mse),psnr)

def main():
    img=cv2.imread("Task 3.jpg")
#read and show image to user
    cv2.imshow("Original Image",img)
    cv2.waitKey() & 0xFF

    noisy=img[861:924,204:308,:]
#cut out a small uniform part and draw its histogram. Shows Gaussian
distribution
    plt.title("Histogram of Clean grey area to estimate type of noise 1")
    plt.hist(noisy.ravel(),256,[0,256])
    plt.show()

    noisy=img[704:807,1108:1182,:]
#for sanity check, cut out another uniform part and check. Shows Gaussian PDF
    plt.title("Histogram of Clean grey area to estimate type of noise 2")
    plt.hist(noisy.ravel(),256,[0,256])
    plt.show()

    print("\nMETRICS USED:\nPSNR:HIGHER IS BETTER \nRMSE:LOWER IS BETTER\n")
#higher PSNR(calculates similarity with original image) and lower RMSE means
better denoised image
```

```
    sp_img=median_filter(img)
#Apply Median Filter and print results and show image
    rmse_sp,psnr_sp=PSNR(sp_img,img)
    print("Median Filter PSNR : "+str(psnr_sp))
#Median Filter PSNR : 42.74279602338066
    print("Median Filter RMSE : "+str(rmse_sp)+"\n")
#Median Filter RMSE : 1.8595179683597514
    cv2.imshow("Median Filter Test", sp_img)
    cv2.waitKey() & 0xFF

    bilateral_img=cv2.bilateralFilter(img,3,100,100)
#Apply Bilateral Filter and print results and show image(BEST PSNR AND RMSE
RATING)
    rmse_bilateral,psnr_bilateral=PSNR(bilateral_img,img)
    print("Bilateral Filter PSNR : "+str(psnr_bilateral))
#Bilateral Filter PSNR : 45.270338974238484
    print("Bilateral Filter RMSE : "+str(rmse_bilateral)+"\n")
#Bilateral Filter RMSE : 1.3900269859615635
    cv2.imshow("Bilateral Filter Test", bilateral_img)
    cv2.waitKey() & 0xFF

    gaussian_img=cv2.GaussianBlur(img,(3,3),0)
#Apply Gaussian Filter and print results and show image
    rmse_gaussian,psnr_gaussian=PSNR(gaussian_img,img)
    print("Gaussian Filter PSNR : "+str(psnr_gaussian))
#Gaussian Filter PSNR : 43.963393881862594
    print("Gaussian Filter RMSE : "+str(rmse_gaussian)+"\n")
#Gaussian Filter RMSE : 1.6157363154753417
    cv2.imshow("Gaussian Filter Test", gaussian_img)
    cv2.waitKey() & 0xFF

    nonlocal_img=cv2.fastNlMeansDenoisingColored(img,h=1)
#Apply Non Local Filter and print results and show image
    rmse_nonlocal,psnr_nonlocal=PSNR(nonlocal_img,img)
    print("Non-Local Means Filter PSNR : "+str(psnr_nonlocal))
#Non-Local Means Filter PSNR : 44.68882223740083
    print("Non-Local Means Filter RMSE : "+str(rmse_nonlocal)+"\n")
#Non-Local Means Filter RMSE : 1.4862746433745664
    cv2.imshow("Non-Local Means Filter Test", nonlocal_img)
    cv2.waitKey() & 0xFF

    arithmean = cv2.boxFilter(img, -1, (3, 3))
#Apply Arithmetic Mean Filter and print results and show image
```

```
    rmse_arithmean,psnr_arithmean=PSNR(arithmean,img)
    print("Arithmetical Mean Filter PSNR : "+str(psnr_arithmean))
#Arithmetical Mean Filter PSNR : 41.819607175768745
    print("Arithmetical Mean Filter RMSE : "+str(rmse_arithmean)+"\n")
#Arithmetical Mean Filter RMSE : 2.068044223872488
    cv2.imshow("Arithmetical Mean Filter Test", arithmean)
    cv2.waitKey() & 0xFF

    log=np.float32(np.log(img))
#Apply Geometric Mean Filter and print results and show image
    geomean2 = np.uint8(np.exp(cv2.boxFilter(log, -1, (3, 3))))
    rmse_geomean,psnr_geomean=PSNR(geomean2,img)
    print("Geometrical Mean Filter PSNR : "+str(psnr_geomean))
#Geometrical Mean Filter PSNR : 41.47952729296938
    print("Geometrical Mean Filter RMSE : "+str(rmse_geomean)+"\n")
#Geometrical Mean Filter RMSE : 2.150620671069943
    cv2.imshow("Geometrical Mean Filter Test", geomean2)
    cv2.waitKey() & 0xFF

    harmmean = 1/cv2.boxFilter(1/img, -1, (3, 3))
#Apply Harmonic Mean Filter and print results and show image
    harmmean=np.uint8(harmmean)
    rmse_harmmean,psnr_harmmean=PSNR(harmmean,img)
    print("Harmonic Mean Filter PSNR : "+str(psnr_harmmean))
#Harmonic Mean Filter PSNR : 41.336710973538416
    print("Harmonic Mean Filter RMSE : "+str(rmse_harmmean)+"\n")
#Harmonic Mean Filter RMSE : 2.1862742101289583
    cv2.imshow("Harmonic Mean Filter Test", harmmean)
    cv2.waitKey() & 0xFF


final=np.concatenate((img[743:,996:,:],sp_img[743:,996:,:],gaussian_img[743:,99
6:,:],nonlocal_img[743:,996:,:],arithmean[743:,996:,:],geomean2[743:,996:,:],ha
rmmean[743:,996:,:],bilateral_img[743:,996:,:]),axis=1)
    cv2.imshow("Comparison of All", final)
#stitch a small corner of all images together for visual check for noise
    cv2.waitKey() & 0xFF

    #On checking visually it was found that the best denoising techique was Non
Local filter. But on analysing its output image, I saw that it has a
significant loss in detail
    #This is not what we want. By looking at all the pictures indivisually and
analysing the PSNR and RMSE scores it is clear that the best filter is
Bilateral Filter
```

```
    #This is because it manages to remove a lot of noise but still has some
noise but there is no significant loss in detail. So by keeping the balance
Bilateral filter works the best

    #In terms of pure denoising: Non local filter is best as the noise is very
less.
    #In terms of denoising as well as retaining detail: Bilateral Filter works
the best(it removes a lot of noise while retaining all the detail)

    #We dont want loss in detail as if we take the low detail picture after
this preprocessing step, the results of edge detection and all other algorithms
will be fuzzy and inaccurate.
    #Hence balance in detail and denoising should be maintained

    cv2.destroyAllWindows()


if __name__=="__main__":
    main()
```

4. How would you isolate the hand from the background in the given image? Explain the advantages and disadvantages of your method of isolation?

Make sure you use the image which is given below the respective question.

A4.:

- I tried to use many methods to isolate it.
- First I tried a basic mask with colors handpicked. An obvious failure as the hand is both in dark and light conditions.
- Then I tried k-means clustering to cluster the hand, camera and background. That failed as some parts of background were also in k means
- Then I tried histogram backprop. But that is based on color segmentation so again it failed as the hand is in multiple lighting conditions(both dark and light)
- Finally i found a code in stack overflow that gave a good mask of the hand(https://stackoverflow.com/questions/8753833/exact-skin-color-hsv-range#:~:text=The%20skin%20in%20channel%20H,for%20small%20values%20of%20V.)
- I used that but there will still noise in the mask so i manually removed it by setting mask to 0 in certain soisy locations

- Then i noticed that there were holes in the hand so i wrote function to make a pixel active if surrounding pixels are active and so the holes were filled
- I applied the mask to the image
- Still there were some false positives and false negatives. I fixed them by applying opening and closing function
- Then i noticed that the camera(with which picture was taken) was parallel to the circular lens so it was a perfect circle
- So i plotted it using pyplot and found the center and radius of the lens and added a black circle in there
- Then i showed the final image

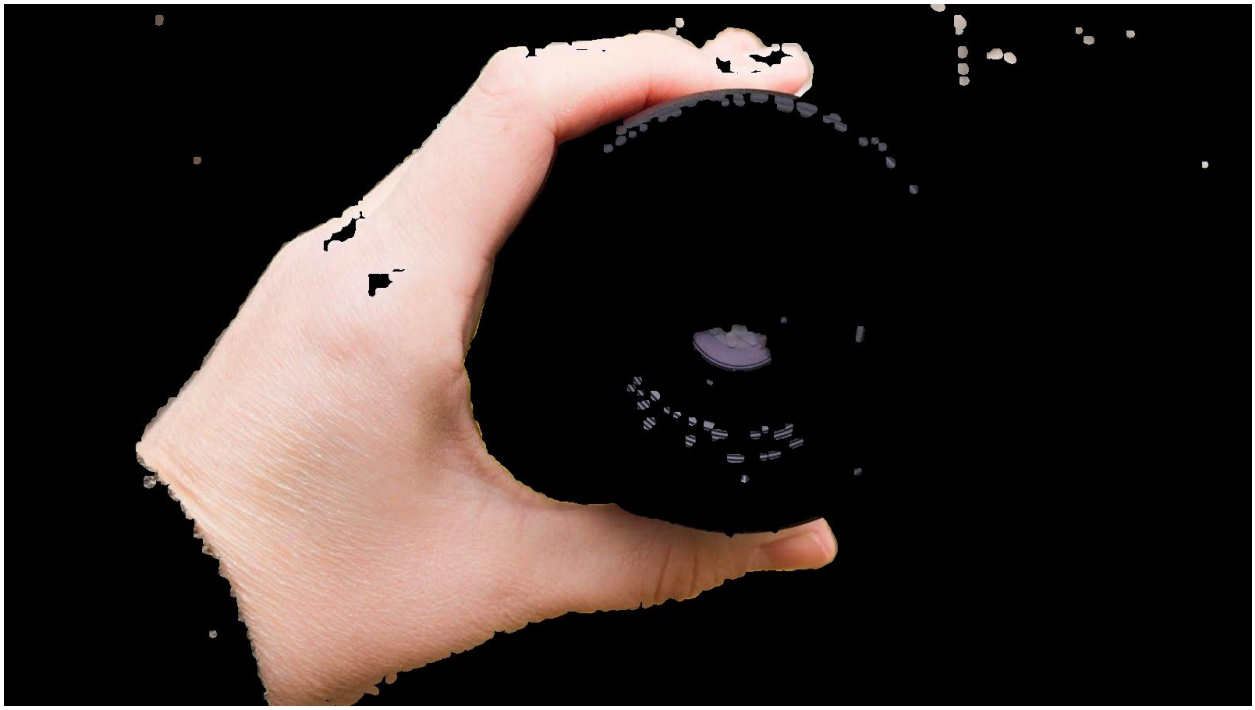
Advantages of method used:

1. As the method used is tailor made for the particular image, it is perfect and nothing except the hand is in the final image. The method achieves the result 100%
2. It does not use any time consuming algorithms like k means so it is very time saving
3. It is efficient and does the job well

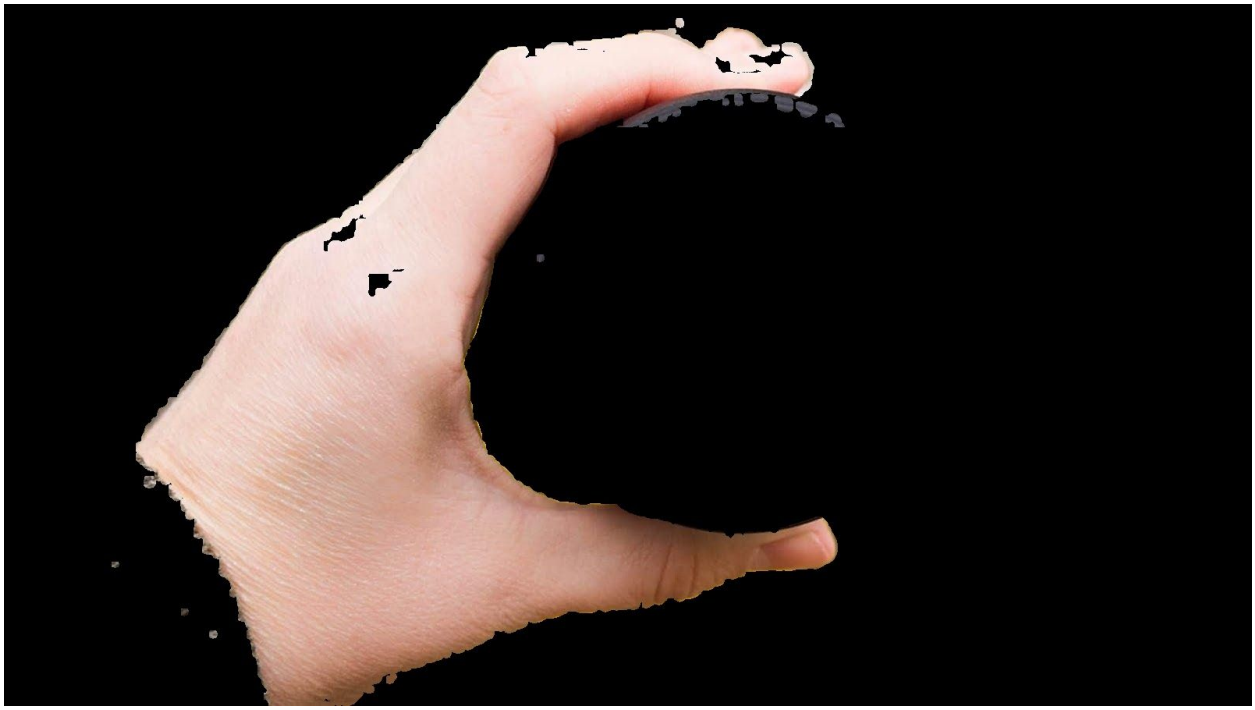
Disadvantages of method:

1. As it has many steps, it cant be generalised for this type of images
2. This only works on this image so it cant be used for any image with a hand holding a camera at different angles
3. It used much of manual work and so cant be used in stuff like hand detection as it cant be automated
4. This will not work for any other shades of skin. This only works to remove that particular shade of skin

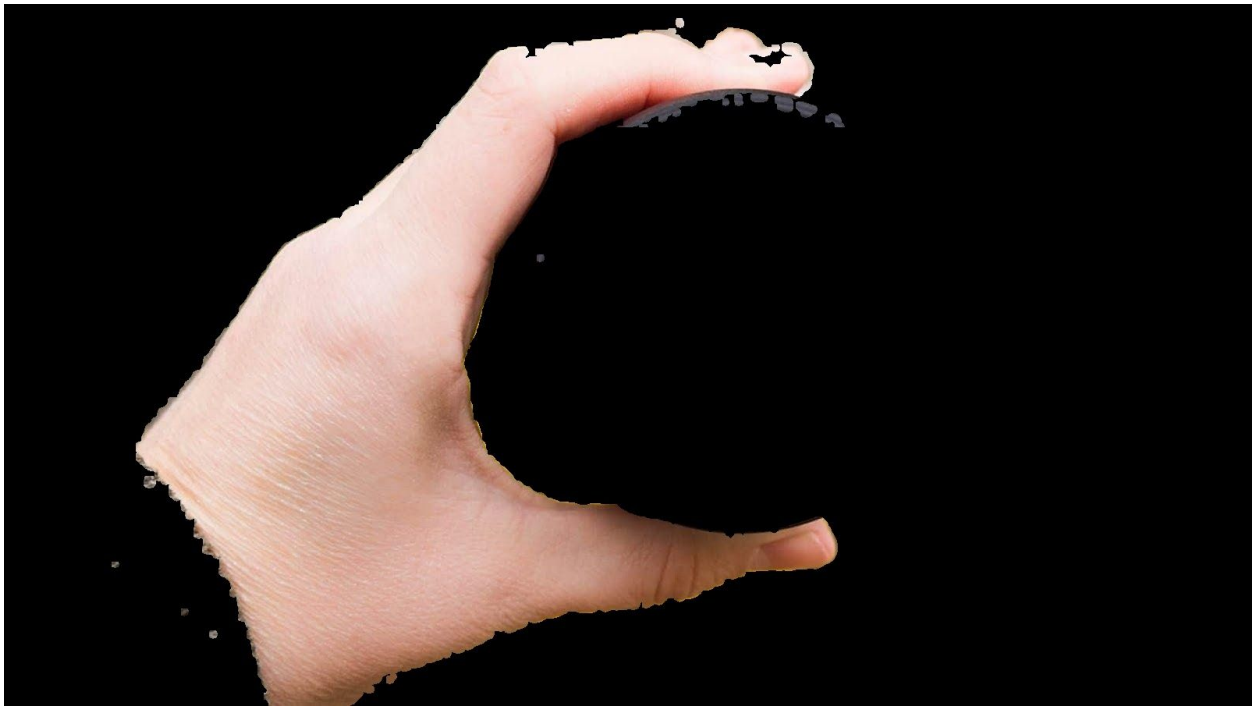
There might be other algorithms that can do a better job and can be automated. Due to the time constraint of 4 days and no prior knowledge in image processing I think that it does a good job. This was all that i could learn from the internet in 4 days. Thank You for letting me do this task as i didnt know about opencv and was not interested in image processing but i think i am interested in it now.



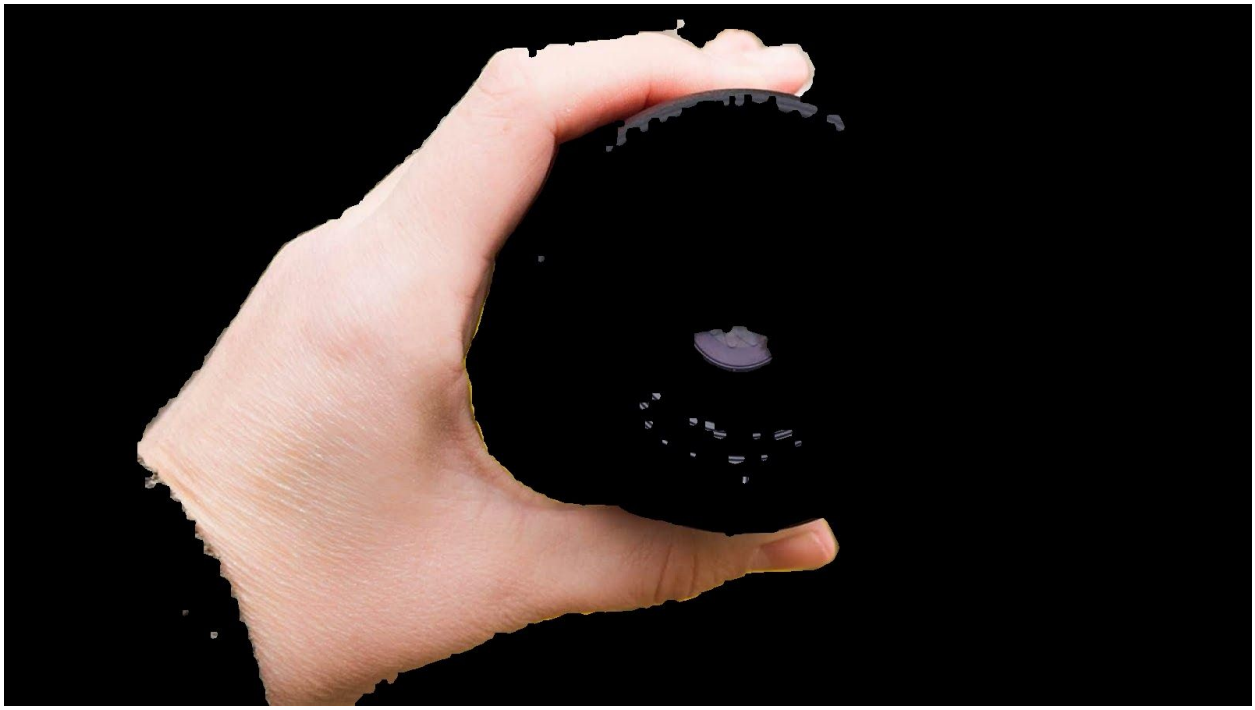
Output without fillhole and noise removal



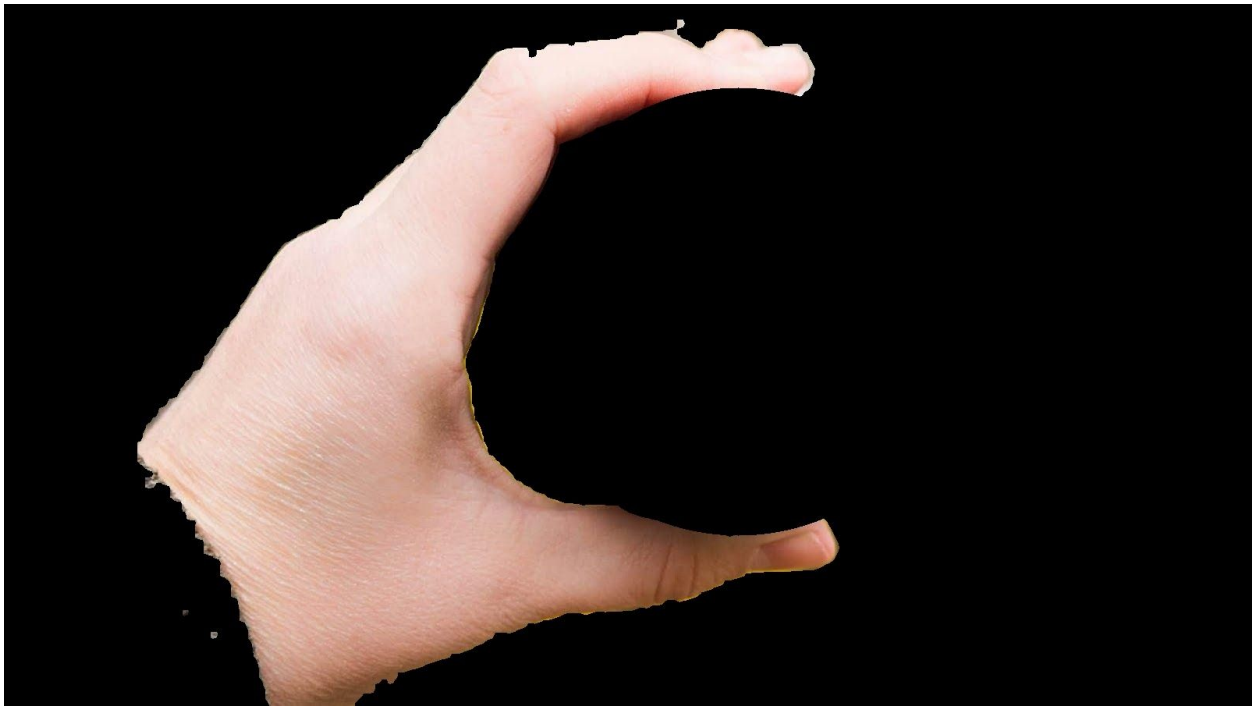
Output after noise removal but without fillhole



Output with fillhole



Output with opening and closing but no black circle



Final Output with all Implementation

Code for 4th SubTask:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

def preprocess(img):                                                    #preprocess
function found at
https://stackoverflow.com/questions/8753833/exact-skin-color-hsv-range#:~:text=
The%20skin%20in%20channel%20H,for%20small%20values%20of%20V.
    kernel=np.ones((5,5),np.uint8)                                      #the function
takes the image, applies a Gaussian Blur of 3x3 for better detection, then uses
a mask specially designed for white human skin
    blur=cv2.GaussianBlur(img,(3,3), 0)                                 #this might
not work for African, African American, Red Indian and people of different
complexion
    hsv=cv2.cvtColor(img,cv2.COLOR_RGB2HSV)                             #it then
blurs the mask further with a median blur of 5x5
    lower_color=np.array([108,23,82])                                   #then it
dilates it so that gaos get filled in
    upper_color=np.array([179,255,255])
    mask=cv2.inRange(hsv,lower_color,upper_color)
    blur=cv2.medianBlur(mask,5)
    kernel=cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(8,8))
    hsv_d=cv2.dilate(blur,kernel)
    hsv_d[:328,:252]=0                                                  #this is
manually added to remove noise and is specific to this image
    hsv_d[:,1081:]=0                                                    #this wont
work for any other image
    opening=cv2.morphologyEx(hsv_d,cv2.MORPH_OPEN,kernel)               #then i use
opening function to remove false positives from the image
    closing=cv2.morphologyEx(opening,cv2.MORPH_CLOSE,kernel)            #then i use
closing on the opening mask to remove false negatives
    return closing                                                      #the function
the returns the final mask

def fillHole(im_in):                                                    #this
function fills up holes in the mask by using the floodfill function
    im_floodfill=im_in.copy()
    h,w=im_in.shape[:2]
    mask=np.zeros((h+2,w+2),np.uint8)
    cv2.floodFill(im_floodfill,mask,(0,0),255);
    im_floodfill_inv=cv2.bitwise_not(im_floodfill)
    im_out=im_in|im_floodfill_inv
```

```
    return im_out

def kmeans(img):                                                    #this was my
try at using a k means clustering algorithm to seperate out the hand. It failed
terribly as the background also was coming into the same cluster
    k=2                                                             #i used
k=2-10 but none clustered the hand as a whole. Hence i commented out its code
in the main function
    img=cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    pixel_values=img.reshape((-1, 3))
    pixel_values=np.float32(pixel_values)
    criteria=(cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100, 0.2)
    _,labels,(centers)=cv2.kmeans(pixel_values, k, None, criteria, 10,
cv2.KMEANS_RANDOM_CENTERS)
    centers=np.uint8(centers)
    labels=labels.flatten()
    segmented_image=centers[labels.flatten()]
    segmented_image=segmented_image.reshape(img.shape)
    masked_image=np.copy(img)
    masked_image=masked_image.reshape((-1, 3))
    cluster=1
    masked_image[labels==cluster]=[0, 0, 0]
    masked_image=masked_image.reshape(img.shape)
    return masked_image

def histogram_backprop(roi,img):                                    #i tried
using histogram backpropagation but as this was based on colour segmentation,
the hand was not being fully seperated
    hsv=cv2.cvtColor(roi,cv2.COLOR_BGR2HSV)                         #i also
commented out this code in main function.
    hsvt=cv2.cvtColor(img,cv2.COLOR_BGR2HSV)
    M=cv2.calcHist([hsv],[0, 1], None, [180, 256], [0, 180, 0, 256] )
    I=cv2.calcHist([hsvt],[0, 1], None, [180, 256], [0, 180, 0, 256] )
    h,s,v=cv2.split(hsvt)
    R=M/I
    B=R[h.ravel(),s.ravel()]
    B=np.minimum(B,1)
    B=B.reshape(hsvt.shape[:2])
    disc=cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(5,5))
    cv2.filter2D(B,-1,disc,B)
    B=np.uint8(B)
    cv2.normalize(B,B,0,255,cv2.NORM_MINMAX)
    _,th=cv2.threshold(B,50,255,0)
    output=cv2.bitwise_and(img,img,mask=th)
```

```
    return output

def main():
    img=cv2.imread('Task 4.jpg')                                        #read
image

    mask=preprocess(img)
#Turns out that the best result was by a code that i found on stack overflow.
So i used this and commenterd out all the other code in main function
    mask=fillHole(mask)                                                 #use
the fillhole function to remove holes in the mask
    output=cv2.bitwise_and(img,img,mask=mask)

    # hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)                        #i
tried basic mask at the start with only hsv colors. An obvious failure
    # mask = cv2.inRange(hsv,(0, 10, 60), (20, 150, 255) )

    # output=kmeans(img)                                                #k
means code commented out(go up to know the reason)
    # output=cv2.cvtColor(output,cv2.COLOR_BGR2GRAY)
    # _,th=cv2.threshold(output,0,255,cv2.THRESH_BINARY)
    # output=cv2.bitwise_and(img,img,mask=th)

    # roi=output[437:777,238:589,:]
#histogram backprop comemnted out
    # output=histogram_backprop(roi,img)

    # output=cv2.circle(output,(941,394),287,(0,0,0),-1)                #as
the lens was parallel to the camera lens at time of capturing it was a perfect
circle so i just calculated the centre and radius
                                                                        #and
put a black circle in. It worked surprisingly well
    cv2.imshow("Output", output)                                        #show
the output
    cv2.waitKey() & 0xFF
    cv2.destroyAllWindows()

if __name__=="__main__":
    main()
```