Wholly owned by UTAR Education Foundation
(Co. No. 578227-M)
DU012(A)

UEMH4334 Machine Vision

Practical 1 Report

2024

Student Name	Ng Yen Keat
Student ID	2202489
Program	MH
Practical Group	P1
Marks	

Table of Contents

0.0 Equipment and Materials

Item Description	*Item category	Quantity estimation (e.g. per set/group of student)
MATLAB (Image Processing Toolbox)	S	1
Python (Anaconda) + OpenCV	S	1
Visual Studio (Visual Studio Code)	S	1
MVTec Halcon	S	1

*Item category	
SP	Sample or specimen
C	Consumable
CH	Chemical
W	Labware, glassware, tool, and components
E	Equipment
S	Software

1.0 Introduction

This lab introduces different preprocessing methodologies to process the image, making them more suitable and helpful information for specific purposes. It also covers proper techniques to capture the specimen using the front lighting method. In addition, by doing this experiment, we can learn how to apply the 'cv2' and 'matplotlib' library to load images, convert colours and utilize the other functions to enhance image presentation. The practical lab teaches us how to take clear and concise photos by adjusting the focal length and the aperture size when capturing the specimen at a certain distance from the lens.

2.0 Theory/Hypothesis

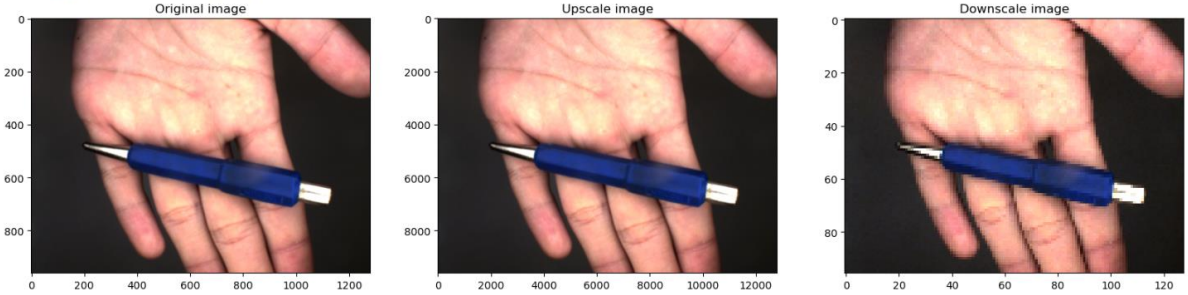
In this experiment, resizing should change the image dimensions and reduce its quality when converting to lower pixels, producing blurriness. Colour-space conversion method will convert the representation type of colour to another, such as converting original photos from BGR either to RGB (Red, Green, Blue) or to GRAY (only Black and White) or HSV (Hue, Saturation, Value). Noise generation will add random pixel variation to the original photo to simulate the real-world scenario as electrical noise is produced in the camera, making it less clear or grainier. Erosion will remove the pixels on object boundaries, whereas dilation will add the pixels to the boundaries, enhancing the object details and reducing noise. The mean/Median/Gaussian filter will mute the noise and soften the image while keeping the edge detail to a certain degree. The edge detection algorithm method will highlight the edges or boundaries of an object to define its structure. Histogram equalization adjusts the image's contrast to let the intensity of all the pixels be well-distributed and allow for a low contrast area to gain higher contrast. Hough transforms will detect any lines, circles, or other structures that can be recognized in the parametric equations. Morphological operations, such as top hat and black hat, are simple transformations applied to an image to change its structure.

3.0 Results

3.1) Resizing

```
# Scale up to ten times larger
Upscale = cv2.resize(image_rgb,(0,0),fx=10.0,fy=10.0)

# Scale down to ten times smaller
Downscale = cv2.resize(image_rgb,(0,0),fx=0.1,fy=0.1)
```

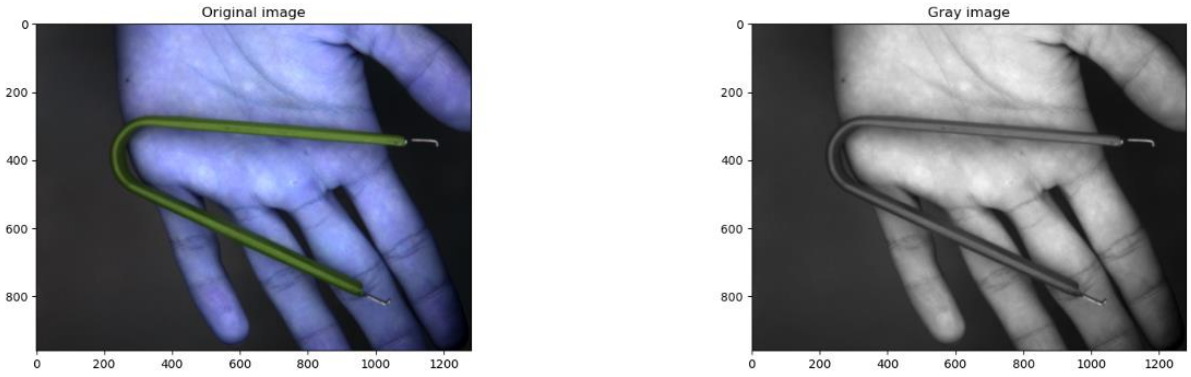
```
Original image size is (960, 1280, 3)
Upscale image size is (9600, 12800, 3)
Downscale image size is (96, 128, 3)
```



3.2) Colour-Space Conversion

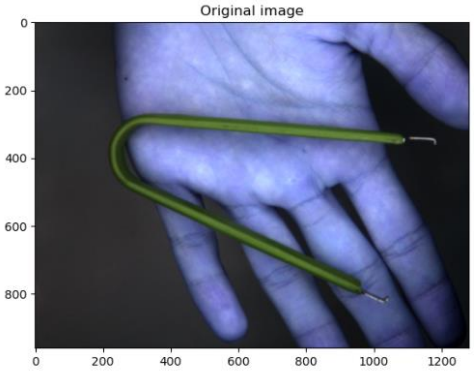
a) BGR to Gray

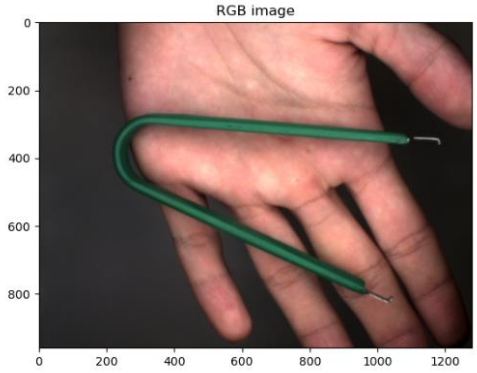
```
# Convert the image from BGR to GRAY
image_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```



b) BGR to RGB

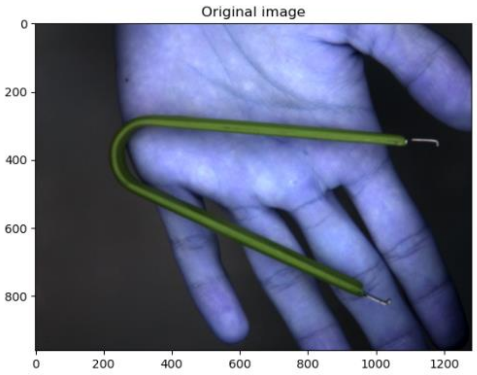
```
# Convert the image from BGR to RGB
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
```

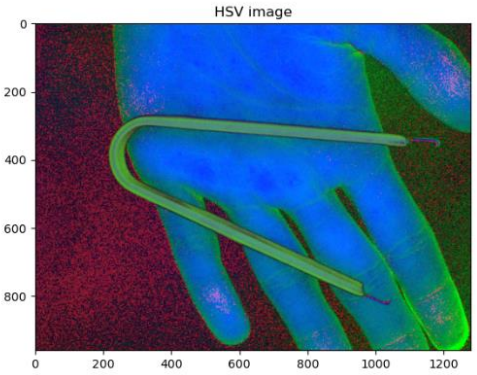




c) BGR to HSV

```
# Convert the image from BGR to HSV
image_hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
```





3.3) Noise generation

a1) Gaussian noise (mean = 0, std = 25) (default)

```
def add_gaussian_noise(image, mean=0, std=25):
    noise = np.random.normal(mean, std, image.shape).astype(np.uint8)
    noisy_image = cv2.add(image, noise)
    return noisy_image

if image is None:
    raise Exception("Image is not loaded properly.Check file path.")

noisy_image = add_gaussian_noise(image_rgb, mean=0, std=25)
```

Original image

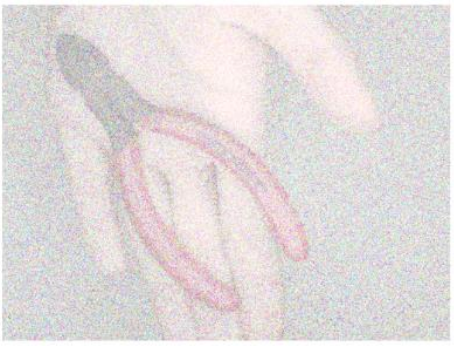
Gaussian noise

a2) Gaussian noise (mean = 1000, std = 25)

```
noisy_image = add_gaussian_noise(image_rgb, mean=1000, std=25)
```

Original image

Gaussian noise

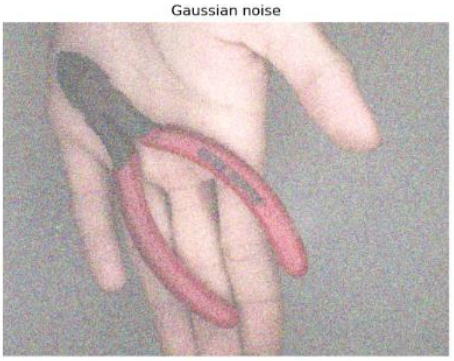


a3) Gaussian noise (mean = 0, std = 1000)

```
noisy_image = add_gaussian_noise(image_rgb, mean=0, std=1000)
```

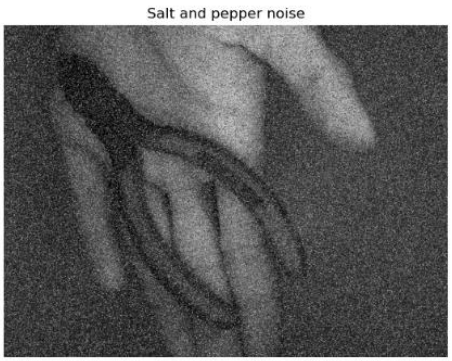
Original image

Gaussian noise



b) Salt and pepper noise (S&P)

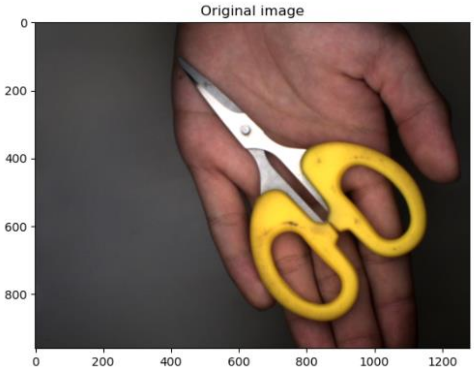
```
def add_noise(img):

    # Getting the dimensions of the image
    row , col = img.shape

    # Randomly pick some pixels in the
    # image for coloring them white
    number_of_pixels = 200000
    for i in range(number_of_pixels):

        # Pick a random y coordinate
        y_coord=random.randint(0, row - 1)

        # Pick a random x coordinate
        x_coord=random.randint(0, col - 1)

        # Color that pixel to white
        img[y_coord][x_coord] = 255

    # Randomly pick some pixels in
    # the image for coloring them black
    number_of_pixels = 200000
    for i in range(number_of_pixels):

        # Pick a random y coordinate
        y_coord=random.randint(0, row - 1)

        # Pick a random x coordinate
        x_coord=random.randint(0, col - 1)

        # Color that pixel to black
        img[y_coord][x_coord] = 0

    return img

img = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Plot the original and filtered images
fig,axes = plt.subplots(1,2,figsize=(20,5))
axes[0].set_title("Original image")
axes[0].imshow(image_rgb)
axes[0].axis("OFF")

axes[1].set_title("Salt and pepper noise")
axes[1].imshow(add_noise(img),cmap="gray")
axes[1].axis("OFF")

plt.show()
```

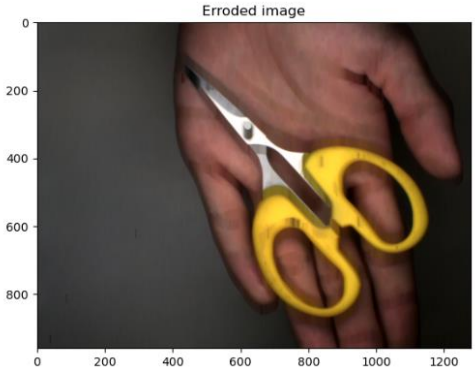


3.4) Erosion

a) Kernel = 5*5, Iterations = 1 (constant variable)

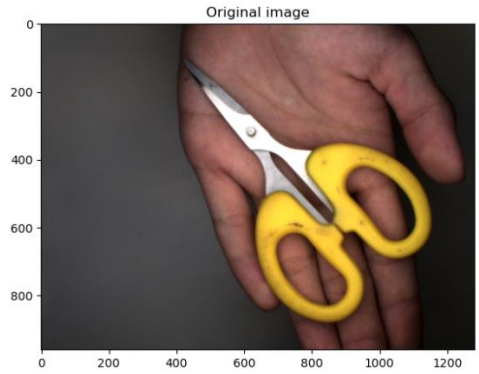
```
kernel = np.ones((5*5),np.uint8)

erodeImage = cv2.erode(image_rgb,kernel,iterations=1)
```

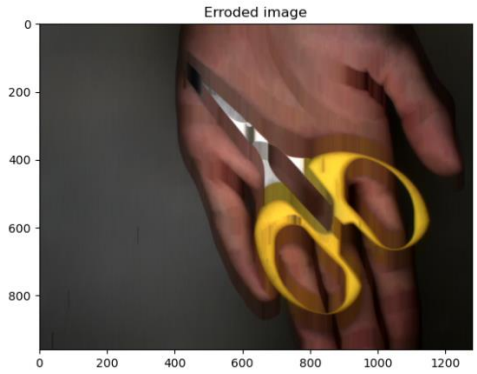




b) Kernel = 7*7, Iterations = 1

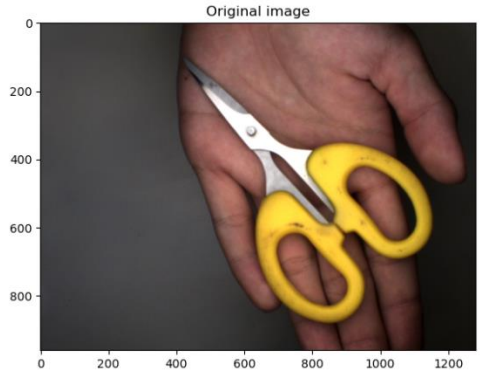
```
kernel = np.ones((7*7),np.uint8)
```

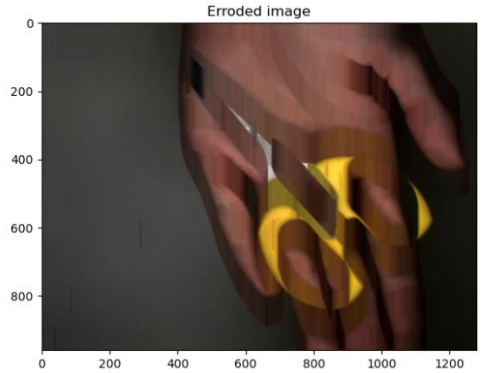




c) Kernel = 9*9, Iterations = 1

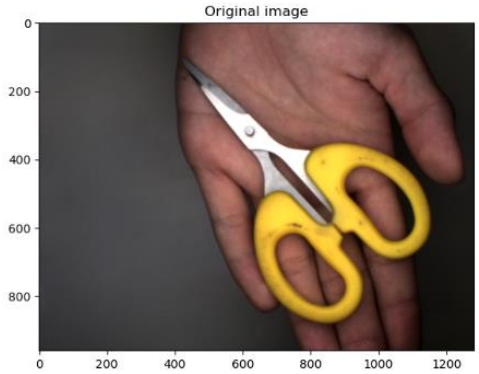
```
kernel = np.ones((9*9),np.uint8)
```

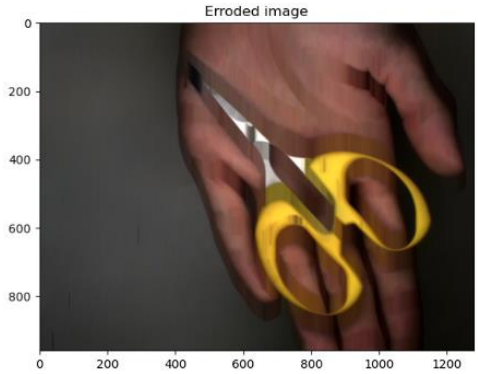




d) Kernel = 5*5, Iterations = 2

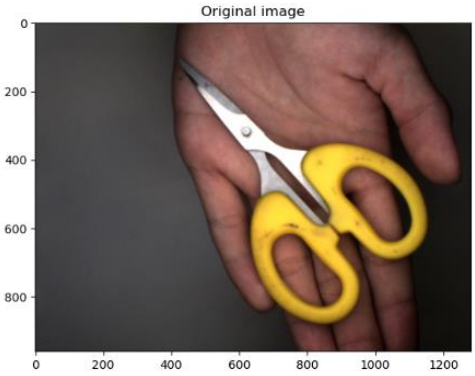
```
erodeImage = cv2.erode(image_rgb,kernel,iterations=2)
```

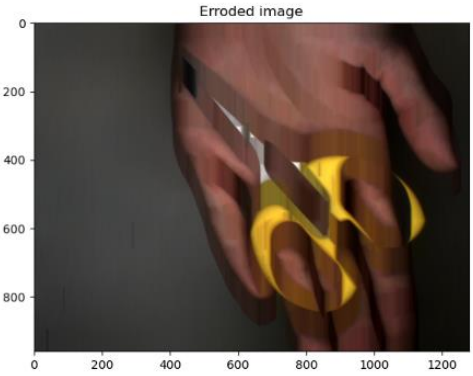




e) Kernel = 5*5, Iterations = 3

```
erodeImage = cv2.erode(image_rgb,kernel,iterations=3)
```

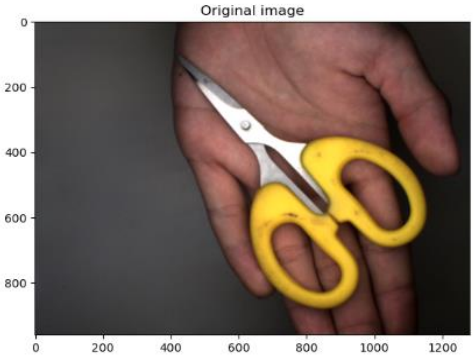


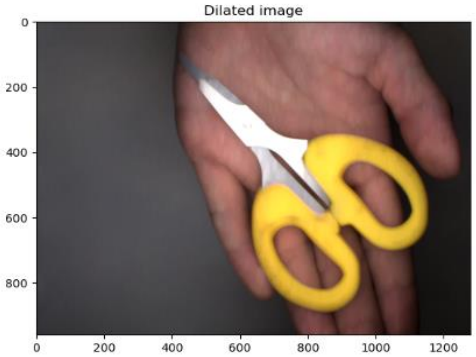


3.5) Dilation

a) Kernel = 5*5, Iterations = 1 (constant variable)

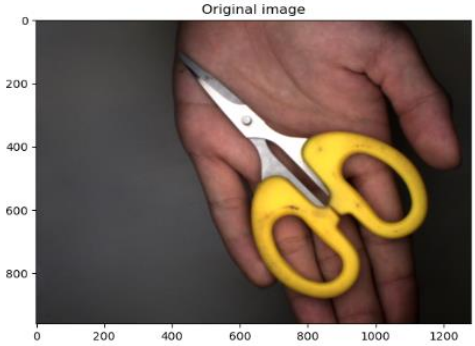
```
kernel = np.ones((5*5),np.uint8)

dilateImage = cv2.dilate(image_rgb,kernel,iterations=1)
```

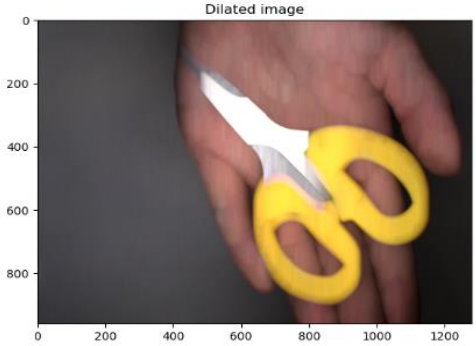




b) Kernel = 7*7, Iterations = 1

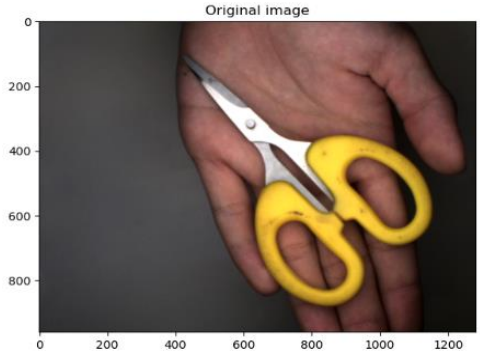
```
kernel = np.ones((7*7),np.uint8)
```

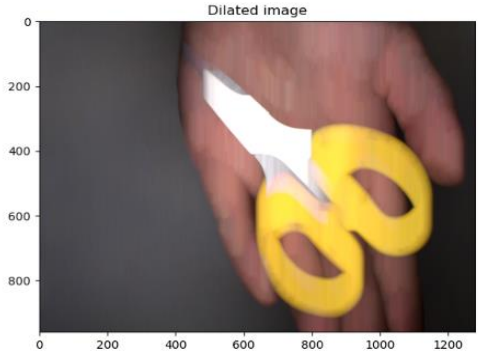




c) Kernel = 9*9, Iterations = 1

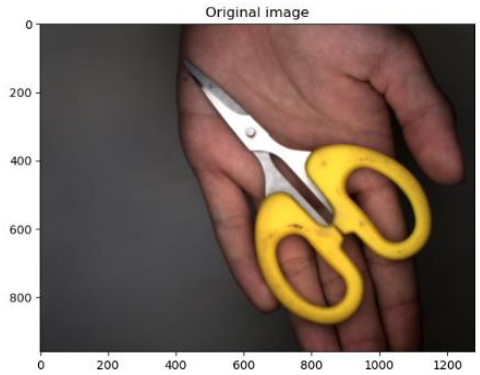
```
kernel = np.ones((9*9),np.uint8)
```

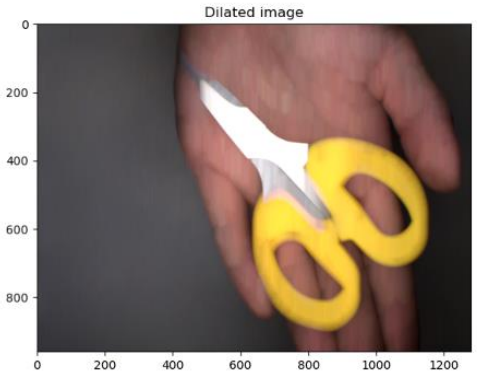




d) Kernel = 5*5, Iterations = 2

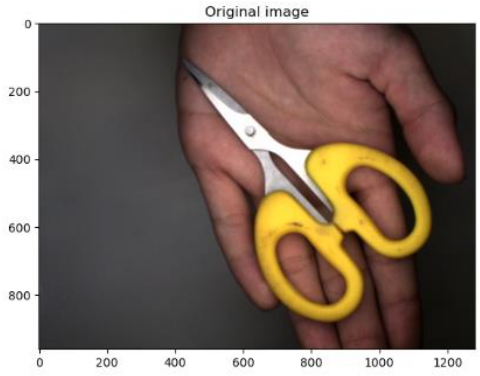
```
dilateImage = cv2.dilate(image_rgb,kernel,iterations=2)
```

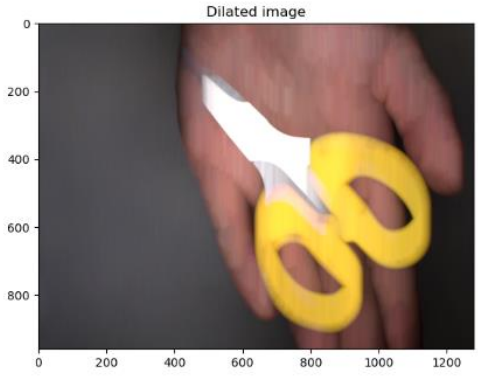




e) Kernel = 5*5, Iterations = 3

```
dilateImage = cv2.dilate(image_rgb,kernel,iterations=3)
```

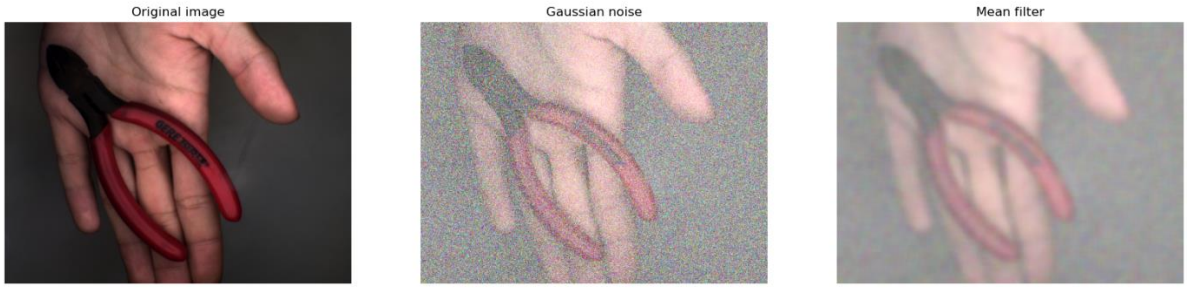




3.6) Mean Filter

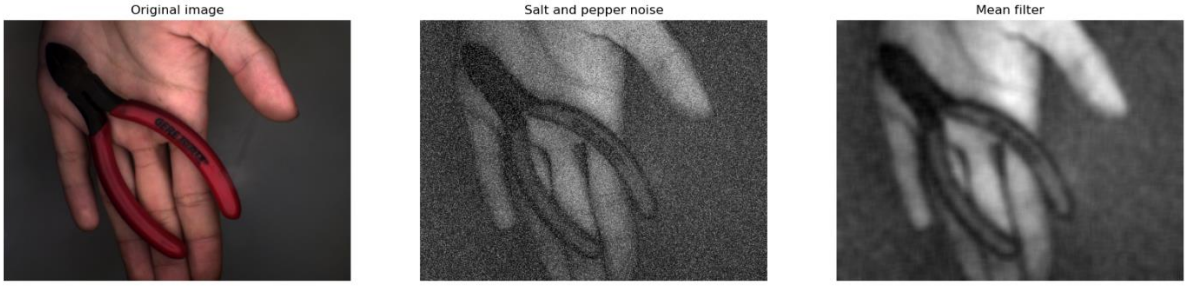
a1) ksize = (31,31) (noise = Gaussian)

```
mean_filter = cv2.blur(noisy_image, ksize=(31,31))
```



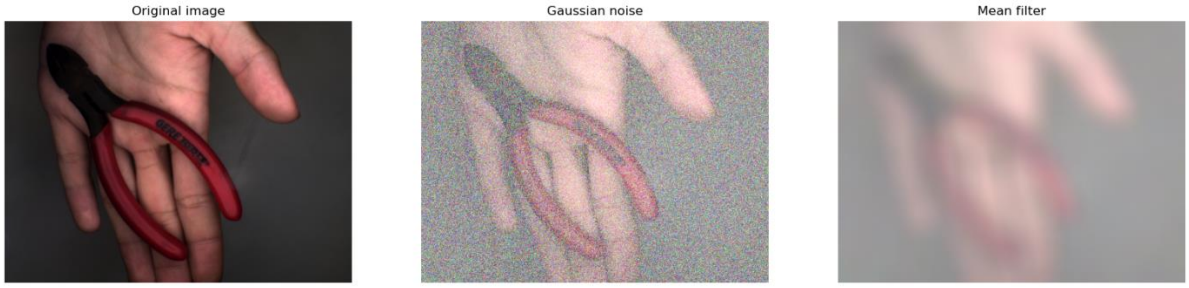
a2) ksize = (31,31) (noise = S&P)

```
mean_filter = cv2.blur(noisy2_image,ksize=(31,31))
```



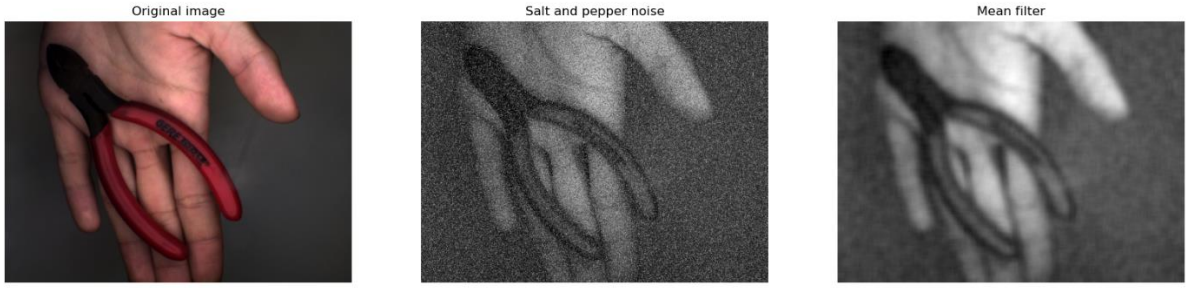
b1) ksize = (101,101) (noise = Gaussian)

```
mean_filter = cv2.blur(noisy_image, ksize=(101,101))
```



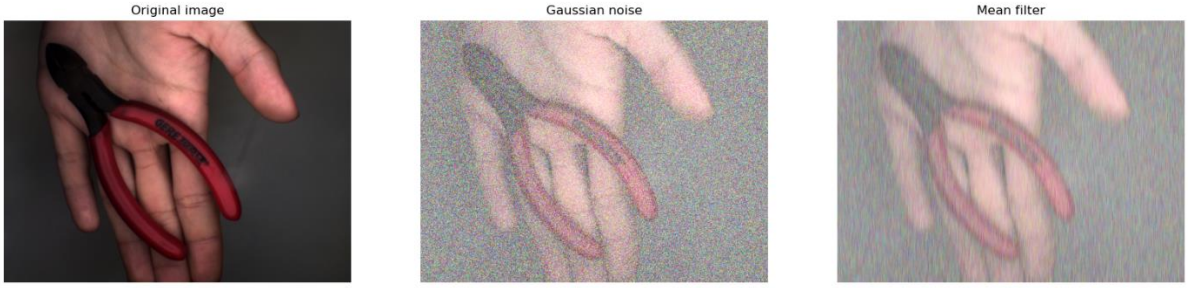
b2) ksize = (101,101) (noise = S&P)

```
mean_filter = cv2.blur(noisy2_image,ksize=(101,101))
```



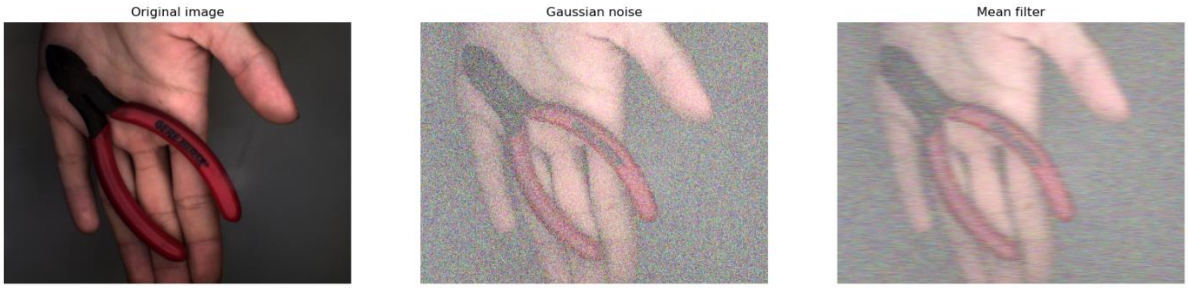
c) ksize = (1,31) (noise = Gaussian)

```
mean_filter = cv2.blur(noisy_image,ksize=(1,31))
```



d) ksize = (31,1) (noise = Gaussian)

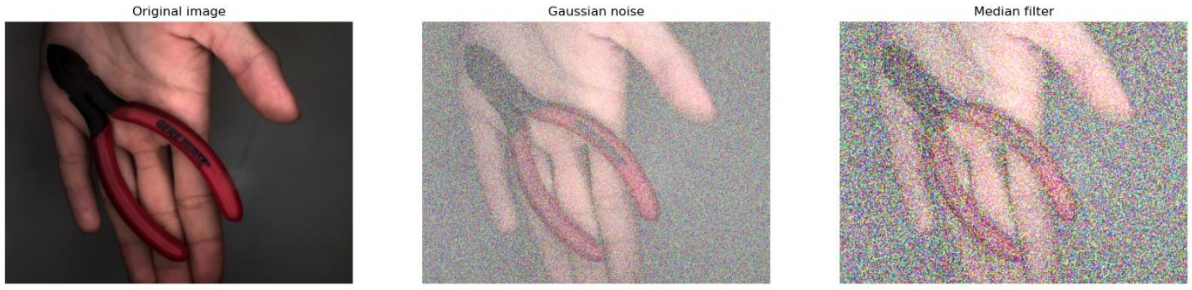
```
mean_filter = cv2.blur(noisy_image,ksize=(31,1))
```



3.7) Median Filter

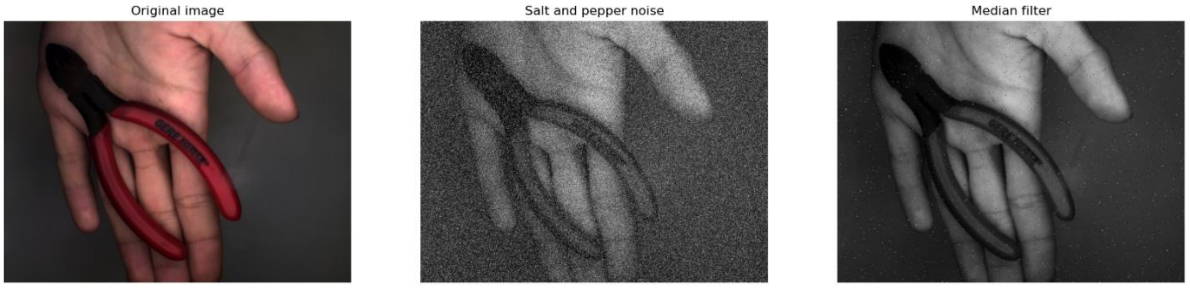
a1) ksize = 3 (noise = Gaussian)

```
median_filter = cv2.medianBlur(noisy_image, ksize=3)
```



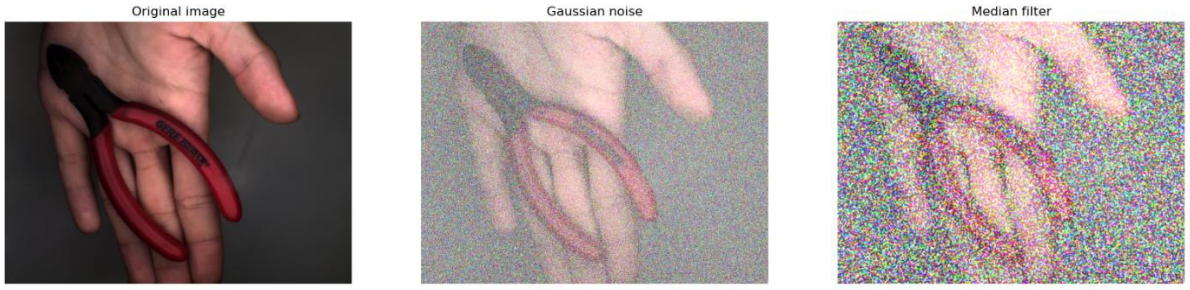
a2) ksize = 3 (noise = S&P)

```
median_filter = cv2.medianBlur(noisy2_image, ksize=3)
```



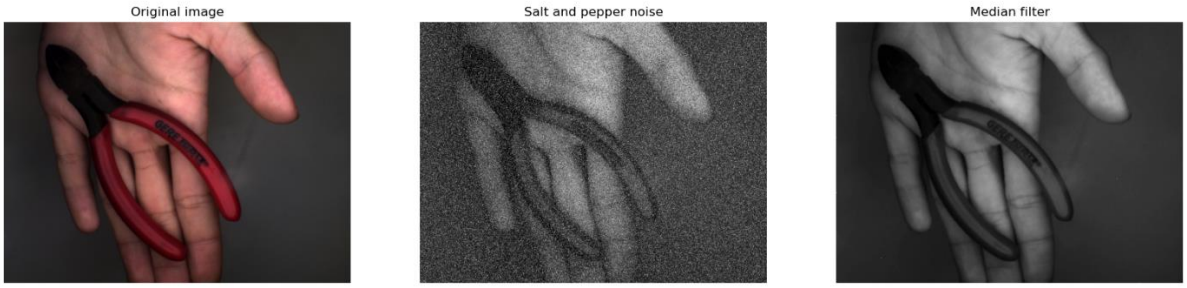
b1) ksize = 5 (noise = Gaussian)

```
median_filter = cv2.medianBlur(noisy_image, ksize=5)
```



b2) ksize = 5 (noise = S&P)

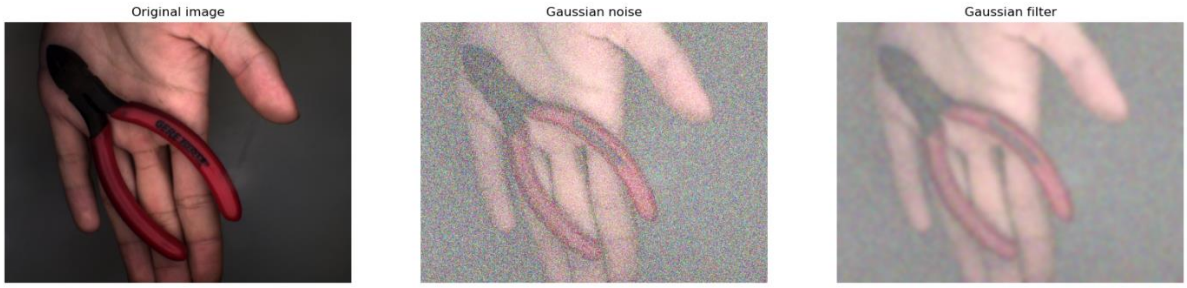
```
median_filter = cv2.medianBlur(noisy2_image, ksize=5)
```



3.8) Gaussian Filter

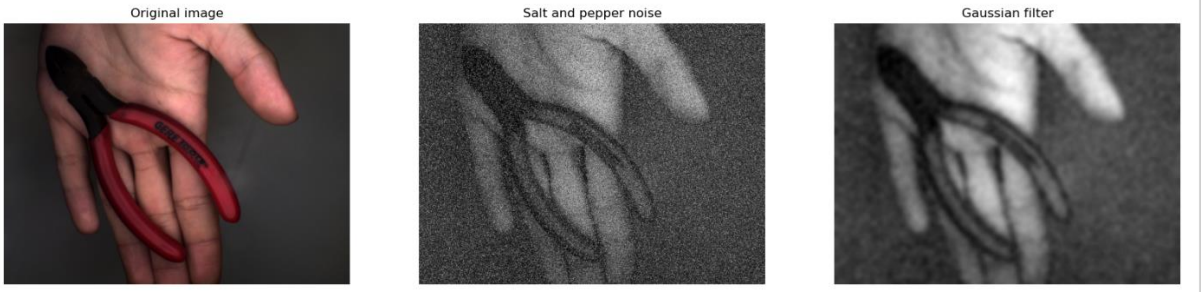
a1) ksize = (31,31), sigmaX = 10, sigmaY = 10 (noise = Gaussian)

```
gaussian_filter = cv2.GaussianBlur(noisy_image, ksize =(31,31), sigmaX = 10,sigmaY=10)
```



a2) ksize = (31,31), sigmaX = 10, sigmaY = 10 (noise = S&P)

```
gaussian_filter = cv2.GaussianBlur(noisy2_image, ksize =(31,31), sigmaX = 10,sigmaY=10)
```

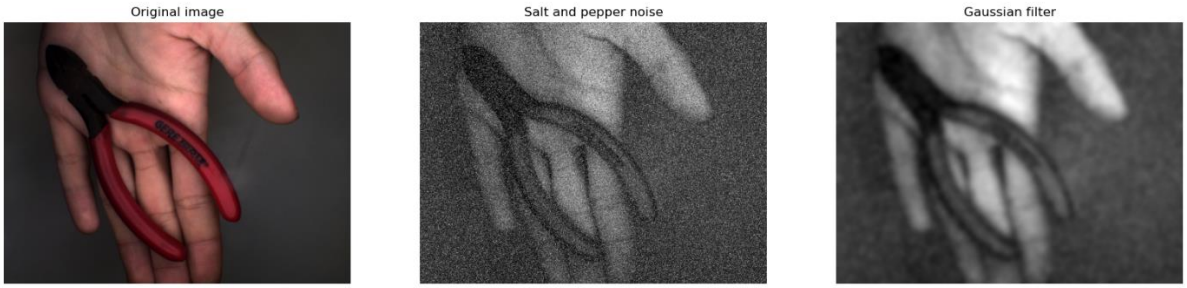


b1) ksize = (101,101), sigmaX = 10, sigmaY = 10 (noise = Gaussian)

```
gaussian_filter = cv2.GaussianBlur(noisy_image, ksize =(101,101), sigmaX = 10,sigmaY=10)
```

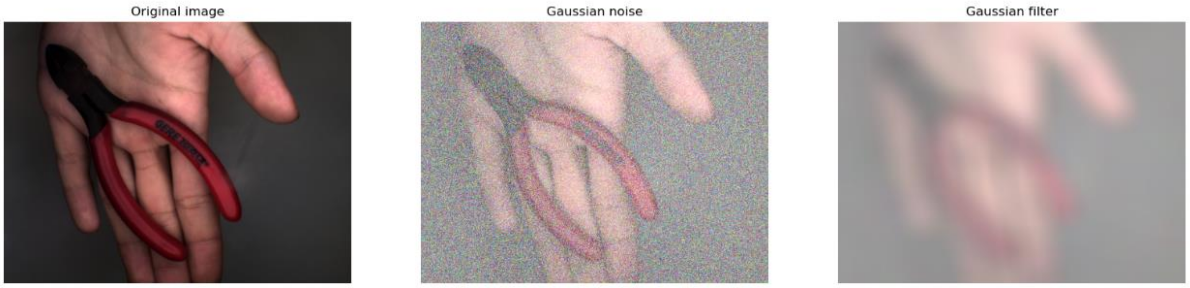
b2) ksize = (101,101), sigmaX = 10, sigmaY = 10 (noise = S&P)

```
gaussian_filter = cv2.GaussianBlur(noisy2_image, ksize =(101,101), sigmaX = 10,sigmaY=10)
```



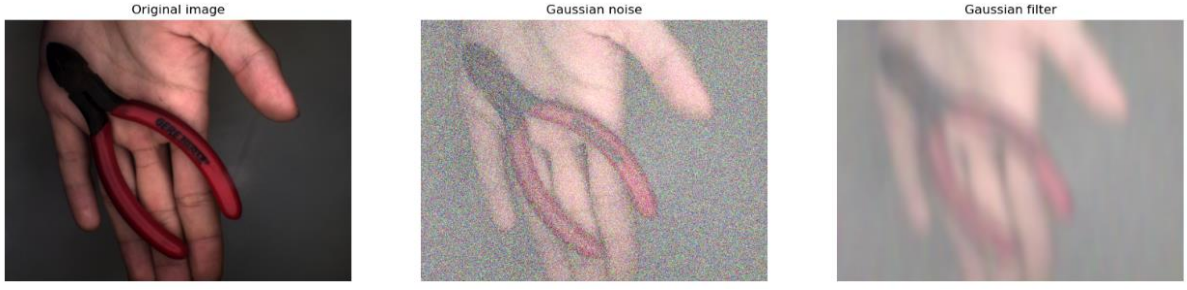
b3) ksize = (101,101), sigmaX = 10^5, sigmaY = 10^5 (noise = Gaussian)

```
gaussian_filter = cv2.GaussianBlur(noisy_image, ksize =(101,101), sigmaX = 100000,sigmaY=100000)
```



b4) ksize = (101,101), sigmaX = 10, sigmaY = 10^5 (noise = Gaussian)

```
gaussian_filter = cv2.GaussianBlur(noisy_image, ksize =(101,101), sigmaX = 10,sigmaY=100000)
```



b5) ksize = (101,101), sigmaX = 10^5, sigmaY = 10 (noise = Gaussian)

```
gaussian_filter = cv2.GaussianBlur(noisy_image, ksize =(101,101), sigmaX = 100000,sigmaY=10)
```

3.9) Edge Detection Algorithm

Canny edge detection

a) Threshold (35,40)

```
def canny_edge_detection(frame):
    # Convert the frame to grayscale for edge detection
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Apply Gaussian blur to reduce noise and smoothen edges
    blurred = cv2.GaussianBlur(src=gray, ksize=(3, 5), sigmaX=0.5)

    # Perform Canny edge detection
    edges = cv2.Canny(blurred, 35, 40)

    return blurred, edges
```

b) Threshold (0,40)

```
# Perform Canny edge detection
edges = cv2.Canny(blurred, 0, 40)
```

c) Threshold (0,0)

```
# Perform Canny edge detection
edges = cv2.Canny(blurred, 0, 0)
```

3.10) Histogram Equalization

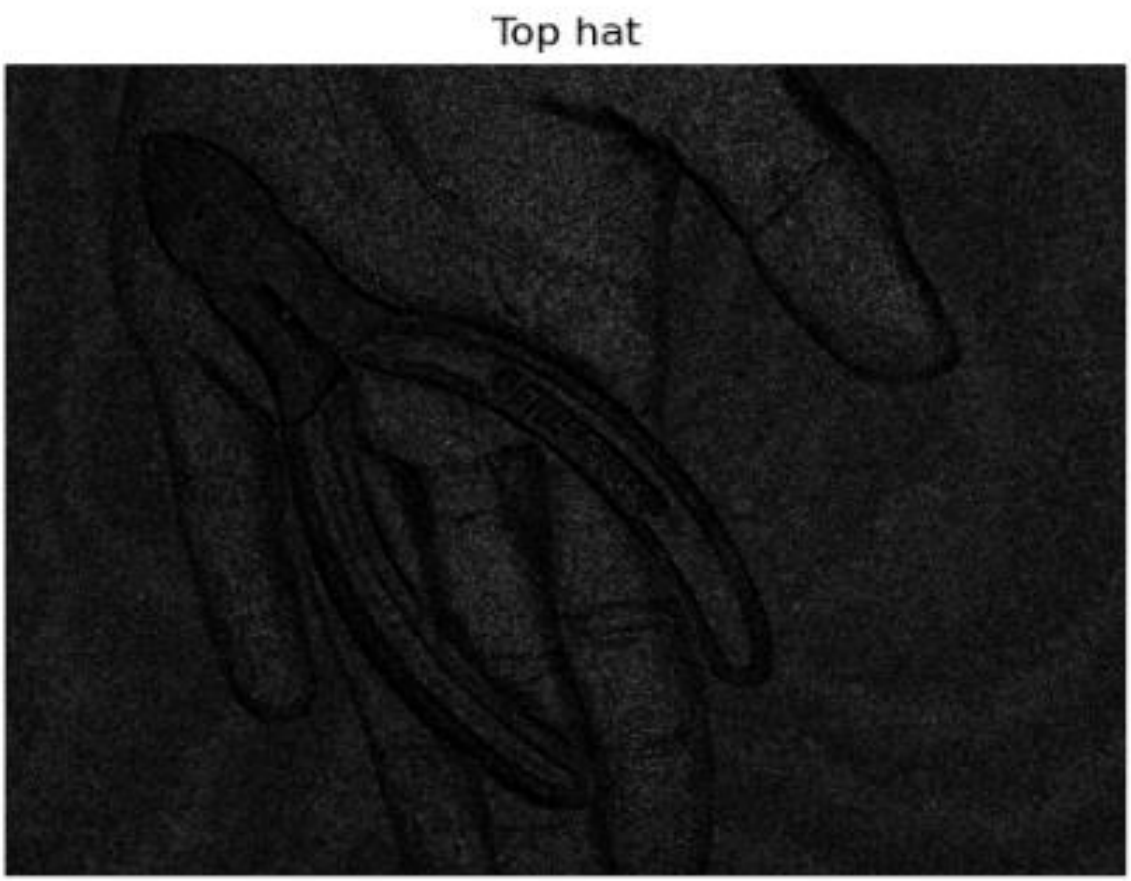
```
# creating a Histograms Equalization
# of a image using cv2.equalizeHist()
equ = cv2.equalizeHist(img)
```

3.11) Hough transforms

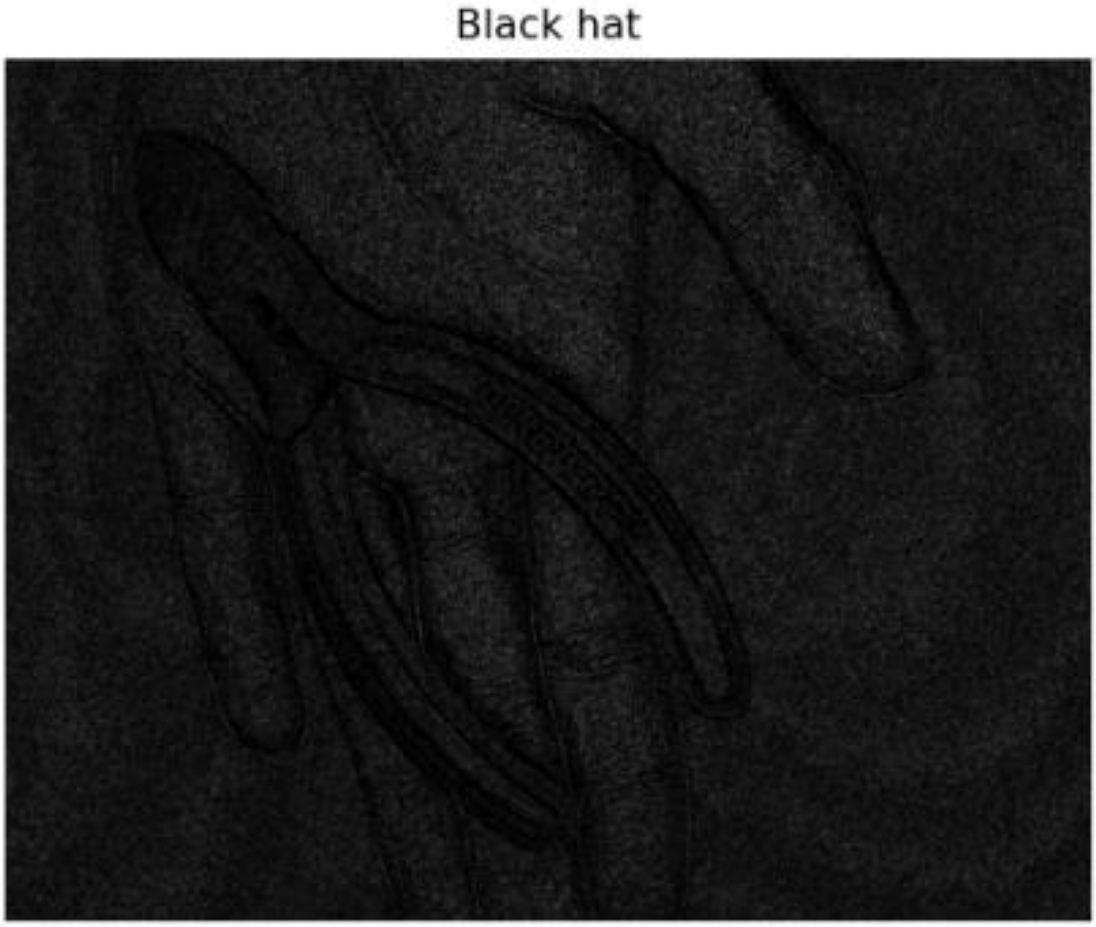
```
edges = cv2.Canny(img,50,150,apertureSize=3)

# Apply HoughLinesP method to
# to directly obtain line end points
lines_list =[]
lines = cv2.HoughLinesP(
            edges, # Input edge image
            1, # Distance resolution in pixels
            np.pi/180, # Angle resolution in radians
            threshold=100, # Min number of votes for valid line
            minLineLength=5, # Min allowed length of line
            maxLineGap=10 # Max allowed gap between line for joining them
            )

# Iterate over points
for points in lines:
      # Extracted points nested in the list
    x1,y1,x2,y2=points[0]
    # Draw the lines joing the points
    # On the original image
    cv2.line(image,(x1,y1),(x2,y2),(0,255,0),2)
    # Maintain a simples lookup list for points
    lines_list.append([(x1,y1),(x2,y2)])
```

3.12) Morphological Operations

a) Top hat



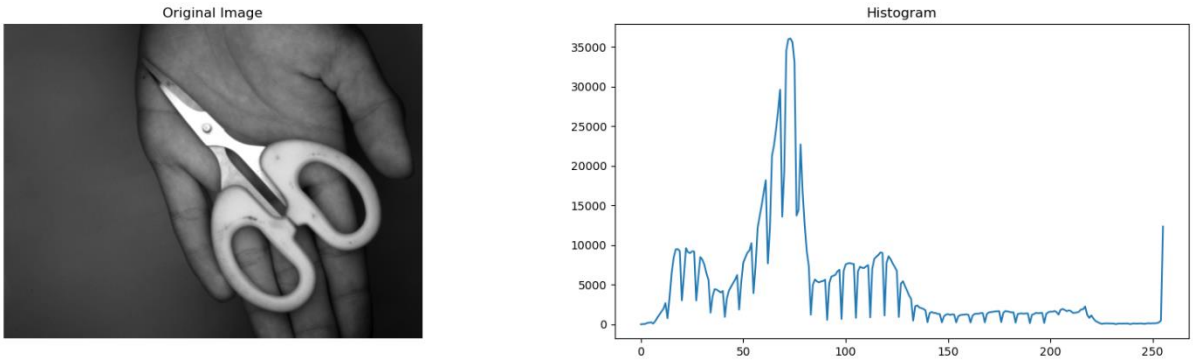
b) Black hat



3.13) Histogram comparison

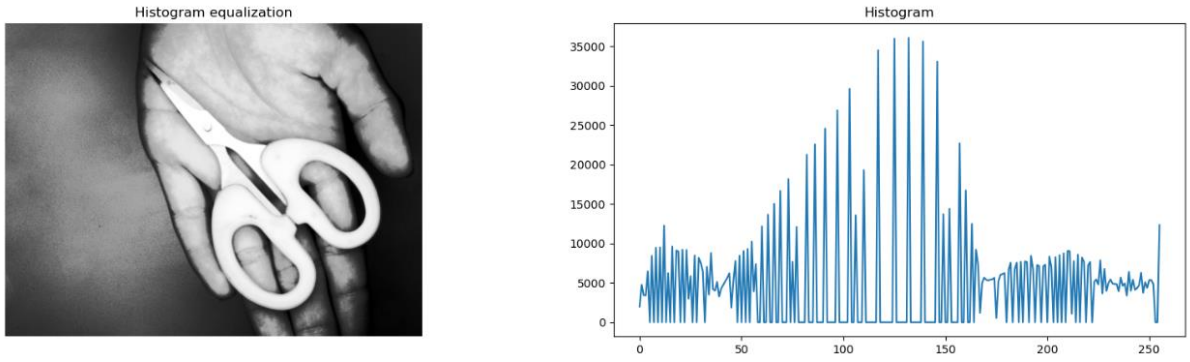
```
image = cv2.imread('o7.bmp',cv2.IMREAD_GRAYSCALE)
```

```
hist = cv2.calcHist([image], [0], None, [256], [0, 256])
#       cv.calcHist(images, channels, mask, histSize, ranges[, hist[, accumulate]]) ->hist


fig, axes = plt.subplots(1, 2, figsize=(20, 5))
axes[0].imshow(image,cmap="gray")
axes[0].set_title("Original Image")
axes[0].axis('off')

axes[1].plot(hist)
axes[1].set_title("Histogram")
axes[1].axis('on')
```




```
equ = cv2.equalizeHist(image)
```

```
hist = cv2.calcHist([equ], [0], None, [256], [0, 256])
#       cv.calcHist(images, channels, mask, histSize, ranges[, hist[, accumulate]]) ->hist


fig, axes = plt.subplots(1, 2, figsize=(20, 5))
axes[0].imshow(equ,cmap="gray")
axes[0].set_title("Histogram equalization")
axes[0].axis('off')

axes[1].plot(hist)
axes[1].set_title("Histogram")
axes[1].axis('on')
```



4.0 Discussion

In the practical session, I noticed that controlling the size of the aperture lens was to control the brightness of the background, whether to reduce the aperture size to prevent excessive exposure to the image or increase the aperture size to allow more light to the photo in low light environment condition. Besides that, the lens's focal length was adjusted depending on the distance between the lens and the object, shortening the focal length, which allowed for a more comprehensive view when the object was closer to the lens and vice versa. In the resize method, f_x and f_y parameters in the cv2.resize() function are the amplification value of the original image shape, if I change the parameters less than 1, it will reduce the size of the image and eventually cause the downgrading of image quality. If the parameters are greater than 1, the picture's size and resolution will increase. The parameters I used for the upscale image is $f_x = f_y = 10$ and for the downscale image is $f_x = f_y = 0.1$ which will enlarge or diminish the photo's pixels by 10 times from the original pixels. Moreover, the cv2.cvtColor() function is used for doing the colour-space conversion as the original photos taken in the lab were saved as BGR colour space format. Thus, we need to convert it to RGB, GRAY or HSV depending on the preprocessing method required.

Gaussian noise is characterized by two parameters, mean (μ) and standard deviation (σ). The mean parameter represents the average value of the noise thereby when I change the mean from 0 to 1000 as the standard deviation acts as a constant variable, it shifts the image toward a brighter appearance. The other comparison is to change the standard deviation from 25 to 1000 when the mean did not change from the default settings. A grainier and broader spread of the noise distribution is noticed when compared to the default noise image. Furthermore, the number_of_pixels parameter in the salt and pepper noise function determines the number of pixels to colour black and white on the image. Therefore, the higher the number_of_pixels will make the image noisier. For the erosion operation, the kernel was created by np.ones((x,x), np.uint8) function. This creates an 'x' by 'x' matrix filled with ones, with an 8-bit unsigned integer data type. The kernel defines the neighbourhood over when the erosion operation is performed. Its mechanism is A $\ominus$ B, where A is the image and B is the structural element. During the operation, the kernel is centered on each pixel of the image. If the kernel is fitted on the image, the pixels on the center of the kernel will be preserved. Otherwise, it will be eroded which will keep repeating

until it checks till the last row of the picture. Dilation, $A \oplus B$ is quite like the erosion operation; the only difference is that on the hit condition, the centered pixel will be preserved, and this tiny condition difference makes the neighbourhood boundary of an object grow larger. The kernel size determines the degree of erosion or dilation, and this can be observed when the kernel size is larger in both operations, the effect will be magnified on the output. The iteration of this operation means that it will repeat the process so that the production will have a similar effect as kernel size increases.

Next, the output difference between the mean, median and Gaussian filter on the Gaussian and salt and pepper noise are that the median filter will worsen on filtering the Gaussian noise. Still, it can filter out the salt and pepper noise entirely among the three filters, whereby the Gaussian filter and mean filter seem to have weak filtering effects on both noises. It is because the noise was removed, and at the same time some of the image quality is also lost due to being blurred. The median filter cannot filter the Gaussian noise because it did not average all the surrounding pixels but picked the median value, which may contain the noise pixels. Kernel size in three filters can affect how robust the filtering is. Unique attributes existed of the Gaussian filter are the sigmaX and the sigmaY representing the extent of the blurred amount in the horizontal or vertical direction. Besides, the canny edge detection method uses the Gaussian blur to filter out the noise following by cv2.Canny(image, (threshold)), and there are two parameters in the threshold: lower threshold and the upper threshold. The edge lower than the lower threshold will be considered as no edge. The value between the lower and upper threshold looks like a weak edge, and the edge higher than the higher threshold will be defined as a strong edge. When the threshold lies between (35,40), it will clearly show the object's edge.

Based on the result, the histogram equalization method will make the original image more contrast and brighter. The Hough transform is a method to define the lines of the object, which can be observed in the result section. The top hat highlights the brighter regions whereas the black hat highlights the darker areas. On the other hand, the histogram analysis on the histogram equalization shows that many of the bright pixels were spread in the middle compared to the original image.

5.0 Conclusion

In conclusion, the experiment was conducted successfully and achieved the objectives labelled in the lab sheet. Different methodologies were applied in this lab, such as resizing, colour-space conversion, canny edge detection, morphologies operation, histogram equalizations, and others, which showed nice and clear results. As a result, a strong discussion and comparison of effects and parameters between each operation could be conducted.

References

1) GeeksforGeeks. (2019). *Image Resizing using OpenCV | Python*. [online] Available at: https://www.geeksforgeeks.org/image-resizing-using-opencv-python/ [Accessed on 7 July 2024].

2) GeeksforGeeks. (2019). *Python OpenCV | cv2.cvtColor() method*. [online] Available at: https://www.geeksforgeeks.org/python-opencv-cv2-cvtcolor-method/ [Accessed on 8 July 2024].

3) Nair, A. (2023). *Guide to Adding Noise Images with Python and OpenCV - AskPython*. [online] AskPython. Available at: https://www.askpython.com/python/examples/adding-noise-images-opencv [Accessed on 9 July 2024].

4) Ceylan, T. (2022). *Blur Filters with OpenCV | by Turgay Ceylan | Medium*. [online] Medium. Available at: https://medium.com/@turgay2317/image-processing-in-python-with-opencv-blur-3e474fda6a52 [Accessed on 9 July 2024].

5) GeeksforGeeks. (2020). *Add a 'salt and pepper' noise to an image with Python*. [online] Available at: https://www.geeksforgeeks.org/add-a-salt-and-pepper-noise-to-an-image-with-python/ [Accessed on 9 July 2024].

6) GeeksforGeeks. (2016). *Real-Time Edge Detection using OpenCV in Python | Canny edge detection method*. [online] Available at: https://www.geeksforgeeks.org/real-time-edge-detection-using-opencv-python/ [Accessed on 11 July 2024].

7) GeeksforGeeks. (2018). *Histograms Equalization in OpenCV*. [online] Available at: https://www.geeksforgeeks.org/histograms-equalization-opencv/ [Accessed on 11 July 2024].

8) GeeksforGeeks. (2017). *Line detection in python with OpenCV | Houghline method*. [online] Available at: https://www.geeksforgeeks.org/line-detection-python-opencv-houghline-method/ [Accessed on 11 July 2024].

9) GeeksforGeeks. (2020). *Top Hat and Black Hat Transform using Python-OpenCV*. [online] Available at: https://www.geeksforgeeks.org/top-hat-and-black-hat-transform-using-python-opencv/ [Accessed on 11 July 2024].

10) Aarthy (2023). *Erosion and Dilation in Image Processing*. [online] Scaler Topics. Available at: https://www.scaler.com/topics/erosion-and-dilation-in-image-processing/ [Accessed on 11 July 2024].

11) samir khanal (2020). *Dilation and Erosion - samir khanal - Medium*. [online] Medium. Available at: https://samirkhanal35.medium.com/dilation-and-erosion-74e7d4c4a503 [Accessed 12 July 2024].

12) Kang & Atul (2019). *Erosion*. [online] TheAILearner. Available at: https://theailearner.com/2019/07/26/erosion/ [Accessed 12 July 2024].

13) Simsangcheol (2023). *OpenCV — Histogram of Grayscale Image - Simsangcheol - Medium*. [online] Medium. Available at: https://medium.com/@sim30217/opencv-histogram-of-grayscale-image-8de86fb248e1 [Accessed 12 Jul. 2024].

Appendix A: Split Original file into R,G,B method

```python
import cv2
import matplotlib.pyplot as plt

# Load the image
image = cv2.imread('o4.bmp')

# Convert the image from BGR to RGB
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Split the image into R, G, B channels
R, G, B = cv2.split(image_rgb)

# Plot the original and filtered images
fig, axes = plt.subplots(1, 4, figsize=(20, 5))
axes[0].imshow(image_rgb)
axes[0].set_title("Original Image")
axes[0].axis('off')

axes[1].imshow(R, cmap='gray')
axes[1].set_title("Red Filter")
axes[1].axis('off')

axes[2].imshow(G, cmap='gray')
axes[2].set_title("Green Filter")
axes[2].axis('off')

axes[3].imshow(B, cmap='gray')
axes[3].set_title("Blue Filter")
axes[3].axis('off')

plt.show()

# save image
#cv2.imwrite(r'C:\Users\HP\Desktop\UTAR       (ALL YEAR)\Y3S1\Machine Vision\Lab\Practical 1\myImage\b4.bmp',B)
```