Value Added Program - Python Vision Techniques

AN INDUSTRIAL INTERNSHIP TRAINING REPORT

Submitted by

Andrew John

18BEC1278

ECE3099 – INDUSTRIAL INTERNSHIP

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

in

ELECTRONICS AND COMMUNICATION ENGINEERING

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

OCTOBER 2021

School of Electronics Engineering

DECLARATION BY THE CANDIDATE

I hereby declare that the Industrial Internship Report entitled "**Value Added Program - Python Vision Techniques"** submitted by me to VIT, Chennai in partial fulfillment of the requirement for the award of the degree of **Bachelor of Technology** in **Electronics and Communication Engineering** is a record of bonafide industrial training undertaken by me under the supervision of **Dr. Sathiya Narayanan Sekar, SENSE Assistant Professor, VIT, Chennai** and **Dr. Annis Fathima, SENSE Associate Professor, VIT, Chennai.** I further declare that the work reported in this report has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Chennai **Signature of the Candidate**

Date: 05/10/2021 **Register Number**: 18BEC1278
Student Name: Andrew John

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

CERTIFICATE OF COMPLETION

This is to certify that Mr./Ms. ANDREW JOHN (18BEC1278) of B.TECH ELECTRONICS AND COMMUNICATION ENGINEERING has successfully completed the Value Added Course **"Python for Vision Techniques"** organized by the **School of Electronics Engineering (SENSE),** Vellore Institute of Technology (VIT) – Chennai, from 25-February-2021 to 10-April-2021. His/her consolidated score is 90 out of 100.

Dr. Annis Fathima A	Dr. Sathya Narayanan S	Dr. Sivasubramanian A	Dr. Kanchana Bhaaskaran
Faculty Coordinator	Faculty Coordinator	Dean-SENSE	Pro Vice Chancellor

VIT – Recognised as Institution of Eminence (IoE) by Government of India

VIT - A place to learn; A chance to grow

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

School of Electronics Engineering

BONAFIDE CERTIFICATE

This is to certify that the Industrial Internship Report entitled "**Value Added Program - Python Vision Techniques"** submitted by **Andrew John (18EC1278)** to VIT University, Chennai in partial fulfillment of the requirement for the award of the degree of **Bachelor of Technology** in **Electronics and Communication Engineering** is a record of bonafide internship undertaken by him/her fulfills the requirements as per the regulations of this institute and in my opinion meets the necessary standards for submission. The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Signature of the Examiner **Signature of the Examiner**

Date: Date:

Head of the Department (B.Tech ECE)

ACKNOWLEDGEMENT

I wish to thank those who were involved in the successful completion of my Value Added Course at VIT Chennai for giving me the opportunity and freedom to learn as per my interests and the faculties, Dr. Sathiya Narayanan and Dr. Annis Fathuma for their support and positivity which made my internship a worthwhile experience.

I would also like to thank my parents, for being my motivation to take up this internship; and last, but not the least, management at Vellore Institute of Technology (VIT), Chennai, for providing me with such an avenue to help realize how interesting it is to work in today's industry.

It is my proud privilege to express my profound gratitude to the Dean of SENSE, Dr. Sivasubramanian. A and Program Chair, HOD, Dr.Vetrivelan.P for providing me this valuable opportunity to have industrial exposure.

ANDREW JOHN

Table Of Contents

List of Figures

List of Tables

S.No	Table Description	Page No.

List of Abbreviations:

1. OpenCV - Open Computer Vision
2. ANN - Artificial Neural Network
3. ML - Machine Learning
4. DL - Deep Learning
5. AI - Artificial Intelligence
6. CNN - Convolutional Neural Network
7. RNN - Recurrent Neural Network
8. TL - Transfer Learning

ABSTRACT

Computer vision is a process by which we can understand the images and videos, how they are stored and how we can manipulate and retrieve data from them. Computer Vision is the base or mostly used for Artificial Intelligence. Computer-Vision is playing a major role in self-driving cars, robotics as well as in photo correction apps.

In image processing we perform some operations on an image, in order to get an enhanced image and or to extract some useful information from it. This image processing and computer vision together gives many insights which are used in many important fields like medical for automatic disease detection, in manufacturing industries for detecting defects in the materials and many other uses.

This course focused on how to use OpenCV - Python to process images and apply some computer vision techniques and get some useful information from it. As well as various real time world applications of Computer Vision are discussed where state-of-the-art technologies are used along with Computer Vision.

CHAPTER 1

Introduction

1.1 About VIT

Founded in 1984, VIT has made a mark in the field of higher education in India imparting quality education in a multi-cultural ambience, intertwined with extensive application-oriented research. VIT was established with the aim to provide quality higher education on par with International Standards. It persistently seeks and adopts innovative methods to improve the quality of higher education on a consistent basis. VIT was established by well-known educationist and former parliamentarian, Dr. G. Viswanathan, Founder and Chancellor, a visionary who transformed VIT into a center of excellence in higher technical education. The Govt. of India recognized VIT as an Institution of Eminence (IoE). VIT Chennai is ably spearheaded by Mr. Sankar Viswanathan, Vice-President, Ms. Kadhambari S. Viswanathan, Assistant Vice-President, Dr Rambabu Kodali, Vice-Chancellor and Dr. V. S. Kanchana Bhaaskaran, Pro-Vice Chancellor. They share in the mission to make VIT a global center towards academic and research excellence.

The focus is to:

- To maximize the interactive Industrial Connectivity.
- To create Centers of Excellence in niche areas.
- To enrich Technological and Managerial Human Capital nurtured in a multicultural ambience.
- To provide a common platform for the agglomeration of ideas of personnel from various walks of life for learning enrichment.
- To create opportunities and exploit the available resources to benefit industry and society.
- To encourage participation in the National Agenda of knowledge building.
- To foster International collaborations for mutual benefits in areas of research.

1.2 About School - SENSE

The School of Electronics Engineering at VIT was established for imparting the state-of-the-art education, training and research in the field of Electronics and Communication Engineering and allied areas. It offers B.Tech Programmes in Electronics and Communication Engineering and Electronics and Computer Engineering, M.Tech Programmes in VLSI Design and Embedded Systems, and Ph.D. in the related domains of ECE. The expertise of the faculty members includes VLSI Design, Communication Engineering, Embedded Systems, Nano-electronics and Nano-technology, Photonics, Signal Processing, Machine Learning, Artificial Intelligence and Data Analytics.

1.3 About the Value Added Program

The motive is to enlighten them on start-of-the-art computer vision techniques and their implementation using Python. The course gave good insights into understanding images. Basic array operations on images using python were performed. Spatial domain operations and image enhancement using python gave good insights into various filters and operators used in image enhancement. Image segmentation and morphology using python helped in detecting edge and corner discrepancies in images and along with this image classification using python through k-Means was performed. CNN for vision applications and Deep Learning concepts in image processing was delivered through guest faculty. State-of-the-art Computer Vision applications were also delivered through the following guest faculty.

CHAPTER 2

Introduction to Image Processing and Basic Array Operations

2.1 Studying the Color-Plane information

In this given task, the layers of color-plane were studied. A color image is represented and stored as a set of three matrices each of size MXN. Each matrix represents a colour plane. Thus if an RGB model is used, we have a red image, blue image and a green image and thus 3 corresponding matrices. The RGB and CMY colour models can be visualized as forming a colour cube shown below. Here, red, green and blue form the three orthogonal edges of the cube while cyan, magenta and yellow form the opposite set of edges of the same cube. Note that the corner (S) where the RGB edges meet corresponds to black colour while the corner (W) where the CMY edges meet corresponds to the white colour. Any point within this cube can therefore be specified in terms of 3 coordinates, namely RGB or CMY values. The diagonal line that connects the black and white points will correspond to the grayscale.

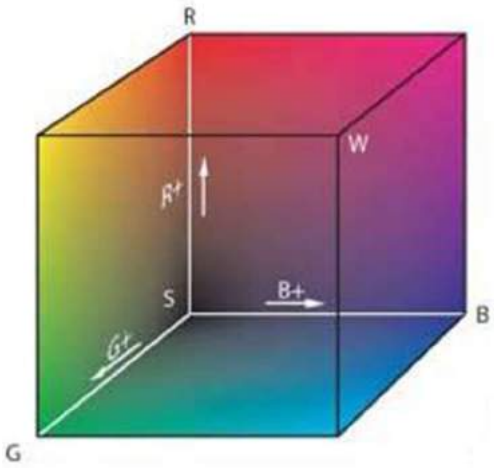


Fig-1: RGB Color Space

Relationship between the RGB and CMY colour model is as follows.

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

There are two ways for processing in colour domain

(a) *Scalar processing*: Process each plane of colour model individually. In this processing, one can process only one or two planes and leave the remaining planes unchanged. For example if we want to modify the red component present in the colour image, then modify only the red plane in the RGB colour model and leave other planes unchanged. After processing the image is converted to RGB space for display.

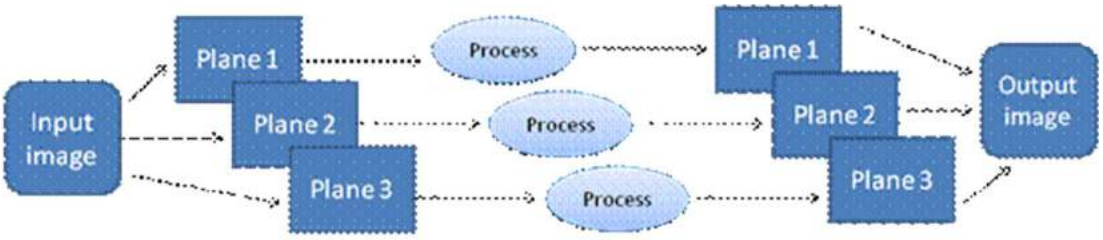


Fig-2: Scalar Processing

(b) *Vector processing* : Consider each pixel of the image as a three element vector, each element corresponding to information from each of the color planes. Instead of processing the image in each plane separately , as we did in the previous case. all planes of colour model are processed simultaneously.

In this task, the following was achieved; where a RGB Layered picture was taken and the aim was to visualize the layers Red, Blue and Green separately.

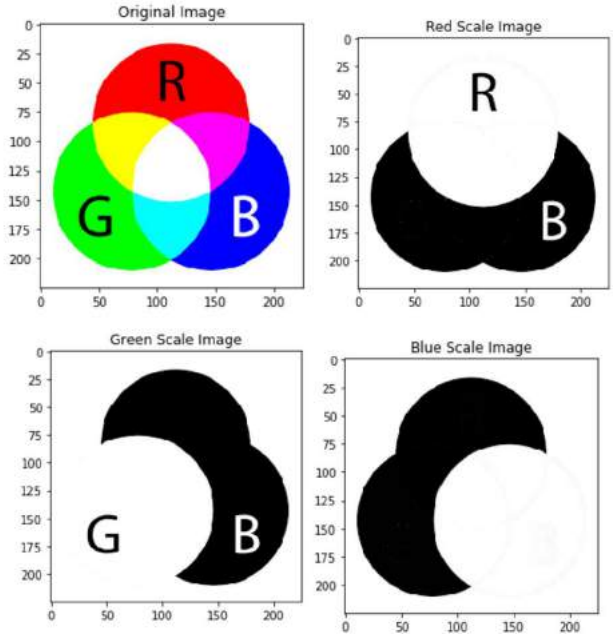


Fig-3: Separate Red, Green and Blue Planes

INFERENCE: The plane that was taken separately, for example, Red, is whitened and the other 2 planes appear black. Similarly it goes for Blue and Green planes.

2.2 Studying the Histogram

The (intensity or brightness) histogram of a digital image with intensity levels in the range [0, L - 1] is a discrete function

$$h(r_k) = n_k, \qquad k = 0, 1, 2, \ldots, L - 1$$

where r_k is the k^{th} intensity value and n_k is the number of pixels in the image with intensity r_k.
The histogram shows how many times a particular intensity level appears. In other words, it shows the distribution of pixel values. Image contrast and brightness can be

assessed from image histogram. The normalized histogram, $p(r_k) = n_k / n$; where n is the total number of pixels in the image, is an estimate of the probability of occurrence of intensity level r_k.

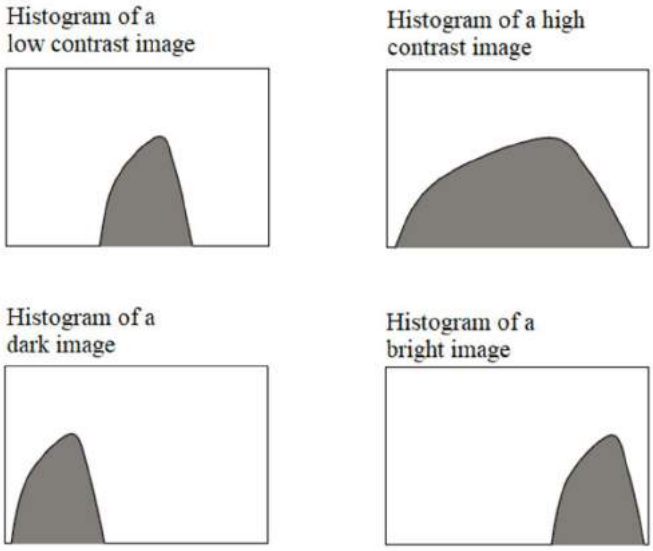


Fig-4: Image contrast and brightness assessment from image histogram.

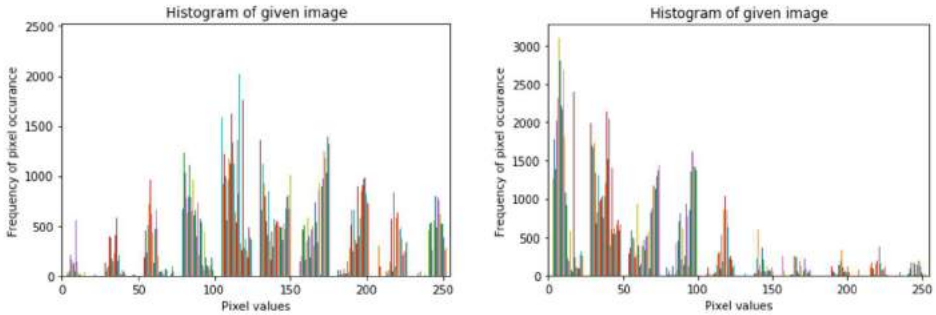
Two types of histogram processing: histogram equalization and histogram matching. The histogram equalization transforms the intensity values such that the histogram of the output image is **nearly flat**. The discrete form of the transformation is:

$$s_k = (L-1)\sum_{j=0}^{k} \frac{n_j}{n}, \qquad k = 0, 1, 2, \ldots, L - 1.$$

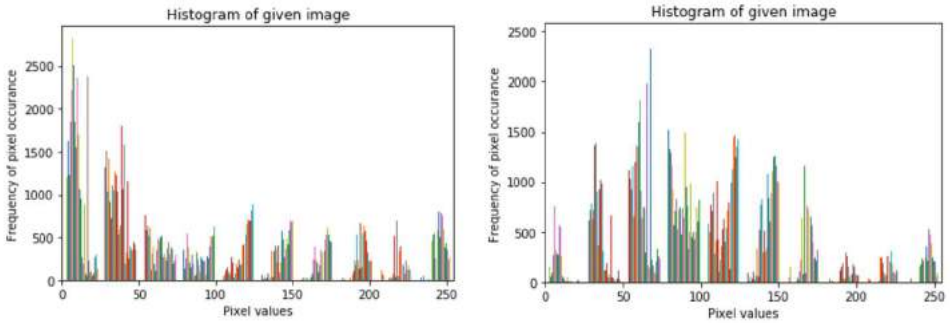
The discrete histogram equalization in general does not yield a flat histogram. The net result is contrast enhancement. The histogram matching transforms the intensity values such that the output image has a specified histogram.

APPLICATIONS: CT lung studies, normalization of MRI images, etc.

In this task, the following was achieved; different images with varying contrast levels (high and low) and brightness levels (dark and bright) were given as input and the histogram of all the images was obtained and assessed.



(a) *(b)*



(c) *(d)*

Fig-5: a-High Brightness, b-Low Brightness, c-High Contrast, d- Low Contrast

INFERENCE: Ideal cases for a Picture with Low Contrast, High Contrast, Dark Image and Bright Image is shown in the fig-3. We can see that the above histograms have the locus of the graphs that is almost similar to that of Ideal case.

2.3 Converting a Grayscale Image to Binary Image

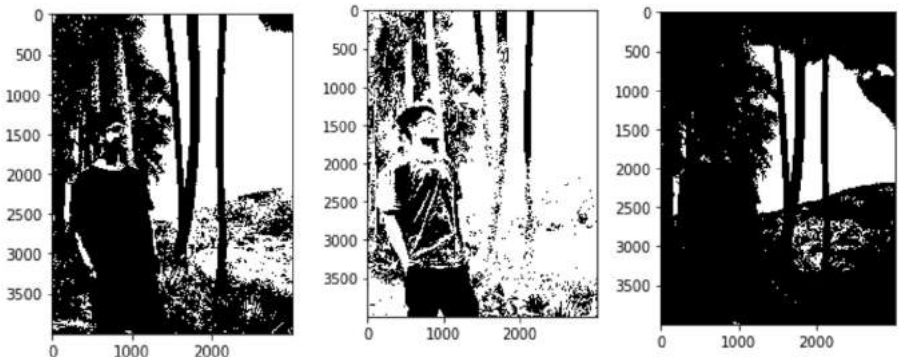
Thresholding is the simplest method of image segmentation and the most common way to convert a grayscale image to a binary image.

In thresholding, we select a threshold value and then all the gray level value which is below the selected threshold value is classified as 0 (black i.e., background) and all the gray levels which are equal to or greater than the threshold value are classified as 1 (white i.e foreground).

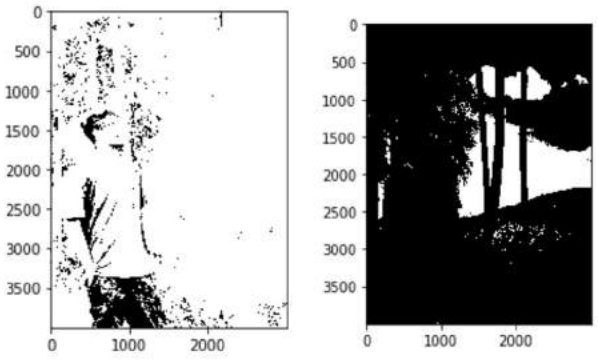
$$g(x,y) = \begin{cases} 1 & \text{if } f(x,y) \geq T \\ 0 & \text{otherwise} \end{cases}$$

Here, g(x, y) represents the threshold image pixel at (x, y) and f(x, y) represents grayscale image pixel at (x, y).

In this task, the following was achieved; The same image at different threshold values was taken. Maximum threshold was given as 255 and minimum was given as 0.



(a) (b) (c)



(d) (e)

Fig-6: Threshold set at a-100, b-50, c-150, d-25, e-175

INFERENCE: If the pixel is more than the threshold, that part of the array will fill up with ones and ones represent white color. If the pixel is less than the threshold, that part of the array won't be filled up with ones, it'll be zeros and zeros represent black color.

CHAPTER 3

Spatial Domain Operations and Image Enhancement using Python

3.1 Gaussian and Median Filter

The Gaussian smoothing operator is a 2-D convolution operator that is used to `blur' images and remove detail and noise. In this sense it is similar to the mean filter, but it uses a different kernel that represents the shape of a Gaussian (`bell-shaped') hump. This kernel has some special properties which are detailed below. The Gaussian distribution in 1-D has the form:

$$G(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}$$

where σ is the standard deviation of the distribution. We have also assumed that the distribution has a mean of zero (i.e. it is centered on the line x=0). The distribution is illustrated in Fig-7.

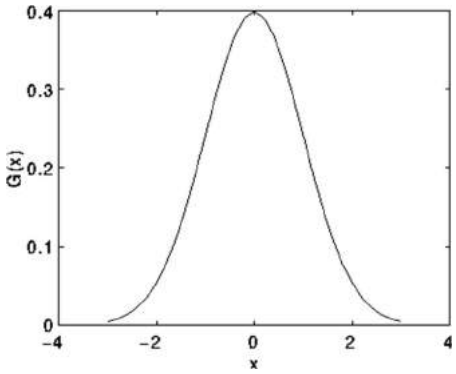


Fig-7: 1-D Gaussian distribution with mean 0 and σ =1

Median filtering is a nonlinear process useful in reducing impulsive, or salt-and-pepper noise. It is also useful in preserving edges in an image while reducing random noise. Impulsive or salt-and pepper noise can occur due to a random bit error in a communication channel. In a median filter, a window slides along the image, and the median intensity value of the pixels within the window becomes the output intensity of the pixel being processed. The median filter is normally used to reduce noise in an image, somewhat like the mean filter. However, it often does a better job than the mean filter of preserving useful detail in the image.

In this task, the following was achieved; The image without noise and with external noise was fed at different σ, size, and internal noise (raised, reasonable & reduced when external noise was not given) values and the outputs were observed. Both the gaussian and median filtering was performed.

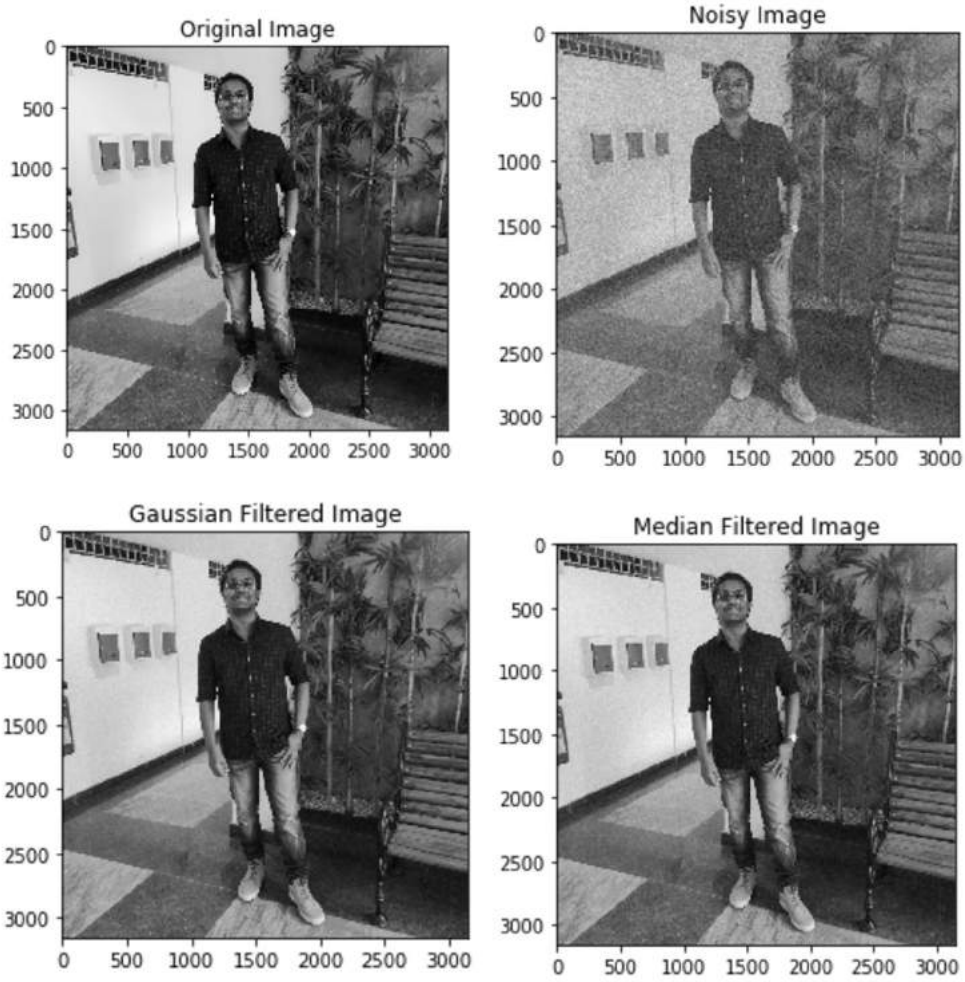
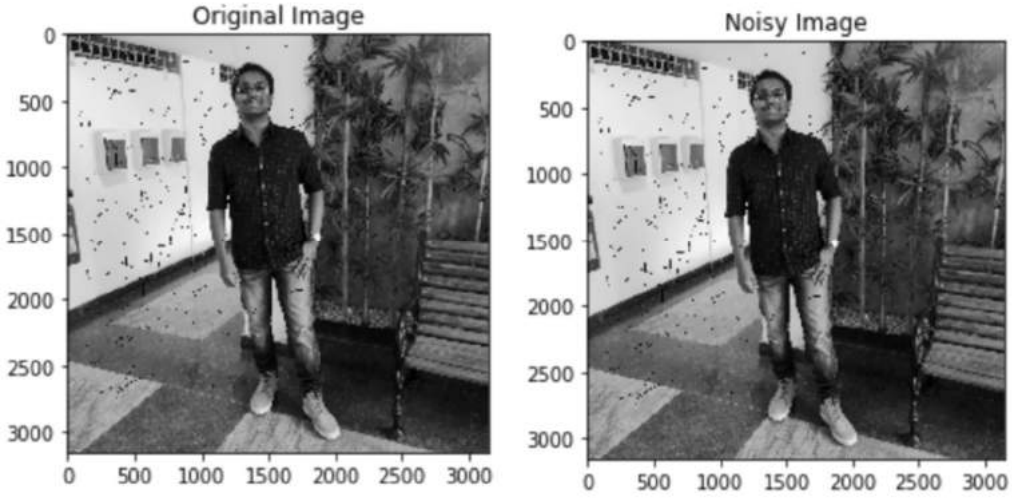


Fig-8: Without External Noise, Internal Noise = 10, sigma=1.5, size=5



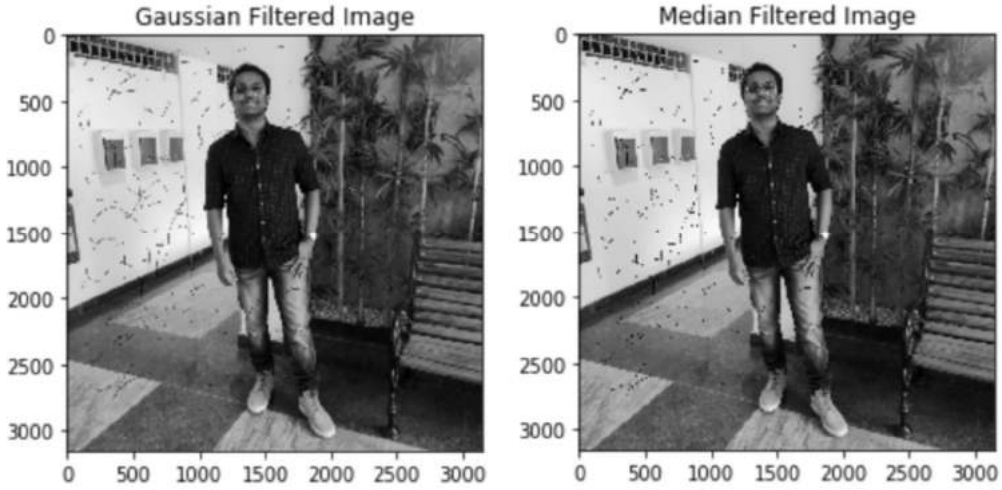


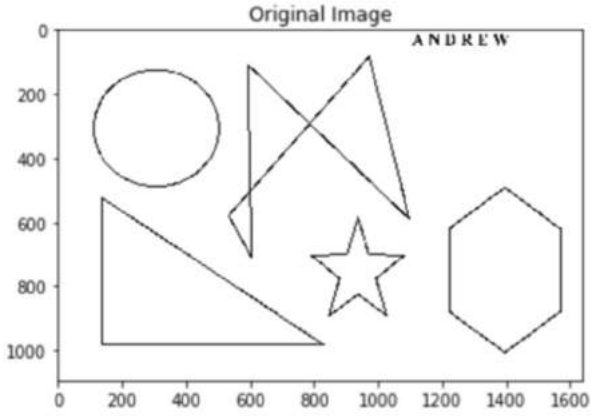
Fig-9: With External Noise, Internal Noise = 0, sigma=1.5, size=5

INFERENCE: We can observe that as we increase the sigma value the picture gets more and more blurred. We can observe that as we increase the median size, these pictures get smoothed but along with that we are starting to observe a loss in the important details of the image. Taking all these into consideration we can infer that gaussian filtering is better than median filtering.

3.2 Minimum and Maximum Filter

The Minimum Filter: The transformation replaces the central pixel with the darkest one in the running window. For example, if you have text that is lightly printed, the minimum filter makes letters thicker.

The Maximum Filter: The maximum and minimum filters are shift-invariant. Whereas the minimum filter replaces the central pixel with the darkest one in the running window, the maximum filter replaces it with the lightest one. For example, if you have a text string drawn with a thick pen, you can make the sign skinnier.
In this task, the following was achieved; Images with clear edges were given and minimum and maximum filter of value 5 was applied and the following was obtained with zero noise.



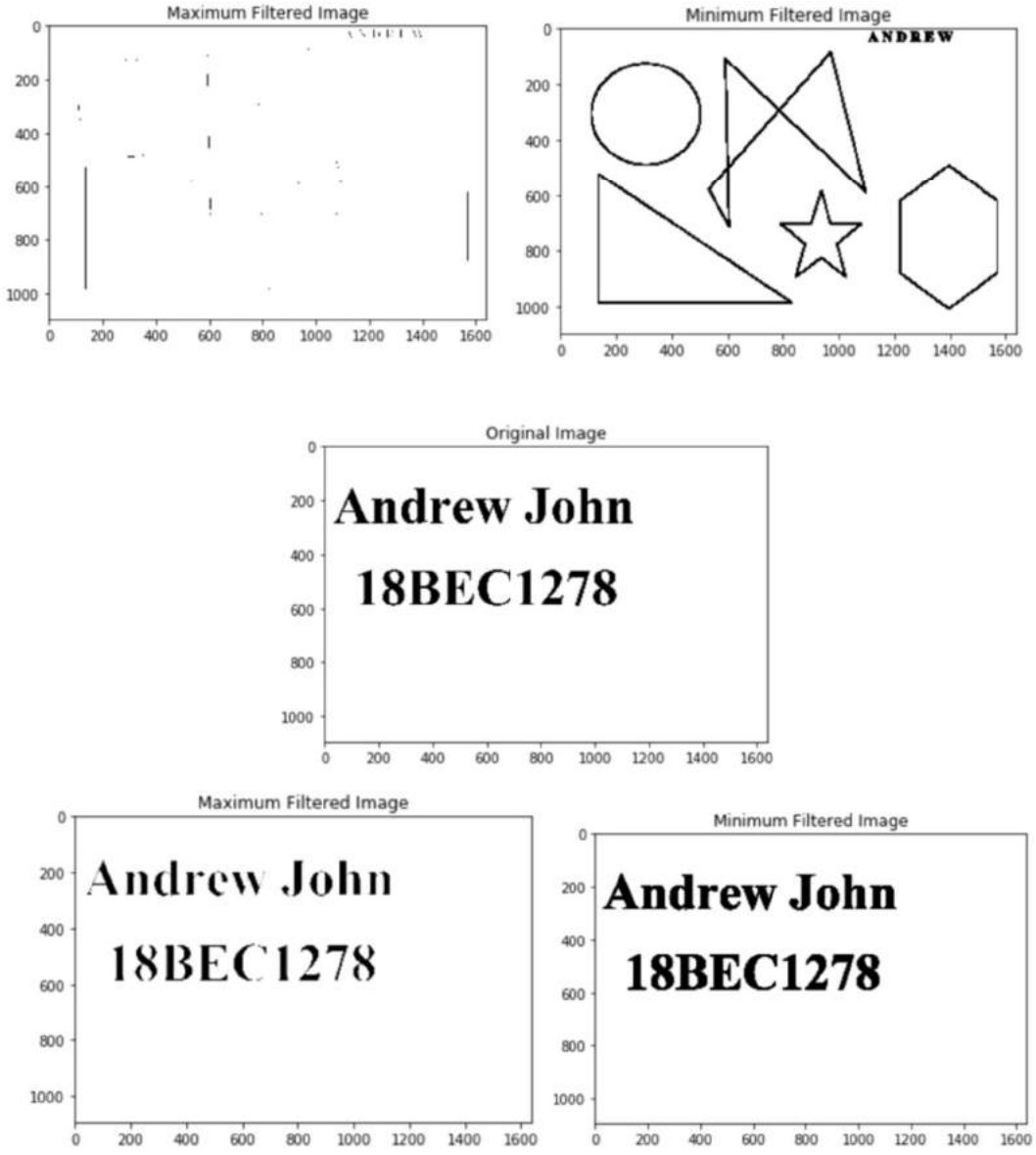


Fig-10: Maximum and Minimum Filters applied to Shapes and Texts

INFERENCE: We can see the clear thinning in Maximum filters meanwhile thickening in Minimum filters as per definition of the filters.

3.3 Sharpening Images - Laplacian Operator on Gaussian Operator

Sharpening as the name suggests is used to sharpen and highlight the edges and make the transitioning of features and details more significant. However sharpening doesn't take into account whether it is highlighting the original features of the image or the noise associated with it. It enhances both.

Blurring vs Sharpening

(a) Blurring : Blurring/smooth is done in spatial domain via taking average of the pixels of its neighbours , thereby producing a blurring effect. It is a process of integration.

(b) *Sharpening* : Sharpening is used to find the difference by the neighborhood and enhance them even more. It is a process of differentiation.

Here Sharpening is done through Laplacian of Gaussian Operator where the image is convolved with the Gaussian smoothing kernel and the resulting smoothened image is convolved with a Laplacian kernel.

In this task, the following was achieved; sharpening of images in both Black and White format and Color format with external noise.

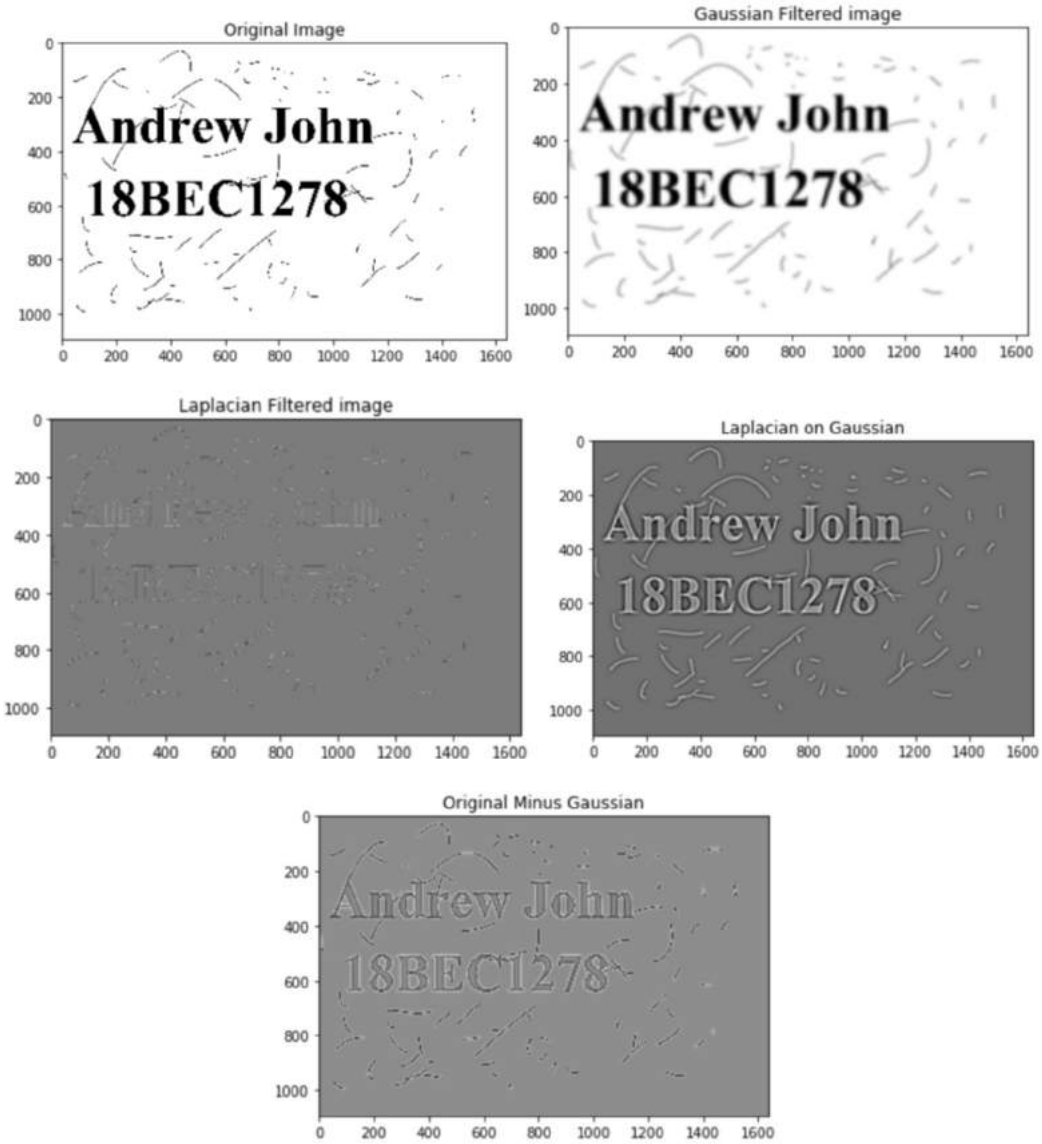


Fig-11: Color Image treatment with External Noise, Sharpened (Sigma=5)

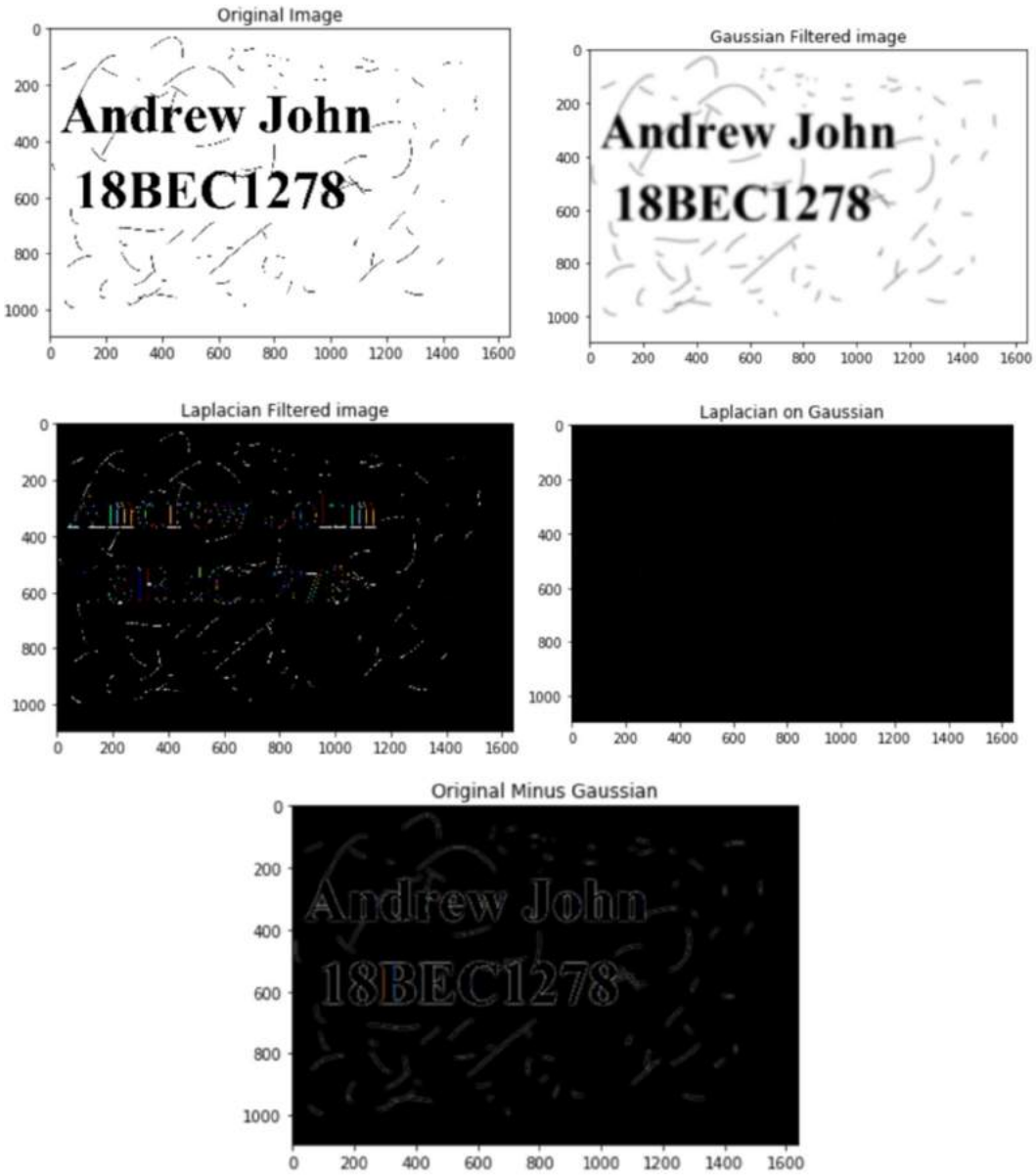


Fig-12: Black & White Image treatment with External Noise, Sharpened (Sigma=5)

3.4 Sobel and Prewitt Operators

Prewitt Operator: Prewitt operator is used for edge detection in an image. It detects two types of edges

- Horizontal Edges
- Vertical Edges

Edges are calculated by using the difference between corresponding pixel intensities of an image. All the masks that are used for edge detection are also known as derivative masks. Prewitt operator provides us two masks one for detecting edges in horizontal direction and another for detecting edges in an vertical direction.

Sobel Operator: The sobel operator is very similar to the Prewitt operator. It is also a derivative mask and is used for edge detection. Like Prewitt operator sobel operator is also used to detect two kinds of edges in an image:

- Vertical direction
- Horizontal direction

The difference between Sobel and Prewitt operators is that in Sobel operators the coefficients of masks are not fixed and they can be adjusted according to our requirement unless they do not violate any property of derivative masks.

In this task, the following was achieved; applying Sobel and Prewitt Operators in both Black and White format and Color format with no external Noise (the same results will be yielded with external noise, where the horizontal and vertical noises will be enhanced horizontally and vertically, respectively)

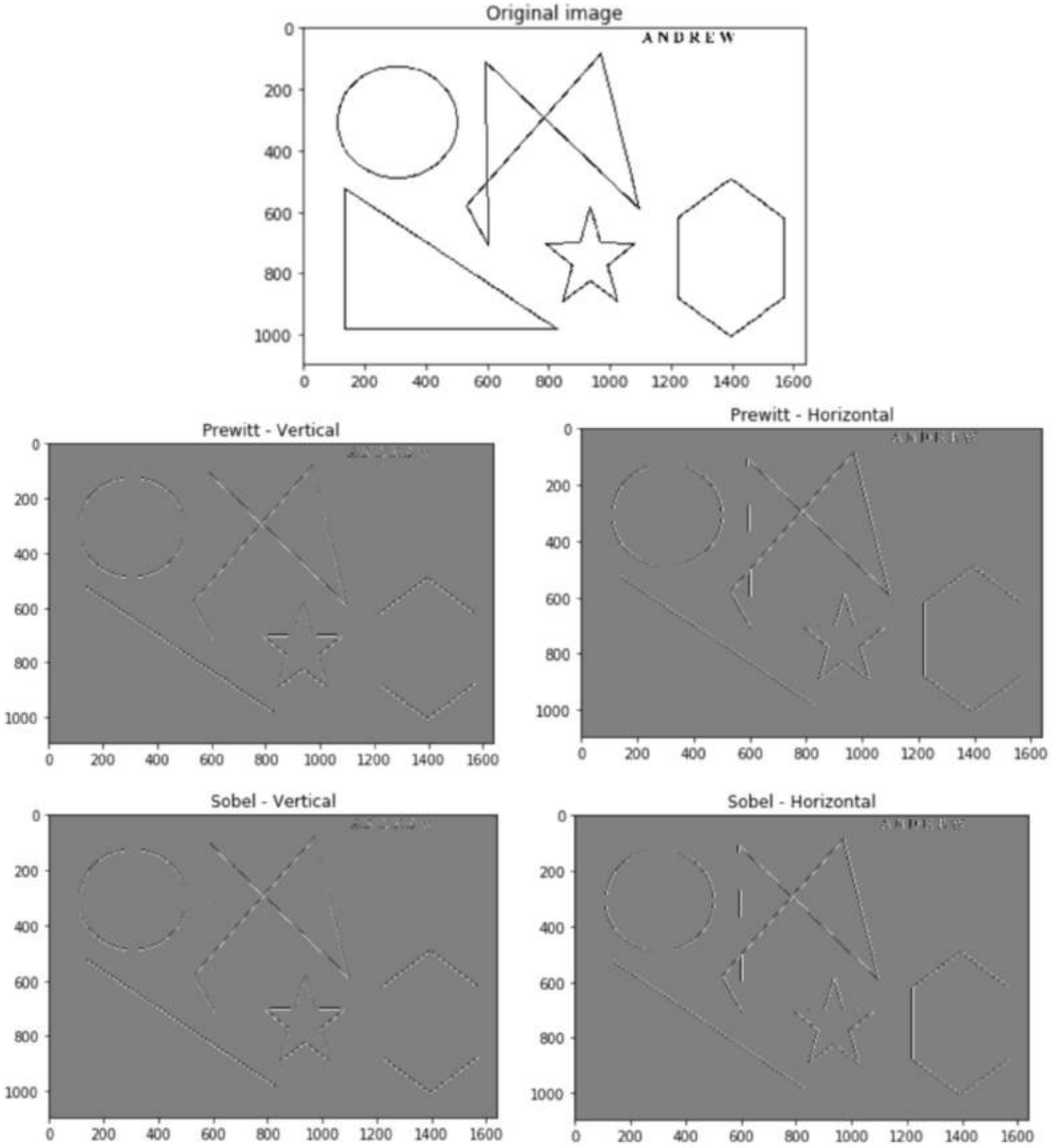


Fig-13: Color Image Treatment with no External Noise

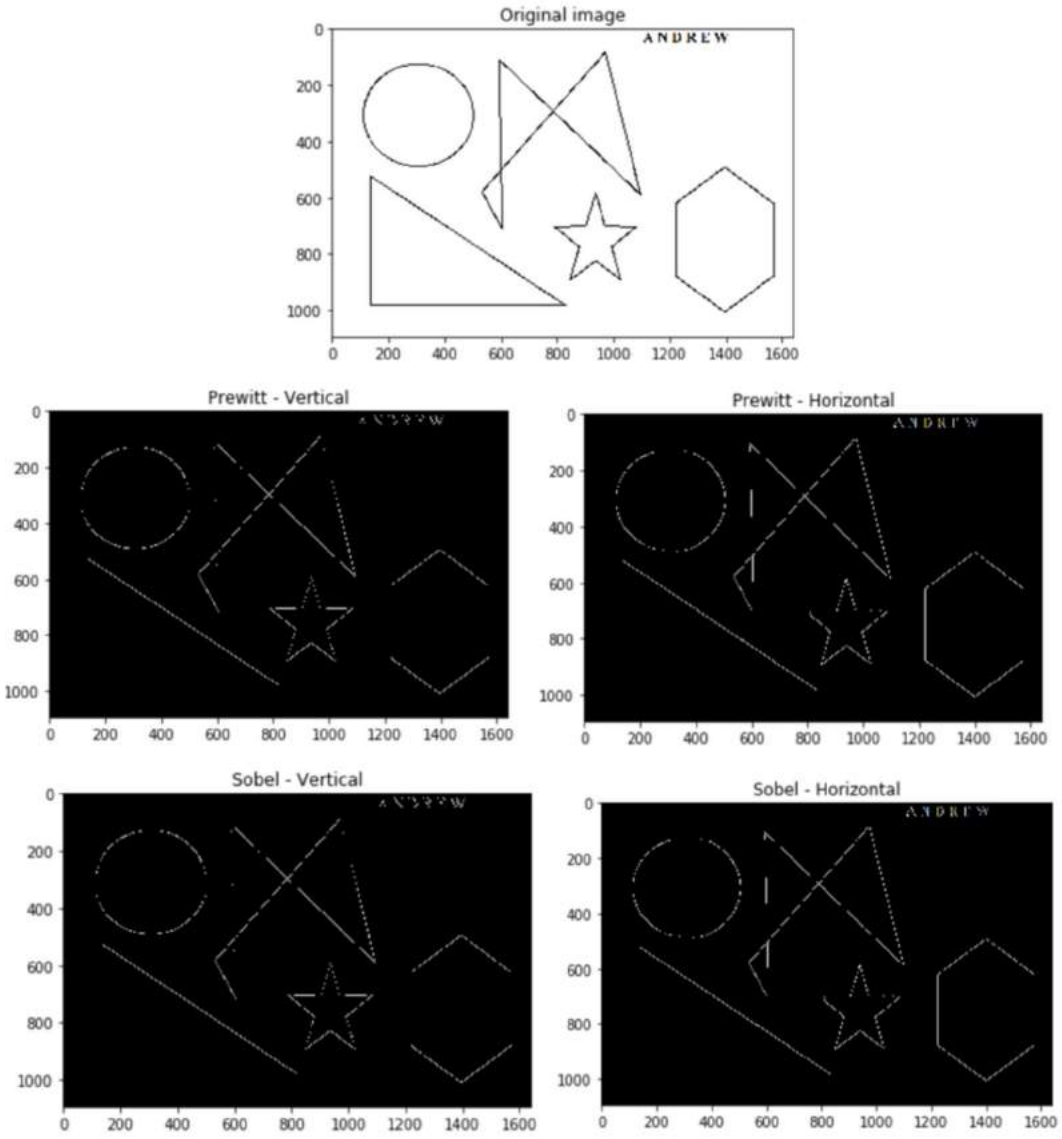


Fig-13: Black & White Image Treatment with no External Noise

3.5 Brightness Function

Changing the brightness of an image is a commonly used point operation. In this operation, the value of each and every pixel in an image should be increased/decreased by a constant. To change the brightness of a video, the same operation should be performed on each frame in the video.

If you want to increase the brightness of an image, you have to add some positive constant value to each and every pixel in the image. If you want to decrease the brightness of an image, you have to subtract some positive constant value from each and every pixel in the image.

In this task, the following was achieved; Brightness function of an Image was taken and drastic change in histogram of the picture was observed.

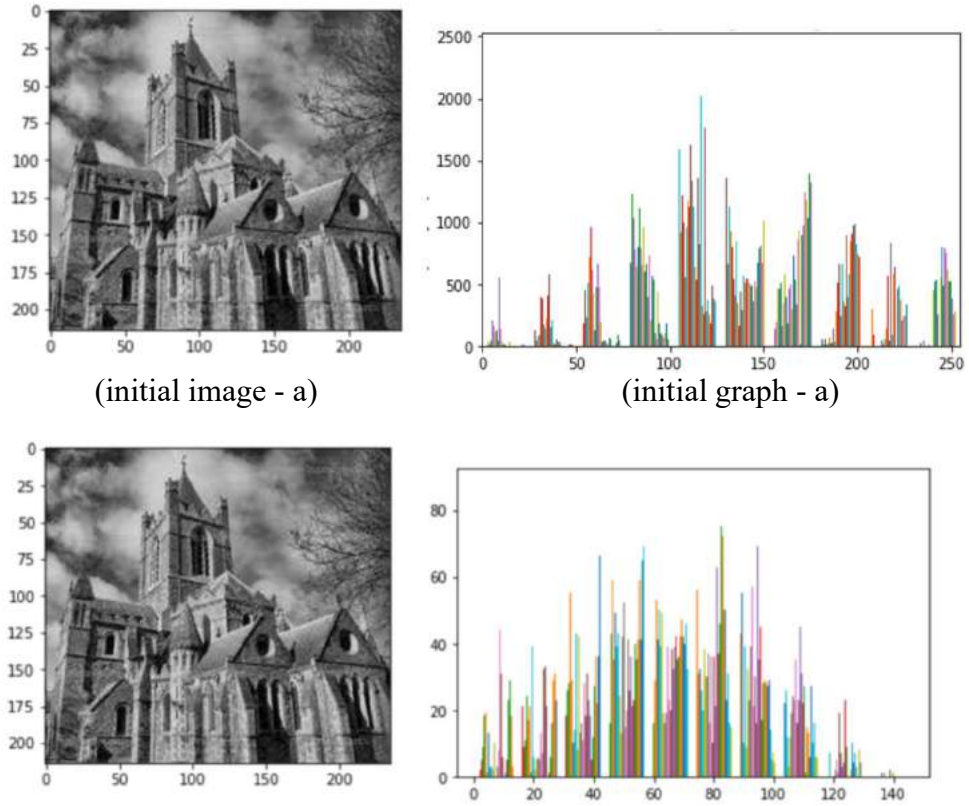


Fig-14: After applying Brightness Function, drastic change is observed in Graph - b

INFERENCE: Here we can observe that , as we add 100 images to the original image, Histogram values have jumped from 140 to 240. Also the image looks a little bit brighter.

3.6 Gamma Transformation

Gamma correction controls the overall brightness of an image. Images which are not properly corrected can look either bleached out, or too dark. Trying to reproduce colors accurately also requires some knowledge of gamma. Varying the amount of gamma correction changes not only the brightness, but also the ratios of red to green to blue.

Power-law (Gamma) transformation: The general form of the power-law transformation is

$$s = cr^{\gamma}$$

where c and γ are positive constants. This transformation is also known as gamma correction. A variety of devices used for image capturing, printing and display respond according to power law. The optimal value for γ is device-dependent.

In this task, the following was achieved; Gamma function was used to the input image where as we increase the value of Gamma we can see the image changes to a bit darker spectrum.

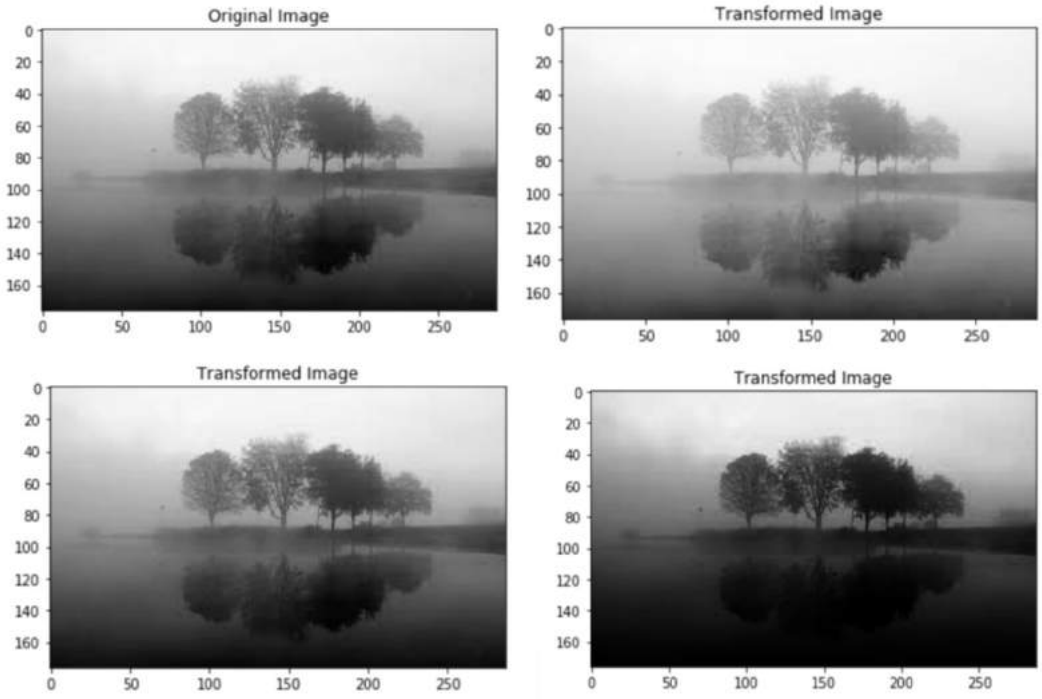


Fig - 15: Gamma Function at different γ values (Original Image, and γ at 0.5, 1.2, 2,2)

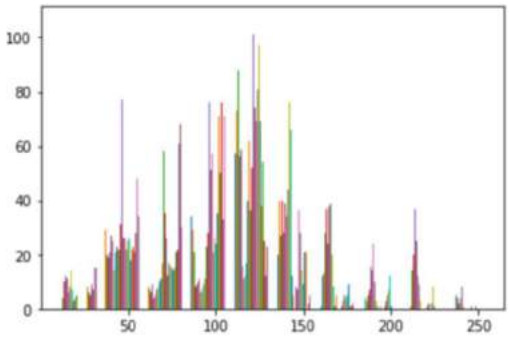
INFERENCE: As the value of gamma increases the less bright/foggy image becomes dark. We can see that the trees are getting darker and darker as we increase the gamma value

3.7 Contrast Stretching

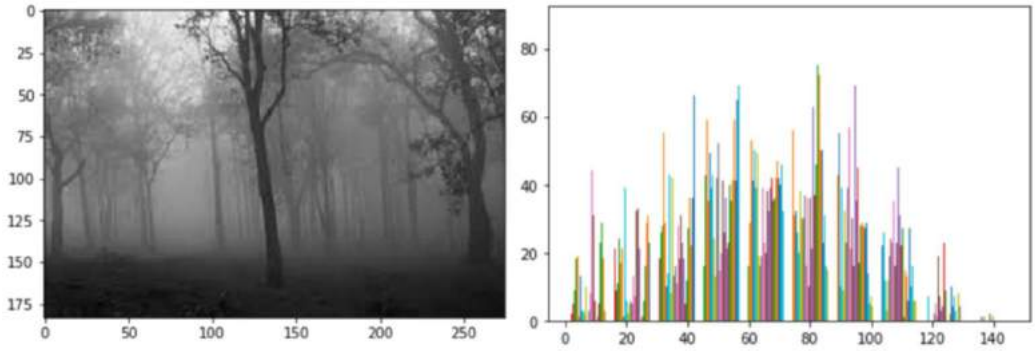
Contrast stretching (often called normalization) is a simple image enhancement technique that attempts to improve the contrast in an image by `stretching' the range of intensity values it contains to span a desired range of values, e.g. the the full range of pixel values that the image type concerned allows. It differs from the more sophisticated histogram equalization in that it can only apply a linear scaling function to the image pixel values. As a result the 'enhancement' is less harsh. (Most implementations accept a gray level image as input and produce another gray level image as output.)

In this task, the following was achieved; Contrast stretching/normalization was performed where input image histogram should have range changes and the image needs to be a bit brighter than the original initial image.

(initial image - a)



(initial graph - a)



(normalized image - b) (normalized graph - b)

Fig-16: After performing Contrast Stretching, range change is observed in Graph - b

INFERENCE: In the above histograms we can observe a range of gaps have been stretched and also the background is more clear than the first pic.

3.8 Log Transformation

Logarithmic transformation of an image is one of the gray level image transformations. Log transformation of an image means replacing all pixel values, present in the image, with its logarithmic values. Log transformation is used for image enhancement as it expands dark pixels of the image as compared to higher pixel values. The general form of log transformation function is

$$s = T(r) = c*\log(1+r)$$

where, ‘s’ and ‘r’ are the output and input pixel values and c is the scaling constant represented by the following expression (for 8-bit)

$$c = 255/(\log(1 + \text{max\_input\_pixel\_value}))$$

The value of c is chosen such that we get the maximum output value corresponding to the bit size used. e.g for 8 bit image, c is chosen such that we get max value equal to 255.

In this task, the following was achieved; when the log transformation was applied, the histogram value of the original image must be flipped and needs to be on the higher side when the initial is on the lower side and vice-versa.

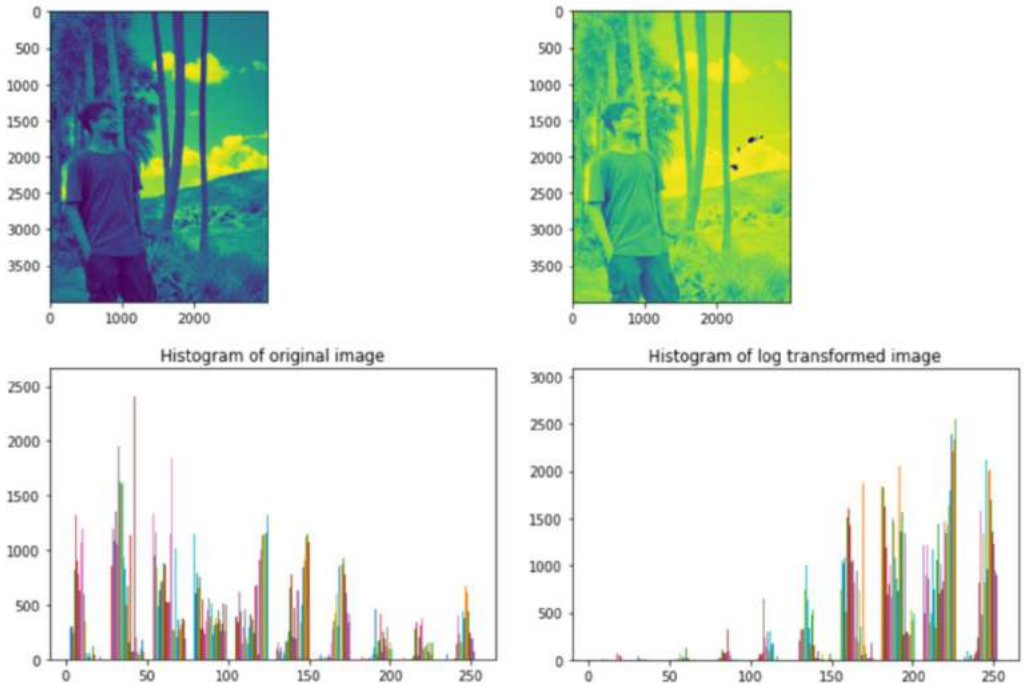


Fig-17: After applying Log transformation, values have been flipped

INFERENCE: The values have been flipped and the image has been transformed when the log function was applied.

CHAPTER 4

Image Segmentation and Image Morphology using Python

4.1 Canny Edge Detection on the Captured Images(s)

The Canny edge detector is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images. The Canny filter is a multi-stage edge detector. It uses a filter based on the derivative of a Gaussian in order to compute the intensity of the gradients.The Gaussian reduces the effect of noise present in the image. Then, potential edges are thinned down to 1-pixel curves by removing non-maximum pixels of the gradient magnitude. Finally, edge pixels are kept or removed using hysteresis thresholding on the gradient magnitude.

The Canny has three adjustable parameters: the width of the Gaussian (the noisier the image, the greater the width), and the low and high threshold for the hysteresis thresholding.

In this task, the following was achieved; Canny Edge Detection of input images was detected in different RGB Border colors.

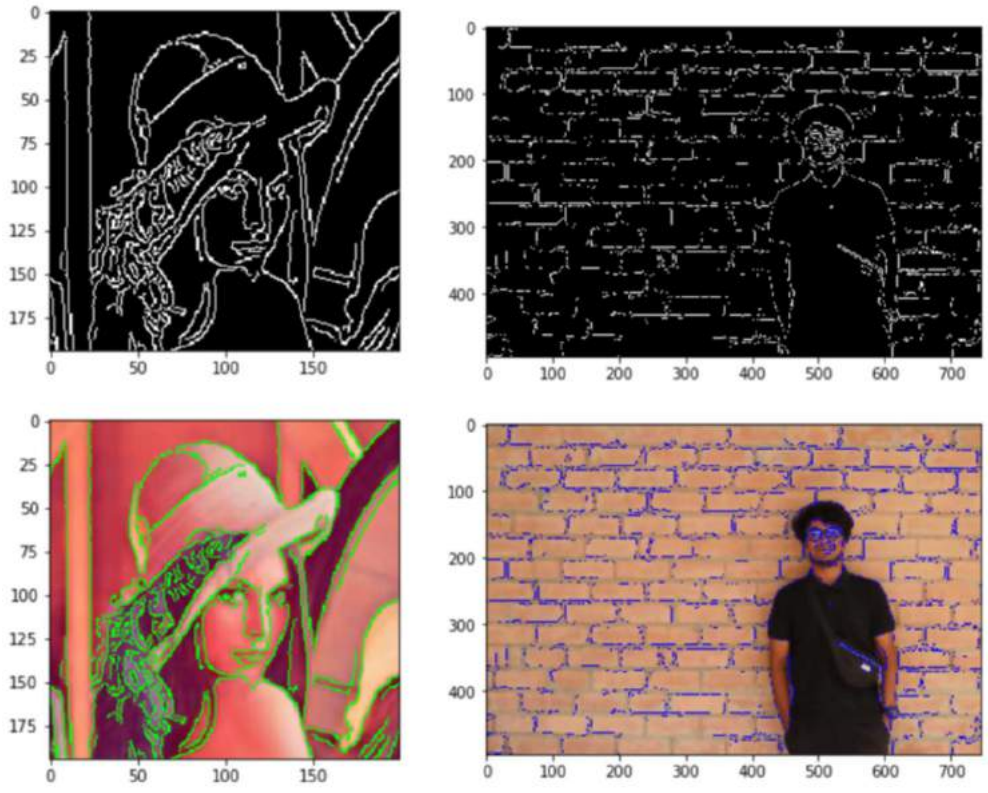


Fig-18: Canny Edge Detected Images

The general criteria for edge detection include:

- Detection of edge with low error rate, which means that the detection should accurately catch as many edges shown in the image as possible

- The edge point detected from the operator should accurately localize on the center of the edge.
- A given edge in the image should only be marked once, and where possible, image noise should not create false edges.

INFERENCE: Threshold value is image dependent. As you increase the upper threshold value more the image is smoothened .

4.2 Harris Corner Detection on the Captured Images(s)

The Harris corner detector is a corner detection operator that is commonly used in computer vision algorithms to extract corners and infer features of an image. Harris' corner detector takes the differential of the corner score into account with reference to direction directly, instead of using shifting patches for every 45 degree angle, and has been proved to be more accurate in distinguishing between edges and corners.

Commonly, Harris corner detector algorithm can be divided into five steps.

1. Color to grayscale
2. Spatial derivative calculation
3. Structure tensor setup
4. Harris response calculation
5. Non-maximum suppression

In this task, the following was achieved; Corners was detected when the Harris Corner function was applied along pictures that had curvature as well as linear pictures.

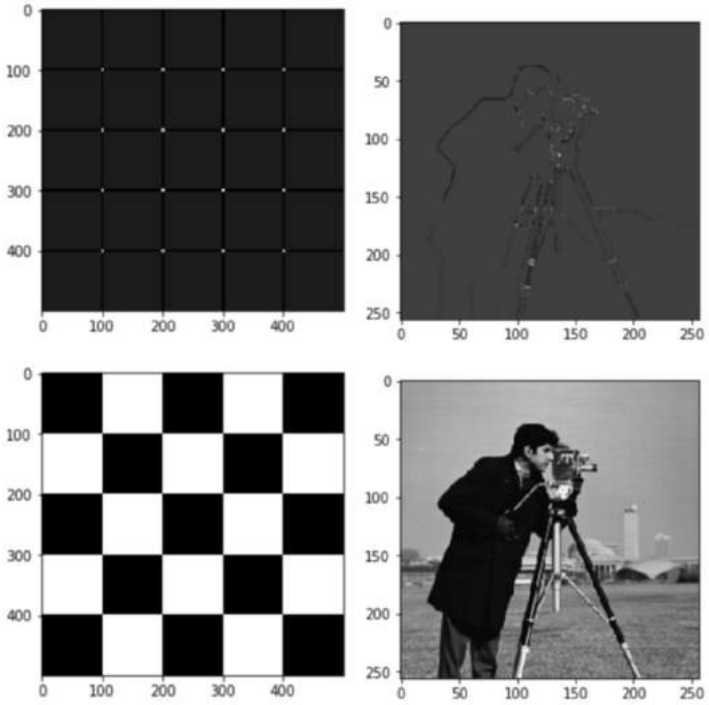


Fig-19: Harris Corner Detected Images

INFERENCE: When we upload the picture , through Harris corner detection it detects all the edges of the image and finally the image is converted into RGB image where only red, blue ,green colors of the image are shown

4.3 Hough Line Detection on Task - 1's Edge Detected Image(s)

The Hough transform is a feature extraction technique used in image analysis, computer vision, and digital image processing. The purpose of the technique is to find imperfect instances of objects within a certain class of shapes by a voting procedure.

This voting procedure is carried out in a parameter space, from which object candidates are obtained as local maxima in a so-called accumulator space that is explicitly constructed by the algorithm for computing the Hough transform. The simplest case of Hough transform is detecting straight lines. In general, the straight line y = mx + b can be represented as a point (b, m) in the parameter space.

In this task the following was achieved; On the edge detected images from Canny Detection, Hough transform was applied and lines along the image were highlighted in RGB Colors.

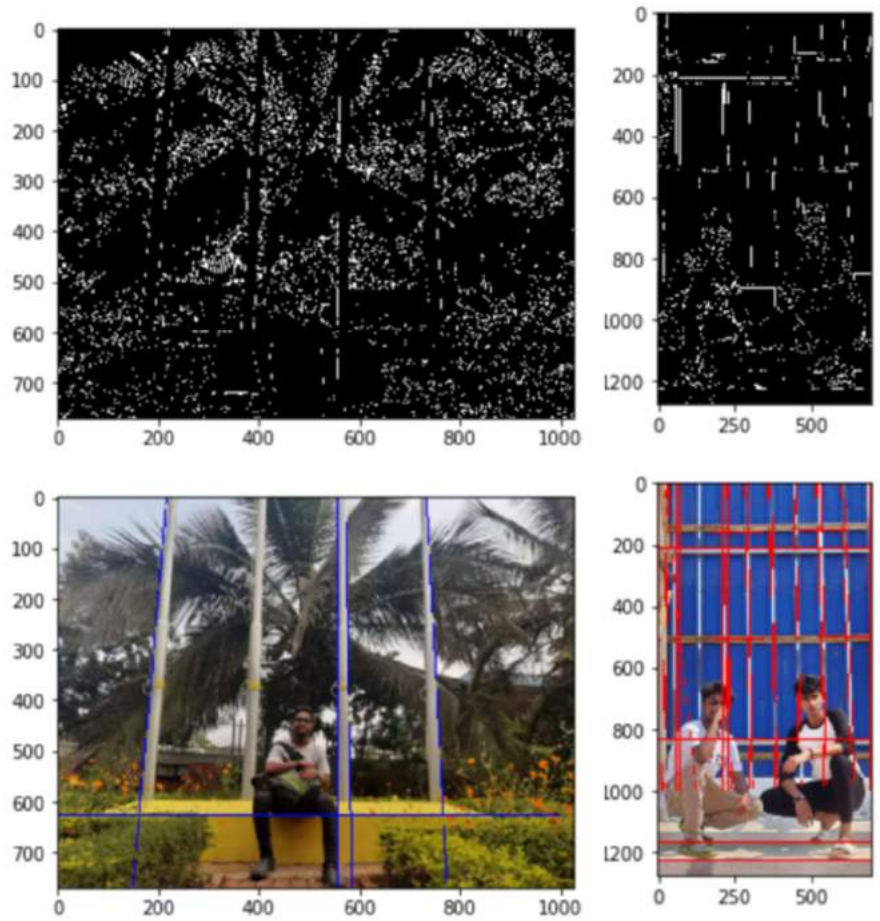


Fig-20: Hough Line Transformed Images

INFERENCE : As we increase the value of 'd' the image gets more blurred. This makes it tough to detect the edges . As a result the red lines in the image get reduced

whereas in the image with ‘d’ value as 2 , the image is more clear and hence it detects more straight lines as edges and there are more red lines in the output.
Hence we have applied the HOUGH transform and detected the lines

4.4 Hough Circle Detection on the Captured Images(s)

The circle Hough Transform (CHT) is a basic feature extraction technique used in digital image processing for detecting circles in imperfect images. The circle candidates are produced by “voting” in the Hough parameter space and then selecting local maxima in an accumulator matrix.

In a two-dimensional space, a circle can be described by:

$$(x-a)^2 + (y-b)^2 = r^2$$

where (a,b) is the center of the circle, and r is the radius. If a 2D point (x,y) is fixed, then the parameters can be found according to the above equation. The parameter space would be three dimensional, (a, b, r). And all the parameters that satisfy (x, y) would lie on the surface of an inverted right-angled cone whose apex is at (x, y, 0).

In this task, the following was achieved; circles were detected similarly to how line was detected in the above Hough transformation. Both real life images and paint images were given as input. The radius of each circle was also detected.

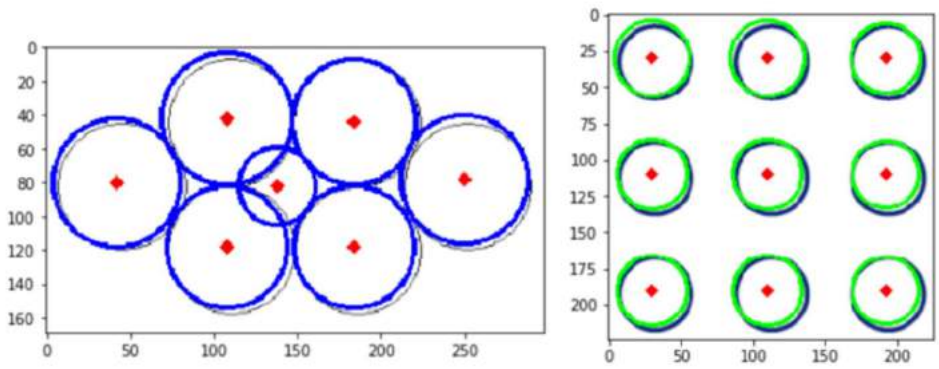


Fig-21: Hough Circle Transformed Images

4.5 k-Means Clustering

k-Means clustering is a method of vector quantization, originally from signal processing, that aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean (cluster centers or cluster centroid), serving as a prototype of the cluster.

Here k-Means is used for Image Segmentation, where Image segmentation is the process of partitioning an image into multiple different regions (or segments). The goal is to change the representation of the image into an easier and more meaningful image. Here k-Means aims to partition N observations into K clusters in which each observation belongs to the cluster with the nearest mean. A cluster refers to a collection of data points aggregated together because of certain similarities. For image segmentation, clusters here are different image colors.

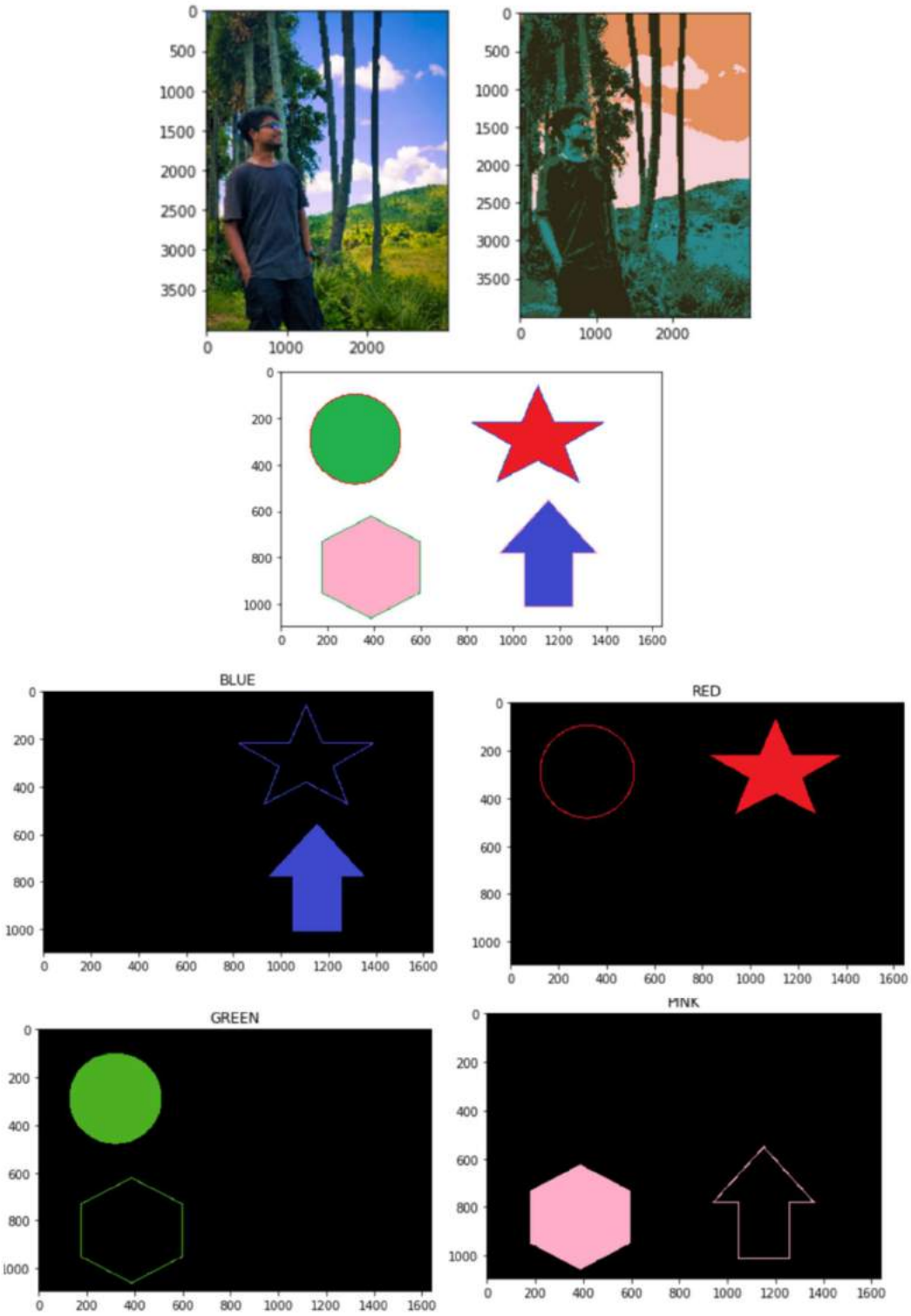


Fig-22: k-Means clustering used for Image Segmentation and Color Segmentation

INFERENCE: When we upload images we note that only the red , blue ,green color portions of the image are highlighted since we cluster the RGB Colors into a segment. Hence we have performed k-means clustering on images and observed the output.

4.6 Morphological Operations

Morphological Operations is a broad set of image processing operations that process digital images based on their shapes. In a morphological operation, each image pixel is corresponding to the value of another pixel in its neighborhood. By choosing the shape and size of the neighborhood pixel, you can construct a morphological operation that is sensitive to specific shapes in the input image.

Types of Morphological operations:

- *Dilation*: Dilation adds pixels on the object boundaries.
- *Erosion*: Erosion removes pixels on object boundaries.
- *Open*: The opening operation erodes an image and then dilates the eroded image, using the same structuring element for both operations.
- *Close*: The closing operation dilates an image and then erodes the dilated image, using the same structuring element for both operations.

In this task, the following was achieved; Dilation, Opening, Erosion, and Closing was performed on the input image.

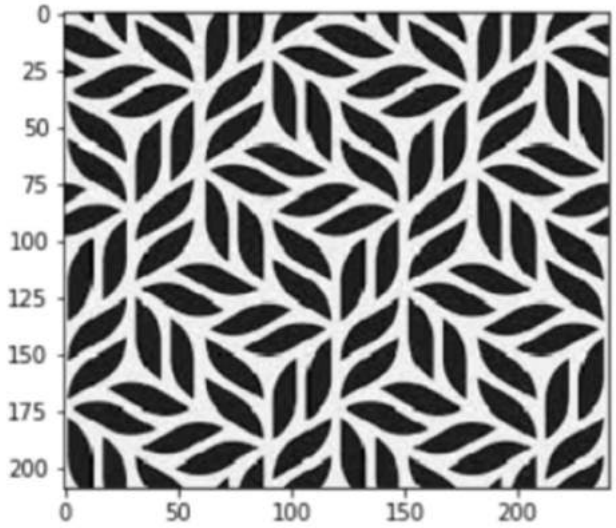


Fig-23: Input Image for Morphological Operations

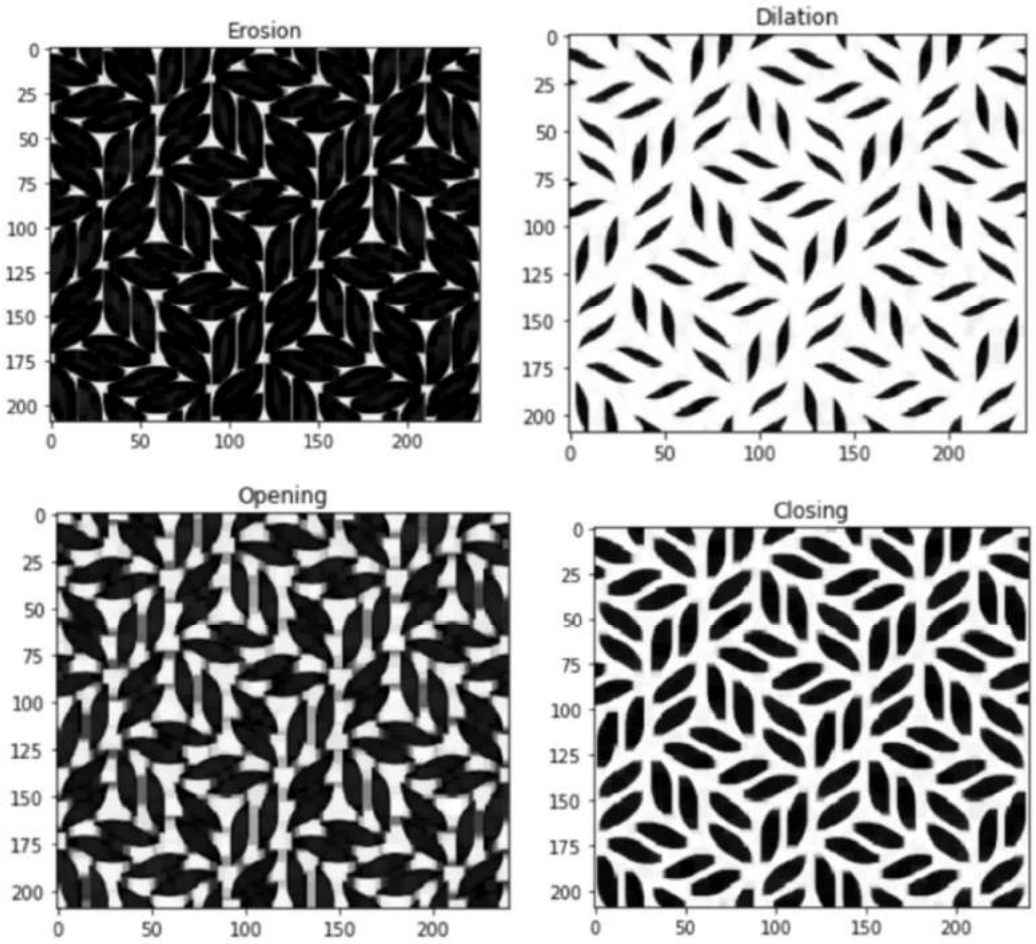


Fig-24: Image after Morphological Operations

INFERENCE: Erosion has thickened (removed pixels) each petal boundaries while Dilation (added pixels) has thinned the boundaries of the petals, Opening has increased the pixel size of each element in the image whereas Closing has refined each element in the image.

CHAPTER 5

Deep Learning and CNN for Vision Applications

Concepts of Neural Network, Deep Learning, CNN & Transfer Learning was discussed by the Guest Lecturer

Neural networks are parallel computing devices, which is basically an attempt to make a computer model of the brain. The main objective is to develop a system to perform various computational tasks faster than the traditional systems. These tasks include pattern recognition and classification, approximation, optimization, and data clustering.

Artificial Neural Network is an efficient computing system whose central theme is borrowed from the analogy of biological neural networks. ANNs are also named as "artificial neural systems," or "parallel distributed processing systems," or "connectionist systems." ANN acquires a large collection of units that are interconnected in some pattern to allow communication between the units. These units, also referred to as nodes or neurons, are simple processors which operate in parallel.

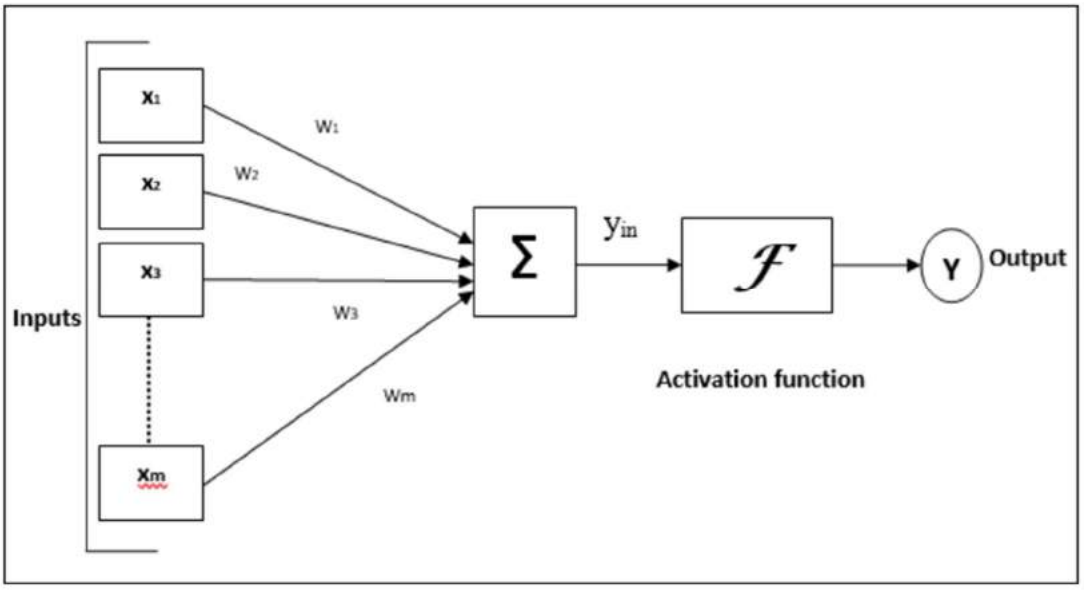


Fig-25: Model of an Artificial Neural Network

Deep structured learning or hierarchical learning or deep learning in short is part of the family of machine learning methods which are themselves a subset of the broader field of Artificial Intelligence. Deep learning is a class of machine learning algorithms that use several layers of nonlinear processing units for feature extraction and transformation. Each successive layer uses the output from the previous layer as input.

Deep neural networks, deep belief networks and recurrent neural networks have been applied to fields such as computer vision, speech recognition, natural language processing, audio recognition, social network filtering, machine translation, and

bioinformatics where they produced results comparable to and in some cases better than human experts have.

Convolutional Neural networks are designed to process data through multiple layers of arrays. This type of neural networks is used in applications like image recognition or face recognition. The primary difference between CNN and any other ordinary neural network is that CNN takes input as a two-dimensional array and operates directly on the images rather than focusing on feature extraction which other neural networks focus on.

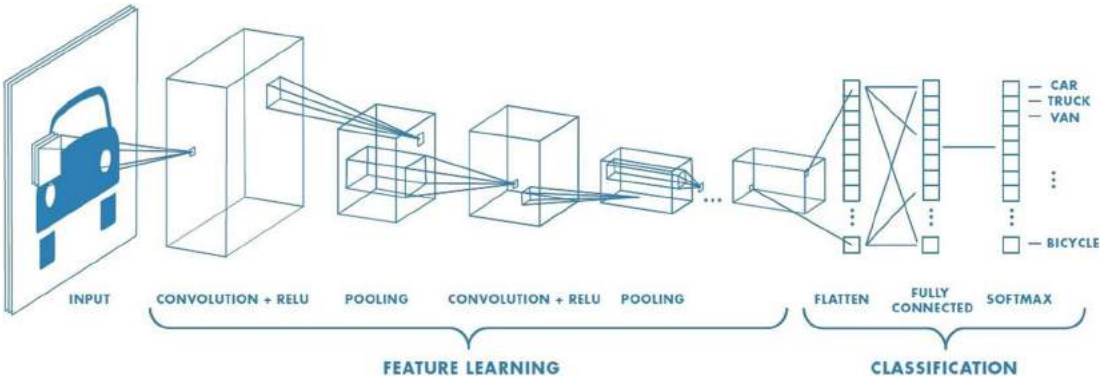


Fig-26: Convolutional Neural Network

Transfer learning (TL) is a research problem in machine learning (ML) that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem. This area of research bears some relation to the long history of psychological literature on transfer of learning, although practical ties between the two fields are limited. From the practical standpoint, reusing or transferring information from previously learned tasks for the learning of new tasks has the potential to significantly improve the sample efficiency of a reinforcement learning agent.

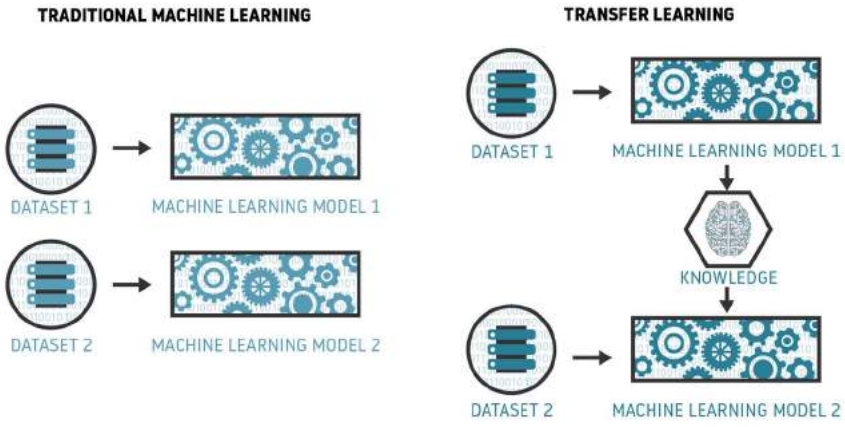


Fig-27: Transfer Learning and Traditional ML Difference.

CHAPTER 6

State-of-the-art Computer Vision Applications

Computer Vision in Manufacturing-

- Productivity Analytics:
    - Productivity analytics track the impact of workplace change, how employees spend their time and resources, and implement various tools. Such data can provide valuable insight into time management, workplace collaboration, and employee productivity. Computer Vision lean management strategies aim to objectively quantify and assess processes with cameras-based vision systems.

- Quality Management:
    - Smart camera applications provide a scalable method to implement automated visual inspection and quality control of production processes and assembly lines in smart factories. Hereby, deep learning uses real-time object detection to provide superior results (detection accuracy, speed, objectiveness, reliability) compared to laborious manual inspection.

Computer Vision in Healthcare-

- Cancer Detection:
    - Machine learning is incorporated in medical industries for purposes such as breast and skin cancer detection. For instance, image recognition allows scientists to detect slight differences between cancerous and non-cancerous images and diagnose data from magnetic resonance imaging (MRI) scans and inputted photos as malignant or benign.
- COVID-19 diagnosis:
    - Computer Vision can be used for coronavirus control. Multiple deep learning computer vision models exist for x-ray based COVID-19 diagnosis. The most popular one for detecting COVID-19 cases with digital chest x-ray radiography (CXR) images is named COVID-Net and was developed by Darwin AI, Canada.

Computer Vision in Agriculture-

- Farm Automation:
    - Technologies such as harvest, seeding, and weeding robots, autonomous tractors, and vision systems to monitor remote farms, drones for visual inspection can maximize productivity with labor shortages. The profitability can be significantly increased by automating manual inspection with AI vision, reducing the ecological footprint, and improving decision-making processes.

- Yield Assessment:

- Through the application of computer vision technology, the functions of soil management, maturity detection, and yield estimation for farms have been realized. Moreover, the existing technology can be well applied to methods such as spectral analysis and deep learning.
- Most of these methods have the advantages of high precision, low cost, good portability, good integration, and scalability and can provide reliable support for management decision making. An example is the estimation of citrus crop yield via fruit detection and counting using computer vision.

Computer Vision in Transportation-

- Moving Violations Detection:
  - Law enforcement agencies and municipalities are increasing the deployment of camera-based roadway monitoring systems with the goal of reducing unsafe driving behavior. Probably the most critical application is the detection of stopped vehicles in dangerous areas.
  - Also, there is increasing use of computer vision techniques in smart cities that involve automating the detection of violations such as speeding, running red lights or stop signs, wrong-way driving, and making illegal turns.

- Traffic Flow Analysis:
  - Traffic flow analysis has been studied extensively for intelligent transportation systems (ITS) using invasive methods (tags, under-pavement coils, etc.) and non-invasive methods such as cameras.
  - With the rise of computer vision and AI, video analytics can now be applied to the ubiquitous traffic cameras, which can generate a vast impact in ITS and smart cities. The traffic flow can be observed using computer vision means and measure some of the variables required by traffic engineers.

Computer Vision in Retail-

- Customer Tracking:
  - Deep learning algorithms can process the video streams in real-time to analyze the customer footfall in retail stores. Camera-based methods allow re-using the video stream of common, inexpensive security surveillance cameras. Machine learning algorithms detect people anonymously and contactless to analyze time spent in different areas, waiting times, queueing time, and assess the service quality.

- Theft Detection:
  - Retailers can detect suspicious behavior such as loitering or accessing areas that are off-limits using computer vision algorithms that are autonomously analyzing the scene.
- Social Distancing:
  - To ensure safety precautions are being followed, companies are using distance detectors. A camera tracks employee or customer movement and uses depth sensors to assess the distance between them. Then,

depending on their position, the system draws a red or green circle around the person. Learn more about Social Distancing Monitoring with deep learning.

Computer Vision in Sports-

- Player Pose Tracking-
    - AI vision can recognize patterns between human body movement and pose over multiple frames in video footage or real-time video streams. For example, human pose estimation has been applied to real-world videos of swimmers where single stationary cameras film above and below the water surface. Those video recordings can be used to quantitatively assess the athletes' performance without manually annotating the body parts in each video frame. Thus, Convolutional Neural Networks are used to automatically infer the required pose information and detect the swimming style of an athlete.
- Goal-Line Technology
    - Camera-based systems can be used to determine if a goal has been scored or not to support the decision-making of referees. Unlike sensors, the AI vision-based method is noninvasive and does not require changes to the typical football devices.
    - Such Goal-Line Technology systems are based on high-speed cameras whose images are used to triangulate the ball's position. A ball detection algorithm that analyzes candidate ball regions in order to recognize the ball pattern.

CHAPTER 7

Conclusion

At the end of the course we were able to : Identify appropriate computer vision techniques for various real-time projects, apply computer vision techniques for better image understanding, implement and debug python codes related to image enhancement, understand and implement object detection techniques in python, apply face detection algorithm in python and studied the importance of computer vision in real time world by understanding their applications in various fields.

REFERENCES

https://chennai.vit.ac.in/

https://chennai.vit.ac.in/events/

https://www.wikipedia.org/

https://www.tutorialspoint.com/index.htm

https://towardsdatascience.com/

https://opencv.org/

https://www.python.org/

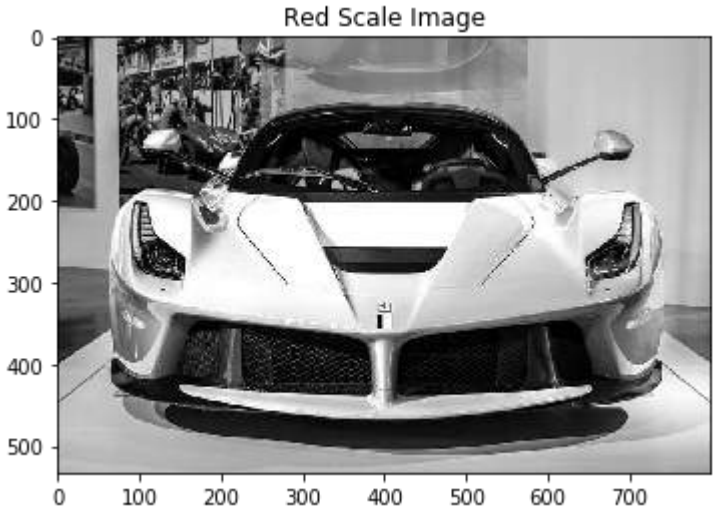
https://jupyter.org/

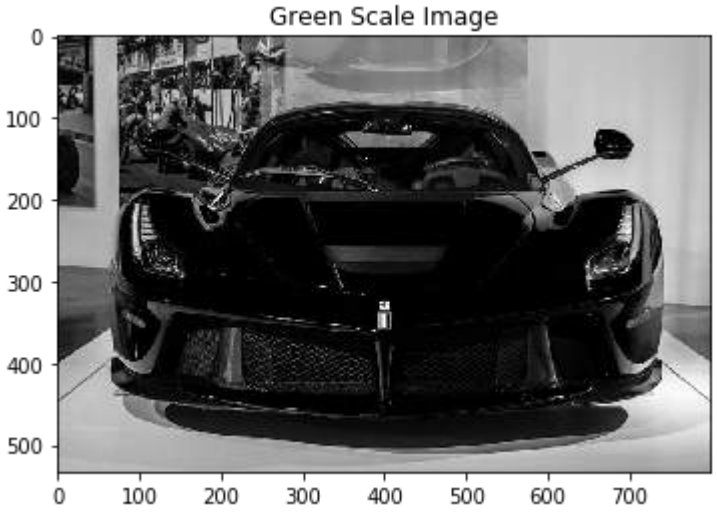
APPENDIX

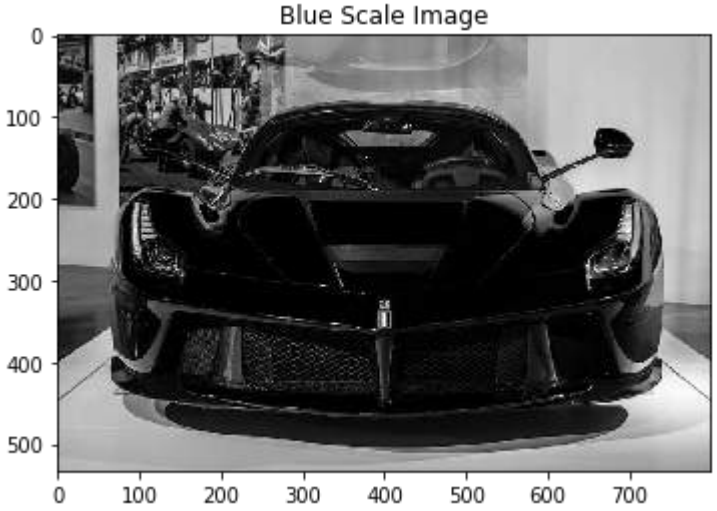
In [ ]:
```
# VAP - PVT, Lecture3_Task-1
# 27/02/21 - Saturday
# 18BEC1278 - Andrew John
```

In [30]:
```
#Studying the color-plane information (Only Red Image)
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/RGBr.jpg')
plt.figure(1)
plt.imshow(img)
plt.title('Original Image')
imgR=img[:,:,0]
imgG=img[:,:,1]
imgB=img[:,:,2]
plt.figure(2)
plt.imshow(imgR,cmap='gray')
plt.title('Red Scale Image')
plt.figure(3)
plt.imshow(imgG,cmap='gray')
plt.title('Green Scale Image')
plt.figure(4)
plt.imshow(imgB,cmap='gray')
plt.title('Blue Scale Image')
```

Out[30]: Text(0.5, 1.0, 'Blue Scale Image')

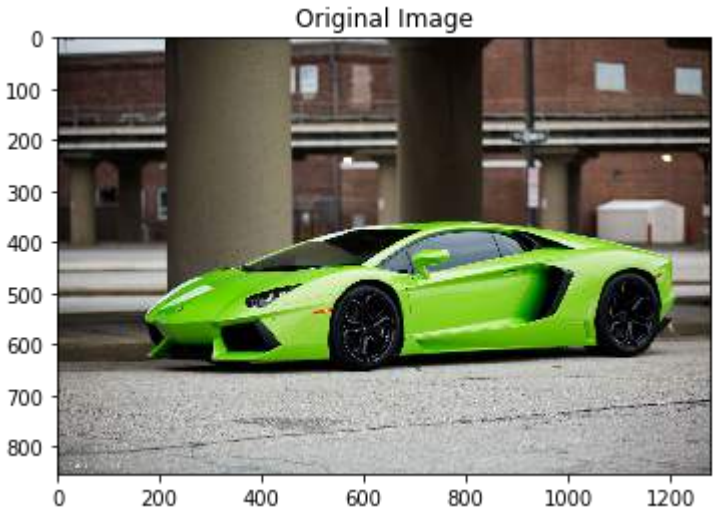


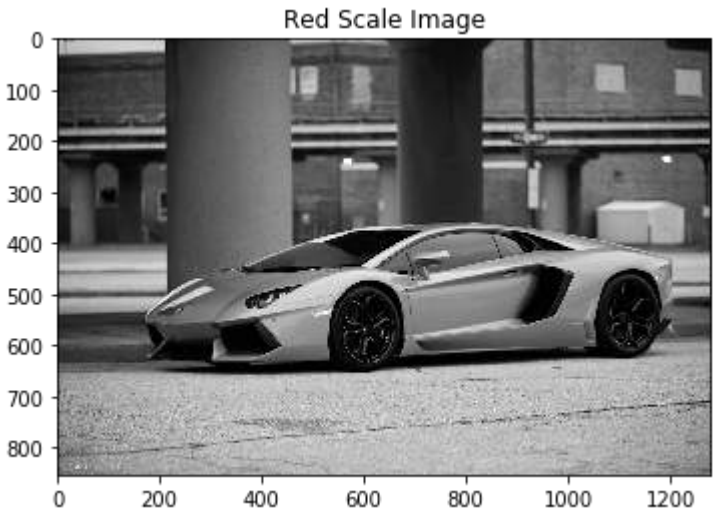


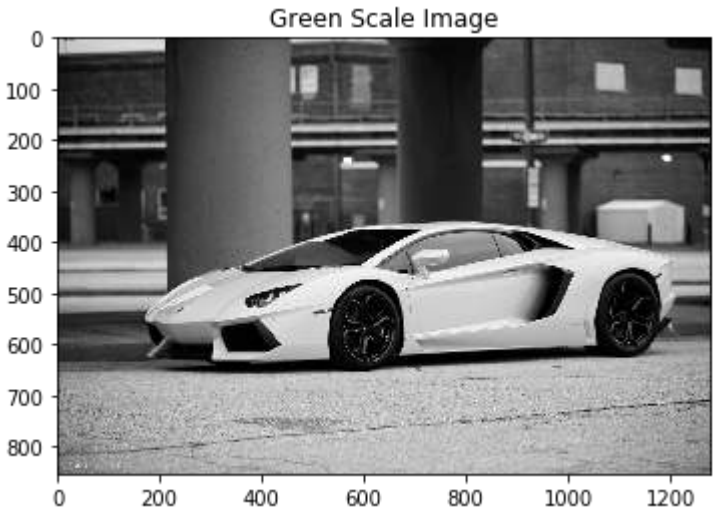

Blue Scale Image
0
100
200
300
400
500
0
100
200
300
400
500
600
700

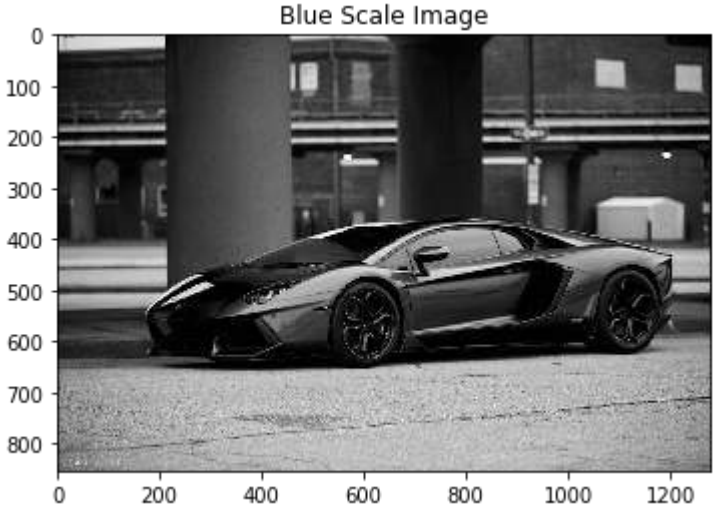
In [31]:
```
#Studying the color-plane information (Only Green Image)
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/RGBg.jpg')
plt.figure(1)
plt.imshow(img)
plt.title('Original Image')
imgR=img[:,:,0]
imgG=img[:,:,1]
imgB=img[:,:,2]
plt.figure(2)
plt.imshow(imgR,cmap='gray')
plt.title('Red Scale Image')
plt.figure(3)
plt.imshow(imgG,cmap='gray')
plt.title('Green Scale Image')
plt.figure(4)
plt.imshow(imgB,cmap='gray')
plt.title('Blue Scale Image')
```

Out[31]: Text(0.5, 1.0, 'Blue Scale Image')



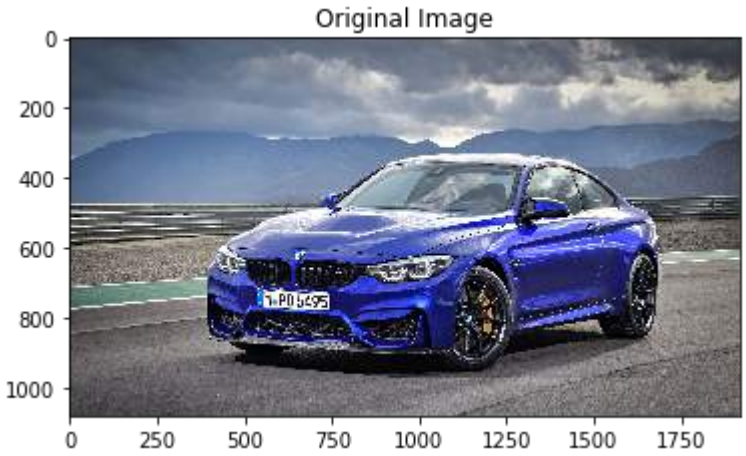


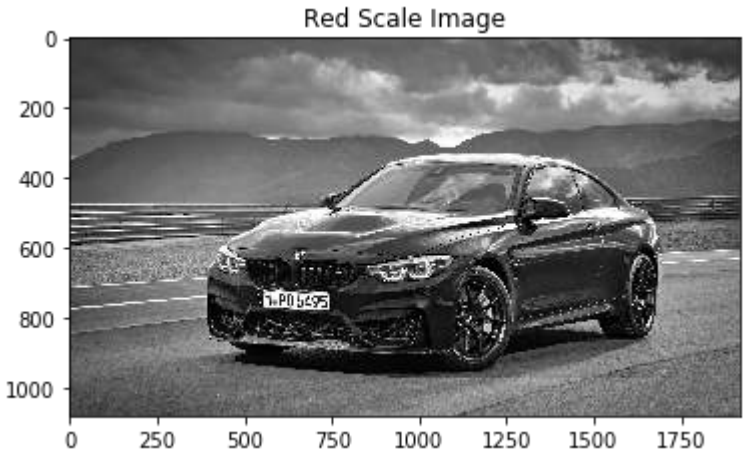


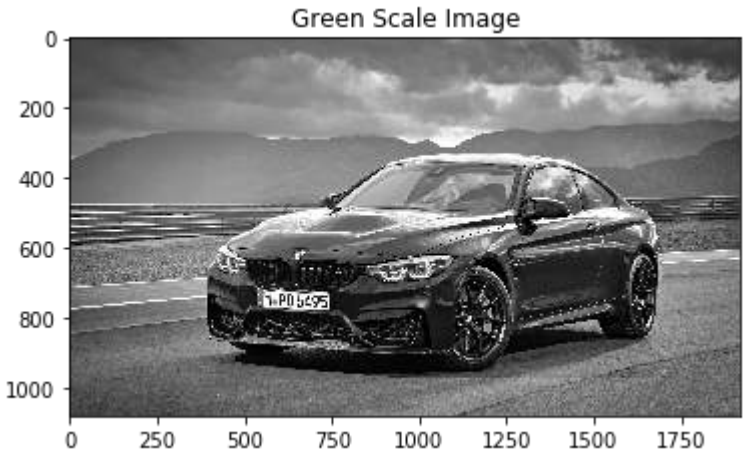

Blue Scale Image
0
100
200
300
400
500
600
700
800
0
200
400
600
800
1000
1200

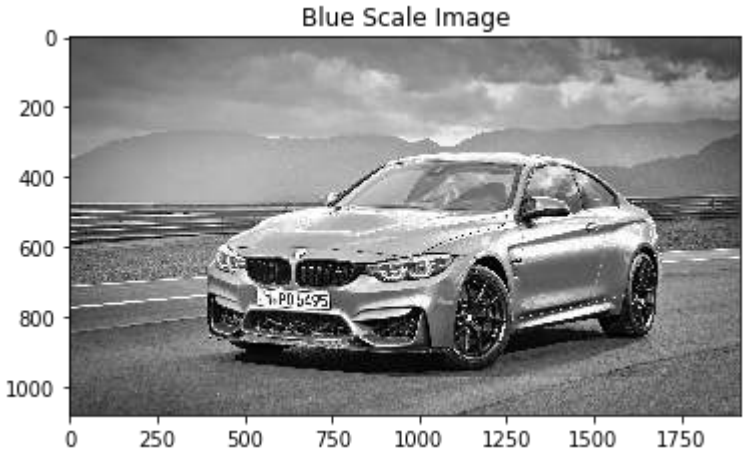
In [32]:
```
#Studying the color-plane information (Only Blue Image)
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/RGBb.jpg')
plt.figure(1)
plt.imshow(img)
plt.title('Original Image')
imgR=img[:,:,0]
imgG=img[:,:,1]
imgB=img[:,:,2]
plt.figure(2)
plt.imshow(imgR,cmap='gray')
plt.title('Red Scale Image')
plt.figure(3)
plt.imshow(imgG,cmap='gray')
plt.title('Green Scale Image')
plt.figure(4)
plt.imshow(imgB,cmap='gray')
plt.title('Blue Scale Image')
```

Out[32]: Text(0.5, 1.0, 'Blue Scale Image')



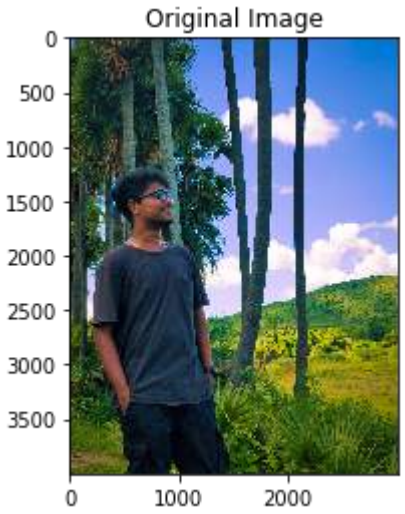


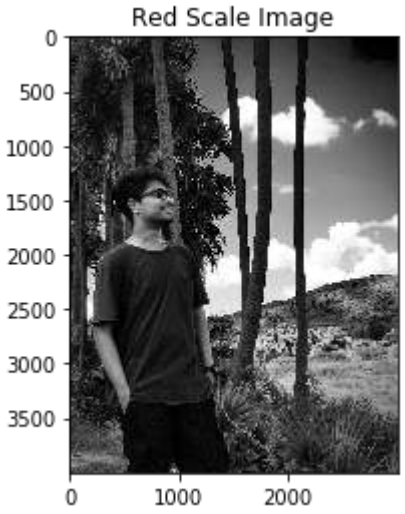


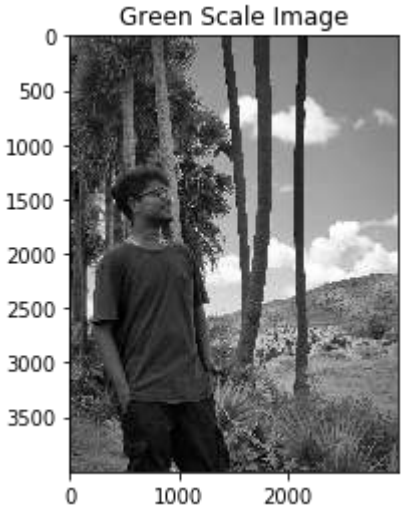

Blue Scale Image
0
200
400
600
800
1000
0
250
500
750
1000
1250
1500
1750

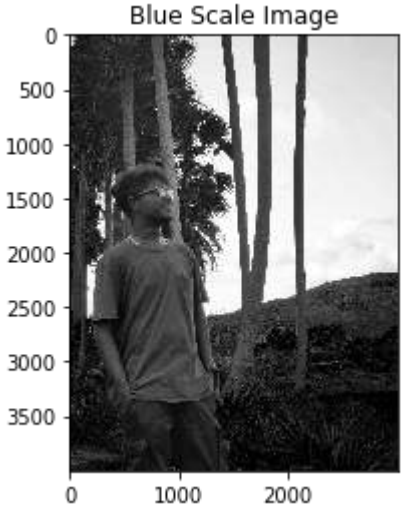
In [33]:
```
#Studying the color-plane information (Combination of Green and Blue)
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/RGB1.jpg')
plt.figure(1)
plt.imshow(img)
plt.title('Original Image')
imgR=img[:,:,0]
imgG=img[:,:,1]
imgB=img[:,:,2]
plt.figure(2)
plt.imshow(imgR,cmap='gray')
plt.title('Red Scale Image')
plt.figure(3)
plt.imshow(imgG,cmap='gray')
plt.title('Green Scale Image')
plt.figure(4)
plt.imshow(imgB,cmap='gray')
plt.title('Blue Scale Image')
```

Out[33]: Text(0.5, 1.0, 'Blue Scale Image')



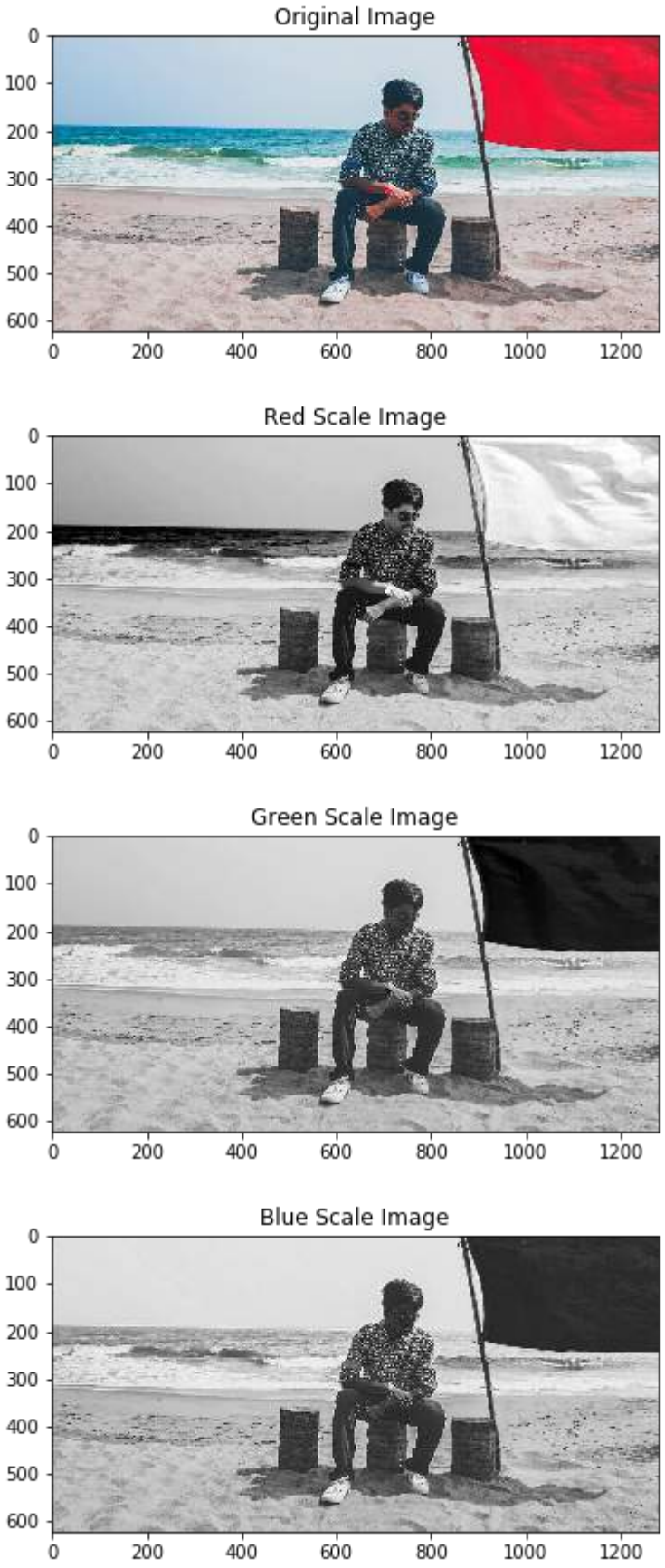





Blue Scale Image
0
500
1000
1500
2000
2500
3000
3500
0
1000
2000

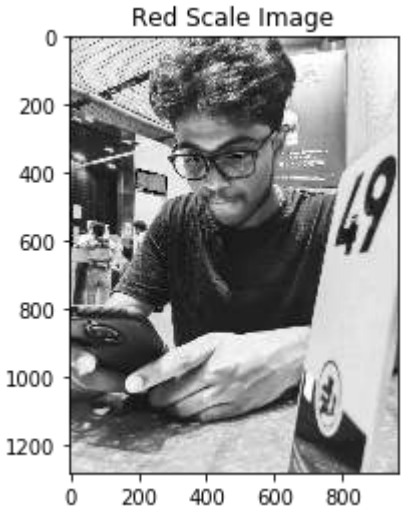
In [34]:
```
#Studying the color-plane information (Combination of Red and Blue)
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/RGB2.jpg')
plt.figure(1)
plt.imshow(img)
plt.title('Original Image')
imgR=img[:,:,0]
imgG=img[:,:,1]
imgB=img[:,:,2]
plt.figure(2)
plt.imshow(imgR,cmap='gray')
plt.title('Red Scale Image')
plt.figure(3)
plt.imshow(imgG,cmap='gray')
plt.title('Green Scale Image')
plt.figure(4)
plt.imshow(imgB,cmap='gray')
plt.title('Blue Scale Image')
```

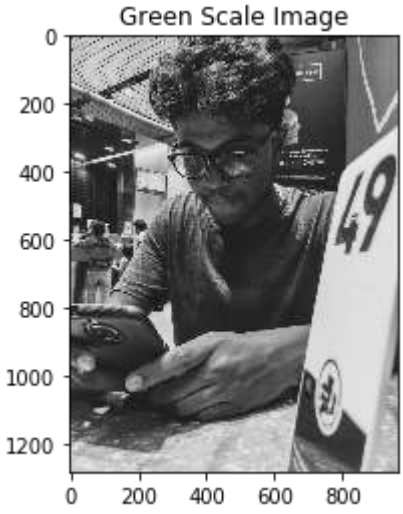
Out[34]: Text(0.5, 1.0, 'Blue Scale Image')



In [35]:
```
#Studying the color-plane information (Combination of Red and Green)
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/RGB3.jpg')
plt.figure(1)
plt.imshow(img)
plt.title('Original Image')
imgR=img[:,:,0]
imgG=img[:,:,1]
imgB=img[:,:,2]
plt.figure(2)
plt.imshow(imgR,cmap='gray')
plt.title('Red Scale Image')
plt.figure(3)
plt.imshow(imgG,cmap='gray')
plt.title('Green Scale Image')
plt.figure(4)
plt.imshow(imgB,cmap='gray')
plt.title('Blue Scale Image')
```

Out[35]: Text(0.5, 1.0, 'Blue Scale Image')

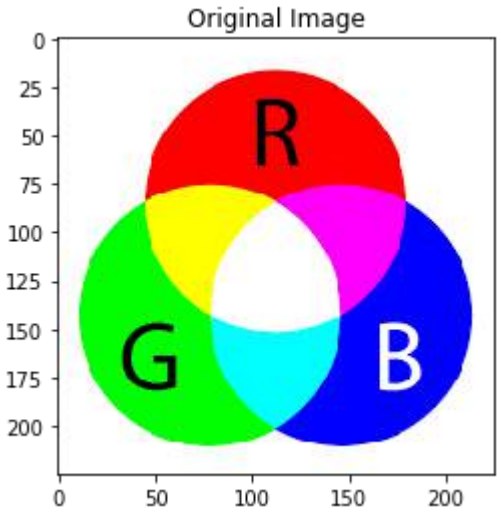


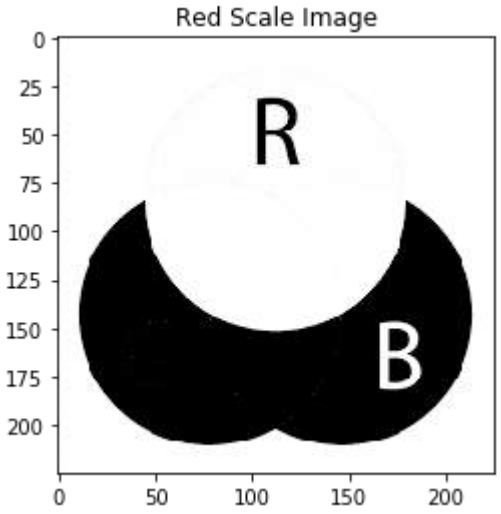


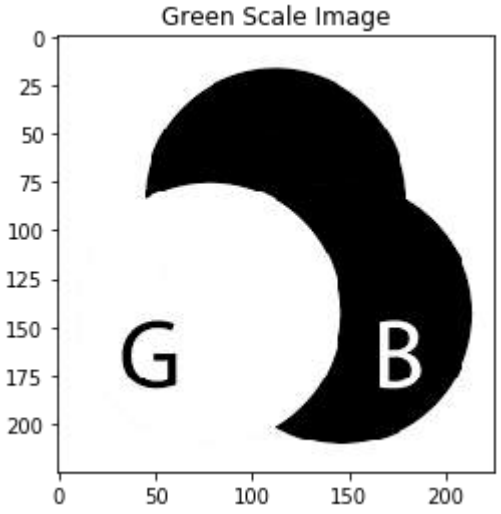

Blue Scale Image
0
200
400
600
800
1000
1200
0
200
400
600
800

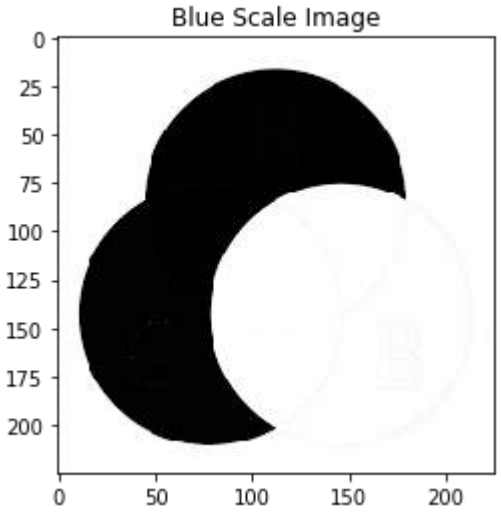
In [36]:
```
#Studying the color-plane information (Combination of Red, Green and Blue)
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/RGBrgb.jpg')
plt.figure(1)
plt.imshow(img)
plt.title('Original Image')
imgR=img[:,:,0]
imgG=img[:,:,1]
imgB=img[:,:,2]
plt.figure(2)
plt.imshow(imgR,cmap='gray')
plt.title('Red Scale Image')
plt.figure(3)
plt.imshow(imgG,cmap='gray')
plt.title('Green Scale Image')
plt.figure(4)
plt.imshow(imgB,cmap='gray')
plt.title('Blue Scale Image')
```

Out[36]: Text(0.5, 1.0, 'Blue Scale Image')







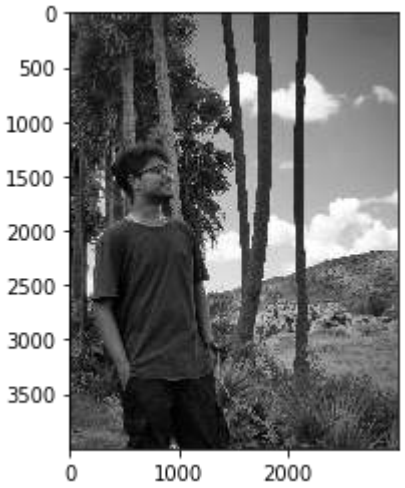


In [ ]:
```
#INFERENCE:

#   1) As we can see in Only Red Image, the Red Scale is whitened up and the B
lue and Green Scale are dark.
#   2) As we can see in Only Green Image, the Green Scale is whitened up and t
he Red and Blue Scale are dark.
#   3) As we can see in Only Blue Image, the Blue Scale is whitened up and the
Red and Green Scale are dark.
#   4) In Green and Blue combination Image, we can see the Blue Sky and Green
 Mountain is whitened in Blue
#      and Green Scale respectively and since my skin is in Maroon color, the
 Red Scale is whitened up.
#   5) In Red and Blue combination Image, we can see the Blue Ocean and Red Fl
ag is whitened up in
#      Blue Scale and Red Scale Respectively
#   6) In Red and Green combination Image, we can see the Red Background and G
reen T-Shirt is whitened up in
#      Red Scale and Green Scale Respectively.
#   7) In combination of all 3 colors, we can see Red, Green and Blue is white
ned up in
#      Red, Green and Blue Scale respectively.
```

In [ ]:
```

```

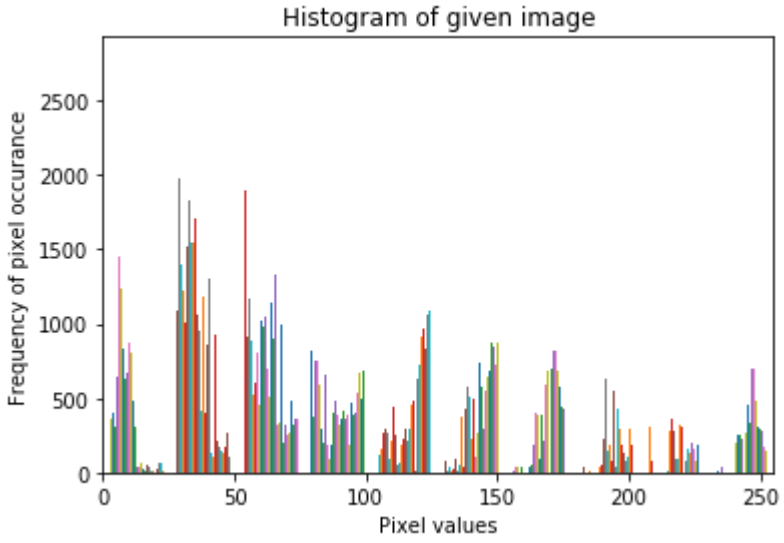
In [1]:
```
#VAP - PVT, Lecture3_Task-2
#27/02/21 - Saturday
#18BEC1278 - Andrew John
```

In [2]:
```
#Studying the histogram - Original Picture
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/RGB1.jpg')
def rgb2gray(rgb):
    r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
    gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
    return gray
img_gray=rgb2gray(img)
plt.figure(1)
plt.imshow(img_gray,cmap='gray')
plt.figure(2)
plt.show()
plt.hist(img_gray)
plt.title("Histogram of given image")
plt.xlabel("Pixel values")
plt.ylabel("Frequency of pixel occurance")
plt.xlim([0, 255])
```

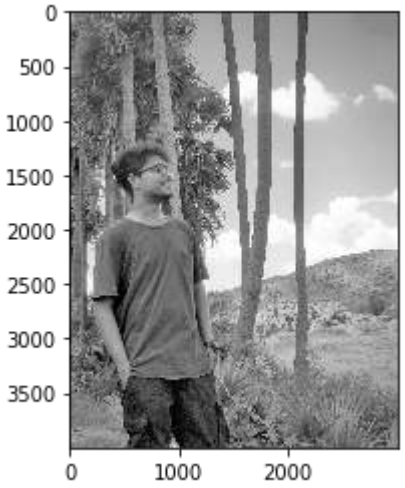



```
<Figure size 432x288 with 0 Axes>
```

Out[2]:
```
(0, 255)
```

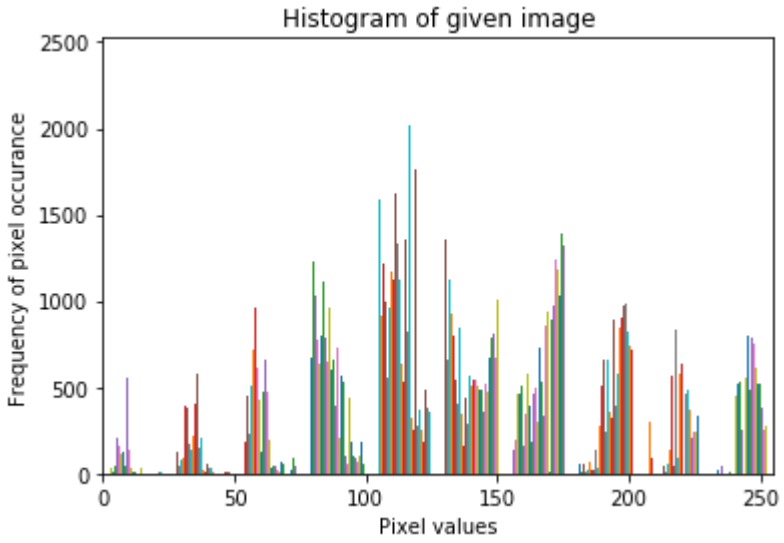


In [3]:
```
#Studying the histogram - High Brightness Picture
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/RGB1_HB.jpg')
def rgb2gray(rgb):
    r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
    gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
    return gray
img_gray=rgb2gray(img)
plt.figure(1)
plt.imshow(img_gray,cmap='gray')
plt.figure(2)
plt.show()
plt.hist(img_gray)
plt.title("Histogram of given image")
plt.xlabel("Pixel values")
plt.ylabel("Frequency of pixel occurance")
plt.xlim([0, 255])
```

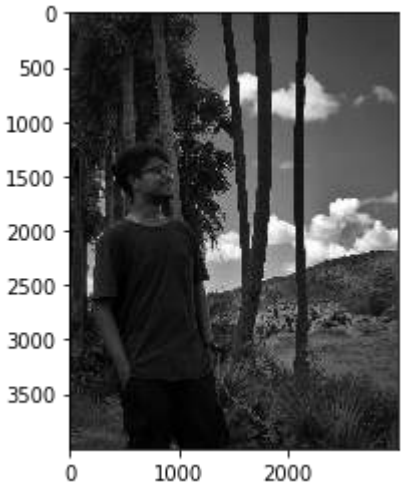



```
<Figure size 432x288 with 0 Axes>
```

Out[3]: (0, 255)

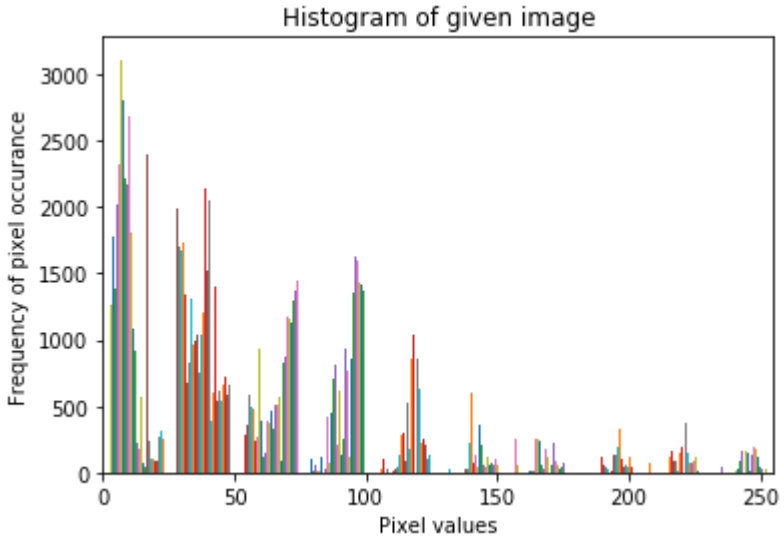


In [4]:
```
#Studying the histogram - Low Brightness Picture
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/RGB1_LB.jpg')
def rgb2gray(rgb):
    r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
    gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
    return gray
img_gray=rgb2gray(img)
plt.figure(1)
plt.imshow(img_gray,cmap='gray')
plt.figure(2)
plt.show()
plt.hist(img_gray)
plt.title("Histogram of given image")
plt.xlabel("Pixel values")
plt.ylabel("Frequency of pixel occurance")
plt.xlim([0, 255])
```

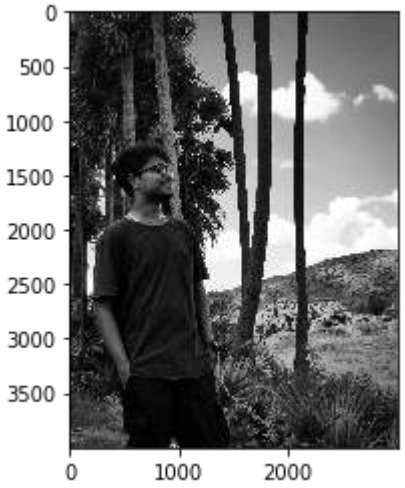



```
<Figure size 432x288 with 0 Axes>
```

Out[4]:
```
(0, 255)
```

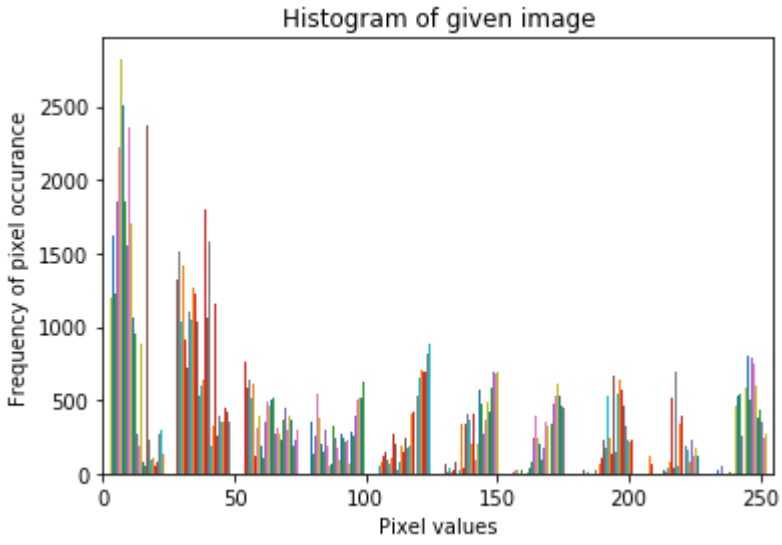


In [7]:
```
#Studying the histogram - High Contrast Picture
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/RGB1_HC.jpg')
def rgb2gray(rgb):
    r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
    gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
    return gray
img_gray=rgb2gray(img)
plt.figure(1)
plt.imshow(img_gray,cmap='gray')
plt.figure(2)
plt.show()
plt.hist(img_gray)
plt.title("Histogram of given image")
plt.xlabel("Pixel values")
plt.ylabel("Frequency of pixel occurance")
plt.xlim([0, 255])
```

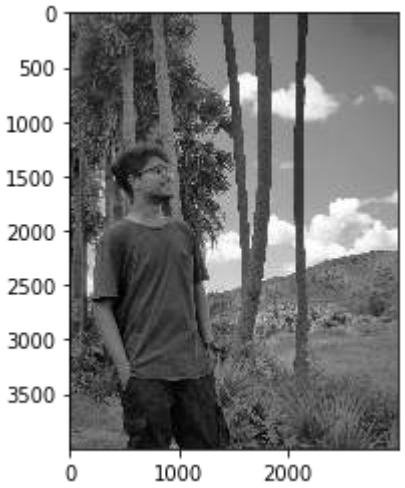



```
<Figure size 432x288 with 0 Axes>
```

Out[7]: (0, 255)

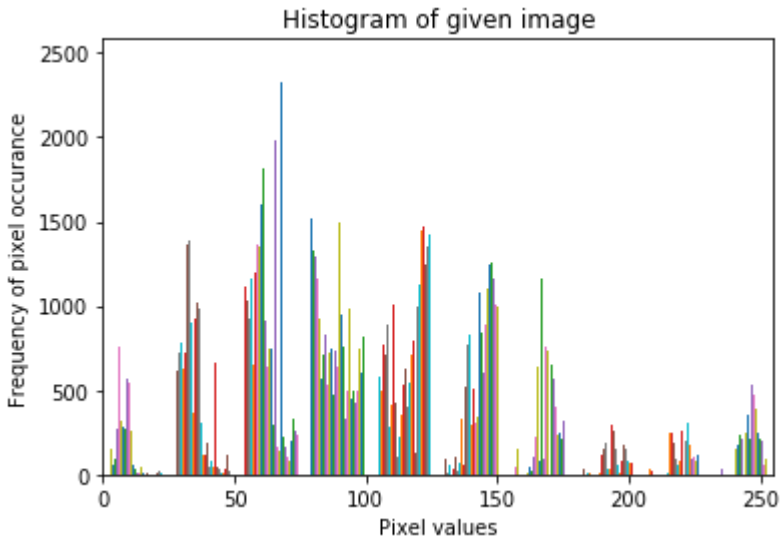


In [6]:
```
#Studying the histogram - Low Contrast Picture
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/RGB1_LC.jpg')
def rgb2gray(rgb):
    r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
    gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
    return gray
img_gray=rgb2gray(img)
plt.figure(1)
plt.imshow(img_gray,cmap='gray')
plt.figure(2)
plt.show()
plt.hist(img_gray)
plt.title("Histogram of given image")
plt.xlabel("Pixel values")
plt.ylabel("Frequency of pixel occurance")
plt.xlim([0, 255])
```



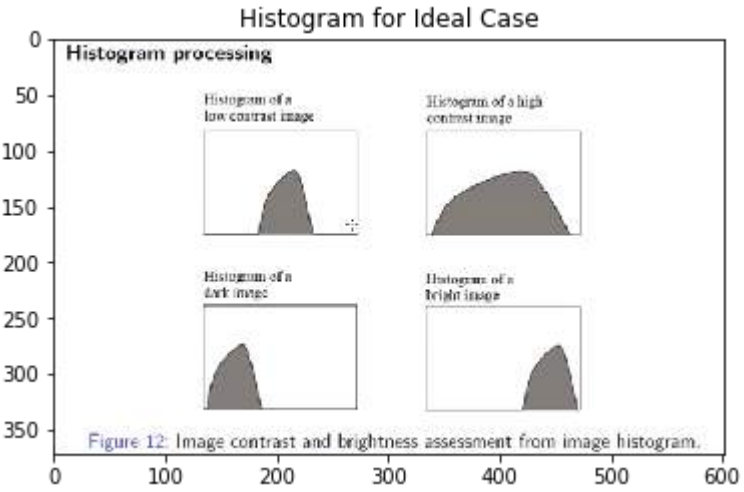

```
<Figure size 432x288 with 0 Axes>
```

Out[6]:
```
(0, 255)
```



In [9]:
```
#INFERENCE:

img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/Histogram.jpg')
plt.title("Histogram for Ideal Case")
plt.figure(1)
plt.imshow(img)

# Ideal cases for a Picture with Low Contrast, High Contrast,
# Dark Image and Bright Image is shown in the figure below.

# We can see that from the JPG picture taken, RGB1, RGB1_HB, RGB1_LB, RGB1_HC
 and RGB1_LC
# we have studied and seen that the locus of the graphs is almost similar to t
hat of Ideal case.
```

Out[9]: <matplotlib.image.AxesImage at 0x2125a227408>

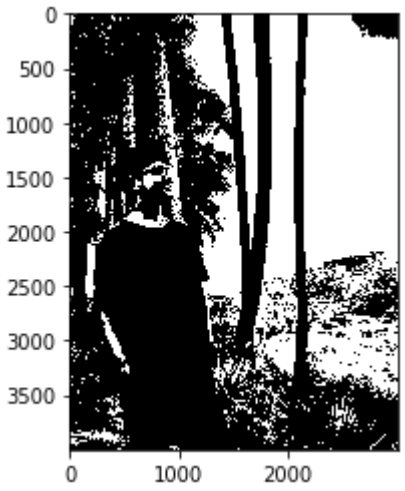


In [ ]:
```

```

In [ ]:
```
#VAP - PVT, Lecture3_Task-3
#27/02/21 - Saturday
#18BEC1278 - Andrew John
```

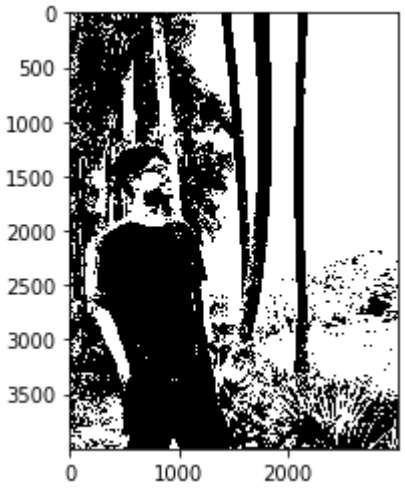
In [2]:
```
#Converting a gray scale image to binary image (thresholding @ 100)
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/RGB1.jpg')
def rgb2gray(rgb):
    r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
    gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
    return gray
img_gray=rgb2gray(img)
img_temp = np.ones((4000,3000), dtype=np.int16)
img_gray=img_temp*(img_gray > 100)
plt.figure(1)
plt.imshow(img_gray,cmap='gray')
```

Out[2]:
```
<matplotlib.image.AxesImage at 0x294a818d5c8>
```



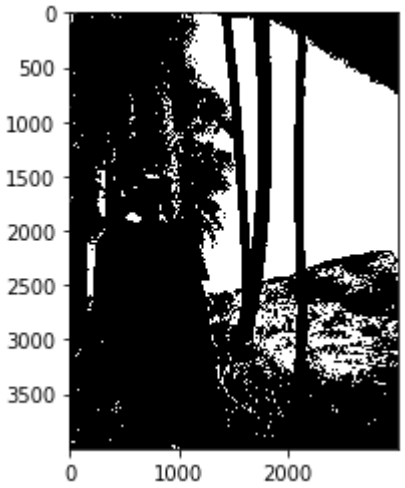
In [3]:
```
#Converting a gray scale image to binary image (thresholding @ 75)
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/RGB1.jpg')
def rgb2gray(rgb):
    r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
    gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
    return gray
img_gray=rgb2gray(img)
img_temp = np.ones((4000,3000), dtype=np.int16)
img_gray=img_temp*(img_gray > 75)
plt.figure(1)
plt.imshow(img_gray,cmap='gray')
```

Out[3]: <matplotlib.image.AxesImage at 0x294a8972688>



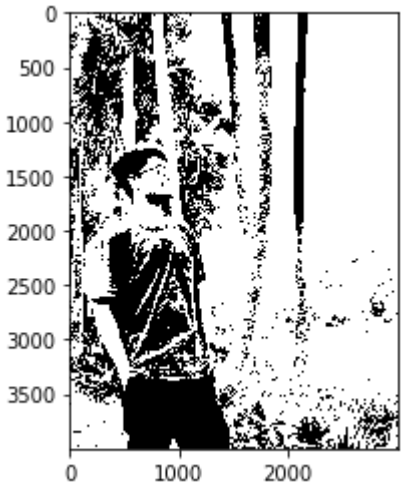
In [4]:
```
#Converting a gray scale image to binary image (thresholding @ 125)
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/RGB1.jpg')
def rgb2gray(rgb):
    r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
    gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
    return gray
img_gray=rgb2gray(img)
img_temp = np.ones((4000,3000), dtype=np.int16)
img_gray=img_temp*(img_gray > 125)
plt.figure(1)
plt.imshow(img_gray,cmap='gray')
```

Out[4]: <matplotlib.image.AxesImage at 0x294a89e0148>



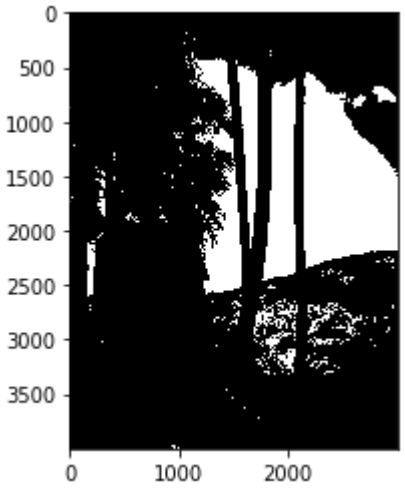
In [5]:
```
#Converting a gray scale image to binary image (thresholding @ 50)
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/RGB1.jpg')
def rgb2gray(rgb):
    r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
    gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
    return gray
img_gray=rgb2gray(img)
img_temp = np.ones((4000,3000), dtype=np.int16)
img_gray=img_temp*(img_gray > 50)
plt.figure(1)
plt.imshow(img_gray,cmap='gray')
```

Out[5]: <matplotlib.image.AxesImage at 0x294a8a41d48>



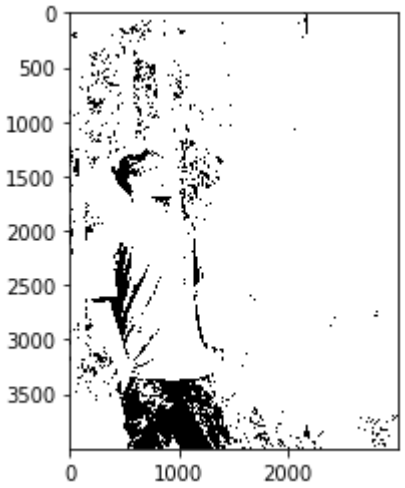
In [6]:
```
#Converting a gray scale image to binary image (thresholding @ 150)
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/RGB1.jpg')
def rgb2gray(rgb):
    r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
    gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
    return gray
img_gray=rgb2gray(img)
img_temp = np.ones((4000,3000), dtype=np.int16)
img_gray=img_temp*(img_gray > 150)
plt.figure(1)
plt.imshow(img_gray,cmap='gray')
```

Out[6]: <matplotlib.image.AxesImage at 0x294a8ab11c8>



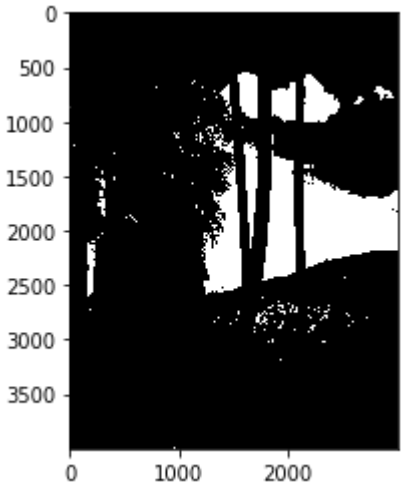
In [7]:
```
#Converting a gray scale image to binary image (thresholding @ 25)
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/RGB1.jpg')
def rgb2gray(rgb):
    r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
    gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
    return gray
img_gray=rgb2gray(img)
img_temp = np.ones((4000,3000), dtype=np.int16)
img_gray=img_temp*(img_gray > 25)
plt.figure(1)
plt.imshow(img_gray,cmap='gray')
```

Out[7]: <matplotlib.image.AxesImage at 0x294a8b1a1c8>



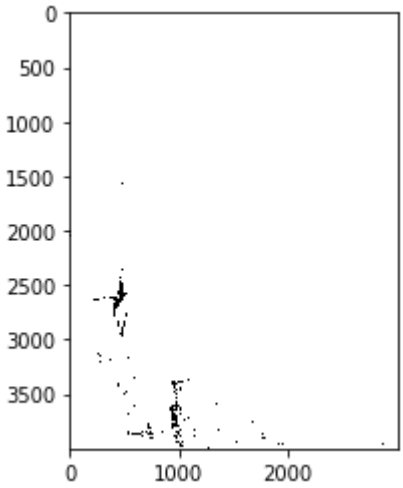
In [8]:
```
#Converting a gray scale image to binary image (thresholding @ 175)
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/RGB1.jpg')
def rgb2gray(rgb):
    r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
    gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
    return gray
img_gray=rgb2gray(img)
img_temp = np.ones((4000,3000), dtype=np.int16)
img_gray=img_temp*(img_gray > 175)
plt.figure(1)
plt.imshow(img_gray,cmap='gray')
```

Out[8]: <matplotlib.image.AxesImage at 0x294a8b7edc8>



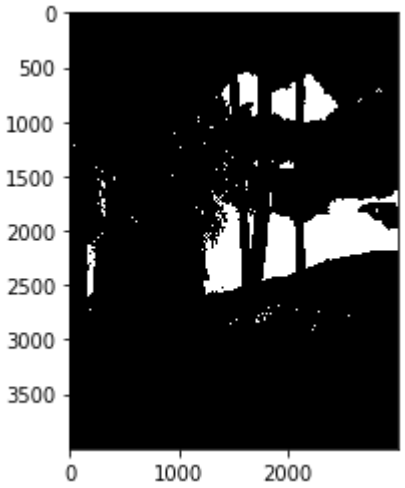
In [10]:
```
#Converting a gray scale image to binary image (thresholding @ 5)
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/RGB1.jpg')
def rgb2gray(rgb):
    r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
    gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
    return gray
img_gray=rgb2gray(img)
img_temp = np.ones((4000,3000), dtype=np.int16)
img_gray=img_temp*(img_gray > 5)
plt.figure(1)
plt.imshow(img_gray,cmap='gray')
```

Out[10]: <matplotlib.image.AxesImage at 0x294a8be8c08>



In [11]:
```
#Converting a gray scale image to binary image (thresholding @ 195)
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/RGB1.jpg')
def rgb2gray(rgb):
    r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
    gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
    return gray
img_gray=rgb2gray(img)
img_temp = np.ones((4000,3000), dtype=np.int16)
img_gray=img_temp*(img_gray > 195)
plt.figure(1)
plt.imshow(img_gray,cmap='gray')
```

Out[11]: <matplotlib.image.AxesImage at 0x294a8c52a48>



In [9]:
```
#INFERENCE:

# 1) If the pixel is more than the threshold, that part of the array will fill
up with ones and ones represent white color.
# 2) If the pixel is less than the threshold, that part of the array won't be
 filled up with ones, it'll be zeroes
# and zeroes represent black color.
```

In [ ]:
```

```

In [ ]:

```
#VAP - PVT, Lecture-6 - Hands On
#20/03/21 - Saturday
#18BEC1278 - Andrew John
```

In [ ]:

```
# Gaussian and Median Filter
# Maximum and Minimum Filter
# Sharpening Images - Laplacian on Gaussian
# Sobel and Prewitt
# Open CV
#  low Contrast Function
#  Gamma Transformation
#  Contrast Strecthing
#  Log Transformation
```

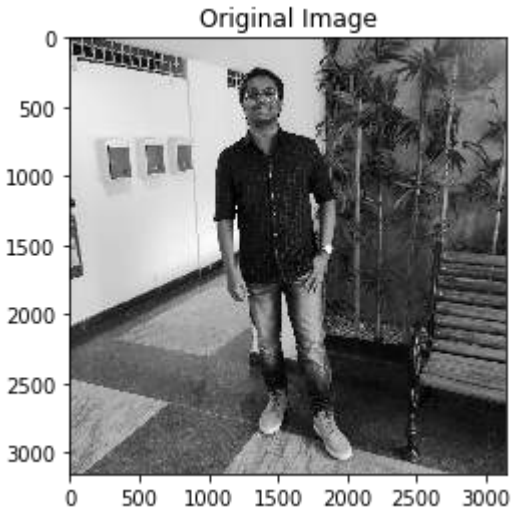
In [49]:

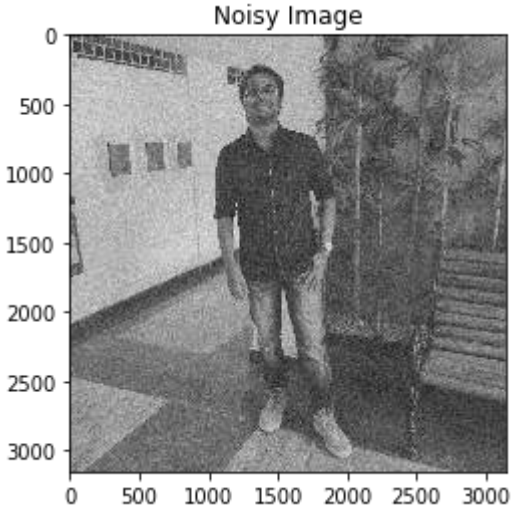
```
# Gaussian and Median Filters
# Raised Noise
# Sigma = 1.5
# Size = 5
# Noise = 20
```

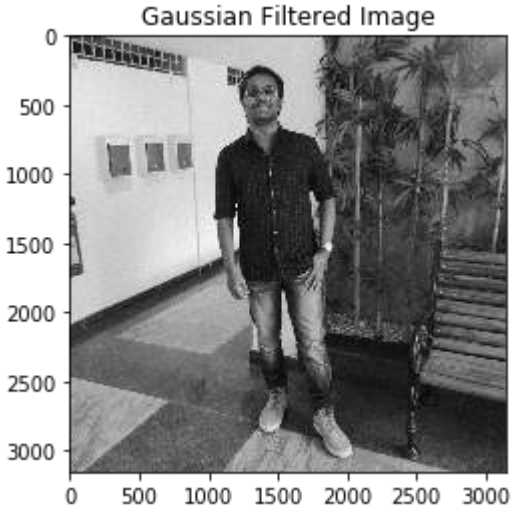
In [24]:

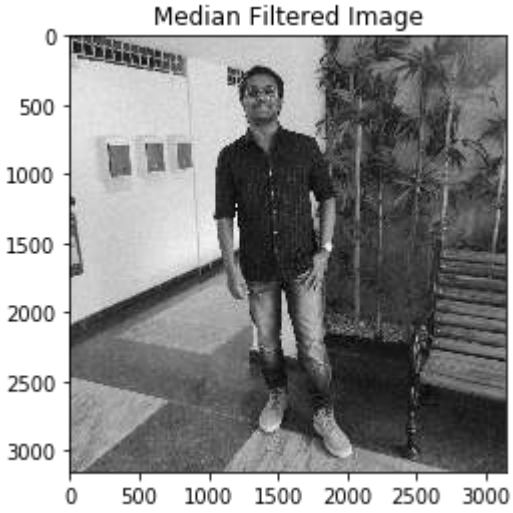
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from scipy import ndimage # Multi-dimensional image processing
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-2/Andrew_PVT.jpg')
def rgb2gray(rgb):
    r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
    gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
    return gray
img=rgb2gray(img)
img_noise = img + 20*np.random.randn(*img.shape)
blurred_img1=ndimage.gaussian_filter(img_noise, sigma=1.5) # Multidimensional Gaussian
 filter
blurred_img2=ndimage.median_filter(img_noise,5) # Calculate a multidimensional median f
ilter
plt.figure(1)
plt.imshow(img,cmap='gray')
plt.title("Original Image")
plt.figure(2)
plt.imshow(img_noise,cmap='gray')
plt.title("Noisy Image")
plt.figure(3)
plt.imshow(blurred_img1,cmap='gray')
plt.title("Gaussian Filtered Image")
plt.figure(4)
plt.imshow(blurred_img2,cmap='gray')
plt.title("Median Filtered Image")
err1=img - blurred_img1
err2=img - blurred_img2
sqerr1 = np.power(err1,2)
sqerr2 = np.power(err2,2)
print(sqerr1.mean())
print(sqerr2.mean())
```

```
33.80623536481327
43.66292778421164
```









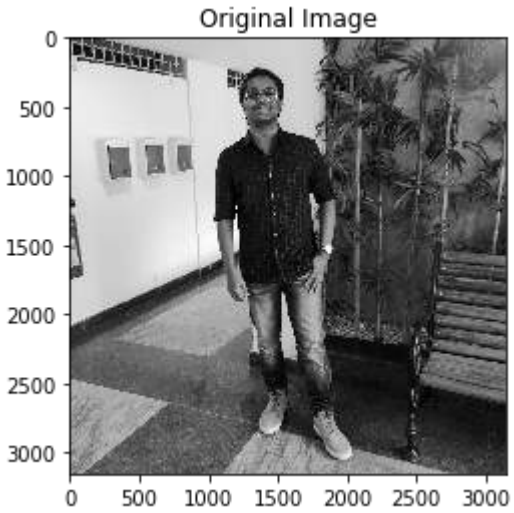
In [ ]:

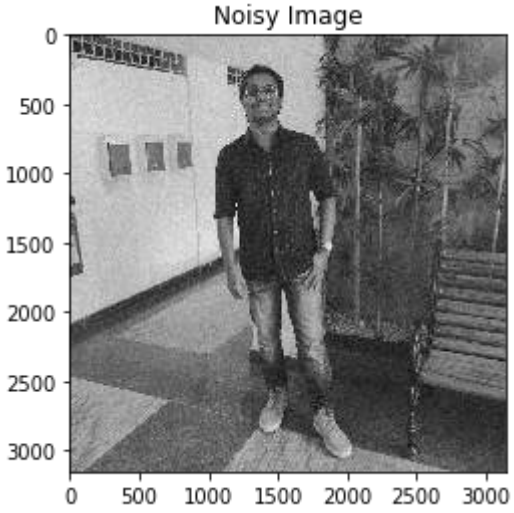
```
# Gaussian and Median Filters
# Reasonable Noise
# Sigma = 1.5
# Size = 5
# Noise = 10
```

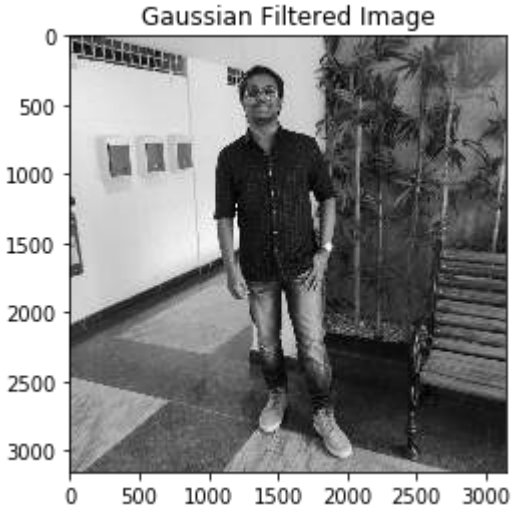
In [25]:

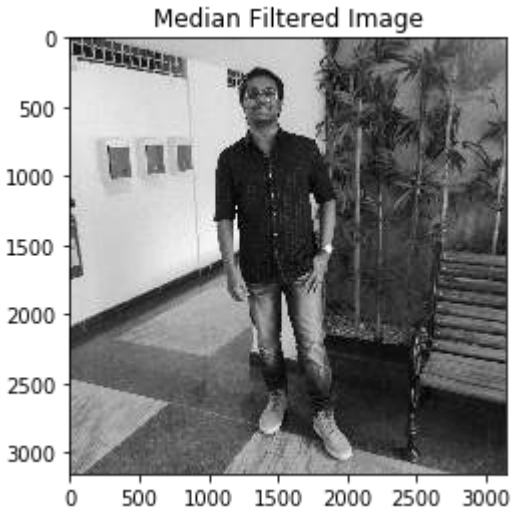
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from scipy import ndimage # Multi-dimensional image processing
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-2/Andrew_PVT.jpg')
def rgb2gray(rgb):
    r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
    gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
    return gray
img=rgb2gray(img)
img_noise = img + 10*np.random.randn(*img.shape)
blurred_img1=ndimage.gaussian_filter(img_noise, sigma=1.5) # Multidimensional Gaussian
 filter
blurred_img2=ndimage.median_filter(img_noise,5) # Calculate a multidimensional median f
ilter
plt.figure(1)
plt.imshow(img,cmap='gray')
plt.title("Original Image")
plt.figure(2)
plt.imshow(img_noise,cmap='gray')
plt.title("Noisy Image")
plt.figure(3)
plt.imshow(blurred_img1,cmap='gray')
plt.title("Gaussian Filtered Image")
plt.figure(4)
plt.imshow(blurred_img2,cmap='gray')
plt.title("Median Filtered Image")
err1=img - blurred_img1
err2=img - blurred_img2
sqerr1 = np.power(err1,2)
sqerr2 = np.power(err2,2)
print(sqerr1.mean())
print(sqerr2.mean())
```

```
23.173673744892195
22.997974195723693
```









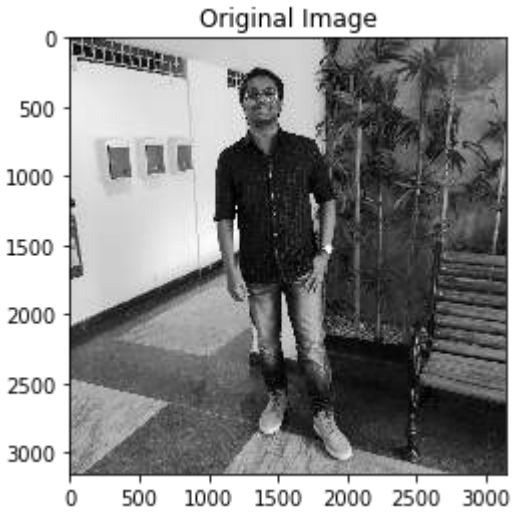
In [ ]:

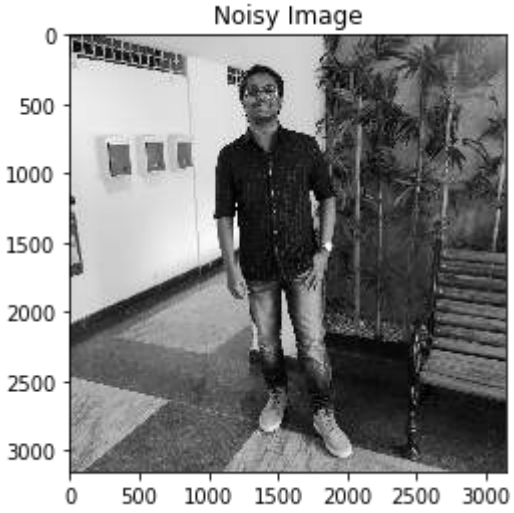
```
# Gaussian and Median Filters
# Reduced Noise
# Sigma = 5
# Size = 7
# Noise = 0.05
```

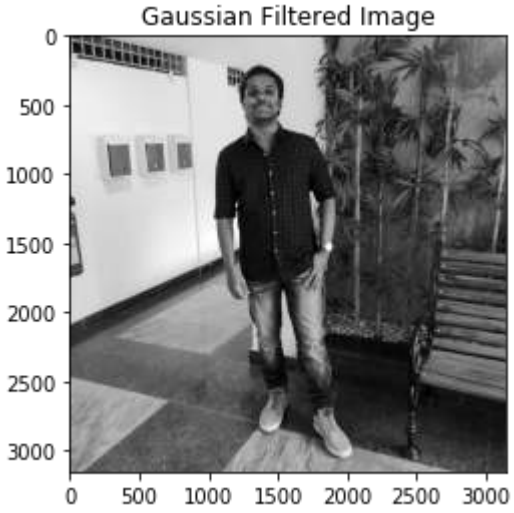
In [50]:

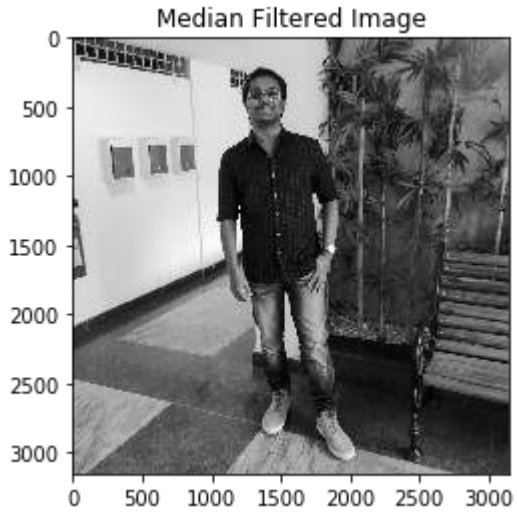
```
#Gaussian and Median Filters
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from scipy import ndimage # Multi-dimensional image processing
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-2/Andrew_PVT.jpg')
def rgb2gray(rgb):
    r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
    gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
    return gray
img=rgb2gray(img)
img_noise = img + 0.05*np.random.randn(*img.shape)
blurred_img1=ndimage.gaussian_filter(img_noise, sigma=5) # Multidimensional Gaussian fi
lter
blurred_img2=ndimage.median_filter(img_noise,7) # Calculate a multidimensional median f
ilter
plt.figure(1)
plt.imshow(img,cmap='gray')
plt.title("Original Image")
plt.figure(2)
plt.imshow(img_noise,cmap='gray')
plt.title("Noisy Image")
plt.figure(3)
plt.imshow(blurred_img1,cmap='gray')
plt.title("Gaussian Filtered Image")
plt.figure(4)
plt.imshow(blurred_img2,cmap='gray')
plt.title("Median Filtered Image")
err1=img - blurred_img1
err2=img - blurred_img2
sqerr1 = np.power(err1,2)
sqerr2 = np.power(err2,2)
print(sqerr1.mean())
print(sqerr2.mean())
```

```
102.75980945368087
22.589657228163567
```









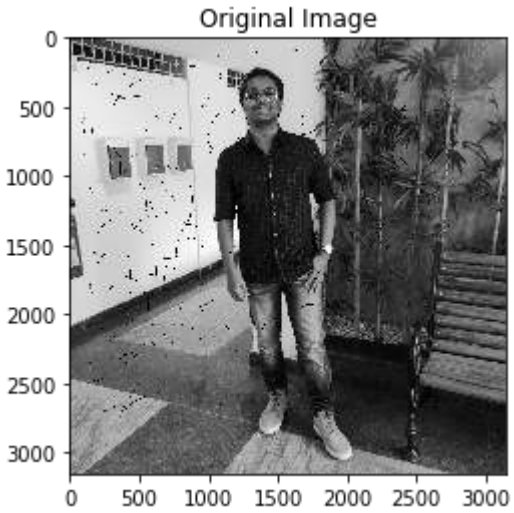
In [ ]:

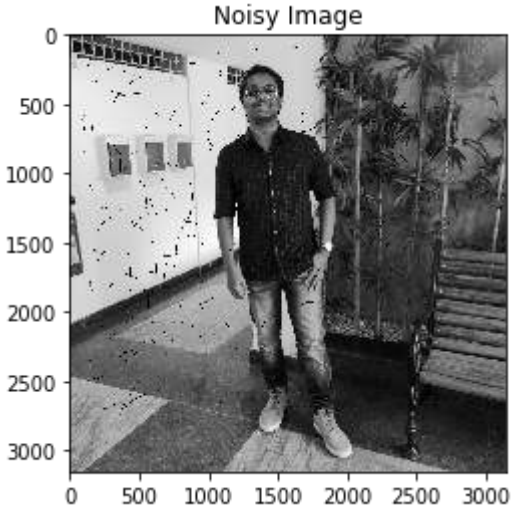
```
#Gaussian and Median Filters
#External Noise
#Sigma = 1.5
#Size = 5
#Noise = 0
```

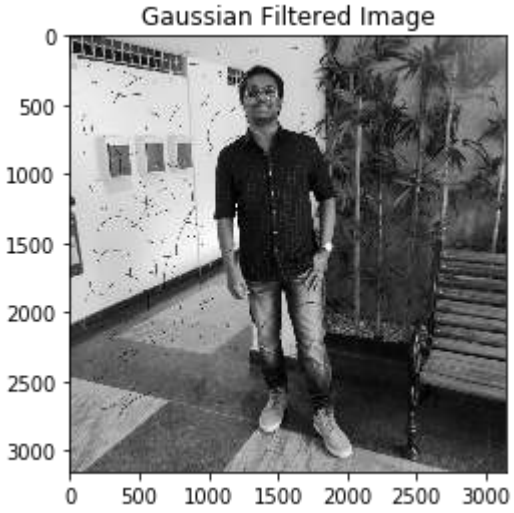
In [27]:

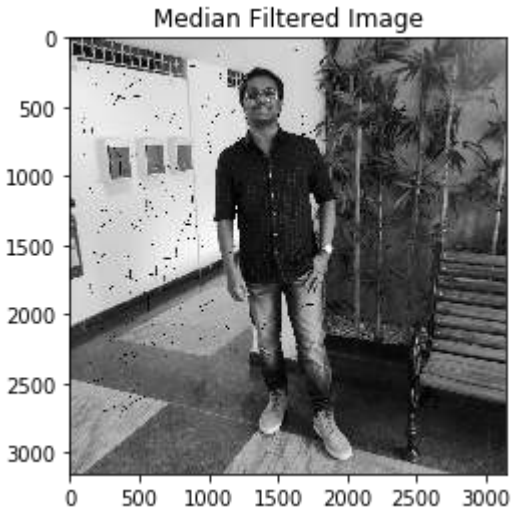
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from scipy import ndimage # Multi-dimensional image processing
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-2/Andrew_PVT_noise.png')
def rgb2gray(rgb):
    r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
    gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
    return gray
img=rgb2gray(img)
img_noise = img #+ 20*np.random.randn(*img.shape)
blurred_img1=ndimage.gaussian_filter(img_noise, sigma=1.5) # Multidimensional Gaussian
 filter
blurred_img2=ndimage.median_filter(img_noise,5) # Calculate a multidimensional median f
ilter
plt.figure(1)
plt.imshow(img,cmap='gray')
plt.title("Original Image")
plt.figure(2)
plt.imshow(img_noise,cmap='gray')
plt.title("Noisy Image")
plt.figure(3)
plt.imshow(blurred_img1,cmap='gray')
plt.title("Gaussian Filtered Image")
plt.figure(4)
plt.imshow(blurred_img2,cmap='gray')
plt.title("Median Filtered Image")
err1=img - blurred_img1
err2=img - blurred_img2
sqerr1 = np.power(err1,2)
sqerr2 = np.power(err2,2)
print(sqerr1.mean())
print(sqerr2.mean())
```

```
0.00075363094
0.00034207854
```









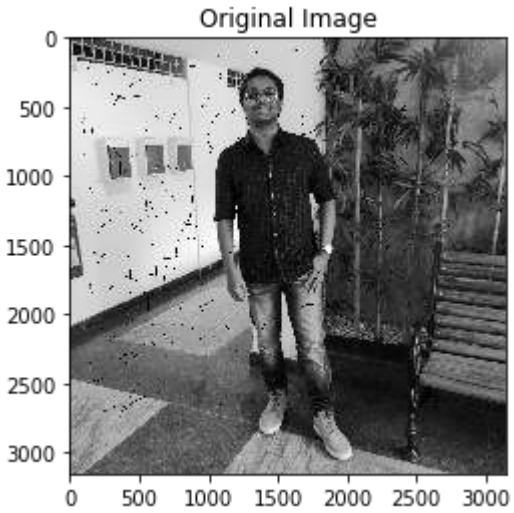
In [ ]:

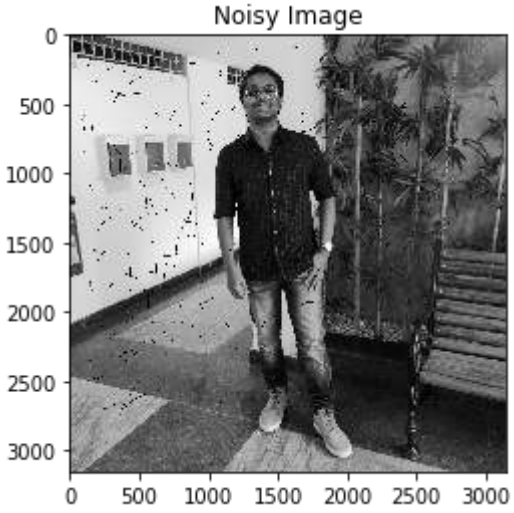
```
#Gaussian and Median Filters
#External Noise
#Sigma = 5
#Size = 7
#Noise = 0
```

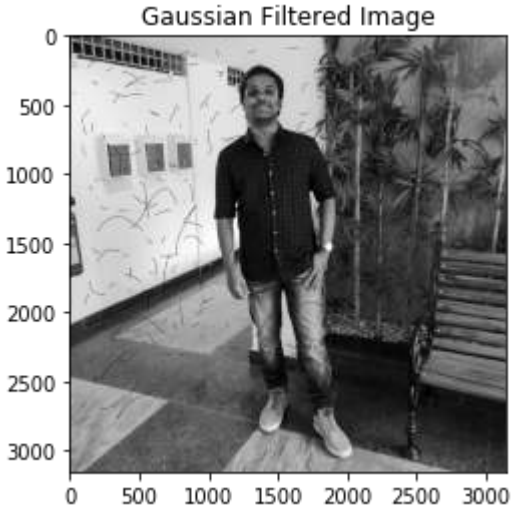
In [28]:

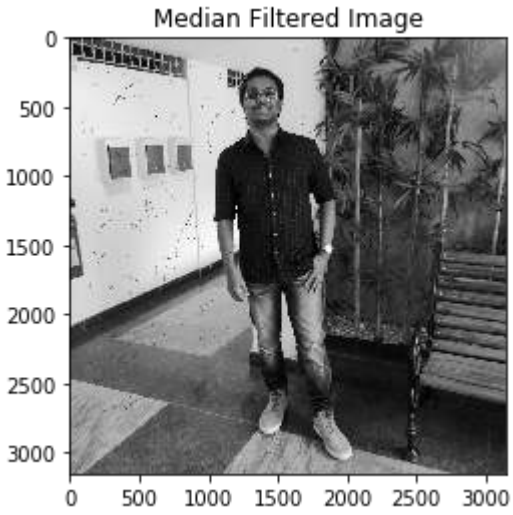
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from scipy import ndimage # Multi-dimensional image processing
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-2/Andrew_PVT_noise.png')
def rgb2gray(rgb):
    r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
    gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
    return gray
img=rgb2gray(img)
img_noise = img #+ 20*np.random.randn(*img.shape)
blurred_img1=ndimage.gaussian_filter(img_noise, sigma=5) # Multidimensional Gaussian fi
lter
blurred_img2=ndimage.median_filter(img_noise,7) # Calculate a multidimensional median f
ilter
plt.figure(1)
plt.imshow(img,cmap='gray')
plt.title("Original Image")
plt.figure(2)
plt.imshow(img_noise,cmap='gray')
plt.title("Noisy Image")
plt.figure(3)
plt.imshow(blurred_img1,cmap='gray')
plt.title("Gaussian Filtered Image")
plt.figure(4)
plt.imshow(blurred_img2,cmap='gray')
plt.title("Median Filtered Image")
err1=img - blurred_img1
err2=img - blurred_img2
sqerr1 = np.power(err1,2)
sqerr2 = np.power(err2,2)
print(sqerr1.mean())
print(sqerr2.mean())
```

```
0.003041504
0.0013957005
```









In [ ]:

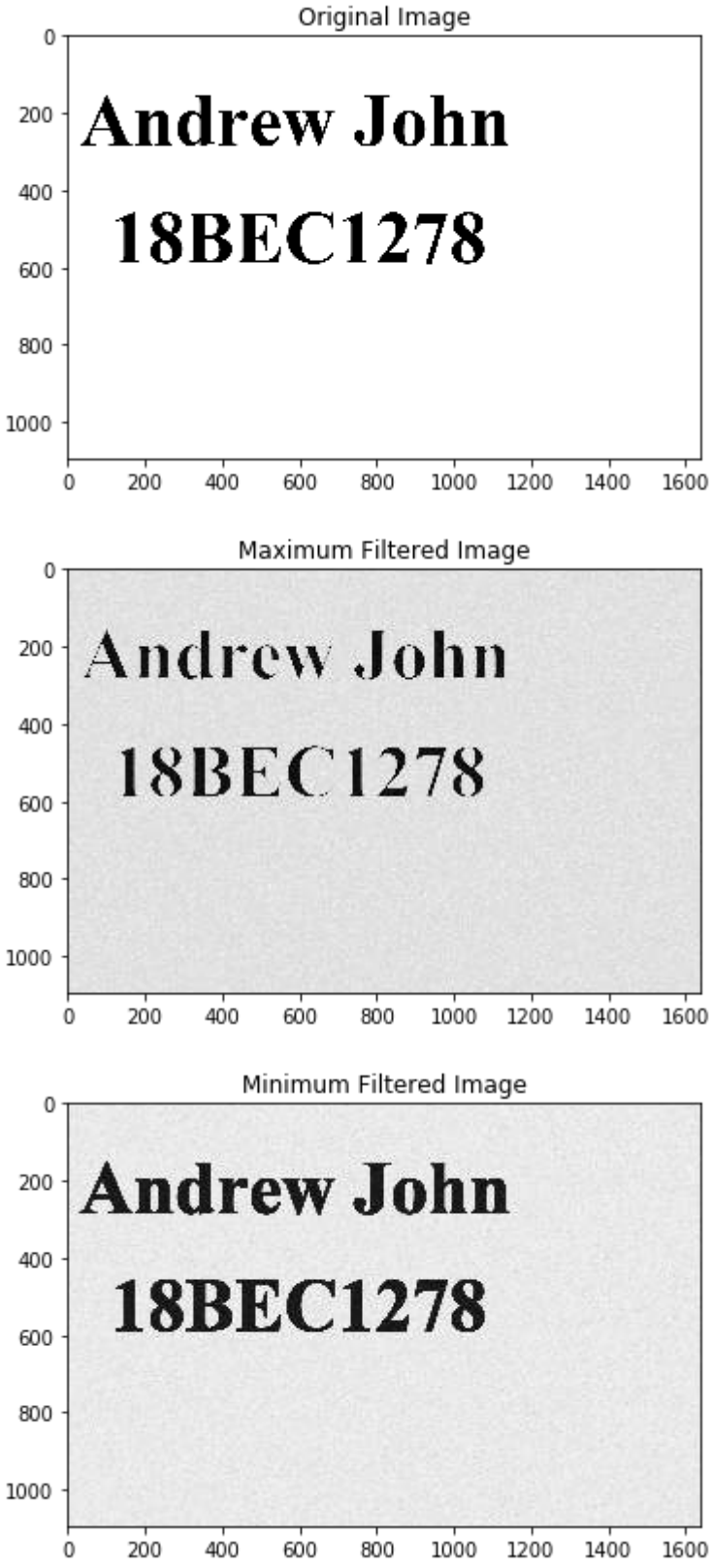
```
'''
-> We can observe that as we increase the sigma value the picture
gets more and more blurred.
-> We can observe that as we increase the median size, they pictures gets
smoothed but along with that we are starting to observe a lose in the
important details of the image.
-> Taking all these into consideration we can infer that gaussin
filtering is better than median filtering.
'''
```

In [58]:

```
#Maximum and Minimum Filters
#Noise = 0.05
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from scipy import ndimage # Multi-dimensional image processing
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-2/Andrew_Reg.png')
def rgb2gray(rgb):
    r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
    gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
    return gray
img=rgb2gray(img)
img_noise = img + 0.05* np.random.randn(*img.shape)
blurred_img1=ndimage.maximum_filter(img_noise,5) # Multidimensional Gaussian filter
blurred_img2=ndimage.minimum_filter(img_noise,5) # Calculate a multidimensional median
 filter
plt.figure(1)
plt.imshow(img,cmap='gray')
plt.title("Original Image")
plt.figure(2)
plt.imshow(blurred_img1,cmap='gray')
plt.title("Maximum Filtered Image")
plt.figure(3)
plt.imshow(blurred_img2,cmap='gray')
plt.title("Minimum Filtered Image")
```

Out[58]:

```
Text(0.5, 1.0, 'Minimum Filtered Image')
```

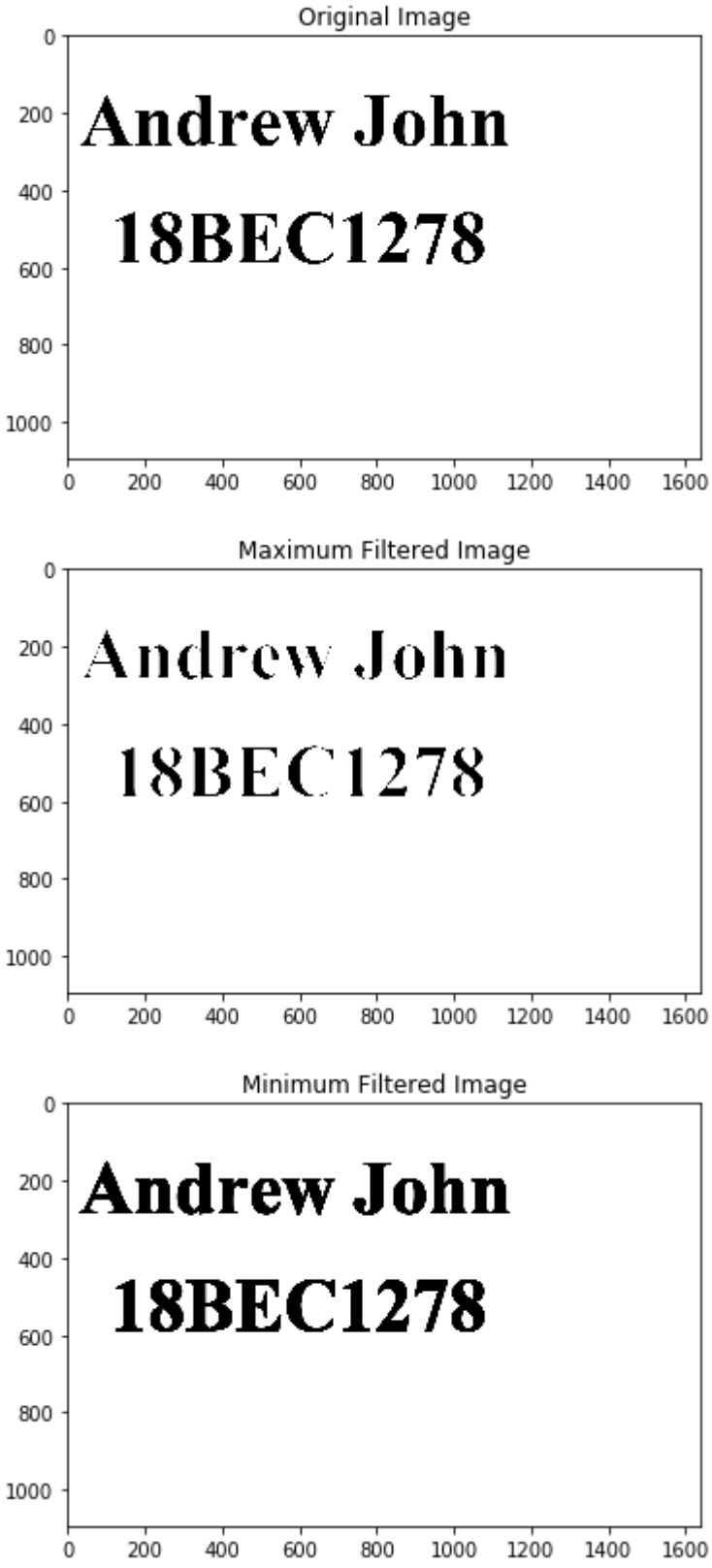


In [59]:

```
#Maximum and Minimum Filters
#Noise = 0
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from scipy import ndimage # Multi-dimensional image processing
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-2/Andrew_Reg.png')
def rgb2gray(rgb):
    r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
    gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
    return gray
img=rgb2gray(img)
img_noise = img + 0* np.random.randn(*img.shape)
blurred_img1=ndimage.maximum_filter(img_noise,5) # Multidimensional Gaussian filter
blurred_img2=ndimage.minimum_filter(img_noise,5) # Calculate a multidimensional median
 filter
plt.figure(1)
plt.imshow(img,cmap='gray')
plt.title("Original Image")
plt.figure(2)
plt.imshow(blurred_img1,cmap='gray')
plt.title("Maximum Filtered Image")
plt.figure(3)
plt.imshow(blurred_img2,cmap='gray')
plt.title("Minimum Filtered Image")
```

Out[59]:

```
Text(0.5, 1.0, 'Minimum Filtered Image')
```

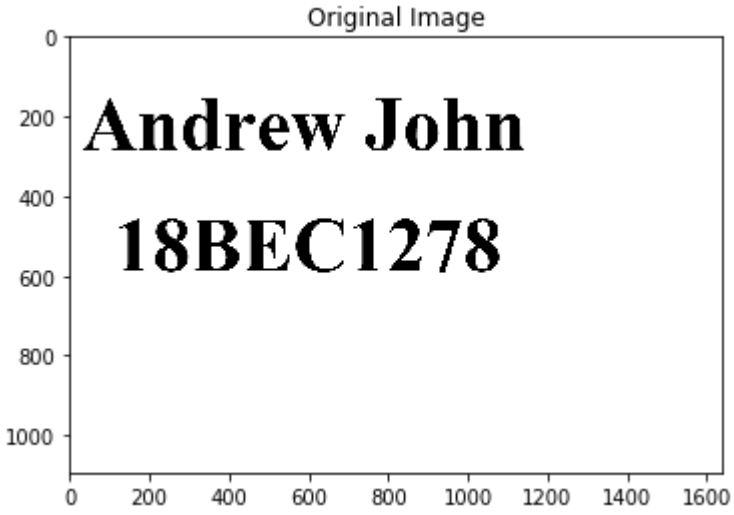


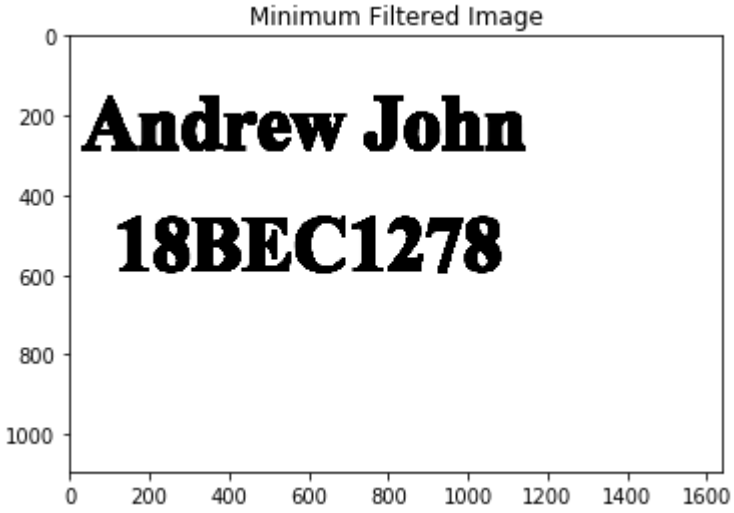
In [60]:

```
#Maximum and Minimum Filters
#Noise = 0
#Increasing the values of Maximum and Minimum Filter to 10
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from scipy import ndimage # Multi-dimensional image processing
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-2/Andrew_Reg.png')
def rgb2gray(rgb):
    r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
    gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
    return gray
img=rgb2gray(img)
img_noise = img + 0* np.random.randn(*img.shape)
blurred_img1=ndimage.maximum_filter(img_noise,10) # Multidimensional Gaussian filter
blurred_img2=ndimage.minimum_filter(img_noise,10) # Calculate a multidimensional median
filter
plt.figure(1)
plt.imshow(img,cmap='gray')
plt.title("Original Image")
plt.figure(2)
plt.imshow(blurred_img1,cmap='gray')
plt.title("Maximum Filtered Image")
plt.figure(3)
plt.imshow(blurred_img2,cmap='gray')
plt.title("Minimum Filtered Image")
```

Out[60]:

```
Text(0.5, 1.0, 'Minimum Filtered Image')
```



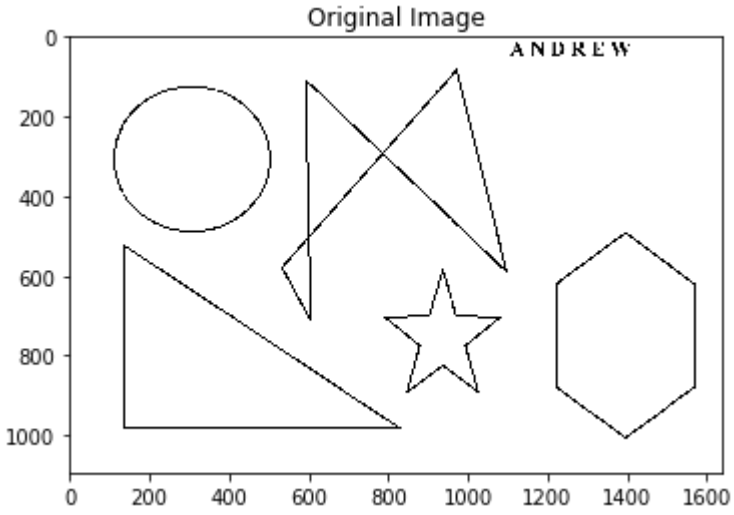

Minimum Filtered Image
Andrew John
18BEC1278
0
200
400
600
800
1000
0
200
400
600
800
1000
1200
1400
1600

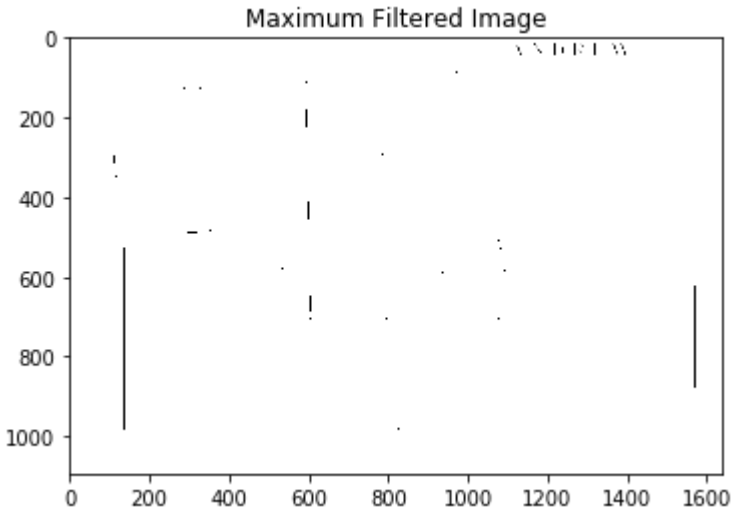
In [61]:

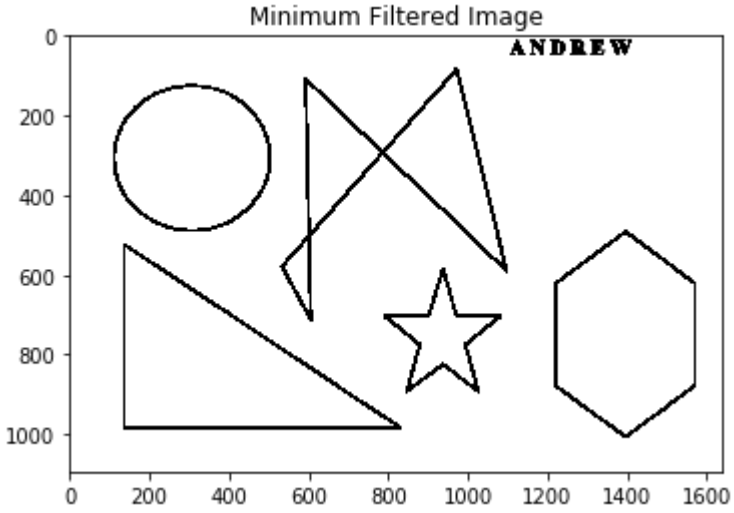
```
#Maximum and Minimum Filters
#Noise = 0
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from scipy import ndimage # Multi-dimensional image processing
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-2/Andrew_shapes.png')
def rgb2gray(rgb):
    r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
    gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
    return gray
img=rgb2gray(img)
img_noise = img + 0* np.random.randn(*img.shape)
blurred_img1=ndimage.maximum_filter(img_noise,5) # Multidimensional Gaussian filter
blurred_img2=ndimage.minimum_filter(img_noise,5) # Calculate a multidimensional median
 filter
plt.figure(1)
plt.imshow(img,cmap='gray')
plt.title("Original Image")
plt.figure(2)
plt.imshow(blurred_img1,cmap='gray')
plt.title("Maximum Filtered Image")
plt.figure(3)
plt.imshow(blurred_img2,cmap='gray')
plt.title("Minimum Filtered Image")
```

Out[61]:

```
Text(0.5, 1.0, 'Minimum Filtered Image')
```





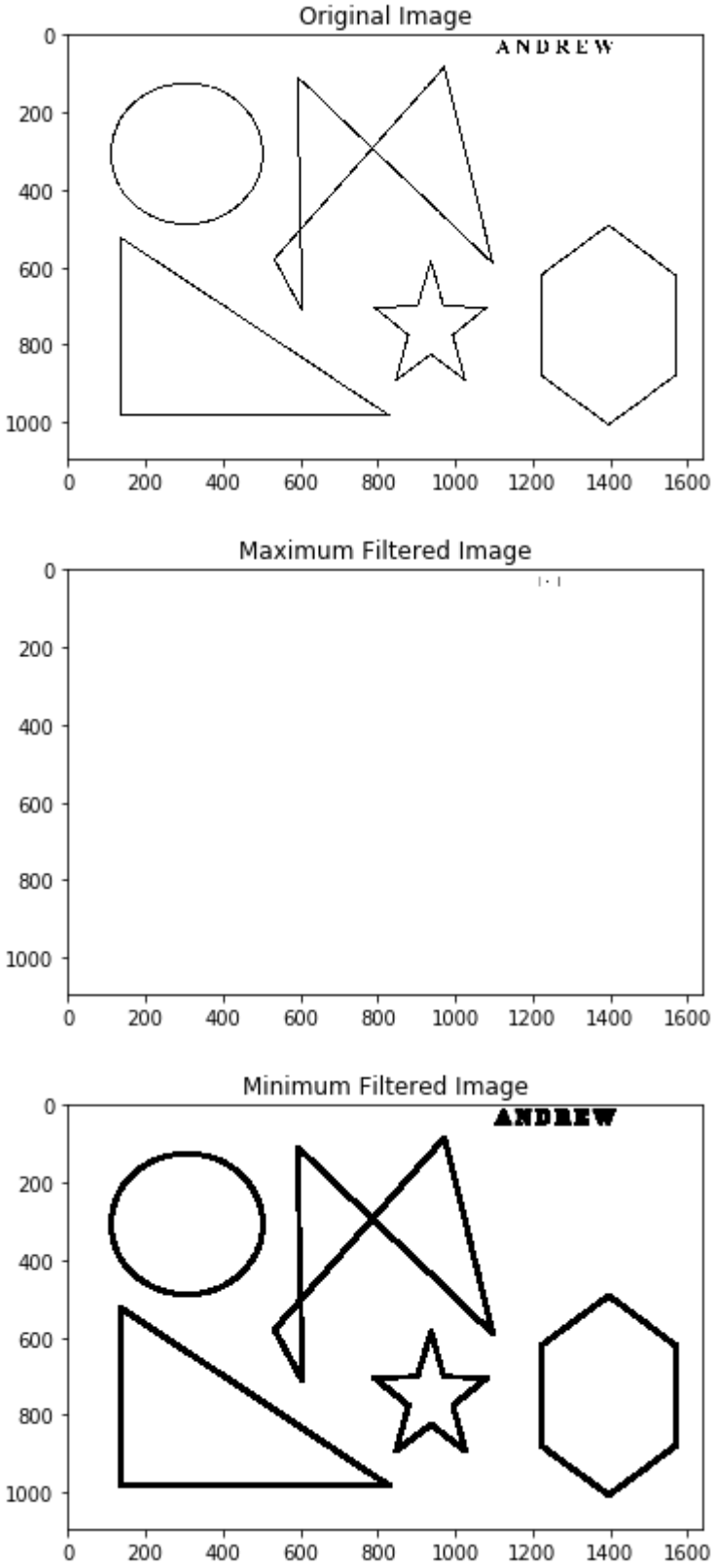

Minimum Filtered Image
ANDREW
0
200
400
600
800
1000
0
200
400
600
800
1000
1200
1400
1600

In [62]:

```
#Maximum and Minimum Filters
#Noise = 0
#Increasing the values of Maximum and Minimum Filter to 10
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from scipy import ndimage # Multi-dimensional image processing
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-2/Andrew_shapes.png')
def rgb2gray(rgb):
    r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
    gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
    return gray
img=rgb2gray(img)
img_noise = img + 0* np.random.randn(*img.shape)
blurred_img1=ndimage.maximum_filter(img_noise,10) # Multidimensional Gaussian filter
blurred_img2=ndimage.minimum_filter(img_noise,10) # Calculate a multidimensional median
filter
plt.figure(1)
plt.imshow(img,cmap='gray')
plt.title("Original Image")
plt.figure(2)
plt.imshow(blurred_img1,cmap='gray')
plt.title("Maximum Filtered Image")
plt.figure(3)
plt.imshow(blurred_img2,cmap='gray')
plt.title("Minimum Filtered Image")
```

Out[62]:

```
Text(0.5, 1.0, 'Minimum Filtered Image')
```



In [ ]:

In [ ]:

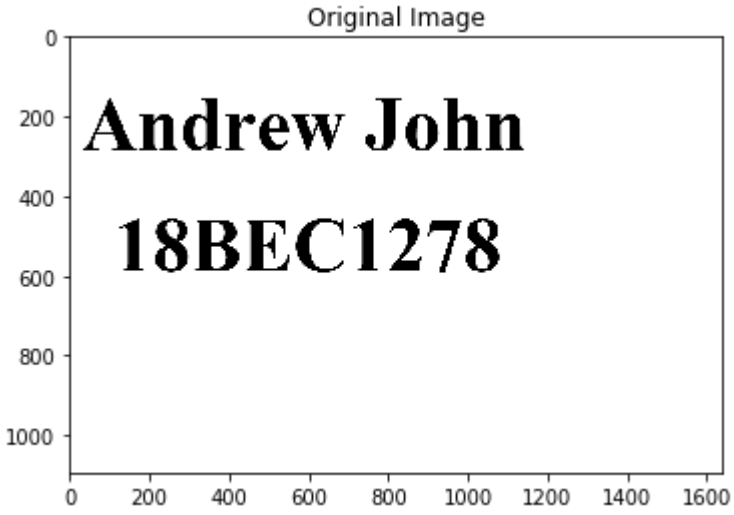
```
#SHARPENING IMAGES - Treating the input image as Color Image
#Sigma = 1.5
```

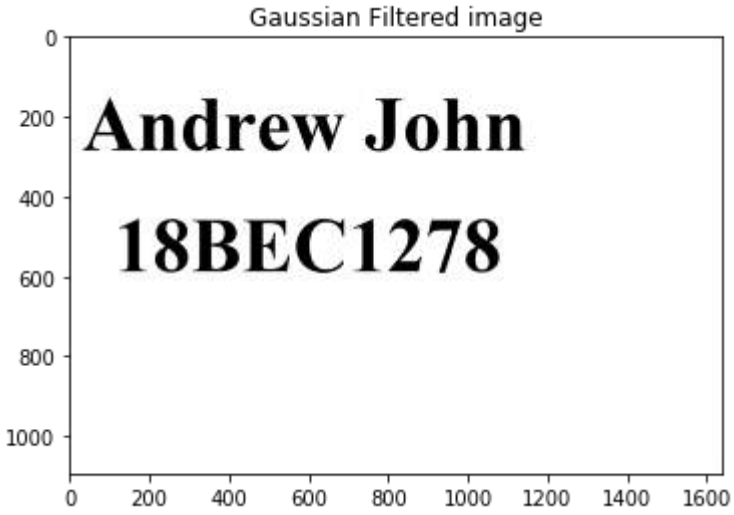
In [32]:

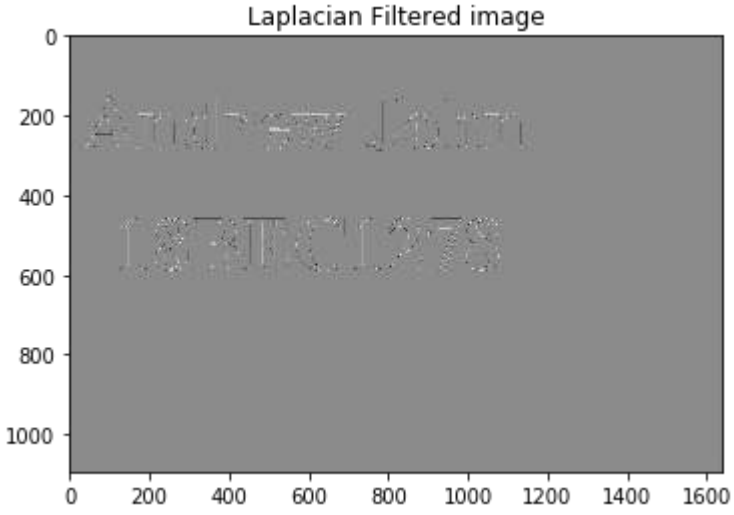
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from scipy import ndimage # Multi-dimensional image processing
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-2/Andrew_Reg.png')
def rgb2gray(rgb):
    r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
    gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
    return gray
img=rgb2gray(img)
blurred_img=ndimage.gaussian_filter(img, sigma=1.5)  # Gaussian Filter
sharpened_img=ndimage.laplace(img)                   # Laplacian Filter
sharpened_img2=ndimage.laplace(blurred_img)          # Laplacian on Gaussian
sharpened_img3=img - blurred_img                     # Original minus Gaussian

plt.figure(1)
plt.imshow(img,cmap='gray')
plt.title("Original Image")
plt.figure(2)
plt.imshow(blurred_img,cmap='gray')
plt.title("Gaussian Filtered image")
plt.figure(3)
plt.imshow(sharpened_img,cmap='gray')
plt.title("Laplacian Filtered image")
plt.figure(4)
plt.imshow(sharpened_img2,cmap='gray')
plt.title("Laplacian on Gaussian")
plt.figure(5)
plt.imshow(sharpened_img3,cmap='gray')
plt.title("Original Minus Gaussian")
```

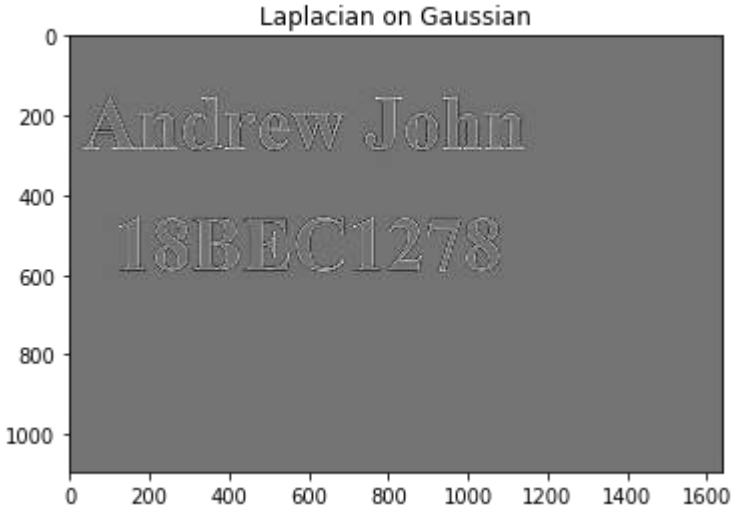
Out[32]:

```
Text(0.5, 1.0, 'Original Minus Gaussian')
```









In [ ]:

```
#SHARPENING IMAGES - Treating the input image as B&W Image
#Sigma = 1.5
```

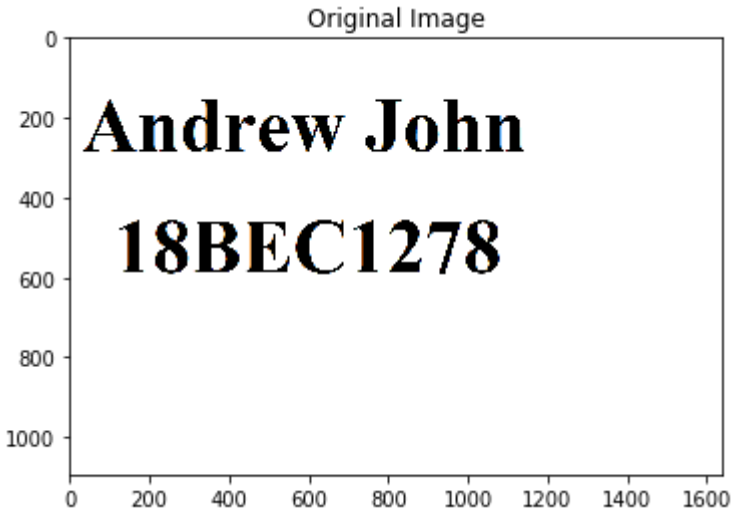
In [31]:

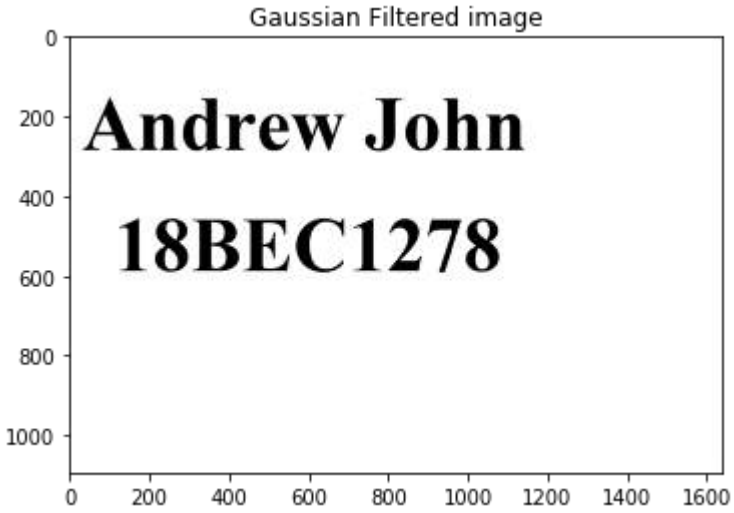
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from scipy import ndimage # Multi-dimensional image processing
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-2/Andrew_Reg.png')
'''def rgb2gray(rgb):
    r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
    gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
    return gray
img=rgb2gray(img)'''
blurred_img=ndimage.gaussian_filter(img, sigma=1.5)  # Gaussian Filter
sharpened_img=ndimage.laplace(img)                   # Laplacian Filter
sharpened_img2=ndimage.laplace(blurred_img)          # Laplacian on Gaussian
sharpened_img3=img - blurred_img                     # Original minus Gaussian

plt.figure(1)
plt.imshow(img,cmap='gray')
plt.title("Original Image")
plt.figure(2)
plt.imshow(blurred_img,cmap='gray')
plt.title("Gaussian Filtered image")
plt.figure(3)
plt.imshow(sharpened_img,cmap='gray')
plt.title("Laplacian Filtered image")
plt.figure(4)
plt.imshow(sharpened_img2,cmap='gray')
plt.title("Laplacian on Gaussian")
plt.figure(5)
plt.imshow(sharpened_img3,cmap='gray')
plt.title("Original Minus Gaussian")
```

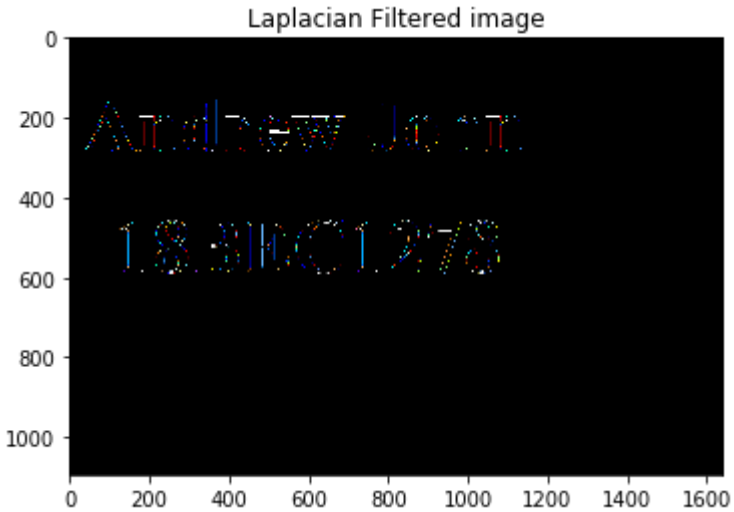
```
Clipping input data to the valid range for imshow with RGB data ([0..1] fo
r floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] fo
r floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] fo
r floats or [0..255] for integers).
```

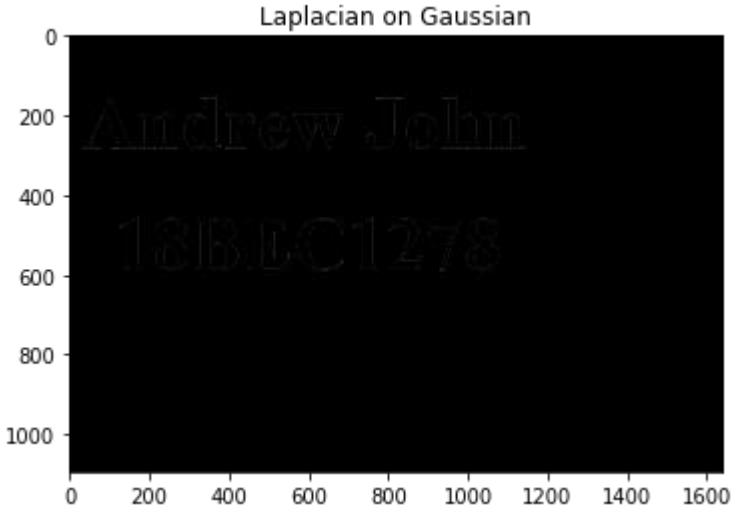
Out[31]:

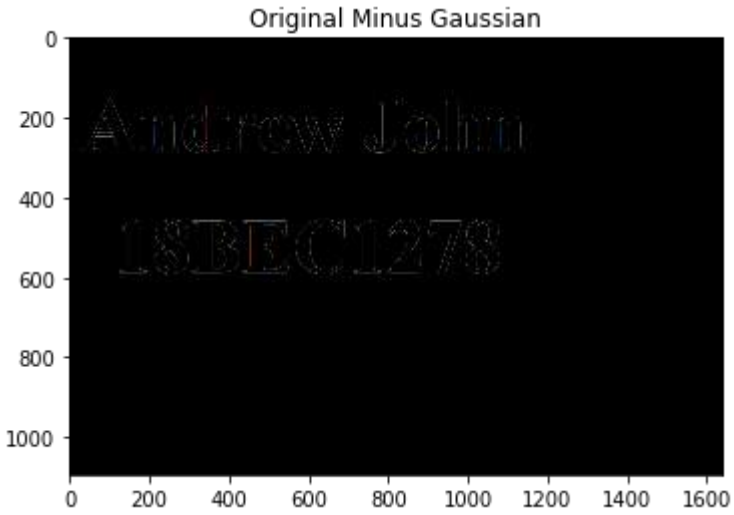
```
Text(0.5, 1.0, 'Original Minus Gaussian')
```











In [ ]:

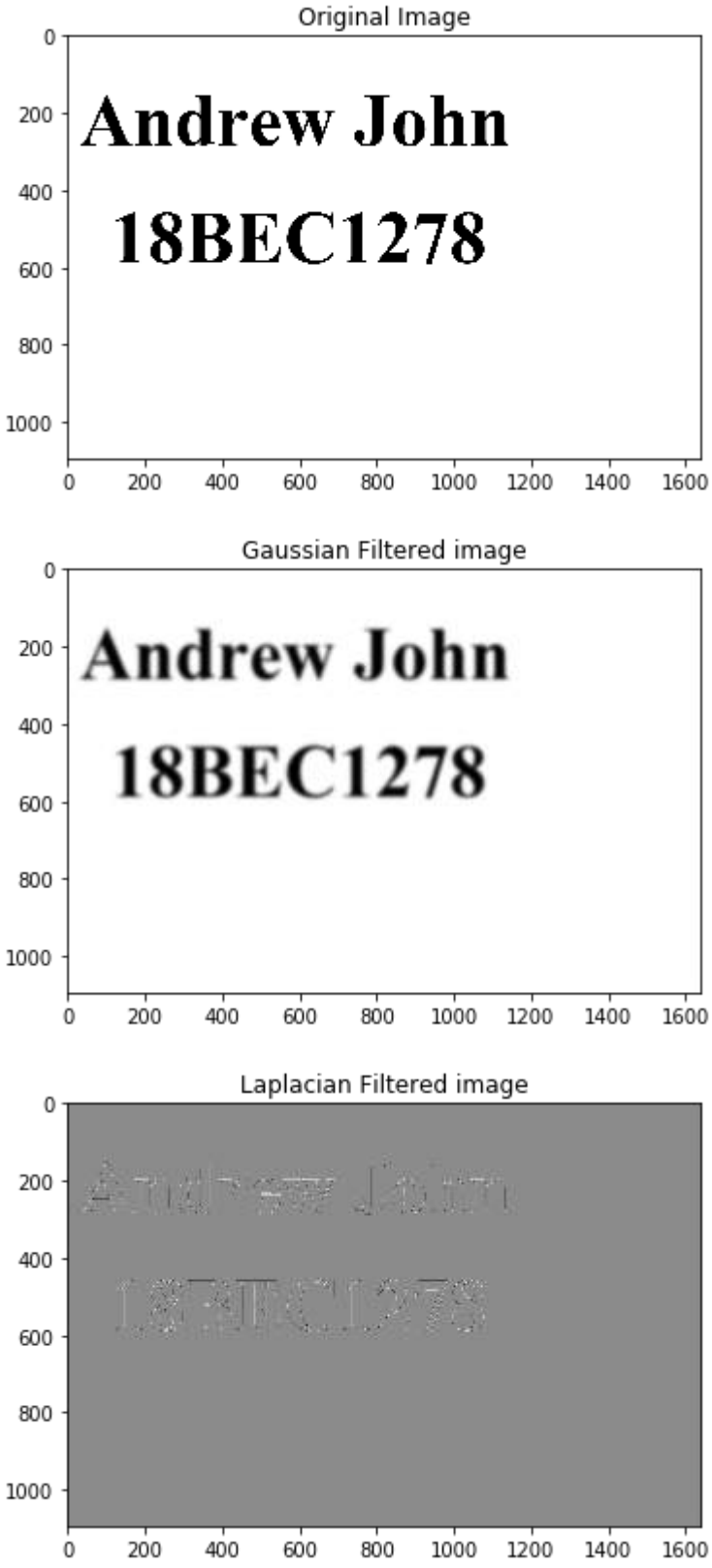
```
#SHARPENING IMAGES - Treating the input image as Color Image
#Sigma = 5
```

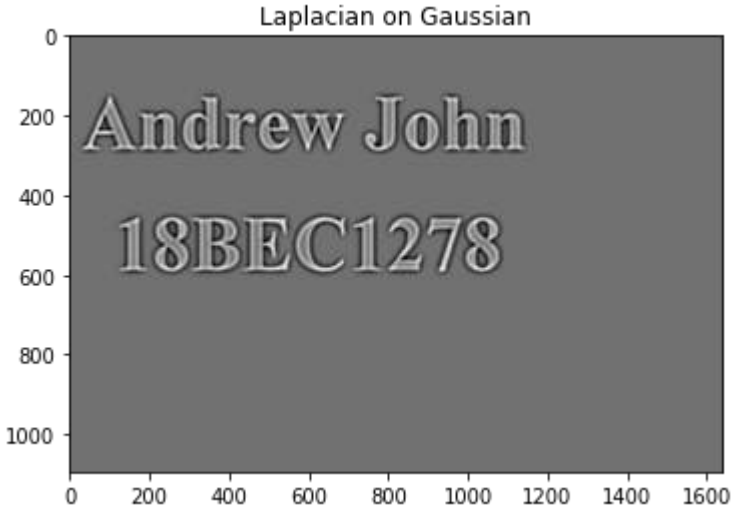
In [33]:

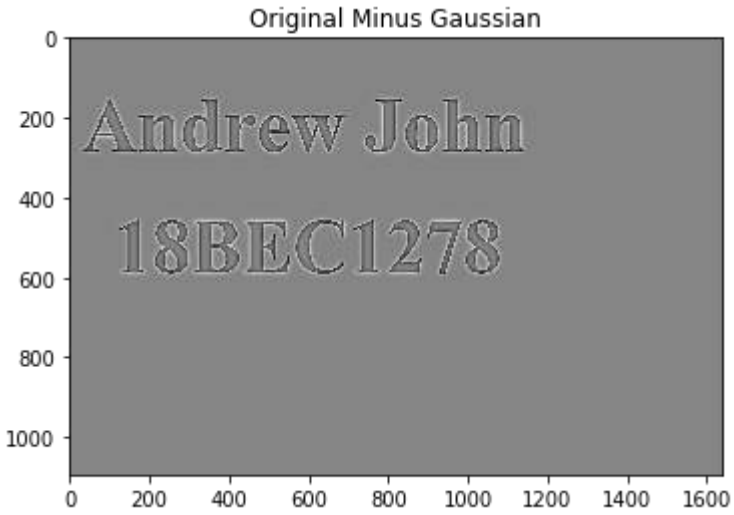
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from scipy import ndimage # Multi-dimensional image processing
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-2/Andrew_Reg.png')
def rgb2gray(rgb):
    r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
    gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
    return gray
img=rgb2gray(img)
blurred_img=ndimage.gaussian_filter(img, sigma=5)  # Gaussian Filter
sharpened_img=ndimage.laplace(img)                   # Laplacian Filter
sharpened_img2=ndimage.laplace(blurred_img)          # Laplacian on Gaussian
sharpened_img3=img - blurred_img                     # Original minus Gaussian

plt.figure(1)
plt.imshow(img,cmap='gray')
plt.title("Original Image")
plt.figure(2)
plt.imshow(blurred_img,cmap='gray')
plt.title("Gaussian Filtered image")
plt.figure(3)
plt.imshow(sharpened_img,cmap='gray')
plt.title("Laplacian Filtered image")
plt.figure(4)
plt.imshow(sharpened_img2,cmap='gray')
plt.title("Laplacian on Gaussian")
plt.figure(5)
plt.imshow(sharpened_img3,cmap='gray')
plt.title("Original Minus Gaussian")
```

Out[33]:

```
Text(0.5, 1.0, 'Original Minus Gaussian')
```







In [ ]:

```
#SHARPENING IMAGES - Treating the input image as B&W Image
#Sigma = 5
```

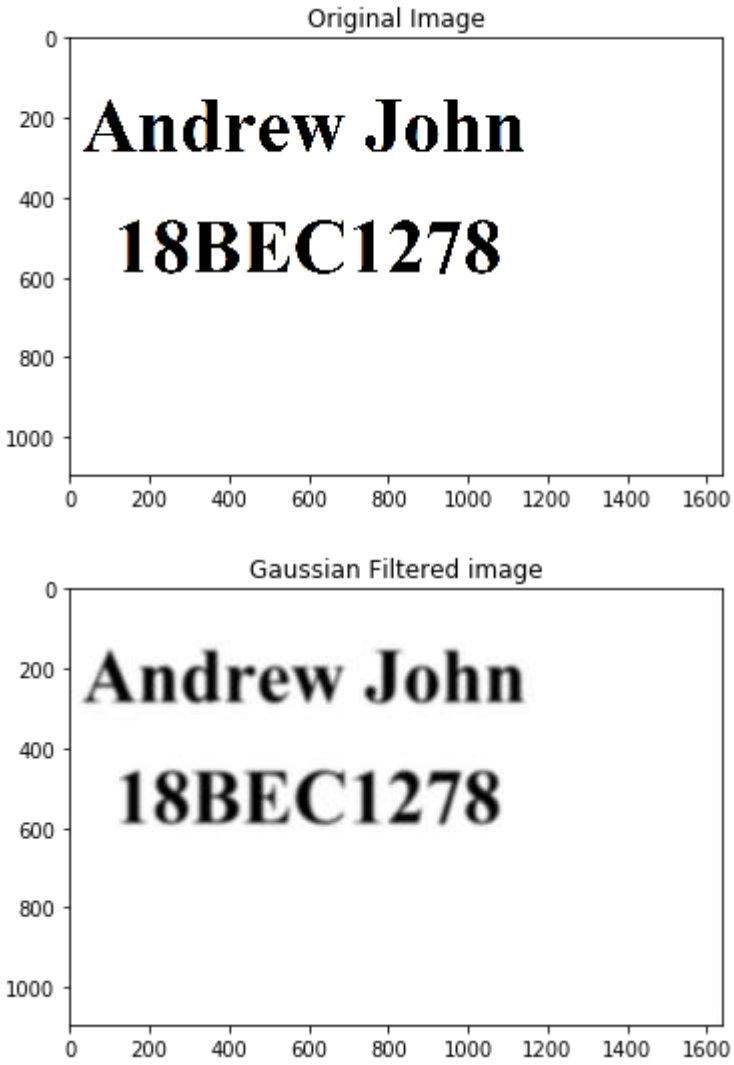
In [34]:

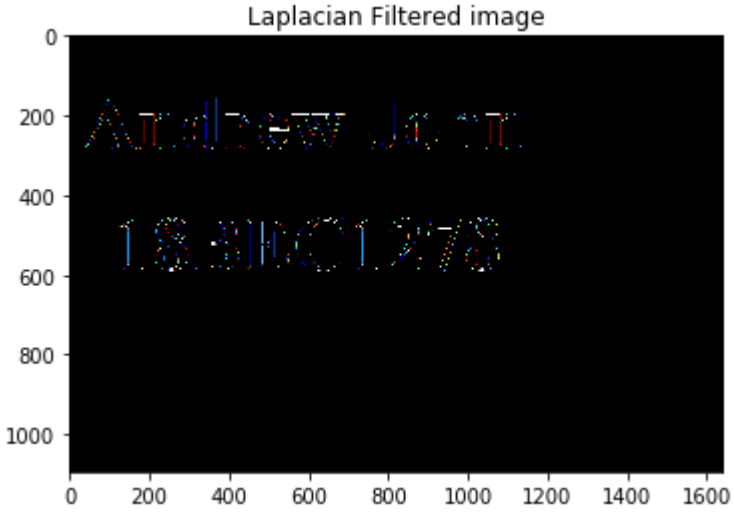
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from scipy import ndimage # Multi-dimensional image processing
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-2/Andrew_Reg.png')
'''def rgb2gray(rgb):
    r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
    gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
    return gray
img=rgb2gray(img)'''
blurred_img=ndimage.gaussian_filter(img, sigma=5)  # Gaussian Filter
sharpened_img=ndimage.laplace(img)                   # Laplacian Filter
sharpened_img2=ndimage.laplace(blurred_img)          # Laplacian on Gaussian
sharpened_img3=img - blurred_img                     # Original minus Gaussian

plt.figure(1)
plt.imshow(img,cmap='gray')
plt.title("Original Image")
plt.figure(2)
plt.imshow(blurred_img,cmap='gray')
plt.title("Gaussian Filtered image")
plt.figure(3)
plt.imshow(sharpened_img,cmap='gray')
plt.title("Laplacian Filtered image")
plt.figure(4)
plt.imshow(sharpened_img2,cmap='gray')
plt.title("Laplacian on Gaussian")
plt.figure(5)
plt.imshow(sharpened_img3,cmap='gray')
plt.title("Original Minus Gaussian")
```

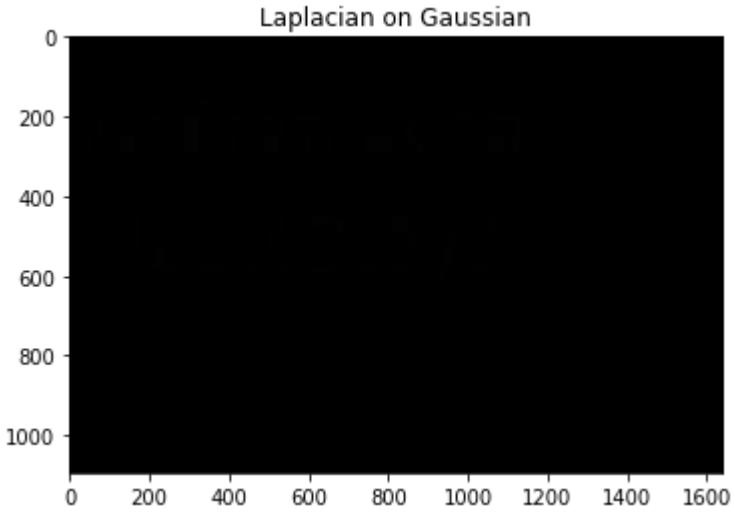
```
Clipping input data to the valid range for imshow with RGB data ([0..1] fo
r floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] fo
r floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] fo
r floats or [0..255] for integers).
```

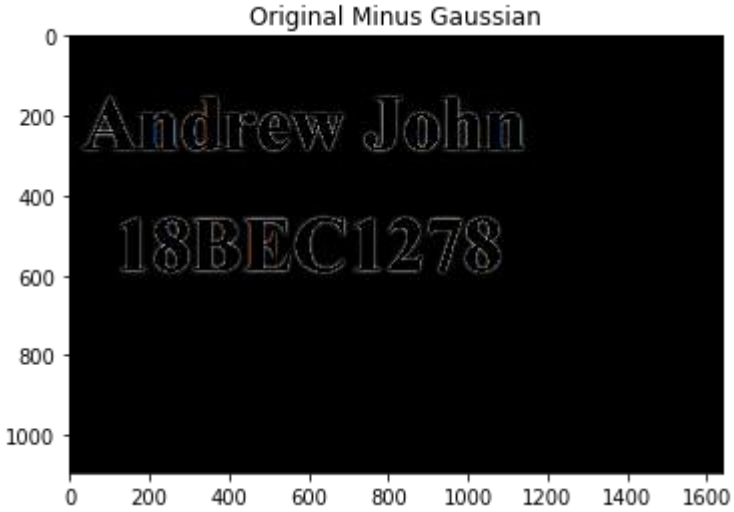
Out[34]:

```
Text(0.5, 1.0, 'Original Minus Gaussian')
```




Laplacian Filtered image
0
200
400
600
800
1000
0
200
400
600
800
1000
1200
1400
1600


Laplacian on Gaussian
0
200
400
600
800
1000
0
200
400
600
800
1000
1200
1400
1600


Original Minus Gaussian
Andrew John
18BEC1278
0
200
400
600
800
1000
0
200
400
600
800
1000
1200
1400
1600

In [ ]:

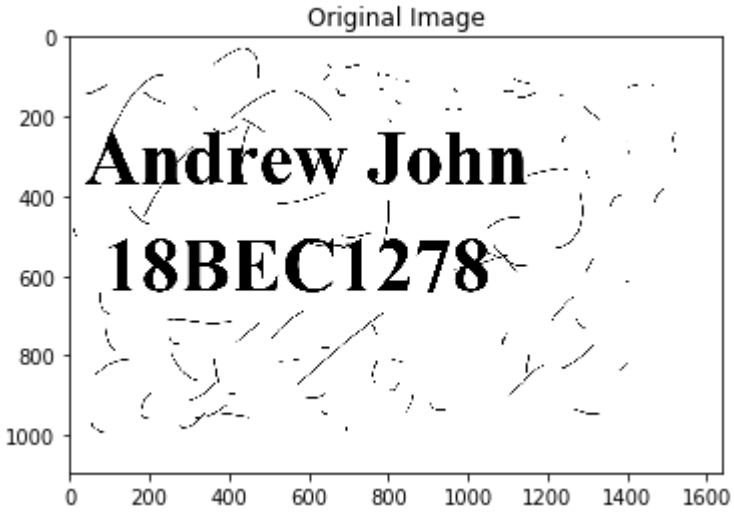
```
#SHARPENING IMAGES - Treating the input image as Color Image
#Sigma = 1.5
#External Noise
```

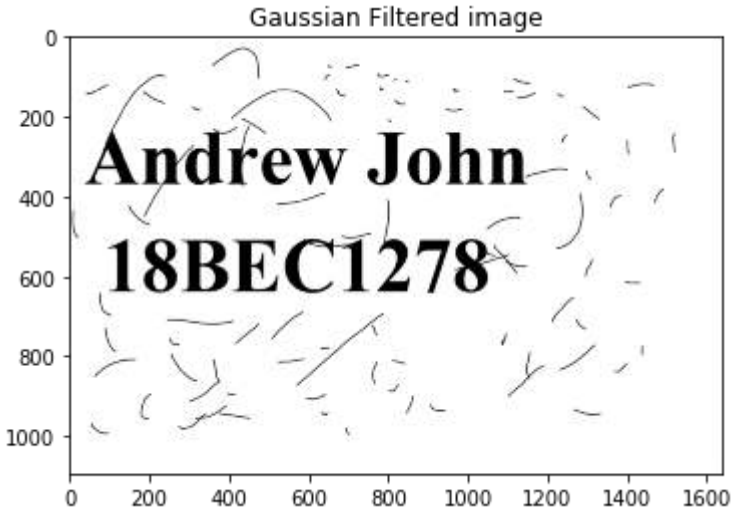
In [53]:

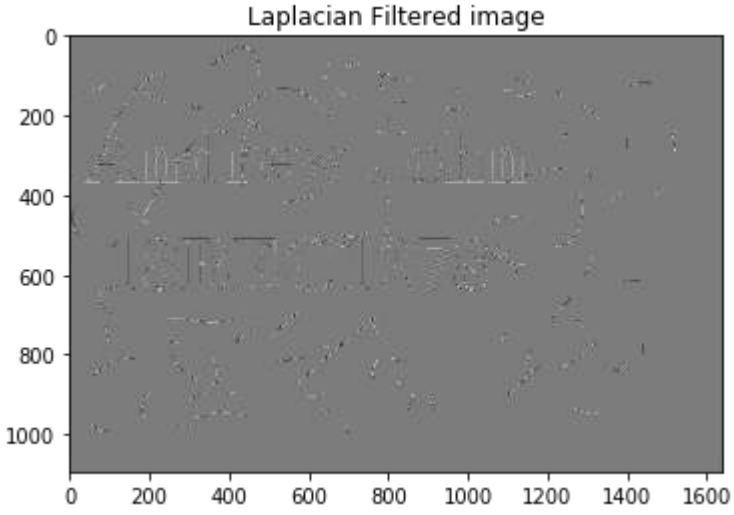
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from scipy import ndimage # Multi-dimensional image processing
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-2/Andrew_Reg_noise.png')
def rgb2gray(rgb):
    r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
    gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
    return gray
img=rgb2gray(img)
blurred_img=ndimage.gaussian_filter(img, sigma=1.5)  # Gaussian Filter
sharpened_img=ndimage.laplace(img)                   # Laplacian Filter
sharpened_img2=ndimage.laplace(blurred_img)          # Laplacian on Gaussian
sharpened_img3=img - blurred_img                     # Original minus Gaussian

plt.figure(1)
plt.imshow(img,cmap='gray')
plt.title("Original Image")
plt.figure(2)
plt.imshow(blurred_img,cmap='gray')
plt.title("Gaussian Filtered image")
plt.figure(3)
plt.imshow(sharpened_img,cmap='gray')
plt.title("Laplacian Filtered image")
plt.figure(4)
plt.imshow(sharpened_img2,cmap='gray')
plt.title("Laplacian on Gaussian")
plt.figure(5)
plt.imshow(sharpened_img3,cmap='gray')
plt.title("Original Minus Gaussian")
```

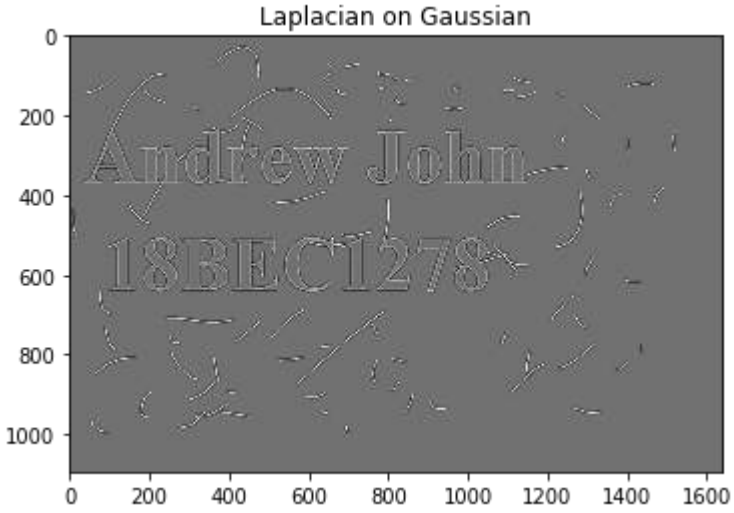
Out[53]:

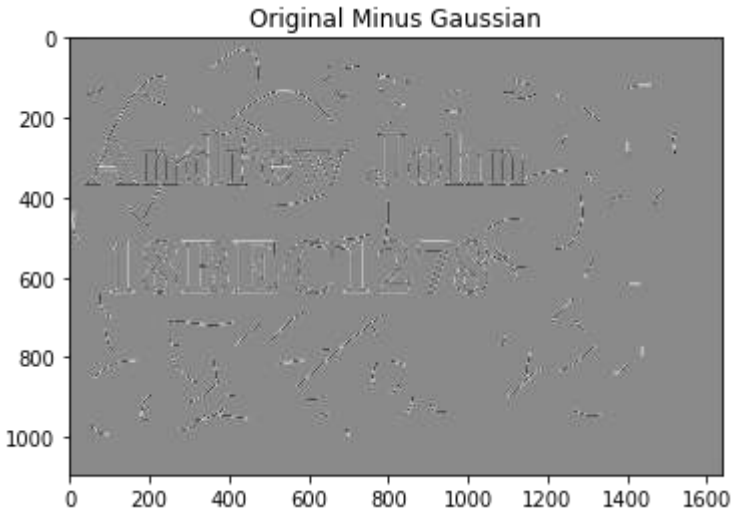
```
Text(0.5, 1.0, 'Original Minus Gaussian')
```











In [54]:

```
#SHARPENING IMAGES - Treating the input image as B&W Image
#Sigma = 1.5
#External Noise
```

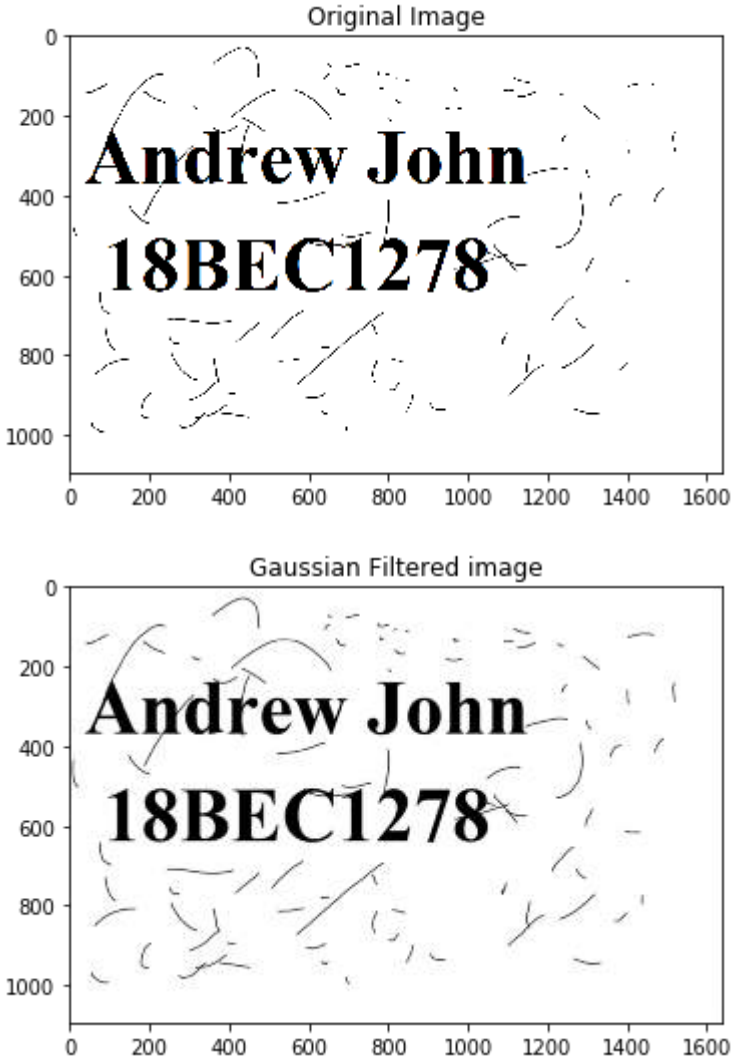
In [55]:

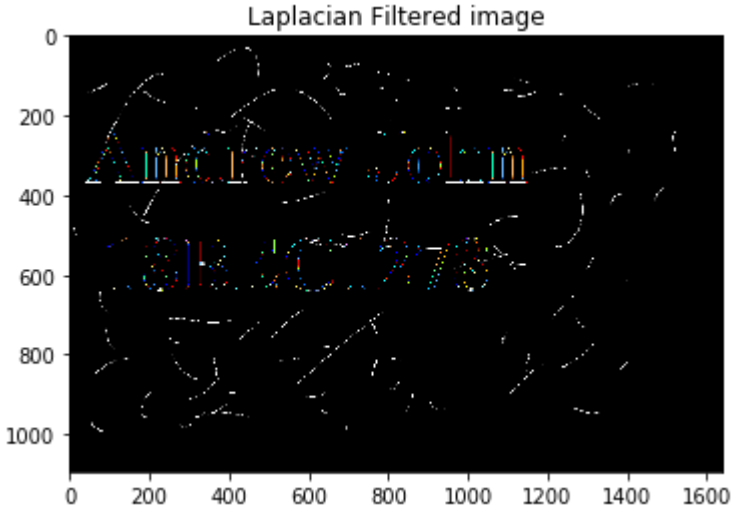
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from scipy import ndimage # Multi-dimensional image processing
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-2/Andrew_Reg_noise.png')
'''def rgb2gray(rgb):
    r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
    gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
    return gray
img=rgb2gray(img)'''
blurred_img=ndimage.gaussian_filter(img, sigma=1.5)  # Gaussian Filter
sharpened_img=ndimage.laplace(img)                   # Laplacian Filter
sharpened_img2=ndimage.laplace(blurred_img)          # Laplacian on Gaussian
sharpened_img3=img - blurred_img                     # Original minus Gaussian

plt.figure(1)
plt.imshow(img,cmap='gray')
plt.title("Original Image")
plt.figure(2)
plt.imshow(blurred_img,cmap='gray')
plt.title("Gaussian Filtered image")
plt.figure(3)
plt.imshow(sharpened_img,cmap='gray')
plt.title("Laplacian Filtered image")
plt.figure(4)
plt.imshow(sharpened_img2,cmap='gray')
plt.title("Laplacian on Gaussian")
plt.figure(5)
plt.imshow(sharpened_img3,cmap='gray')
plt.title("Original Minus Gaussian")
```

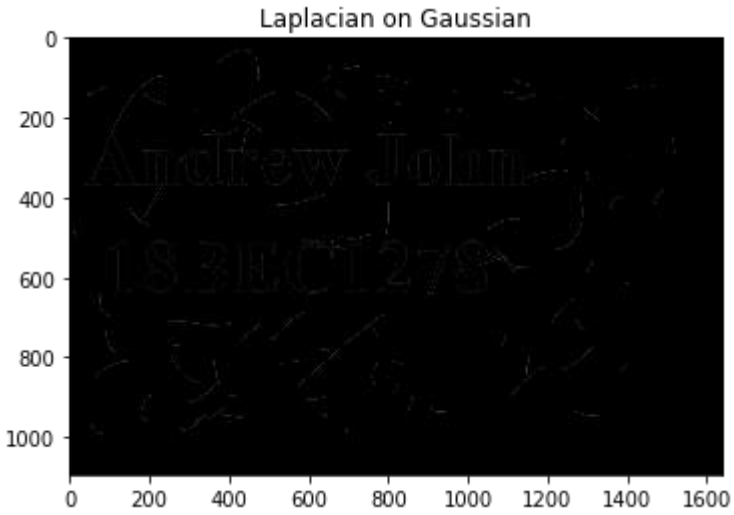
```
Clipping input data to the valid range for imshow with RGB data ([0..1] fo
r floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] fo
r floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] fo
r floats or [0..255] for integers).
```

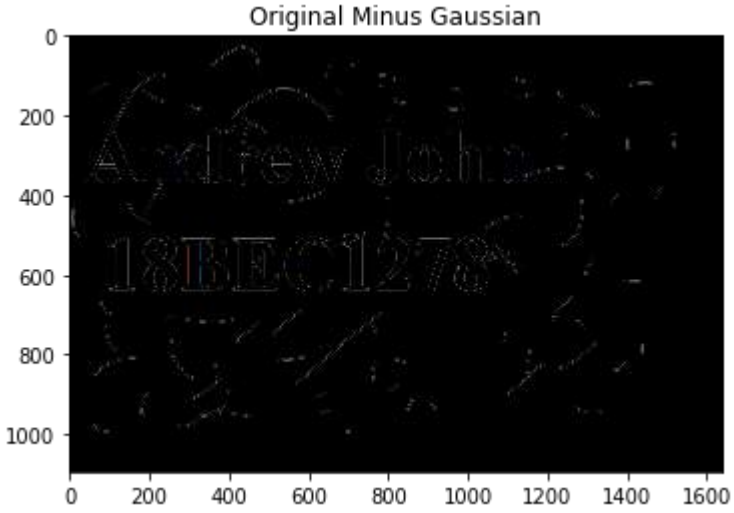
Out[55]:

```
Text(0.5, 1.0, 'Original Minus Gaussian')
```




Laplacian Filtered image
0
200
400
600
800
1000
0
200
400
600
800
1000
1200
1400
1600


Laplacian on Gaussian
0
200
400
600
800
1000
0
200
400
600
800
1000
1200
1400
1600



In [ ]:

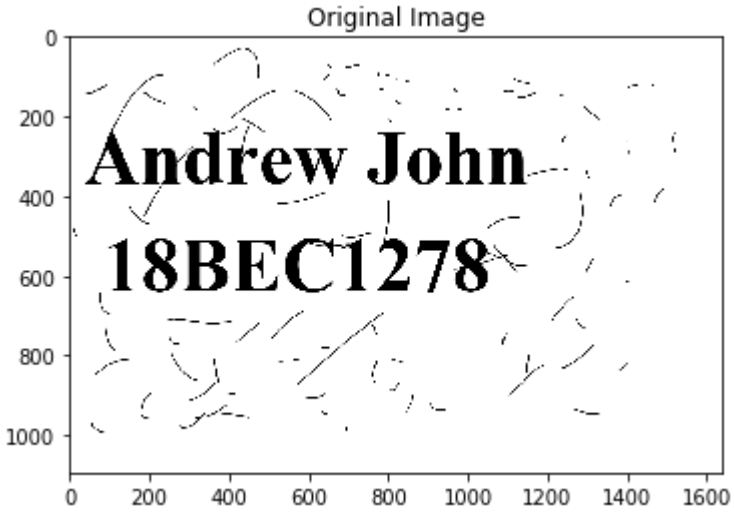
```
#SHARPENING IMAGES - Treating the input image as Color Image
#Sigma = 5
#External Noise
```

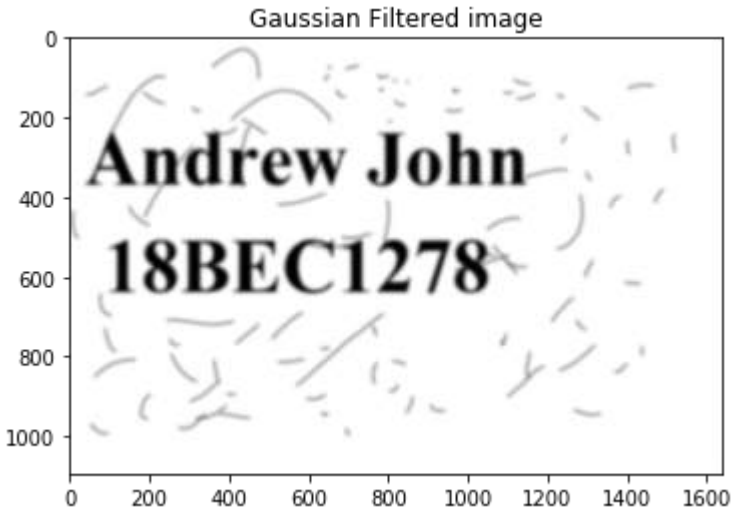
In [56]:

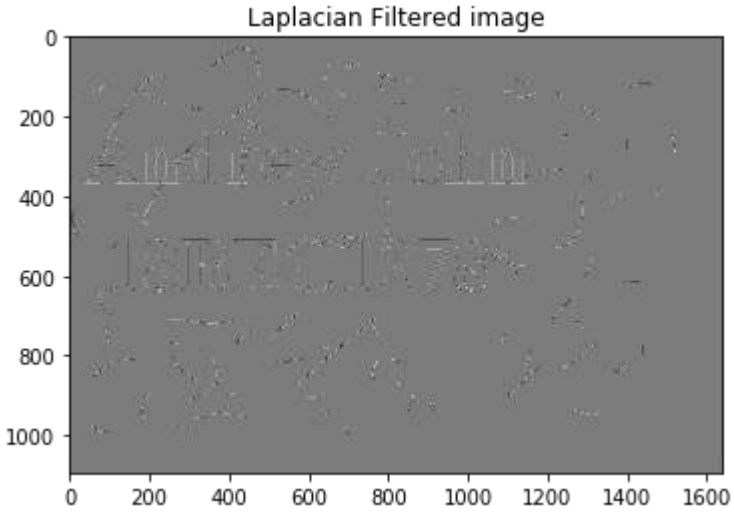
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from scipy import ndimage # Multi-dimensional image processing
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-2/Andrew_Reg_noise.png')
def rgb2gray(rgb):
    r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
    gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
    return gray
img=rgb2gray(img)
blurred_img=ndimage.gaussian_filter(img, sigma=5)  # Gaussian Filter
sharpened_img=ndimage.laplace(img)                   # Laplacian Filter
sharpened_img2=ndimage.laplace(blurred_img)          # Laplacian on Gaussian
sharpened_img3=img - blurred_img                     # Original minus Gaussian

plt.figure(1)
plt.imshow(img,cmap='gray')
plt.title("Original Image")
plt.figure(2)
plt.imshow(blurred_img,cmap='gray')
plt.title("Gaussian Filtered image")
plt.figure(3)
plt.imshow(sharpened_img,cmap='gray')
plt.title("Laplacian Filtered image")
plt.figure(4)
plt.imshow(sharpened_img2,cmap='gray')
plt.title("Laplacian on Gaussian")
plt.figure(5)
plt.imshow(sharpened_img3,cmap='gray')
plt.title("Original Minus Gaussian")
```

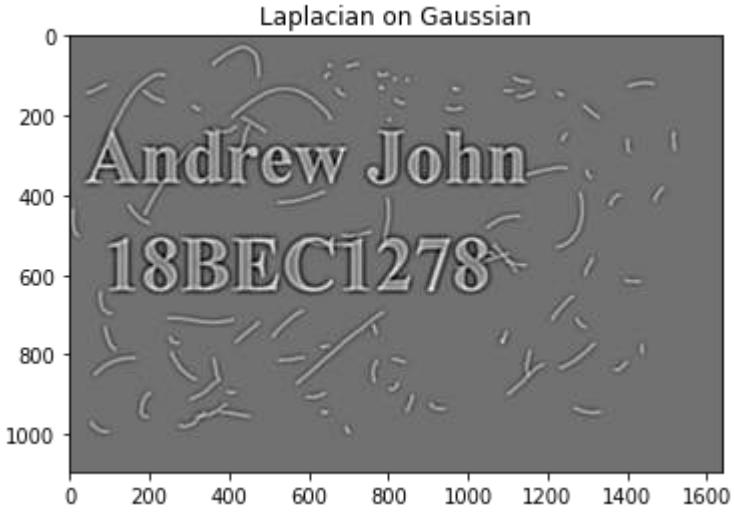
Out[56]:

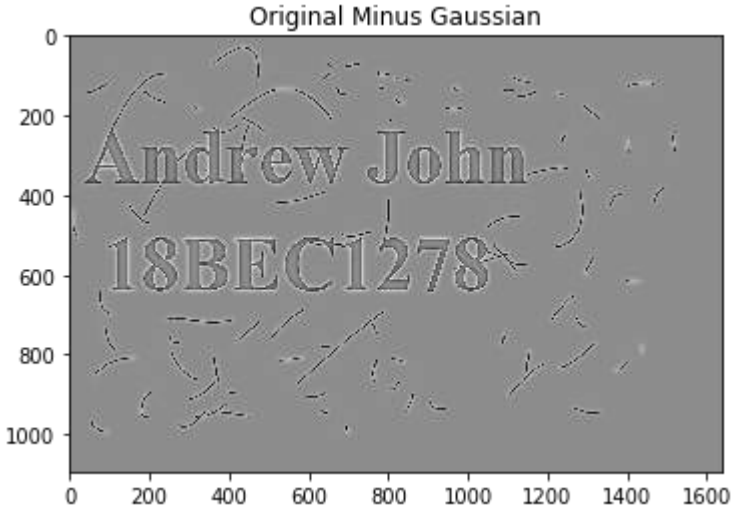
```
Text(0.5, 1.0, 'Original Minus Gaussian')
```











In [ ]:

```
#SHARPENING IMAGES - Treating the input image as B&W Image
#Sigma = 5
#External Noise
```

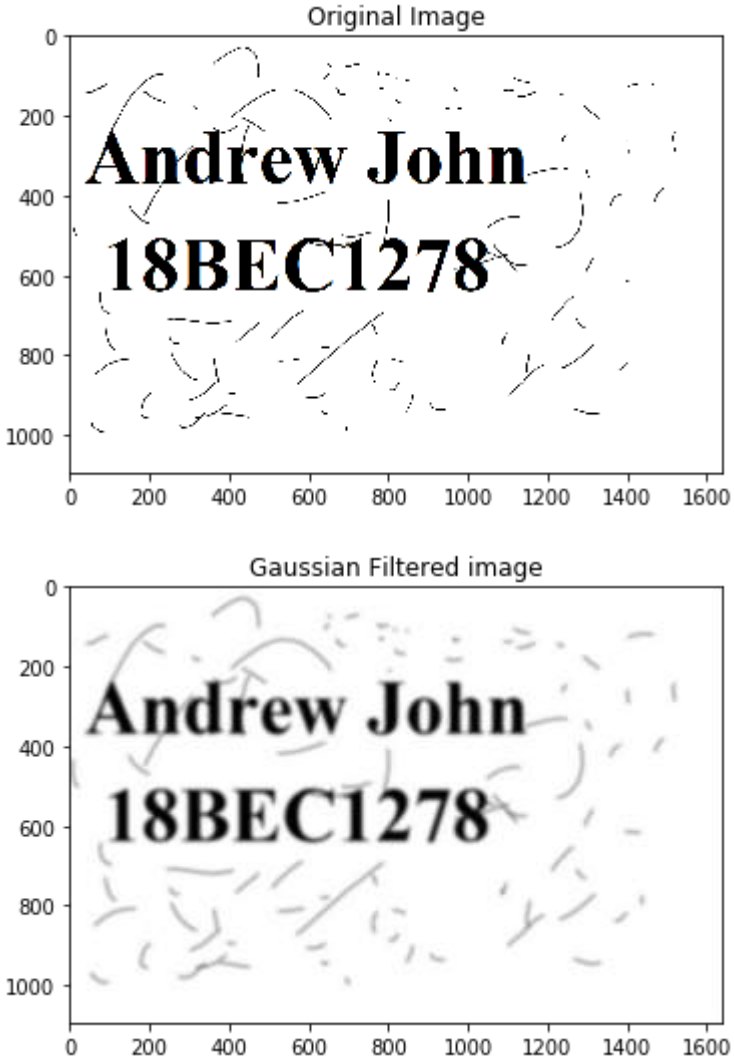
In [40]:

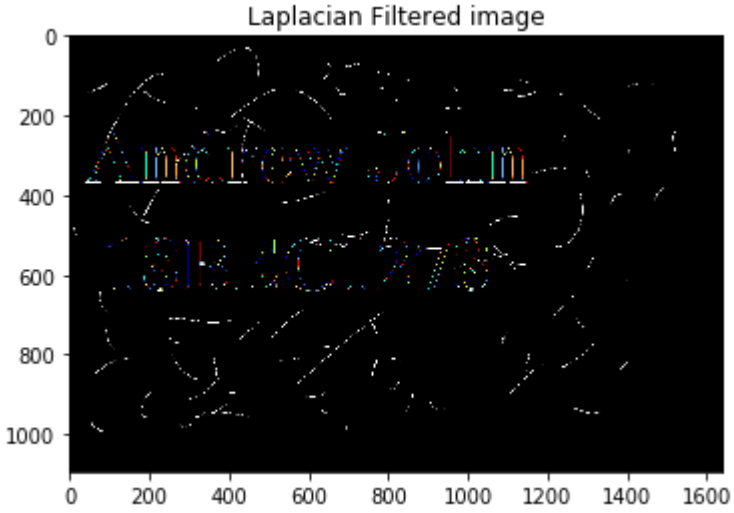
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from scipy import ndimage # Multi-dimensional image processing
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-2/Andrew_Reg_noise.png')
'''def rgb2gray(rgb):
    r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
    gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
    return gray
img=rgb2gray(img)'''
blurred_img=ndimage.gaussian_filter(img, sigma=5)  # Gaussian Filter
sharpened_img=ndimage.laplace(img)                   # Laplacian Filter
sharpened_img2=ndimage.laplace(blurred_img)          # Laplacian on Gaussian
sharpened_img3=img - blurred_img                     # Original minus Gaussian

plt.figure(1)
plt.imshow(img,cmap='gray')
plt.title("Original Image")
plt.figure(2)
plt.imshow(blurred_img,cmap='gray')
plt.title("Gaussian Filtered image")
plt.figure(3)
plt.imshow(sharpened_img,cmap='gray')
plt.title("Laplacian Filtered image")
plt.figure(4)
plt.imshow(sharpened_img2,cmap='gray')
plt.title("Laplacian on Gaussian")
plt.figure(5)
plt.imshow(sharpened_img3,cmap='gray')
plt.title("Original Minus Gaussian")
```

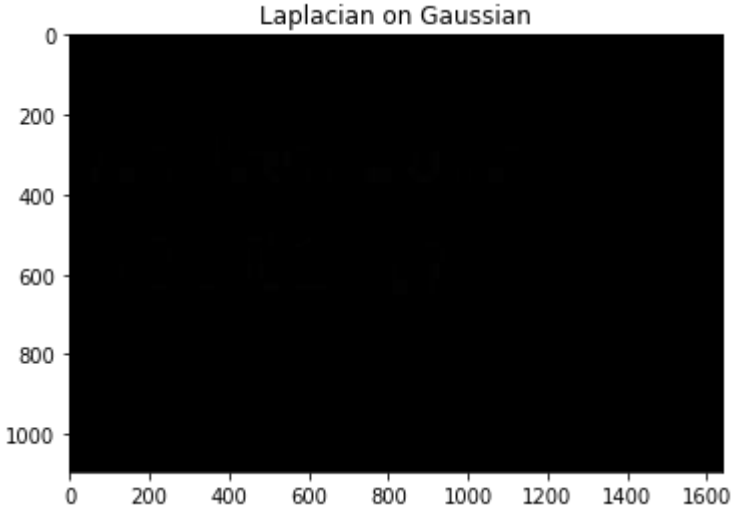
```
Clipping input data to the valid range for imshow with RGB data ([0..1] fo
r floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] fo
r floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] fo
r floats or [0..255] for integers).
```

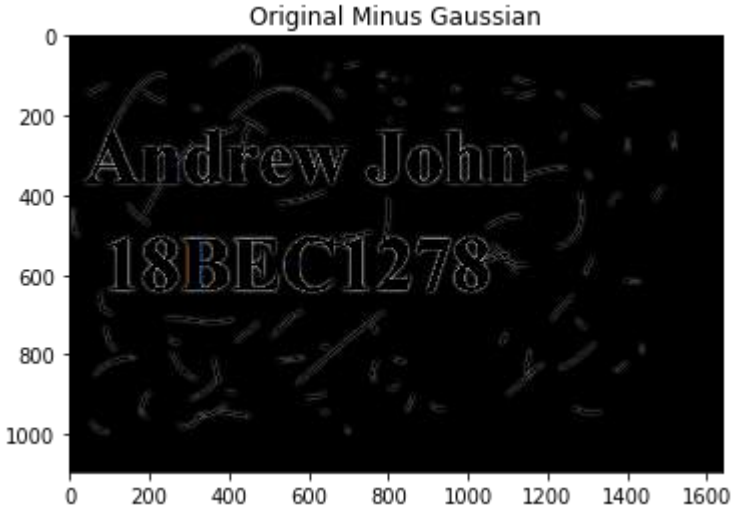
Out[40]:

```
Text(0.5, 1.0, 'Original Minus Gaussian')
```




Laplacian Filtered image
0
200
400
600
800
1000
0
200
400
600
800
1000
1200
1400
1600


Laplacian on Gaussian
0
200
400
600
800
1000
0
200
400
600
800
1000
1200
1400
1600



In [ ]:

```
```

In [57]:

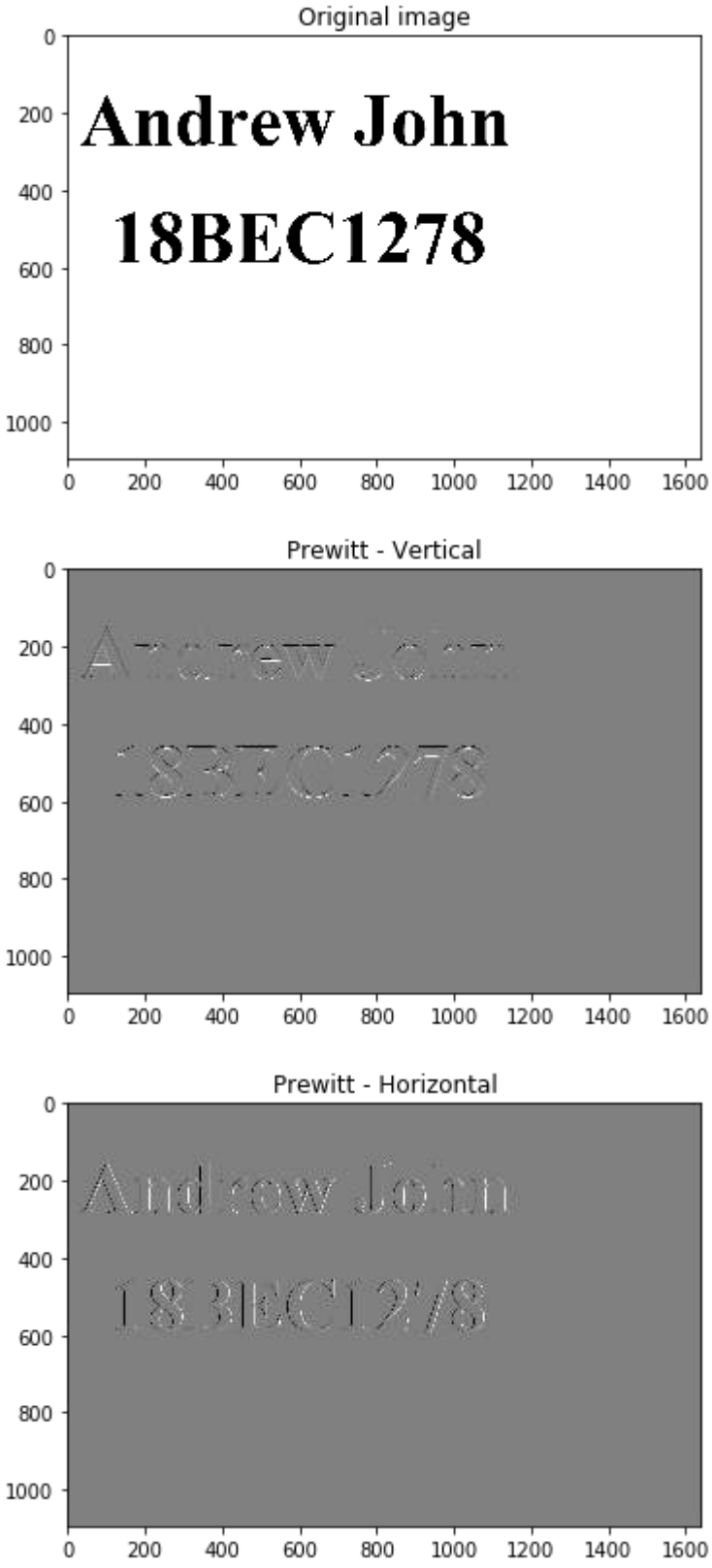
```
#SOBEL and PREWITT
#No External Noise
#Treating as Color Image
```

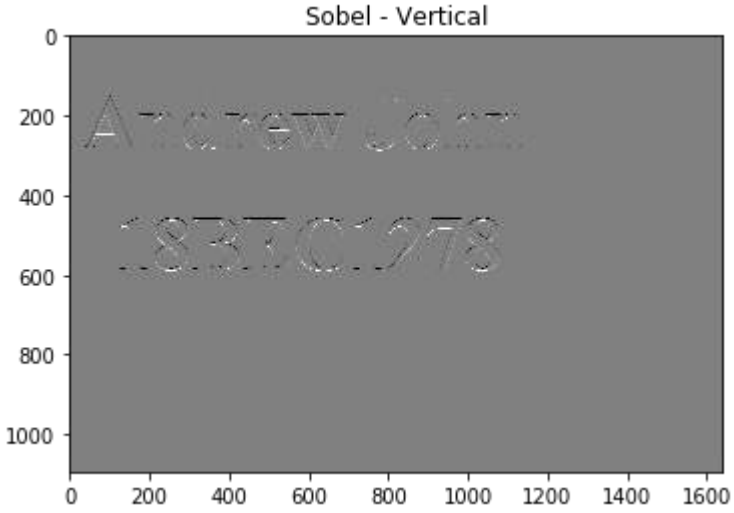
In [42]:

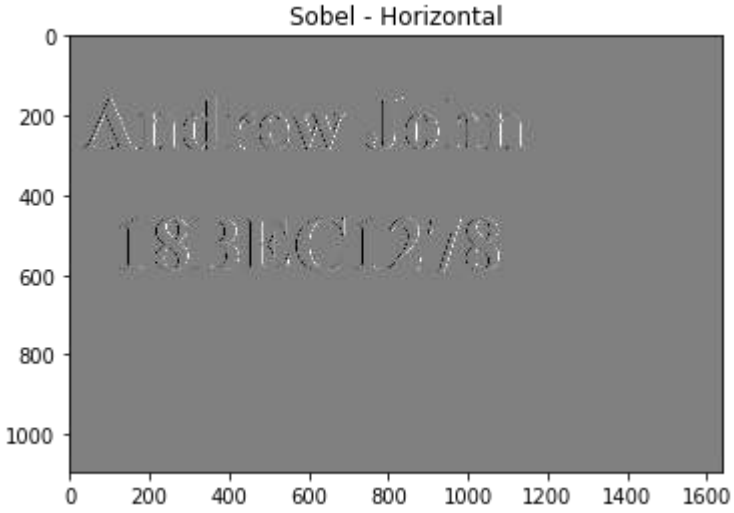
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from scipy import ndimage # Multi-dimensional image processing
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-2/Andrew_Reg.png')
def rgb2gray(rgb):
    r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
    gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
    return gray
img=rgb2gray(img)
sharpened_img1=ndimage.prewitt(img,axis=0)  # Prewitt - Vertical
sharpened_img2=ndimage.prewitt(img, axis=1) # Prewitt - Horizontal
sharpened_img3=ndimage.sobel(img,axis=0)    # Sobel - Vertical
sharpened_img4=ndimage.sobel(img,axis=1)    # Sobel - Horizontal
plt.figure(1)
plt.imshow(img,cmap='gray')
plt.title("Original image")
plt.figure(2)
plt.imshow(sharpened_img1,cmap='gray')
plt.title("Prewitt - Vertical")
plt.figure(3)
plt.imshow(sharpened_img2,cmap='gray')
plt.title("Prewitt - Horizontal")
plt.figure(4)
plt.imshow(sharpened_img3,cmap='gray')
plt.title("Sobel - Vertical")
plt.figure(5)
plt.imshow(sharpened_img4,cmap='gray')
plt.title("Sobel - Horizontal")
```

Out[42]:

```
Text(0.5, 1.0, 'Sobel - Horizontal')
```







In [ ]:

```
#SOBEL and PREWITT
#No External Noise
#Treating as B&W Image
```

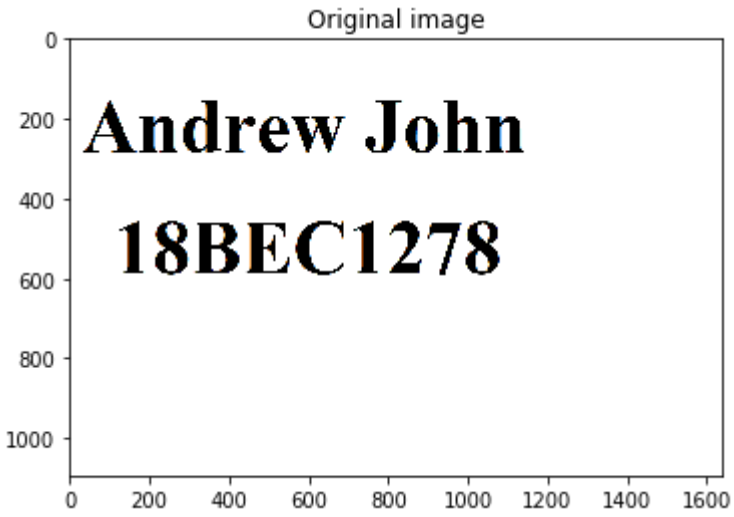
In [48]:

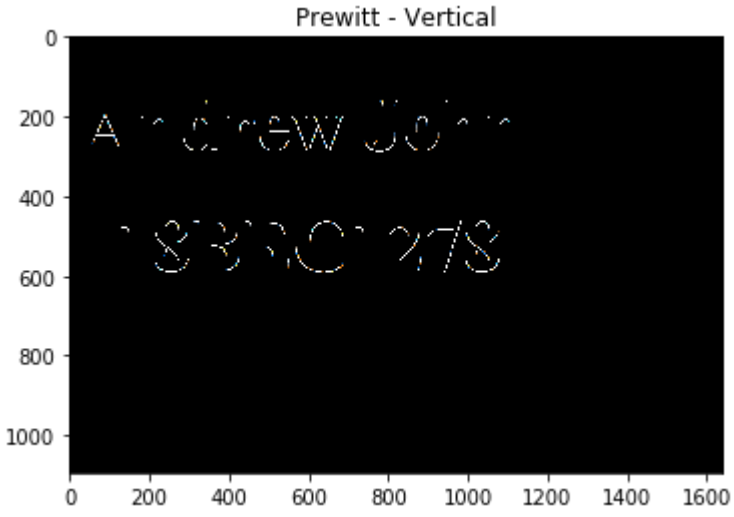
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from scipy import ndimage # Multi-dimensional image processing
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-2/Andrew_Reg.png')
'''def rgb2gray(rgb):
    r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
    gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
    return gray
img=rgb2gray(img)'''
sharpened_img1=ndimage.prewitt(img,axis=0)  # Prewitt - Vertical
sharpened_img2=ndimage.prewitt(img, axis=1) # Prewitt - Horizontal
sharpened_img3=ndimage.sobel(img,axis=0)    # Sobel - Vertical
sharpened_img4=ndimage.sobel(img,axis=1)    # Sobel - Horizontal
plt.figure(1)
plt.imshow(img,cmap='gray')
plt.title("Original image")
plt.figure(2)
plt.imshow(sharpened_img1,cmap='gray')
plt.title("Prewitt - Vertical")
plt.figure(3)
plt.imshow(sharpened_img2,cmap='gray')
plt.title("Prewitt - Horizontal")
plt.figure(4)
plt.imshow(sharpened_img3,cmap='gray')
plt.title("Sobel - Vertical")
plt.figure(5)
plt.imshow(sharpened_img4,cmap='gray')
plt.title("Sobel - Horizontal")
```

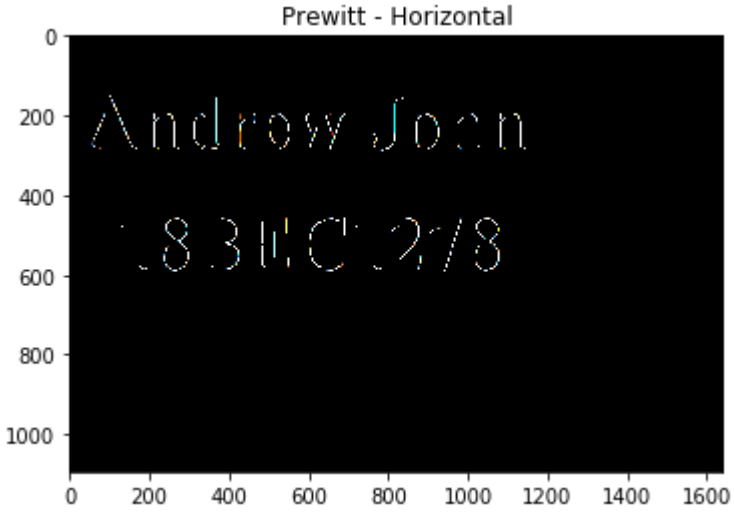
```
Clipping input data to the valid range for imshow with RGB data ([0..1] fo
r floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] fo
r floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] fo
r floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] fo
r floats or [0..255] for integers).
```

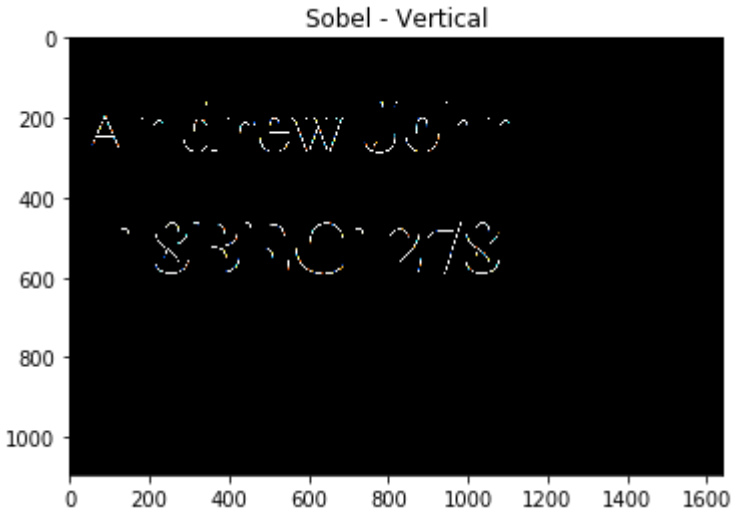
Out[48]:

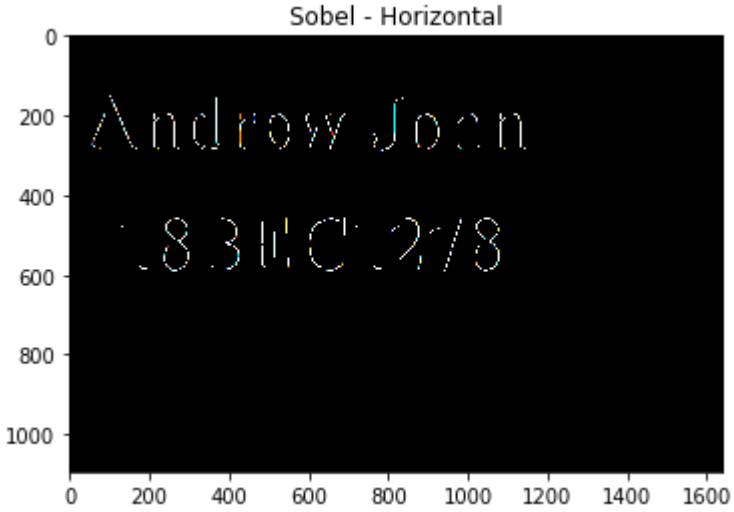
```
Text(0.5, 1.0, 'Sobel - Horizontal')
```











In [ ]:

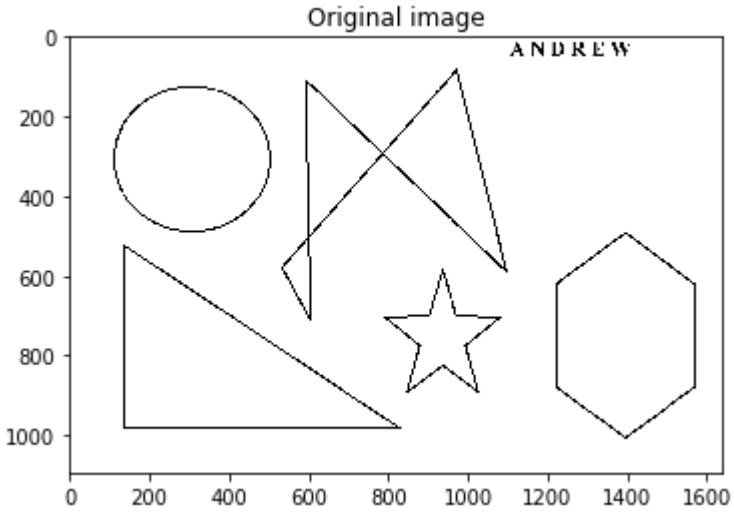
```
#SOBEL and PREWITT
#No External Noise
#Treating as Color Image
```

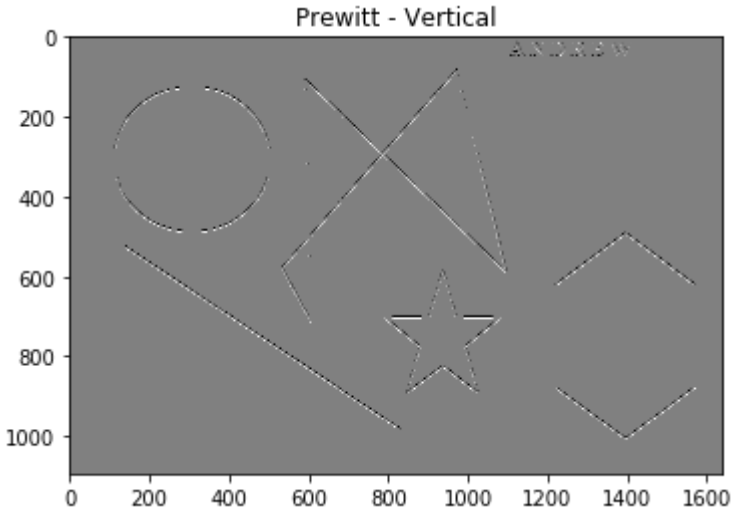
In [43]:

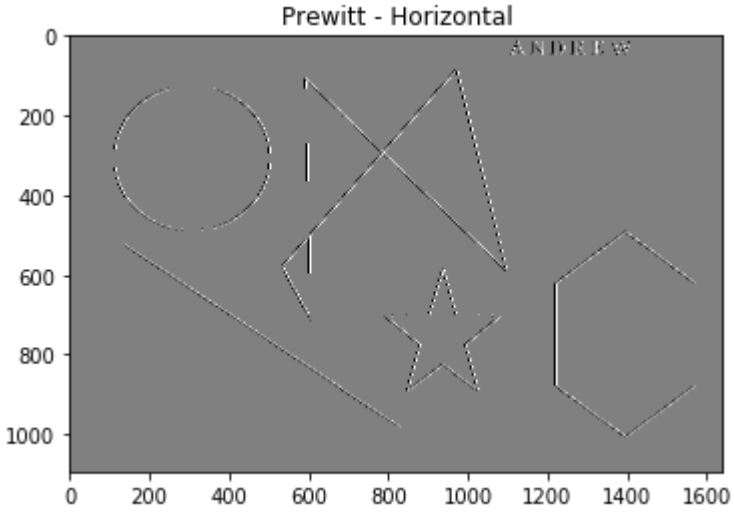
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from scipy import ndimage # Multi-dimensional image processing
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-2/Andrew_Shapes.png')
def rgb2gray(rgb):
    r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
    gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
    return gray
img=rgb2gray(img)
sharpened_img1=ndimage.prewitt(img,axis=0)  # Prewitt - Vertical
sharpened_img2=ndimage.prewitt(img, axis=1) # Prewitt - Horizontal
sharpened_img3=ndimage.sobel(img,axis=0)    # Sobel - Vertical
sharpened_img4=ndimage.sobel(img,axis=1)    # Sobel - Horizontal
plt.figure(1)
plt.imshow(img,cmap='gray')
plt.title("Original image")
plt.figure(2)
plt.imshow(sharpened_img1,cmap='gray')
plt.title("Prewitt - Vertical")
plt.figure(3)
plt.imshow(sharpened_img2,cmap='gray')
plt.title("Prewitt - Horizontal")
plt.figure(4)
plt.imshow(sharpened_img3,cmap='gray')
plt.title("Sobel - Vertical")
plt.figure(5)
plt.imshow(sharpened_img4,cmap='gray')
plt.title("Sobel - Horizontal")
```

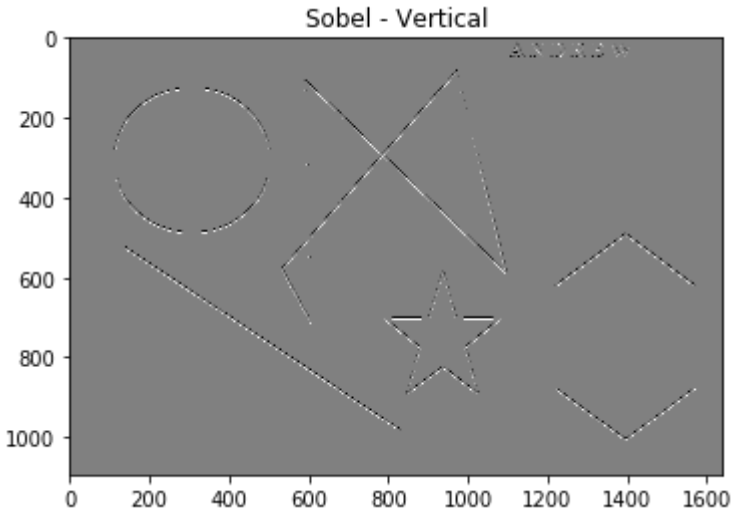
Out[43]:

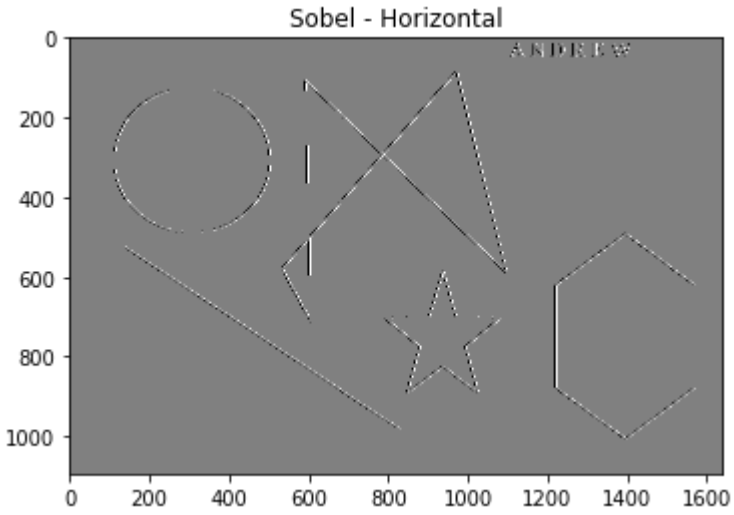
```
Text(0.5, 1.0, 'Sobel - Horizontal')
```






Prewitt - Horizontal
0
200
400
600
800
1000
0
200
400
600
800
1000
1200
1400
1600


Sobel - Vertical
0
200
400
600
800
1000
0
200
400
600
800
1000
1200
1400
1600



In [ ]:

```
#SOBEL and PREWITT
#No External Noise
#Treating as B&W Image
```

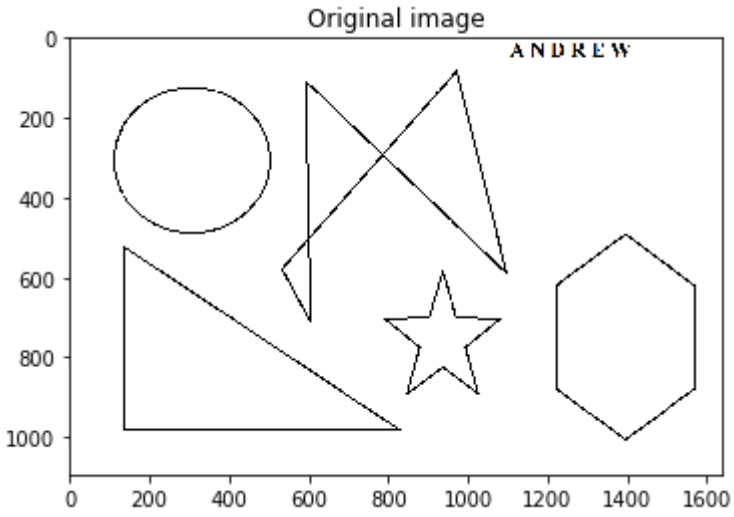
In [47]:

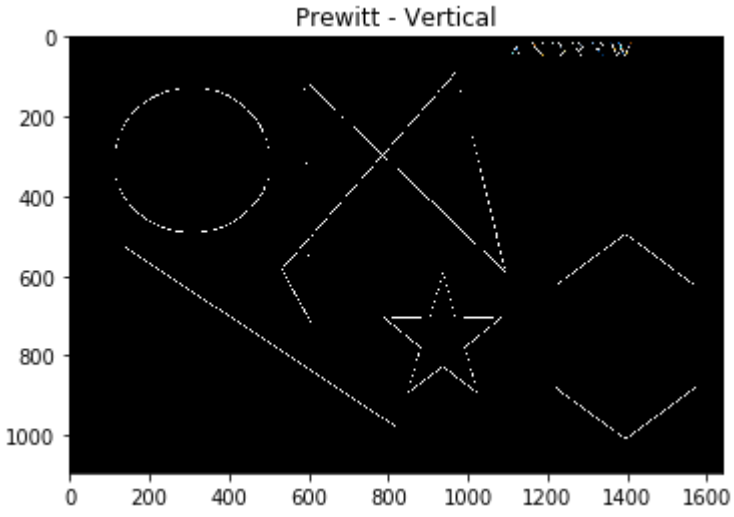
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from scipy import ndimage # Multi-dimensional image processing
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-2/Andrew_Shapes.png')
'''def rgb2gray(rgb):
    r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
    gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
    return gray
img=rgb2gray(img)'''
sharpened_img1=ndimage.prewitt(img,axis=0)  # Prewitt - Vertical
sharpened_img2=ndimage.prewitt(img, axis=1) # Prewitt - Horizontal
sharpened_img3=ndimage.sobel(img,axis=0)    # Sobel - Vertical
sharpened_img4=ndimage.sobel(img,axis=1)    # Sobel - Horizontal
plt.figure(1)
plt.imshow(img,cmap='gray')
plt.title("Original image")
plt.figure(2)
plt.imshow(sharpened_img1,cmap='gray')
plt.title("Prewitt - Vertical")
plt.figure(3)
plt.imshow(sharpened_img2,cmap='gray')
plt.title("Prewitt - Horizontal")
plt.figure(4)
plt.imshow(sharpened_img3,cmap='gray')
plt.title("Sobel - Vertical")
plt.figure(5)
plt.imshow(sharpened_img4,cmap='gray')
plt.title("Sobel - Horizontal")
```

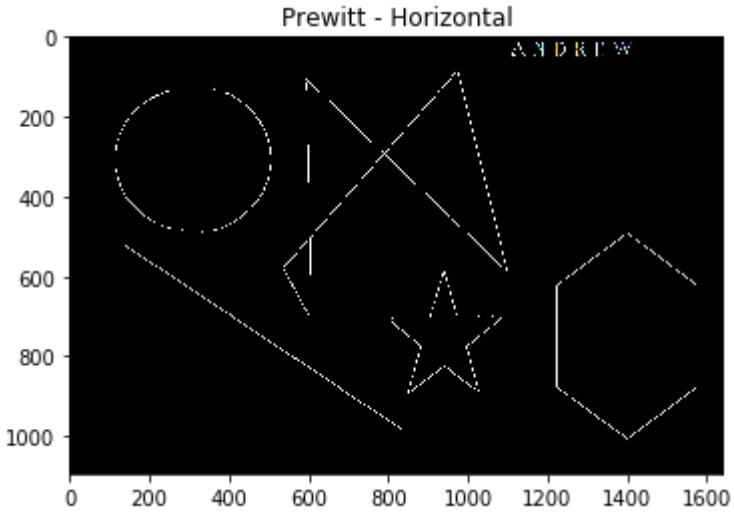
```
Clipping input data to the valid range for imshow with RGB data ([0..1] fo
r floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] fo
r floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] fo
r floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] fo
r floats or [0..255] for integers).
```

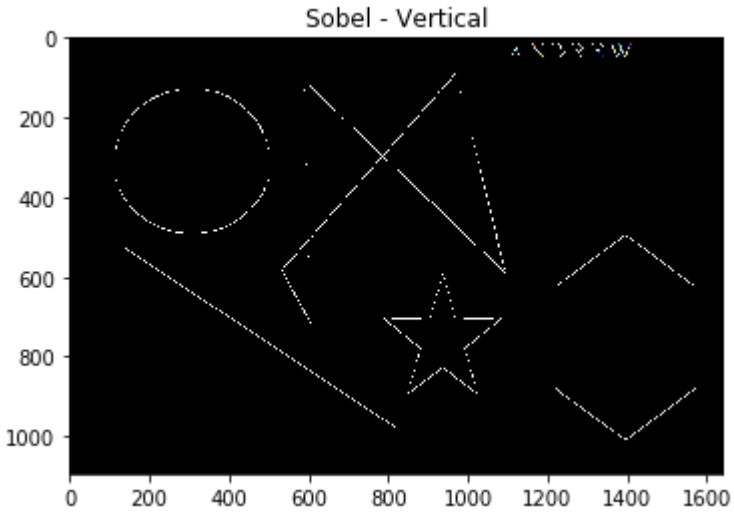
Out[47]:

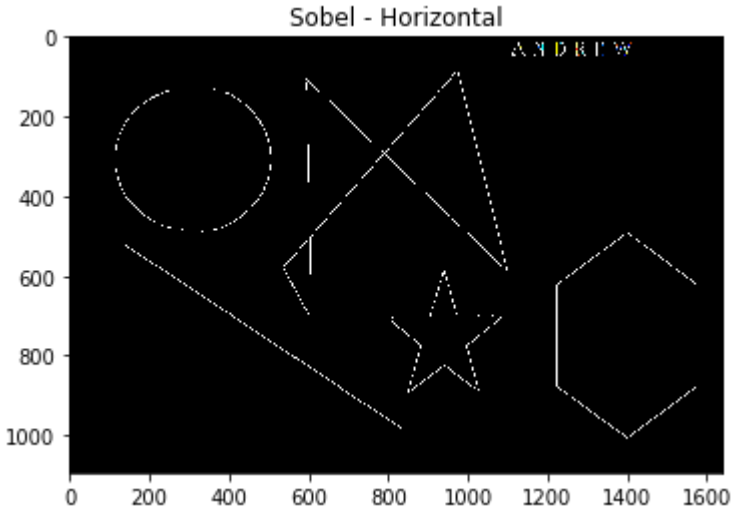
```
Text(0.5, 1.0, 'Sobel - Horizontal')
```











In [ ]:

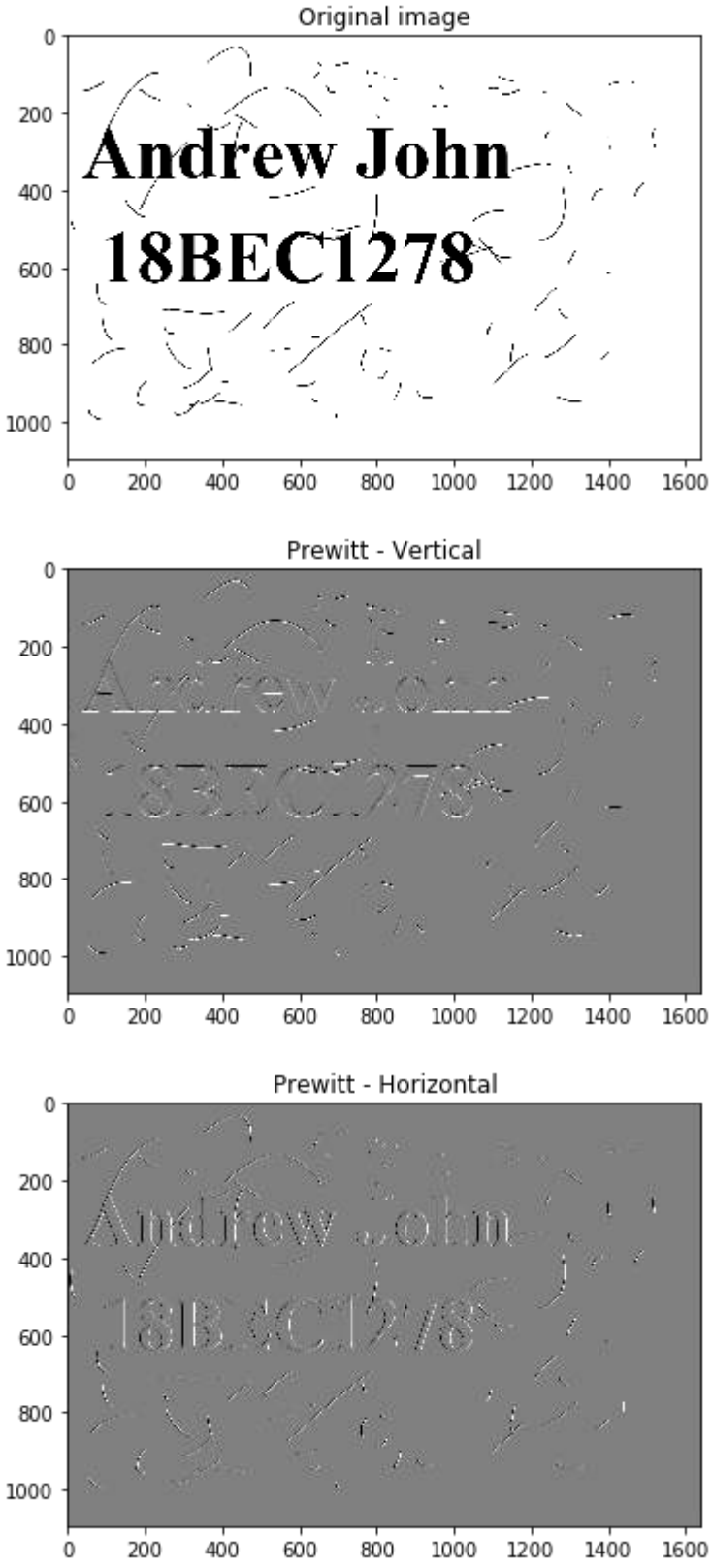
```
#SOBEL and PREWITT
#External Noise
#Treating as Color Image
```

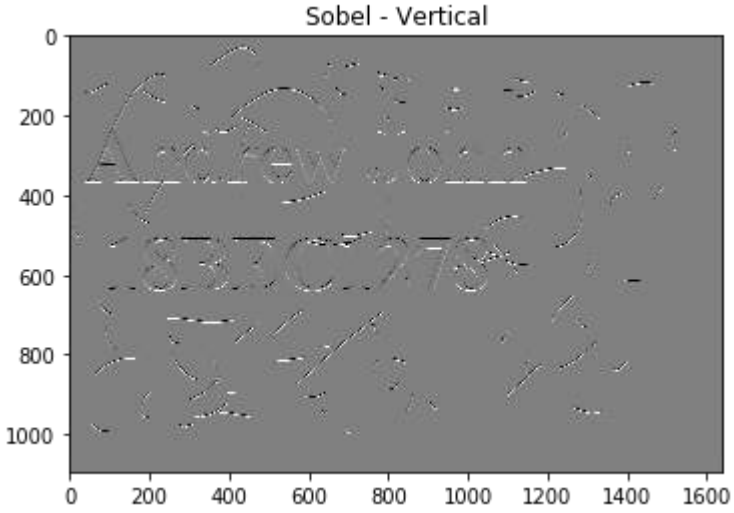
In [44]:

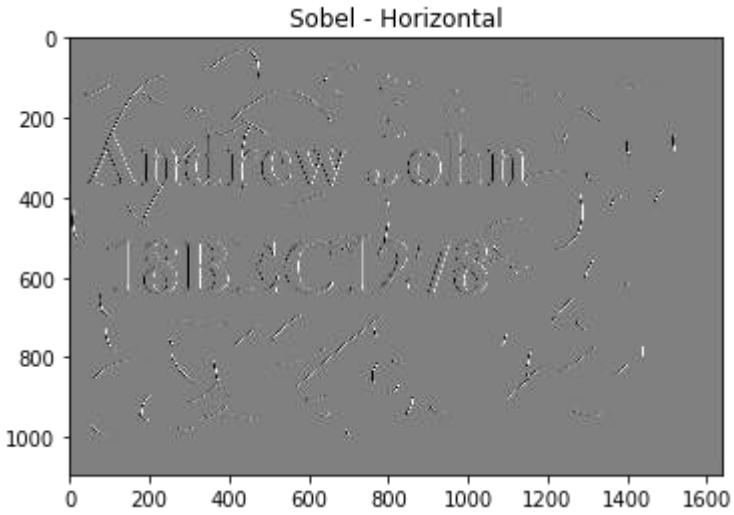
```
#SOBEL and PREWITT
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from scipy import ndimage # Multi-dimensional image processing
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-2/Andrew_Reg_noise.png')
def rgb2gray(rgb):
    r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
    gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
    return gray
img=rgb2gray(img)
sharpened_img1=ndimage.prewitt(img,axis=0)  # Prewitt - Vertical
sharpened_img2=ndimage.prewitt(img, axis=1) # Prewitt - Horizontal
sharpened_img3=ndimage.sobel(img,axis=0)    # Sobel - Vertical
sharpened_img4=ndimage.sobel(img,axis=1)    # Sobel - Horizontal
plt.figure(1)
plt.imshow(img,cmap='gray')
plt.title("Original image")
plt.figure(2)
plt.imshow(sharpened_img1,cmap='gray')
plt.title("Prewitt - Vertical")
plt.figure(3)
plt.imshow(sharpened_img2,cmap='gray')
plt.title("Prewitt - Horizontal")
plt.figure(4)
plt.imshow(sharpened_img3,cmap='gray')
plt.title("Sobel - Vertical")
plt.figure(5)
plt.imshow(sharpened_img4,cmap='gray')
plt.title("Sobel - Horizontal")
```

Out[44]:

```
Text(0.5, 1.0, 'Sobel - Horizontal')
```







In [ ]:

```
#SOBEL and PREWITT
#External Noise
#Treating as B&W Image
```

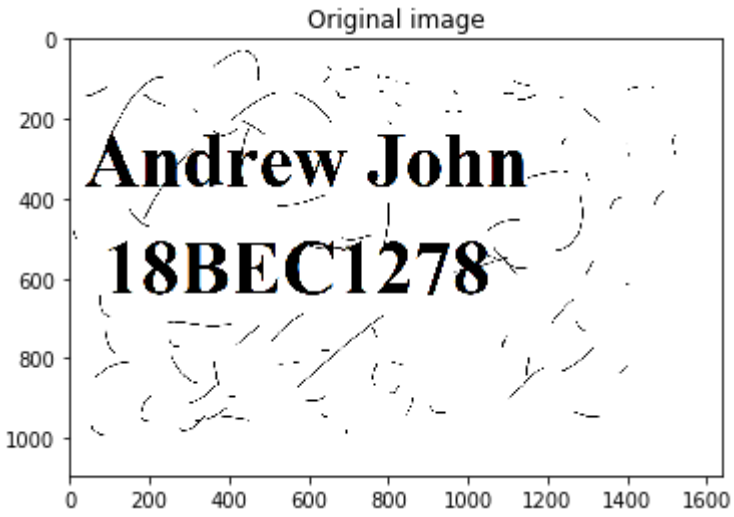
In [46]:

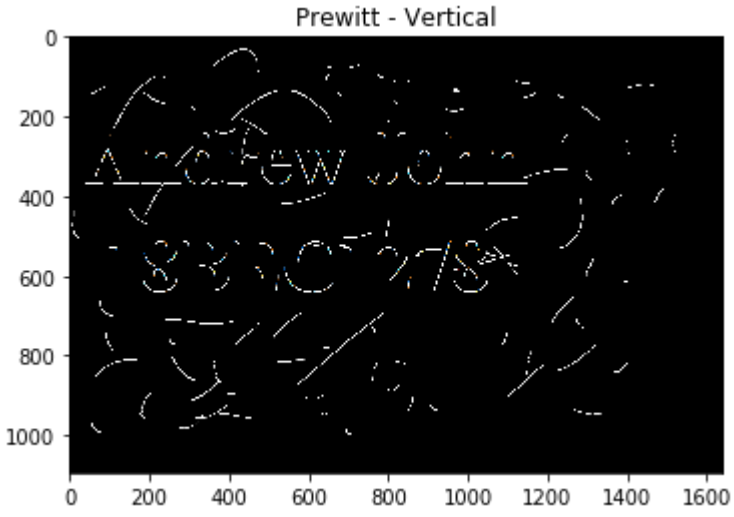
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from scipy import ndimage # Multi-dimensional image processing
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-2/Andrew_Reg_noise.png')
'''def rgb2gray(rgb):
    r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
    gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
    return gray
img=rgb2gray(img)'''
sharpened_img1=ndimage.prewitt(img,axis=0)  # Prewitt - Vertical
sharpened_img2=ndimage.prewitt(img, axis=1) # Prewitt - Horizontal
sharpened_img3=ndimage.sobel(img,axis=0)    # Sobel - Vertical
sharpened_img4=ndimage.sobel(img,axis=1)    # Sobel - Horizontal
plt.figure(1)
plt.imshow(img,cmap='gray')
plt.title("Original image")
plt.figure(2)
plt.imshow(sharpened_img1,cmap='gray')
plt.title("Prewitt - Vertical")
plt.figure(3)
plt.imshow(sharpened_img2,cmap='gray')
plt.title("Prewitt - Horizontal")
plt.figure(4)
plt.imshow(sharpened_img3,cmap='gray')
plt.title("Sobel - Vertical")
plt.figure(5)
plt.imshow(sharpened_img4,cmap='gray')
plt.title("Sobel - Horizontal")
```

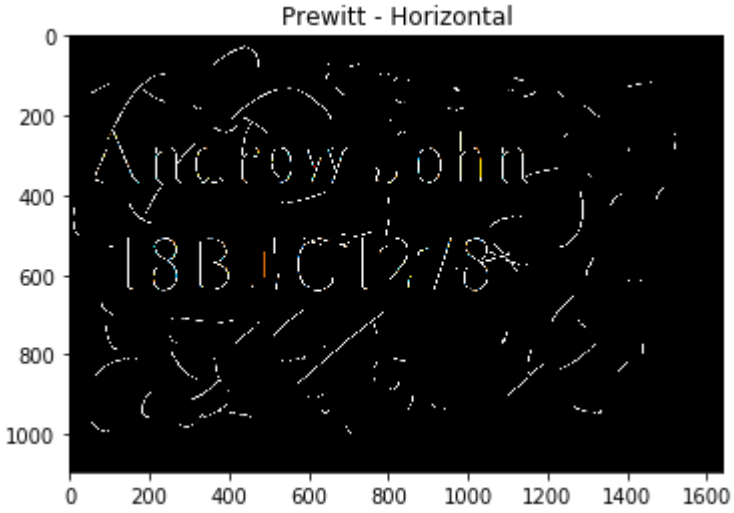
```
Clipping input data to the valid range for imshow with RGB data ([0..1] fo
r floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] fo
r floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] fo
r floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] fo
r floats or [0..255] for integers).
```

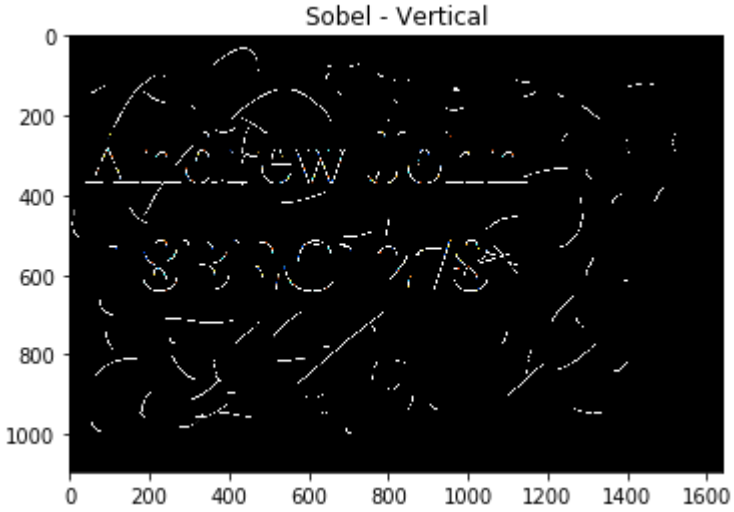
Out[46]:

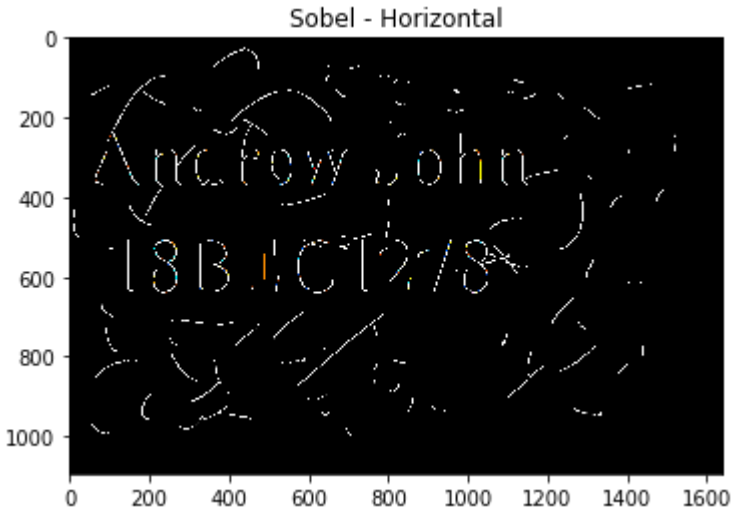
```
Text(0.5, 1.0, 'Sobel - Horizontal')
```






Prewitt - Horizontal
0
200
400
600
800
1000
0
200
400
600
800
1000
1200
1400
1600


Sobel - Vertical
0
200
400
600
800
1000
0
200
400
600
800
1000
1200
1400
1600


Sobel - Horizontal
0
200
400
600
800
1000
0
200
400
600
800
1000
1200
1400
1600

In [ ]:

```
'''
We have observed that with Prewitt and Sobel that, as the pixels
moves inone axis then it will bolden or sharpen the pixels that
are parallel to the axis movement.
'''
```

In [ ]:

```
#OpenCV
```

In [ ]:

```
#Brightness
```

In [107]:

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

In [110]:

```
img=cv2.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-2/LowContrast.png')
b=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
#b_bright=255*(b/255)+100
b_bright=b+100
c=np.full(np.shape(b),100)
#b_bright=255*(b/255)+c
plt.figure(1)
plt.imshow(b, cmap='gray')
plt.figure(2)
plt.hist(b)
plt.figure(3)
plt.imshow(b_bright,cmap='gray')
plt.figure(4)
plt.hist(b_bright)
```

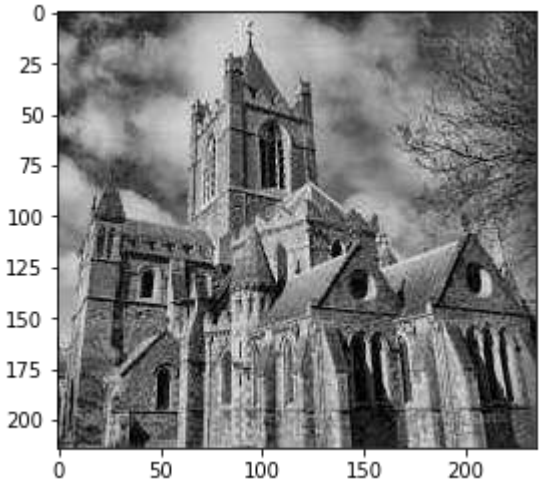
Out[110]:

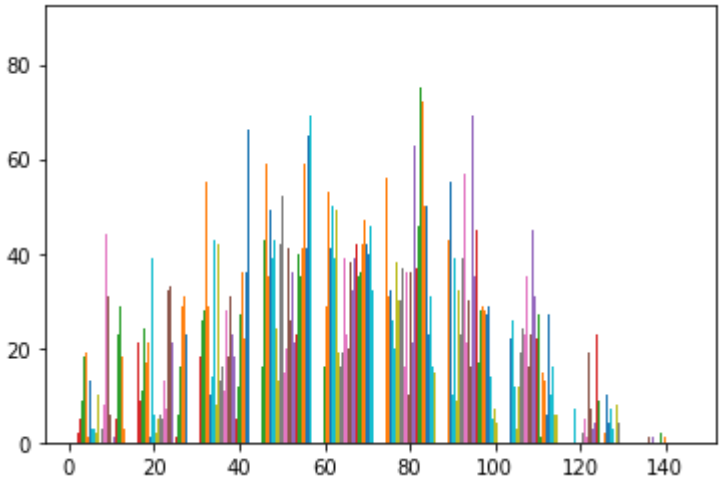
```
([array([12., 21., 24., 16., 22., 53., 44., 22.,  0.,  0.]),
  array([ 1., 19., 17., 19., 23., 56., 43., 34.,  1.,  1.]),
  array([ 1., 19., 13., 16., 16., 54., 65., 28.,  2.,  0.]),
  array([ 3., 21., 18., 19., 10., 56., 53., 34.,  0.,  0.]),
  array([ 0., 16., 15., 23., 22., 45., 55., 33.,  4.,  1.]),
  array([ 0., 10., 19., 16., 30., 42., 61., 34.,  2.,  0.]),
  array([ 4., 12., 18., 16., 34., 47., 44., 37.,  2.,  0.]),
  array([13.,  9., 17., 24., 26., 45., 48., 29.,  3.,  0.]),
  array([ 1.,  2., 13., 42., 35., 47., 43., 27.,  4.,  0.]),
  array([ 0.,  7., 18., 29., 44., 48., 35., 26.,  7.,  0.]),
  array([ 0.,  8., 20., 37., 36., 34., 55., 22.,  2.,  0.]),
  array([ 0.,  7., 29., 43., 29., 31., 42., 27.,  5.,  1.]),
  array([ 2., 15., 26., 43., 32., 28., 37., 19., 12.,  0.]),
  array([ 2.,  9., 30., 42., 27., 25., 52., 16., 11.,  0.]),
  array([ 1.,  5., 22., 47., 41., 25., 57., 14.,  2.,  0.]),
  array([ 3.,  9., 27., 42., 47., 27., 36., 20.,  3.,  0.]),
  array([ 4., 10., 24., 55., 39., 26., 37., 19.,  0.,  0.]),
  array([ 3., 10., 29., 54., 45., 31., 22., 20.,  0.,  0.]),
  array([ 4., 11., 29., 56., 60., 26., 14., 14.,  0.,  0.]),
  array([ 1., 11., 22., 74., 54., 30., 10., 12.,  0.,  0.]),
  array([ 5., 12., 21., 70., 47., 32., 13., 14.,  0.,  0.]),
  array([ 3., 17., 23., 59., 53., 32., 15., 12.,  0.,  0.]),
  array([ 6., 11., 28., 60., 48., 33., 14., 14.,  0.,  0.]),
  array([ 5.,  7., 22., 59., 53., 37., 13., 18.,  0.,  0.]),
  array([ 4.,  6., 14., 55., 60., 39., 19., 17.,  0.,  0.]),
  array([ 3., 11., 27., 36., 46., 46., 23., 22.,  0.,  0.]),
  array([ 3., 15., 28., 43., 47., 41., 26., 11.,  0.,  0.]),
  array([ 4., 17., 31., 39., 50., 42., 25.,  6.,  0.,  0.]),
  array([13., 14., 35., 31., 45., 36., 37.,  3.,  0.,  0.]),
  array([10., 30., 32., 43., 29., 26., 39.,  5.,  0.,  0.]),
  array([20., 44., 29., 27., 41., 30., 23.,  0.,  0.,  0.]),
  array([19., 31., 55., 35., 24., 26., 24.,  0.,  0.,  0.]),
  array([ 9., 24., 35., 50., 39., 44.,  6.,  7.,  0.,  0.]),
  array([11., 17., 47., 41., 53., 26., 12.,  7.,  0.,  0.]),
  array([ 3., 24., 37., 49., 51., 32.,  6., 10.,  2.,  0.]),
  array([11., 22., 34., 38., 65., 25.,  9., 10.,  0.,  0.]),
  array([13., 16., 30., 43., 67., 26.,  9., 10.,  0.,  0.]),
  array([15., 13., 35., 53., 55., 25.,  7., 11.,  0.,  0.]),
  array([12., 13., 37., 61., 50., 20.,  9., 12.,  0.,  0.]),
  array([12., 16., 37., 45., 50., 20., 20., 14.,  0.,  0.]),
  array([10., 17., 21., 49., 59., 32., 16.,  8.,  2.,  0.]),
  array([13., 17., 29., 47., 52., 29., 14., 13.,  0.,  0.]),
  array([18., 13., 29., 38., 48., 37., 17., 14.,  0.,  0.]),
  array([16., 16., 20., 49., 39., 44., 15., 15.,  0.,  0.]),
  array([15., 15., 22., 52., 35., 38., 23., 14.,  0.,  0.]),
  array([16., 14., 19., 52., 38., 38., 25., 11.,  1.,  0.]),
  array([14.,  9., 19., 45., 52., 29., 37.,  9.,  0.,  0.]),
  array([ 5.,  9., 18., 45., 58., 29., 29., 19.,  2.,  0.]),
  array([ 6., 14., 20., 36., 46., 38., 32., 21.,  1.,  0.]),
  array([16., 16., 15., 39., 39., 38., 37., 14.,  0.,  0.]),
  array([25., 12., 10., 40., 46., 38., 31., 12.,  0.,  0.]),
  array([19., 21., 18., 40., 46., 27., 28., 15.,  0.,  0.]),
  array([18., 13., 19., 41., 47., 36., 22., 18.,  0.,  0.]),
  array([11., 19., 22., 39., 40., 43., 18., 22.,  0.,  0.]),
  array([ 9., 16., 22., 41., 32., 36., 23., 31.,  4.,  0.]),
  array([ 4.,  4.,  8., 42., 38., 33., 32., 47.,  6.,  0.]),
  array([ 2.,  1., 17., 41., 44., 46., 26., 32.,  5.,  0.]),
  array([ 2.,  3., 16., 45., 53., 46., 23., 24.,  2.,  0.]),
  array([ 2.,  4., 24., 43., 49., 30., 32., 27.,  3.,  0.]),
```

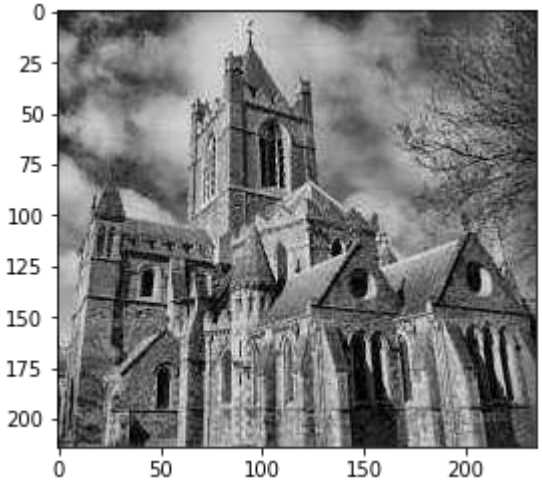
```
array([ 5.,  3., 14., 43., 46., 37., 33., 28.,  5.,  0.]),
array([ 1.,  1., 13., 33., 46., 37., 33., 40., 10.,  0.]),
array([ 1.,  1., 13., 22., 42., 32., 42., 47., 14.,  0.]),
array([ 4.,  2., 11., 21., 28., 35., 43., 49., 21.,  0.]),
array([11., 11.,  6., 19., 33., 35., 49., 37., 12.,  1.]),
array([19., 14., 16., 29., 24., 32., 43., 27., 10.,  0.]),
array([ 4., 18., 23., 27., 22., 22., 60., 32.,  5.,  1.]),
array([ 4., 23., 27., 22., 24., 36., 54., 23.,  1.,  0.]),
array([ 4., 26., 48., 19., 18., 30., 39., 29.,  1.,  0.]),
array([ 8., 31., 40., 24., 19., 19., 30., 38.,  5.,  0.]),
array([ 5., 39., 43., 17., 14., 16., 39., 40.,  0.,  1.]),
array([13., 38., 41., 12., 13.,  9., 39., 47.,  2.,  0.]),
array([16., 31., 38., 21., 15.,  8., 19., 54., 12.,  0.]),
array([ 7., 11., 19., 21., 27., 36., 42., 46.,  4.,  1.]),
array([ 2.,  8.,  4., 16., 22., 22., 40., 88., 10.,  2.]),
array([ 5.,  5.,  1., 33., 27., 36., 39., 58., 10.,  0.]),
array([ 6.,  4., 10., 23., 35., 22., 42., 53., 19.,  0.]),
array([11.,  5., 13., 20., 17., 28., 57., 35., 28.,  0.]),
array([15., 13.,  2., 19., 16., 37., 48., 34., 30.,  0.]),
array([11.,  3.,  8., 13., 10., 35., 44., 41., 49.,  0.]),
array([ 3.,  6.,  9.,  7., 13., 25., 70., 51., 30.,  0.]),
array([ 3.,  8.,  7., 13., 18., 32., 55., 53., 25.,  0.]),
array([ 5.,  8.,  3., 18., 19., 39., 55., 48., 15.,  4.]),
array([ 6.,  8.,  8., 19., 33., 23., 55., 50., 12.,  0.]),
array([ 5., 14., 16., 23., 21., 38., 49., 38., 10.,  0.]),
array([ 5.,  9., 21., 26., 20., 35., 53., 29., 15.,  1.]),
array([ 2.,  4., 22., 72., 48., 26., 16., 16.,  7.,  1.]),
array([ 0., 10., 28., 46., 15., 16., 21., 50., 27.,  1.]),
array([ 1.,  8., 33., 42., 19., 10., 12., 55., 33.,  1.]),
array([ 3.,  2., 42., 38., 16., 25., 30., 37., 21.,  0.]),
array([ 3.,  9., 34., 53., 22., 39., 15., 20., 17.,  2.]),
array([ 0.,  8., 39., 28., 23., 14., 31., 46., 23.,  2.]),
array([12., 29.,  7.,  8., 31., 31., 21., 48., 27.,  0.]),
array([ 3., 38., 22., 11., 21., 31., 50., 24., 12.,  2.]),
array([ 4.,  9., 25., 49., 32., 41., 26., 23.,  5.,  0.]),
array([ 4., 13., 18., 65., 45., 33., 25.,  8.,  3.,  0.]),
array([ 6.,  9., 13., 56., 32., 36., 30., 23.,  8.,  1.]),
array([ 3.,  9., 26., 57., 39., 36., 13., 24.,  7.,  0.]),
array([ 2.,  3., 13., 52., 54., 31., 24., 32.,  3.,  0.]),
array([ 2.,  5., 23., 34., 49., 31., 29., 36.,  5.,  0.]),
array([ 8.,  9., 26., 50., 39., 39., 14., 24.,  5.,  0.]),
array([31., 10., 18., 31., 36., 16., 27., 41.,  4.,  0.]),
array([25., 16., 21., 38., 27., 32., 20., 32.,  3.,  0.]),
array([ 6., 23., 42., 43., 28., 18., 13., 36.,  3.,  2.]),
array([13., 32., 31., 51., 17., 14., 18., 36.,  2.,  0.]),
array([29., 25., 36., 33., 20., 13.,  9., 45.,  4.,  0.]),
array([33., 30., 29., 23., 26., 10., 16., 43.,  4.,  0.]),
array([21., 45., 30., 15., 23., 19., 13., 46.,  2.,  0.]),
array([ 6.,  6., 16., 28., 46., 38., 14., 56.,  3.,  1.]),
array([10., 11., 20., 28., 26., 34., 43., 40.,  2.,  0.]),
array([23., 11., 17., 21., 27., 33., 35., 46.,  1.,  0.]),
array([12., 10., 12., 22., 18., 31., 43., 49., 17.,  0.]),
array([ 2.,  6., 12., 23., 29., 39., 53., 46.,  4.,  0.]),
array([ 0.,  3.,  6., 18., 25., 39., 73., 48.,  2.,  0.]),
array([ 0.,  2.,  3., 17., 31., 23., 56., 59., 23.,  0.]),
array([ 1.,  1., 12., 26., 21., 41., 69., 31., 12.,  0.]),
array([ 1.,  4., 18., 24., 20., 36., 78., 29.,  4.,  0.]),
array([ 2.,  2., 11., 20., 16., 40., 78., 36.,  9.,  0.]),
array([ 0.,  5., 16., 22., 25., 46., 73., 23.,  4.,  0.]),
array([ 2., 10., 21., 16., 16., 50., 72., 23.,  4.,  0.]),
array([ 0.,  5., 13., 16., 28., 41., 71., 34.,  6.,  0.]),
```

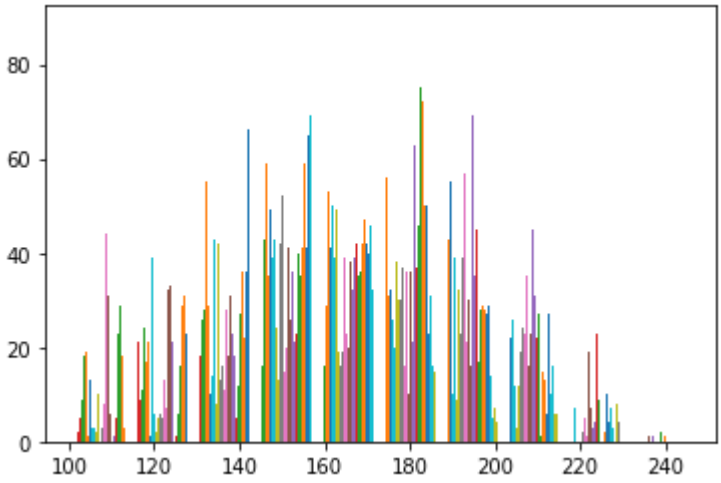
```
array([ 1.,  5., 21., 26., 17., 46., 67., 26.,  5.,  0.]),
array([ 1.,  6., 20., 31., 35., 33., 53., 32.,  3.,  0.]),
array([ 0.,  7., 25., 42., 36., 26., 40., 27.,  9.,  2.]),
array([ 1.,  8., 46., 49., 20., 25., 35., 22.,  7.,  1.]),
array([ 6., 13., 44., 46., 34., 21., 35., 15.,  0.,  0.]),
array([ 8., 14., 34., 41., 38., 32., 32., 15.,  0.,  0.]),
array([ 1., 13., 28., 32., 25., 34., 60., 20.,  1.,  0.]),
array([ 3.,  6., 12., 26., 34., 49., 56., 27.,  1.,  0.]),
array([ 4.,  5., 18., 18., 32., 70., 48., 18.,  1.,  0.]),
array([ 5.,  8., 10., 19., 39., 68., 43., 22.,  0.,  0.]),
array([ 4.,  8., 13., 25., 28., 66., 47., 23.,  0.,  0.]),
array([ 3.,  8., 10., 25., 32., 67., 38., 27.,  4.,  0.]),
array([ 4.,  8., 21., 22., 32., 56., 44., 27.,  0.,  0.]),
array([ 3., 11., 15., 25., 24., 67., 45., 22.,  2.,  0.]),
array([ 3.,  7., 10., 34., 32., 63., 38., 27.,  0.,  0.]),
array([ 5.,  7., 18., 26., 36., 56., 33., 27.,  6.,  0.]),
array([ 8.,  7., 25., 31., 32., 47., 33., 27.,  4.,  0.]),
array([ 7., 10., 29., 38., 26., 43., 29., 23.,  9.,  0.]),
array([11., 14., 24., 36., 37., 45., 32., 11.,  4.,  0.]),
array([ 9., 13., 23., 48., 47., 48., 23.,  3.,  0.,  0.]),
array([ 6., 16., 24., 51., 60., 33., 18.,  5.,  1.,  0.]),
array([11., 11., 29., 54., 47., 38., 20.,  3.,  0.,  1.]),
array([10., 15., 19., 52., 57., 43., 17.,  1.,  0.,  0.]),
array([16., 16., 29., 44., 51., 37., 21.,  0.,  0.,  0.]),
array([16., 42., 23., 36., 39., 41., 16.,  1.,  0.,  0.]),
array([29., 32., 31., 33., 37., 39., 13.,  0.,  0.,  0.]),
array([44., 34., 16., 21., 43., 43., 11.,  2.,  0.,  0.]),
array([50., 30., 18., 30., 33., 38., 12.,  2.,  1.,  0.]),
array([55., 26., 20., 29., 28., 38., 14.,  3.,  1.,  0.]),
array([45., 31., 17., 31., 25., 41., 19.,  5.,  0.,  0.]),
array([42., 25., 16., 32., 29., 40., 24.,  5.,  1.,  0.]),
array([18., 33., 21., 33., 29., 40., 23., 15.,  2.,  0.]),
array([ 8., 35., 13., 30., 33., 46., 28., 18.,  3.,  0.]),
array([12., 29., 12., 27., 42., 48., 29., 15.,  0.,  0.]),
array([19., 27., 23., 21., 46., 42., 24., 10.,  2.,  0.]),
array([31., 33., 14., 19., 43., 41., 25.,  6.,  2.,  0.]),
array([30., 29., 15., 17., 49., 44., 26.,  3.,  1.,  0.]),
array([36., 22., 13., 18., 37., 52., 30.,  4.,  2.,  0.]),
array([27., 22., 18., 20., 30., 59., 26., 10.,  2.,  0.]),
array([18., 16.,  8., 22., 30., 76., 27., 13.,  4.,  0.]),
array([15., 15., 14., 14., 35., 75., 21., 15., 10.,  0.]),
array([11., 14., 16., 15., 34., 77., 29., 13.,  4.,  1.]),
array([12., 19., 13., 18., 35., 75., 29., 10.,  3.,  0.]),
array([11., 18., 17., 23., 28., 73., 33., 11.,  0.,  0.]),
array([ 9., 21., 18., 21., 29., 71., 32., 12.,  1.,  0.]),
array([ 6., 22., 25., 19., 25., 66., 38., 12.,  1.,  0.]),
array([ 1., 22., 22., 19., 23., 65., 50., 11.,  1.,  0.]),
array([ 5.,  7., 23., 25., 31., 59., 52.,  9.,  1.,  2.]),
array([ 8., 11., 16., 26., 29., 72., 40., 11.,  1.,  0.]),
array([14., 10., 19., 16., 28., 80., 38.,  7.,  2.,  0.]),
array([ 8.,  7., 21., 35., 37., 65., 31.,  6.,  4.,  0.]),
array([ 0.,  5., 14., 41., 39., 72., 28., 13.,  1.,  1.]),
array([ 0.,  1.,  7., 40., 36., 64., 45., 16.,  4.,  1.]),
array([ 0.,  1.,  5., 26., 53., 44., 61., 16.,  8.,  0.]),
array([ 0.,  0.,  4., 22., 46., 55., 65., 16.,  6.,  0.]),
array([ 1.,  3.,  4., 31., 46., 55., 56., 14.,  4.,  0.]),
array([ 0.,  0.,  3., 43., 36., 51., 60., 20.,  1.,  0.]),
array([ 0.,  0., 16., 36., 29., 65., 46., 16.,  6.,  0.]),
array([ 0.,  0., 14., 31., 38., 45., 54., 28.,  4.,  0.]),
array([ 0.,  1., 11., 38., 38., 44., 53., 22.,  7.,  0.]),
array([ 0.,  1., 13., 37., 49., 57., 27., 27.,  3.,  0.]),
```

```
 array([ 0.,  1.,  9., 41., 42., 50., 42., 26.,  3.,  0.]),
 array([ 0.,  0., 12., 35., 47., 51., 41., 23.,  5.,  0.]),
 array([ 0.,  1., 21., 34., 49., 49., 41., 17.,  2.,  0.]),
 array([ 1.,  6., 28., 41., 41., 45., 34., 16.,  2.,  0.]),
 array([ 1.,  5., 30., 49., 44., 51., 21., 10.,  3.,  0.]),
 array([ 3.,  0., 28., 47., 45., 44., 32., 10.,  5.,  0.]),
 array([ 5.,  0., 32., 37., 48., 40., 36., 15.,  1.,  0.]),
 array([ 3.,  4., 35., 37., 41., 51., 30., 10.,  3.,  0.]),
 array([ 5.,  6., 33., 33., 39., 57., 28., 10.,  3.,  0.]),
 array([ 5.,  6., 20., 46., 34., 50., 29., 20.,  4.,  0.]),
 array([ 4.,  2., 24., 41., 47., 47., 32., 13.,  4.,  0.]),
 array([ 5.,  6., 27., 48., 28., 54., 32., 11.,  3.,  0.]),
 array([ 5.,  4., 26., 46., 39., 42., 40., 12.,  0.,  0.]),
 array([ 4.,  7., 27., 44., 43., 34., 37., 17.,  1.,  0.]),
 array([ 5.,  2., 31., 45., 45., 39., 31., 13.,  3.,  0.]),
 array([ 4.,  7., 29., 60., 41., 37., 22., 13.,  1.,  0.]),
 array([ 3., 16., 38., 48., 37., 38., 20., 10.,  4.,  0.]),
 array([ 3., 16., 31., 54., 41., 27., 31., 11.,  0.,  0.]),
 array([ 6., 18., 35., 55., 39., 28., 14., 16.,  3.,  0.]),
 array([ 6., 11., 51., 40., 42., 23., 26.,  9.,  6.,  0.]),
 array([ 5., 22., 36., 59., 31., 32., 16., 11.,  2.,  0.]),
 array([23., 16., 25., 58., 39., 28., 14., 11.,  0.,  0.]),
 array([21., 16., 23., 72., 39., 24.,  9., 10.,  0.,  0.]),
 array([22., 15., 26., 51., 54., 32., 13.,  1.,  0.,  0.]),
 array([21., 18., 18., 66., 53., 32.,  6.,  0.,  0.,  0.]),
 array([23., 16., 25., 76., 42., 23.,  5.,  3.,  1.,  0.]),
 array([11., 19., 34., 53., 55., 22., 13.,  5.,  2.,  0.]),
 array([16., 11., 40., 47., 46., 29., 11.,  6.,  8.,  0.]),
 array([22., 23., 23., 57., 44., 31.,  5.,  9.,  0.,  0.]),
 array([30., 25., 32., 41., 40., 27., 10.,  5.,  4.,  0.]),
 array([31., 29., 22., 54., 34., 26., 10.,  3.,  5.,  0.]),
 array([29., 28., 22., 48., 39., 32., 15.,  1.,  0.,  0.]),
 array([21., 21., 21., 51., 60., 27., 12.,  1.,  0.,  0.]),
 array([15., 20., 25., 55., 46., 17., 22., 13.,  1.,  0.]),
 array([10., 29., 19., 43., 48., 24., 16., 20.,  5.,  0.]),
 array([ 7., 27., 26., 57., 45., 23., 10., 17.,  2.,  0.]),
 array([17., 25., 30., 45., 56., 23.,  4., 10.,  4.,  0.]),
 array([18., 32., 33., 44., 52., 19.,  7.,  6.,  3.,  0.]),
 array([19., 40., 26., 58., 46., 16.,  5.,  3.,  1.,  0.]),
 array([ 8., 41., 36., 65., 39., 22.,  3.,  0.,  0.,  0.]),
 array([18., 31., 22., 66., 40., 19., 12.,  5.,  1.,  0.]),
 array([ 9., 27., 49., 50., 32., 28.,  6., 11.,  2.,  0.]),
 array([ 8., 24., 39., 61., 38., 27.,  8.,  9.,  0.,  0.]),
 array([ 4., 17., 38., 68., 47., 24., 12.,  4.,  0.,  0.]),
 array([ 6., 17., 59., 66., 41., 16.,  9.,  0.,  0.,  0.]),
 array([ 6., 22., 44., 71., 47., 18.,  6.,  0.,  0.,  0.]),
 array([ 1., 16., 52., 76., 45., 22.,  2.,  0.,  0.,  0.]),
 array([ 3., 22., 64., 60., 46., 15.,  4.,  0.,  0.,  0.]),
 array([ 2., 24., 65., 69., 32., 19.,  3.,  0.,  0.,  0.]),
 array([ 1., 23., 66., 73., 32., 18.,  1.,  0.,  0.,  0.]),
 array([ 3., 35., 59., 65., 36., 16.,  0.,  0.,  0.,  0.]),
 array([12., 19., 79., 54., 40., 10.,  0.,  0.,  0.,  0.]),
 array([11., 28., 68., 63., 34., 10.,  0.,  0.,  0.,  0.]),
 array([15., 39., 62., 54., 34., 10.,  0.,  0.,  0.,  0.])],
array([100. , 114.6, 129.2, 143.8, 158.4, 173. , 187.6, 202.2, 216.8,
       231.4, 246. ]),
<a list of 235 Lists of Patches objects>)
```


0
25
50
75
100
125
150
175
200
0
50
100
150
200


80
60
40
20
0
0
20
40
60
80
100
120
140


0
25
50
75
100
125
150
175
200
0
50
100
150
200



In [ ]:

```
'''Here we can observe that , as we add 100 images matrix to the original image,
Historgram values have jumped from 140 to 240. Also the image look lil bit more brighte
r
'''
```

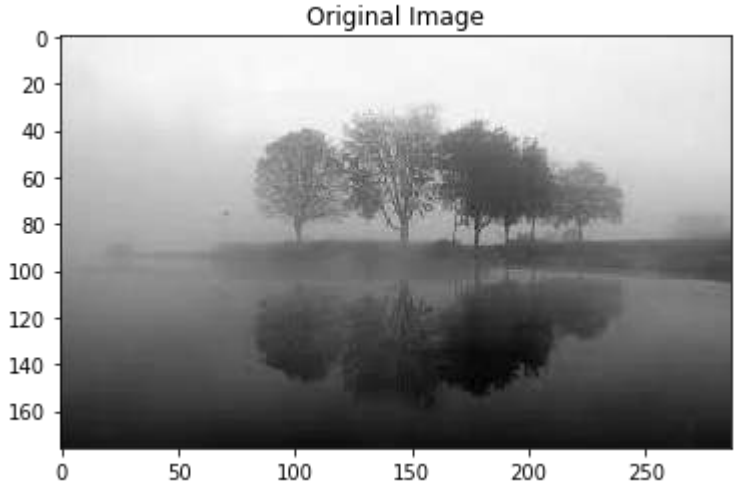
In [ ]:

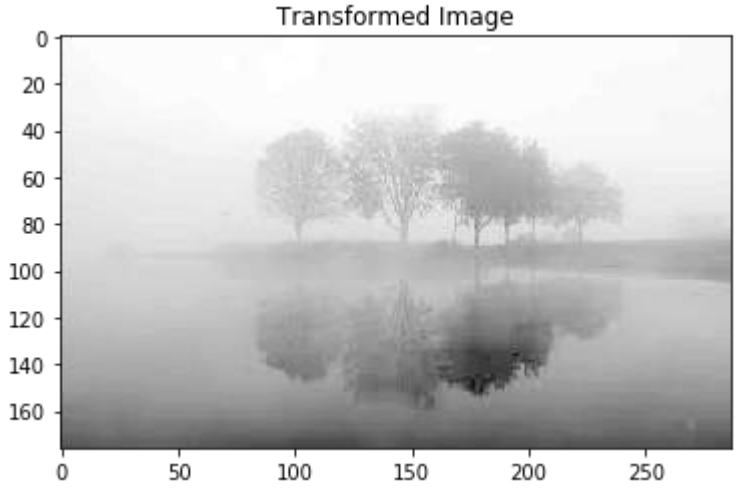
```
# Gamma Transformation
```

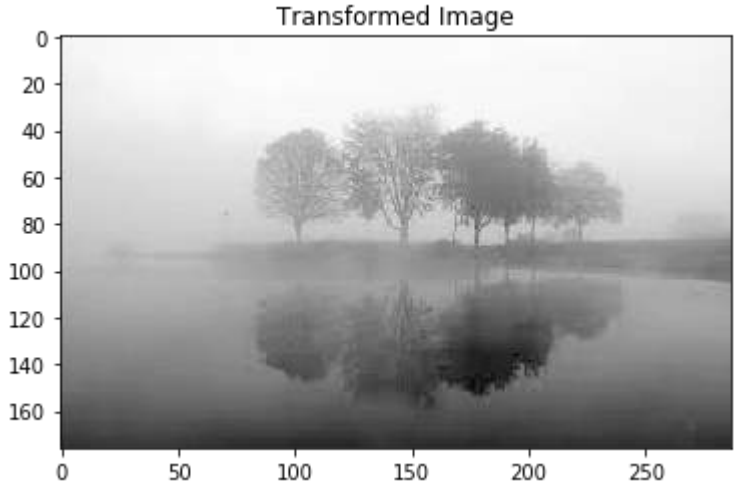
In [81]:

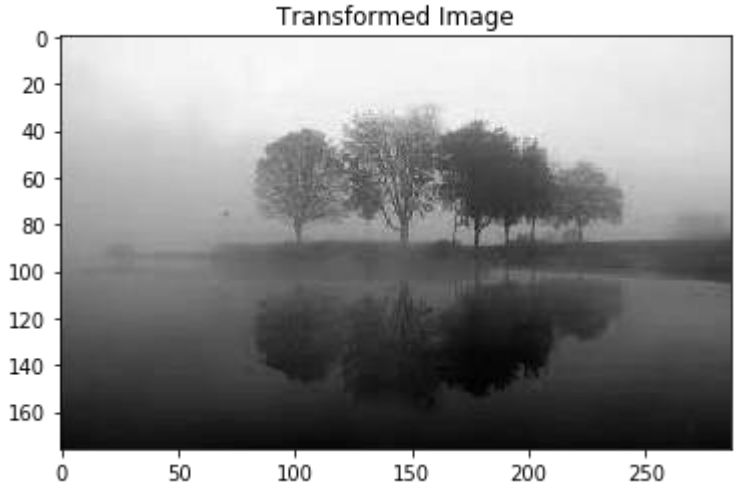
```
img = cv2.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-2/Foggy.png')
def rgb2gray(rgb):
    r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
    gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
    return gray
# Trying 4 gamma values.
a=rgb2gray(img)
plt.figure()
plt.imshow(a, cmap='gray')
plt.title("Original Image")
for gamma in [0.1, 0.5, 1.2, 2.2]:
    gamma_corrected = np.array(255*(a / 255) ** gamma, dtype = 'uint8')
    print('gamma_transformed '+str(gamma)+'.jpg')
    plt.figure()
    plt.imshow(gamma_corrected, cmap='gray')
    plt.title("Transformed Image")
```

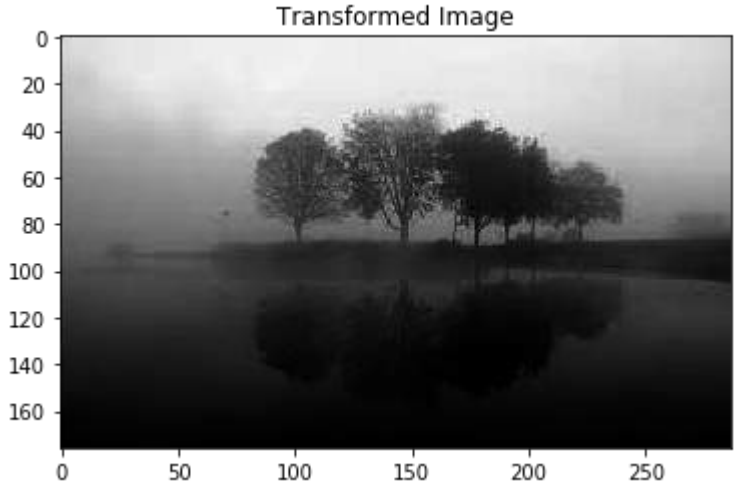
```
gamma_transformed 0.1.jpg
gamma_transformed 0.5.jpg
gamma_transformed 1.2.jpg
gamma_transformed 2.2.jpg
```











In [ ]:

```
'''
As the values of gamma increases the less bright/foggy image becomes darks.
We can see that the trees are getting darker and darker as we increase the
sigma value
'''
```

In [ ]:

```
#Contrast Strectching
```

In [87]:

```
img = cv2.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-2/Foggy2.png')
a=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
[row, col]=np.shape(a)
r_max=np.max(a)
r_min=np.min(a)
s_max=255
s_min=50
b=np.zeros(np.shape(a))

for i in range(row):
    for j in range(col):
       # if (0 <= a[i][j] and a[i][j] <= r_min):
       #     b[i][j]=(s_min / r_min)*a[i][j]
       # elif (r_min < a[i][j] and a[i][j] <= r_max):
            b[i][j]=((s_max - s_min)/(r_max - r_min)) * (a[i][j] - r_min) + s_min
       # else:
       #     b[i][j]=((255 - s_max)/(255 - r_max)) * (a[i][j] - r_max) + s_max
plt.figure(1)
plt.imshow(a, cmap='gray')
plt.figure(2)
plt.hist(a)
plt.figure(3)
plt.imshow(b, cmap='gray')
plt.figure(4)
plt.hist(b)
```

Out[87]:

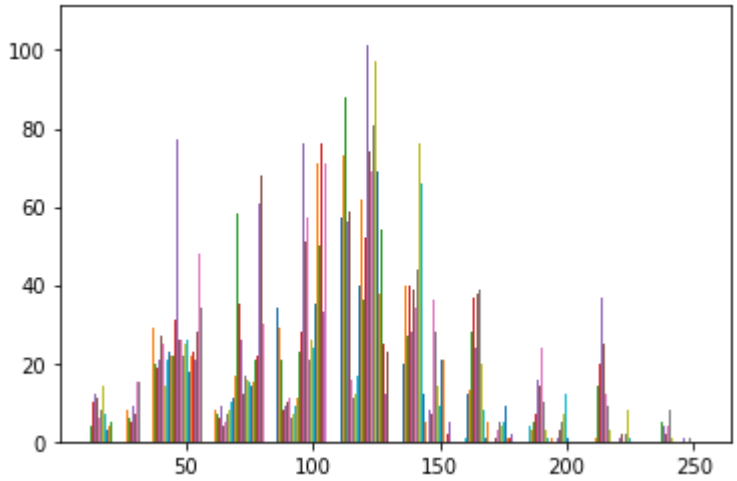
```
([array([ 4., 29.,  8., 34., 72., 34.,  2.,  0.,  0.,  0.]),
  array([ 4., 29.,  8., 28., 74., 38.,  2.,  0.,  0.,  0.]),
  array([ 4., 29.,  7., 37., 62., 38.,  6.,  0.,  0.,  0.]),
  array([ 4., 28.,  8., 35., 69., 36.,  3.,  0.,  0.,  0.]),
  array([ 5., 26., 12., 33., 85., 22.,  0.,  0.,  0.,  0.]),
  array([ 7., 24., 26., 26., 84., 13.,  3.,  0.,  0.,  0.]),
  array([ 9., 22., 21., 60., 58., 13.,  0.,  0.,  0.,  0.]),
  array([ 9., 22., 20., 66., 46., 18.,  1.,  1.,  0.,  0.]),
  array([13., 18., 21., 57., 54., 16.,  4.,  0.,  0.,  0.]),
  array([13., 18., 21., 57., 56., 16.,  1.,  1.,  0.,  0.]),
  array([15., 16., 22., 51., 57., 20.,  2.,  0.,  0.,  0.]),
  array([14., 17.,  8., 29., 80., 31.,  4.,  0.,  0.,  0.]),
  array([12., 20.,  7., 16., 79., 47.,  2.,  0.,  0.,  0.]),
  array([10., 23.,  7., 16., 69., 49.,  9.,  0.,  0.,  0.]),
  array([13., 20.,  6., 17., 85., 36.,  6.,  0.,  0.,  0.]),
  array([16., 17.,  7., 19., 71., 44.,  6.,  2.,  1.,  0.]),
  array([14., 19.,  6., 16., 68., 45., 11.,  2.,  2.,  0.]),
  array([16., 17.,  6., 16., 69., 44.,  9.,  3.,  3.,  0.]),
  array([14., 19.,  7., 20., 73., 40.,  5.,  3.,  2.,  0.]),
  array([12., 21.,  6., 20., 83., 29.,  8.,  4.,  0.,  0.]),
  array([ 9., 24.,  7., 21., 76., 30., 12.,  3.,  1.,  0.]),
  array([ 9., 24.,  6., 22., 73., 40.,  4.,  5.,  0.,  0.]),
  array([14., 19.,  6., 21., 76., 30.,  7.,  8.,  2.,  0.]),
  array([14., 19.,  6., 25., 73., 34., 12.,  0.,  0.,  0.]),
  array([12., 21.,  6., 35., 64., 31., 14.,  0.,  0.,  0.]),
  array([12., 21.,  6., 29., 72., 25., 14.,  3.,  1.,  0.]),
  array([12., 21.,  6., 23., 79., 31., 11.,  0.,  0.,  0.]),
  array([12., 21.,  6., 26., 88., 18., 11.,  1.,  0.,  0.]),
  array([12., 21.,  6., 19., 89., 23., 12.,  0.,  1.,  0.]),
  array([12., 21.,  6., 13., 86., 24., 19.,  2.,  0.,  0.]),
  array([12., 21.,  6., 14., 92., 17., 16.,  5.,  0.,  0.]),
  array([12., 21.,  6., 12., 89., 27., 13.,  3.,  0.,  0.]),
  array([10., 23.,  6., 11., 88., 27., 14.,  3.,  1.,  0.]),
  array([11., 21.,  7.,  8., 96., 22., 14.,  4.,  0.,  0.]),
  array([11., 21.,  9., 18., 84., 25., 11.,  0.,  4.,  0.]),
  array([11., 21.,  9., 14., 88., 24., 12.,  1.,  3.,  0.]),
  array([12., 21.,  8., 10., 82., 31., 15.,  1.,  2.,  1.]),
  array([13., 21.,  7., 14., 79., 30., 14.,  3.,  2.,  0.]),
  array([12., 21.,  7., 19., 77., 28., 15.,  4.,  0.,  0.]),
  array([ 9., 23.,  7., 15., 81., 31., 14.,  1.,  2.,  0.]),
  array([ 9., 24.,  6.,  8., 81., 32., 16.,  6.,  1.,  0.]),
  array([10., 23.,  6.,  8., 70., 39., 20.,  6.,  1.,  0.]),
  array([ 8., 25.,  6., 13., 61., 36., 28.,  5.,  1.,  0.]),
  array([ 3., 31.,  5., 12., 59., 40., 25.,  5.,  3.,  0.]),
  array([ 4., 30.,  5.,  9., 56., 46., 24.,  7.,  2.,  0.]),
  array([ 7., 27.,  4.,  9., 61., 35., 29.,  9.,  2.,  0.]),
  array([ 6., 28.,  4.,  9., 50., 45., 28., 12.,  1.,  0.]),
  array([ 9., 26.,  2.,  9., 49., 46., 26., 14.,  2.,  0.]),
  array([ 8., 25.,  6.,  7., 48., 47., 24.,  5., 13.,  0.]),
  array([ 8., 25.,  6.,  7., 51., 31., 36.,  5., 12.,  2.]),
  array([ 8., 25.,  5.,  8., 49., 28., 35.,  9., 15.,  1.]),
  array([ 8., 25.,  5.,  8., 45., 30., 39., 10., 13.,  0.]),
  array([ 8., 24.,  6.,  8., 38., 39., 39.,  6., 14.,  1.]),
  array([ 8., 24.,  6.,  8., 43., 35., 37.,  7., 13.,  2.]),
  array([ 8., 24.,  6.,  8., 60., 28., 27., 13.,  9.,  0.]),
  array([ 8., 24.,  5., 10., 59., 28., 23., 13., 11.,  2.]),
  array([ 6., 25.,  5.,  8., 70., 13., 24., 12., 17.,  3.]),
  array([ 8., 23.,  5.,  9., 68., 12., 27., 10., 18.,  3.]),
  array([10., 21.,  4., 16., 60., 14., 30., 10., 15.,  3.]),
```

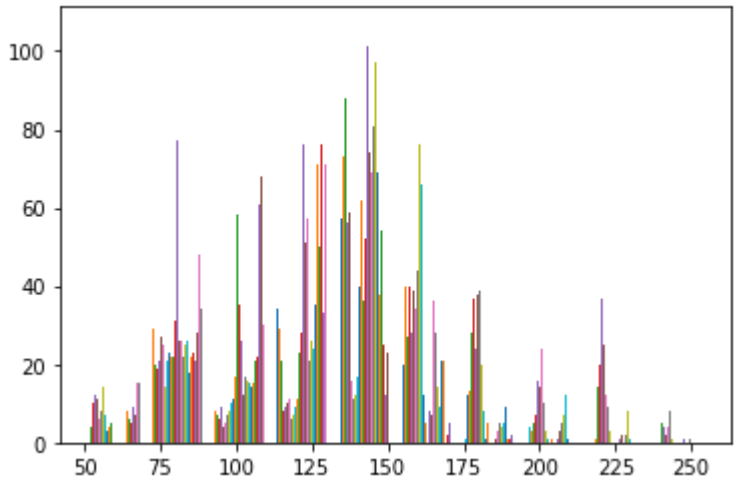
```
array([12., 19.,  4., 22., 50., 16., 21., 22., 13.,  4.]),
array([11., 20.,  4., 19., 48., 27., 12., 21., 21.,  0.]),
array([10., 21.,  4., 18., 40., 35., 12., 24., 16.,  3.]),
array([ 8., 23.,  4., 12., 35., 44., 15., 20., 17.,  5.]),
array([ 7., 24.,  4., 21., 30., 36., 19., 18., 20.,  4.]),
array([16., 13.,  8., 25., 12., 43., 24., 16., 23.,  3.]),
array([16., 13.,  7., 16., 25., 39., 21., 16., 28.,  2.]),
array([15., 14.,  7., 11., 16., 56., 18., 18., 27.,  1.]),
array([14., 15.,  7.,  8., 15., 55., 22., 18., 29.,  0.]),
array([14., 14.,  8.,  7., 13., 53., 25., 18., 27.,  4.]),
array([ 6., 22.,  8.,  7., 13., 50., 26., 19., 30.,  2.]),
array([ 5., 23.,  8.,  7., 13., 51., 21., 16., 39.,  0.]),
array([ 5., 23.,  8.,  8., 11., 45., 29., 19., 33.,  2.]),
array([10., 19.,  7.,  7., 14., 32., 41., 19., 29.,  5.]),
array([12., 17.,  7.,  7., 13., 36., 37., 17., 32.,  5.]),
array([ 9., 20.,  7.,  7., 12., 35., 38., 14., 37.,  4.]),
array([ 9., 20.,  7.,  7., 12., 33., 38., 14., 38.,  5.]),
array([ 8., 20.,  8.,  6., 11., 34., 41., 18., 31.,  6.]),
array([ 7., 21.,  8.,  6., 11., 32., 47., 18., 27.,  6.]),
array([ 4., 24.,  8.,  6., 12., 31., 43., 23., 29.,  3.]),
array([ 7., 21.,  8.,  6., 15., 30., 40., 25., 26.,  5.]),
array([ 6., 22.,  8.,  9., 13., 34., 38., 24., 26.,  3.]),
array([ 5., 23.,  8.,  7., 13., 32., 42., 23., 24.,  6.]),
array([ 5., 23.,  8.,  7., 11., 34., 46., 15., 29.,  5.]),
array([ 5., 22.,  9.,  6., 12., 38., 40., 20., 25.,  6.]),
array([ 4., 23., 10.,  6., 11., 39., 41., 27., 20.,  2.]),
array([ 4., 23.,  9.,  6., 12., 43., 37., 22., 25.,  2.]),
array([ 5., 22.,  9.,  7., 17., 40., 40., 24., 19.,  0.]),
array([ 4., 23.,  9.,  7., 13., 44., 39., 18., 23.,  3.]),
array([ 6., 20., 10.,  7., 12., 43., 41., 29., 14.,  1.]),
array([ 8., 18., 10., 11., 14., 42., 36., 21., 20.,  3.]),
array([ 3., 23., 10.,  9., 15., 46., 36., 21., 20.,  0.]),
array([ 6., 20., 10., 11., 15., 46., 34., 24., 16.,  1.]),
array([ 5., 21., 11.,  7., 19., 51., 31., 24., 14.,  0.]),
array([ 5., 21., 10.,  7., 15., 47., 33., 30., 12.,  3.]),
array([ 9., 17., 10.,  7., 16., 48., 37., 24., 11.,  4.]),
array([ 7., 19., 10.,  8., 16., 50., 44., 12., 13.,  4.]),
array([ 3., 23., 10.,  5., 17., 64., 32., 13., 12.,  4.]),
array([ 4., 22., 10.,  7., 17., 75., 24., 10., 10.,  4.]),
array([ 4., 22., 12.,  9., 19., 76., 20., 13.,  6.,  2.]),
array([ 5., 21., 11.,  9., 17., 81., 22.,  6.,  6.,  5.]),
array([ 4., 22., 11., 15., 23., 82.,  8.,  4.,  6.,  8.]),
array([ 4., 22., 10., 14., 41., 66., 11., 10.,  5.,  0.]),
array([ 5., 21., 11., 12., 55., 41., 16.,  9.,  8.,  5.]),
array([ 4., 22., 12., 40., 27., 45., 12.,  4., 13.,  4.]),
array([ 5., 18., 14., 34., 41., 40.,  8.,  5., 11.,  7.]),
array([ 7., 16., 13., 34., 33., 43., 13.,  3., 11., 10.]),
array([ 8., 14., 15., 21., 18., 71., 11.,  2., 12., 11.]),
array([ 9., 13., 16., 19., 24., 70.,  9.,  6.,  9.,  8.]),
array([10., 13., 14., 13., 31., 72.,  8.,  3., 11.,  8.]),
array([ 7., 16., 15., 11., 37., 66.,  8.,  8.,  9.,  6.]),
array([ 5., 18., 16., 12., 40., 64., 10.,  8.,  6.,  4.]),
array([ 8., 15., 17., 11., 51., 56., 12.,  4.,  7.,  2.]),
array([ 5., 22., 12., 15., 42., 66., 13.,  5.,  3.,  0.]),
array([ 5., 23., 14., 14., 51., 60., 11.,  4.,  1.,  0.]),
array([ 5., 22., 15., 18., 58., 50., 13.,  0.,  2.,  0.]),
array([ 5., 22., 17., 18., 79., 35.,  3.,  2.,  0.,  2.]),
array([ 4., 24., 24., 20., 64., 42.,  3.,  0.,  1.,  1.]),
array([ 4., 25., 23., 21., 66., 38.,  4.,  0.,  0.,  2.]),
array([ 4., 25., 25., 26., 51., 41.,  3.,  4.,  3.,  1.]),
array([ 3., 26., 31., 23., 67., 24.,  3.,  1.,  3.,  2.]),
```

```
array([ 3., 24., 49., 24., 70., 12.,  1.,  0.,  0.,  0.]),
array([ 3., 24., 58., 20., 62., 16.,  0.,  0.,  0.,  0.]),
array([ 2., 30., 58., 23., 69.,  1.,  0.,  0.,  0.,  0.]),
array([ 1., 31., 40., 18., 81.,  7.,  1.,  4.,  0.,  0.]),
array([ 0., 27., 38., 22., 83.,  4.,  2.,  3.,  1.,  3.]),
array([ 0., 29., 56., 27., 60.,  5.,  4.,  2.,  0.,  0.]),
array([ 0., 34., 78., 24., 45.,  2.,  0.,  0.,  0.,  0.]),
array([ 0., 38., 92., 24., 29.,  0.,  0.,  0.,  0.,  0.]),
array([  0.,  35., 106.,  19.,  23.,   0.,   0.,   0.,   0.,   0.]),
array([ 0., 32., 99., 21., 28.,  3.,  0.,  0.,  0.,  0.]),
array([ 0., 34., 76., 25., 43.,  5.,  0.,  0.,  0.,  0.]),
array([ 0., 40., 59., 35., 37.,  5.,  5.,  2.,  0.,  0.]),
array([ 0., 41., 58., 31., 36.,  6.,  8.,  0.,  2.,  1.]),
array([ 0., 65., 35., 28., 46.,  4.,  1.,  3.,  0.,  1.]),
array([ 0., 77., 18., 34., 43.,  4.,  4.,  2.,  1.,  0.]),
array([ 0., 59., 25., 40., 51.,  4.,  2.,  2.,  0.,  0.]),
array([ 0., 63., 17., 46., 47.,  5.,  3.,  1.,  1.,  0.]),
array([ 0., 59., 15., 42., 57.,  7.,  1.,  1.,  1.,  0.]),
array([ 0., 50., 18., 52., 52.,  3.,  2.,  3.,  3.,  0.]),
array([ 0., 51., 10., 51., 63.,  6.,  1.,  1.,  0.,  0.]),
array([ 0., 41., 18., 55., 65.,  1.,  2.,  0.,  1.,  0.]),
array([ 0., 35., 22., 66., 54.,  2.,  3.,  1.,  0.,  0.]),
array([ 0., 33., 51., 54., 45.,  0.,  0.,  0.,  0.,  0.]),
array([ 1., 42., 40., 48., 52.,  0.,  0.,  0.,  0.,  0.]),
array([ 0., 31., 26., 76., 45.,  1.,  4.,  0.,  0.,  0.]),
array([ 0., 26., 21., 69., 62.,  3.,  1.,  1.,  0.,  0.]),
array([ 0., 26., 22., 76., 59.,  0.,  0.,  0.,  0.,  0.]),
array([ 0., 25., 13., 72., 62.,  6.,  3.,  2.,  0.,  0.]),
array([ 0., 25., 13., 66., 70.,  2.,  4.,  3.,  0.,  0.]),
array([ 0., 24., 14., 59., 78.,  4.,  2.,  2.,  0.,  0.]),
array([ 0., 24., 17., 47., 91.,  1.,  3.,  0.,  0.,  0.]),
array([ 0., 24., 18., 46., 91.,  4.,  0.,  0.,  0.,  0.]),
array([ 0., 25., 12., 52., 93.,  1.,  0.,  0.,  0.,  0.]),
array([  0.,  25.,  14.,  37., 105.,   2.,   0.,   0.,   0.,   0.]),
array([  0.,  25.,  13.,  35., 101.,   8.,   1.,   0.,   0.,   0.]),
array([ 0., 25., 12., 51., 88.,  7.,  0.,  0.,  0.,  0.]),
array([ 0., 26., 12., 39., 98.,  7.,  1.,  0.,  0.,  0.]),
array([ 0., 26., 16., 43., 92.,  6.,  0.,  0.,  0.,  0.]),
array([ 1., 26., 13., 43., 94.,  6.,  0.,  0.,  0.,  0.]),
array([ 1., 25., 13., 49., 88.,  6.,  0.,  1.,  0.,  0.]),
array([ 0., 25., 15., 75., 58., 10.,  0.,  0.,  0.,  0.]),
array([ 0., 25., 16., 71., 64.,  7.,  0.,  0.,  0.,  0.]),
array([ 0., 25., 17., 55., 75., 11.,  0.,  0.,  0.,  0.]),
array([ 0., 25., 20., 44., 89.,  5.,  0.,  0.,  0.,  0.]),
array([ 0., 27., 15., 50., 86.,  5.,  0.,  0.,  0.,  0.]),
array([ 1., 24., 16., 60., 74.,  7.,  1.,  0.,  0.,  0.]),
array([ 2., 23., 17., 57., 70., 10.,  2.,  2.,  0.,  0.]),
array([ 3., 22., 17., 64., 57., 16.,  1.,  1.,  2.,  0.]),
array([ 0., 26., 20., 65., 37., 32.,  1.,  2.,  0.,  0.]),
array([ 0., 26., 26., 49., 52., 27.,  2.,  1.,  0.,  0.]),
array([ 0., 26., 24., 32., 73., 26.,  1.,  1.,  0.,  0.]),
array([ 0., 26., 16., 33., 84., 23.,  1.,  0.,  0.,  0.]),
array([ 0., 25., 17., 43., 74., 23.,  1.,  0.,  0.,  0.]),
array([ 0., 25., 16., 26., 89., 22.,  5.,  0.,  0.,  0.]),
array([ 0., 25., 17., 20., 79., 37.,  3.,  1.,  1.,  0.]),
array([ 0., 25., 23., 18., 92., 22.,  1.,  2.,  0.,  0.]),
array([ 0., 25., 19., 29., 69., 36.,  3.,  1.,  1.,  0.]),
array([ 0., 25., 18., 21., 89., 25.,  3.,  2.,  0.,  0.]),
array([ 0., 25., 16., 23., 94., 21.,  1.,  1.,  2.,  0.]),
array([ 0., 28., 17., 23., 86., 25.,  2.,  2.,  0.,  0.]),
array([ 0., 28., 14., 42., 73., 22.,  4.,  0.,  0.,  0.]),
```

```
array([ 0., 26., 15., 30., 73., 34.,  3.,  2.,  0.,  0.]),
array([ 0., 27., 13., 21., 81., 35.,  5.,  1.,  0.,  0.]),
array([ 0., 27., 12., 20., 84., 34.,  4.,  2.,  0.,  0.]),
array([ 0., 27., 12., 39., 64., 36.,  1.,  2.,  1.,  1.]),
array([ 0., 27., 12., 23., 80., 34.,  2.,  3.,  2.,  0.]),
array([ 1., 26., 14., 25., 77., 32.,  3.,  3.,  2.,  0.]),
array([ 5., 22., 13., 25., 81., 28.,  6.,  3.,  0.,  0.]),
array([ 5., 22., 14., 26., 81., 29.,  3.,  2.,  1.,  0.]),
array([ 1., 26., 15., 25., 81., 28.,  4.,  2.,  1.,  0.]),
array([ 0., 27., 15., 26., 78., 35.,  1.,  1.,  0.,  0.]),
array([ 0., 27., 15., 19., 83., 35.,  3.,  1.,  0.,  0.]),
array([ 6., 23., 10., 22., 87., 26.,  8.,  1.,  0.,  0.]),
array([ 6., 23., 10., 22., 82., 31.,  4.,  4.,  1.,  0.]),
array([ 7., 22., 10., 23., 79., 32.,  8.,  1.,  1.,  0.]),
array([ 9., 20., 10., 23., 88., 25.,  4.,  4.,  0.,  0.]),
array([ 9., 20., 10., 24., 92., 19.,  4.,  5.,  0.,  0.]),
array([ 9., 20., 10., 24., 86., 23.,  5.,  5.,  1.,  0.]),
array([ 8., 20., 11., 24., 97., 14.,  3.,  6.,  0.,  0.]),
array([ 8., 20., 11., 24., 87., 23.,  7.,  2.,  1.,  0.]),
array([ 9., 18., 14., 16., 92., 20., 10.,  3.,  1.,  0.]),
array([ 8., 19., 14., 26., 81., 22.,  6.,  7.,  0.,  0.]),
array([ 3., 24., 14., 54., 56., 15., 11.,  6.,  0.,  0.]),
array([ 3., 24., 15., 56., 48., 18., 13.,  6.,  0.,  0.]),
array([ 4., 23., 15., 39., 70., 16.,  7.,  7.,  2.,  0.]),
array([ 9., 18., 15., 39., 70., 16.,  7.,  7.,  2.,  0.]),
array([10., 18., 14., 41., 67., 17.,  6.,  8.,  2.,  0.]),
array([11., 17., 15., 30., 82., 14.,  5.,  7.,  2.,  0.]),
array([ 8., 20., 16., 37., 70., 17.,  4.,  7.,  4.,  0.]),
array([ 8., 20., 16., 35., 76.,  9.,  9.,  8.,  2.,  0.]),
array([ 8., 20., 15., 35., 69., 17.,  8.,  3.,  8.,  0.]),
array([ 6., 22., 15., 35., 65., 22.,  4.,  6.,  8.,  0.]),
array([ 6., 22., 14., 40., 60., 18.,  8.,  3., 12.,  0.]),
array([ 5., 23., 14., 36., 64., 18.,  8.,  7.,  8.,  0.]),
array([ 5., 23., 13., 31., 77., 14.,  8., 11.,  1.,  0.]),
array([ 5., 23., 13., 33., 75., 11., 10.,  9.,  4.,  0.]),
array([ 5., 23., 13., 37., 75.,  9.,  7.,  9.,  5.,  0.]),
array([ 5., 23., 13., 48., 59., 14.,  6., 10.,  4.,  1.]),
array([ 5., 23., 17., 62., 43., 13.,  6.,  6.,  8.,  0.]),
array([ 5., 23., 19., 54., 40., 24.,  5., 12.,  1.,  0.]),
array([ 5., 23., 21., 57., 41., 21.,  7.,  8.,  0.,  0.]),
array([ 5., 23., 21., 71., 38., 11.,  8.,  6.,  0.,  0.]),
array([ 5., 23., 21., 49., 60., 11.,  9.,  5.,  0.,  0.]),
array([ 5., 23., 22., 39., 62., 20., 11.,  1.,  0.,  0.]),
array([ 7., 22., 16., 45., 64., 22.,  6.,  1.,  0.,  0.]),
array([ 7., 22., 14., 43., 64., 30.,  2.,  0.,  1.,  0.]),
array([ 7., 22., 15., 45., 64., 22.,  6.,  1.,  0.,  1.]),
array([ 7., 22., 16., 39., 68., 18.,  9.,  1.,  3.,  0.]),
array([ 7., 22., 16., 45., 59., 22.,  7.,  3.,  2.,  0.]),
array([ 7., 22., 16., 48., 59., 18.,  6.,  5.,  1.,  1.]),
array([ 7., 22., 19., 48., 52., 22.,  9.,  1.,  2.,  1.]),
array([ 7., 22., 23., 48., 52., 21.,  5.,  5.,  0.,  0.]),
array([ 9., 20., 23., 50., 54., 16.,  8.,  2.,  0.,  1.]),
array([ 9., 20., 22., 55., 52., 19.,  5.,  0.,  1.,  0.]),
array([ 9., 21., 24., 58., 51., 19.,  1.,  0.,  0.,  0.]),
array([ 9., 21., 26., 75., 40., 12.,  0.,  0.,  0.,  0.]),
array([ 9., 22., 28., 82., 37.,  5.,  0.,  0.,  0.,  0.]),
array([ 9., 23., 38., 80., 26.,  7.,  0.,  0.,  0.,  0.]),
array([ 9., 24., 33., 84., 27.,  4.,  1.,  1.,  0.,  0.]),
array([ 9., 23., 33., 91., 25.,  2.,  0.,  0.,  0.,  0.]),
array([ 7., 25., 34., 90., 25.,  1.,  1.,  0.,  0.,  0.]),
array([ 7., 24., 44., 79., 26.,  2.,  1.,  0.,  0.,  0.]),
```

```
 array([ 7., 25., 48., 73., 28.,  0.,  2.,  0.,  0.,  0.]),
 array([ 7., 25., 49., 76., 25.,  1.,  0.,  0.,  0.,  0.]),
 array([ 7., 24., 61., 71., 20.,  0.,  0.,  0.,  0.,  0.]),
 array([ 7., 28., 64., 63., 21.,  0.,  0.,  0.,  0.,  0.]),
 array([ 7., 27., 64., 56., 26.,  3.,  0.,  0.,  0.,  0.]),
 array([ 7., 29., 69., 48., 28.,  2.,  0.,  0.,  0.,  0.]),
 array([11., 24., 74., 45., 27.,  1.,  1.,  0.,  0.,  0.]),
 array([11., 26., 72., 53., 17.,  3.,  0.,  1.,  0.,  0.]),
 array([10., 28., 78., 53., 13.,  0.,  1.,  0.,  0.,  0.]),
 array([11., 29., 78., 48., 11.,  4.,  2.,  0.,  0.,  0.]),
 array([12., 29., 75., 53., 11.,  1.,  2.,  0.,  0.,  0.]),
 array([13., 35., 91., 32.,  9.,  2.,  1.,  0.,  0.,  0.]),
 array([14., 47., 72., 33., 12.,  3.,  2.,  0.,  0.,  0.]),
 array([15., 48., 68., 25., 17.,  5.,  2.,  3.,  0.,  0.]),
 array([15., 48., 66., 26., 17.,  7.,  2.,  1.,  1.,  0.]),
 array([15., 55., 71., 16., 14.,  9.,  2.,  1.,  0.,  0.]),
 array([15., 53., 69., 20., 12.,  8.,  2.,  1.,  3.,  0.]),
 array([15., 50., 70., 26., 13.,  5.,  4.,  0.,  0.,  0.]),
 array([15., 52., 73., 16., 13.,  7.,  3.,  3.,  1.,  0.]),
 array([15., 53., 60., 29., 17.,  5.,  2.,  1.,  1.,  0.]),
 array([15., 50., 36., 50., 22.,  5.,  5.,  0.,  0.,  0.]),
 array([15., 50., 24., 64., 21.,  8.,  1.,  0.,  0.,  0.]),
 array([17., 46., 22., 70., 19.,  5.,  2.,  2.,  0.,  0.]),
 array([16., 45., 17., 76., 23.,  3.,  2.,  1.,  0.,  0.]),
 array([15., 41., 30., 71., 21.,  2.,  3.,  0.,  0.,  0.]),
 array([15., 34., 34., 72., 24.,  0.,  2.,  1.,  1.,  0.]),
 array([15., 23., 48., 69., 20.,  5.,  2.,  0.,  1.,  0.]),
 array([15., 23., 47., 82., 14.,  1.,  1.,  0.,  0.,  0.]),
 array([15., 26., 47., 82., 12.,  1.,  0.,  0.,  0.,  0.]),
 array([15., 29., 47., 80., 11.,  1.,  0.,  0.,  0.,  0.]),
 array([13., 27., 56., 77.,  8.,  2.,  0.,  0.,  0.,  0.]),
 array([13., 26., 63., 74.,  7.,  0.,  0.,  0.,  0.,  0.]),
 array([13., 31., 73., 60.,  6.,  0.,  0.,  0.,  0.,  0.])],
array([ 50. ,  70.5,  91. , 111.5, 132. , 152.5, 173. , 193.5, 214. ,
       234.5, 255. ]),
<a list of 275 Lists of Patches objects>)
```





In [ ]:

```
'''
In the above histograms we can observe a range gaps have been stretched
and also the background is more clear than the first pic.
'''
```

In [ ]:

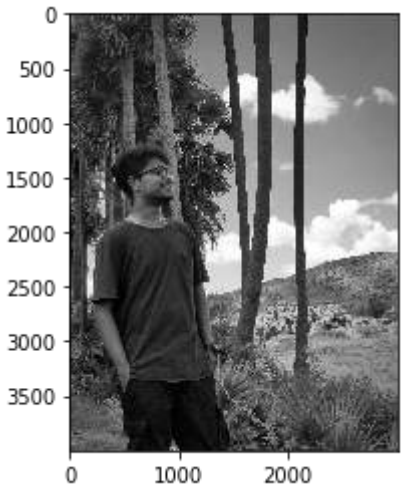
```
#Flip, Transpose and Roatation of Pictures
```

In [101]:

```
a=cv2.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-2/Andrew_B&W.jpg')
#cv2.imshow('Normal',a)
#cv2.waitKey(0)
#cv2.destroyAllWindows()
plt.imshow(a)
```

Out[101]:

```
<matplotlib.image.AxesImage at 0x21c05cf1e88>
```



In [102]:

```
flip_a=cv2.flip(a,0)
plt.figure()
plt.imshow(flip_a,cmap='brg')
```

Out[102]:

```
<matplotlib.image.AxesImage at 0x21c06051688>
```

In [104]:

```
transpose_a=cv2.transpose(a)
plt.imshow(transpose_a)
```

Out[104]:

```
<matplotlib.image.AxesImage at 0x21c06330d08>
```

In [105]:

```
rot_a=cv2.rotate(a, cv2.ROTATE_90_COUNTERCLOCKWISE)
plt.imshow(rot_a)
```

Out[105]:

```
<matplotlib.image.AxesImage at 0x21c06921288>
```

In [106]:

```
rot_a=cv2.rotate(a, cv2.ROTATE_90_CLOCKWISE)
plt.imshow(rot_a)
```

Out[106]:

```
<matplotlib.image.AxesImage at 0x21c06c68f88>
```

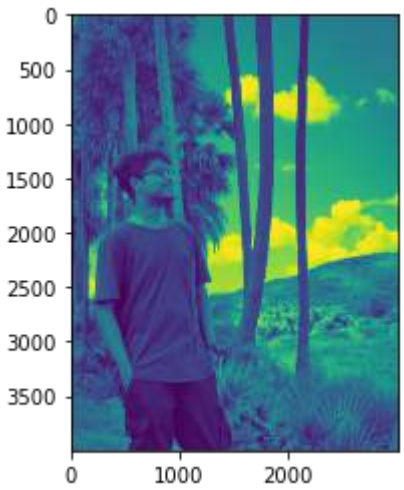
In [ ]:

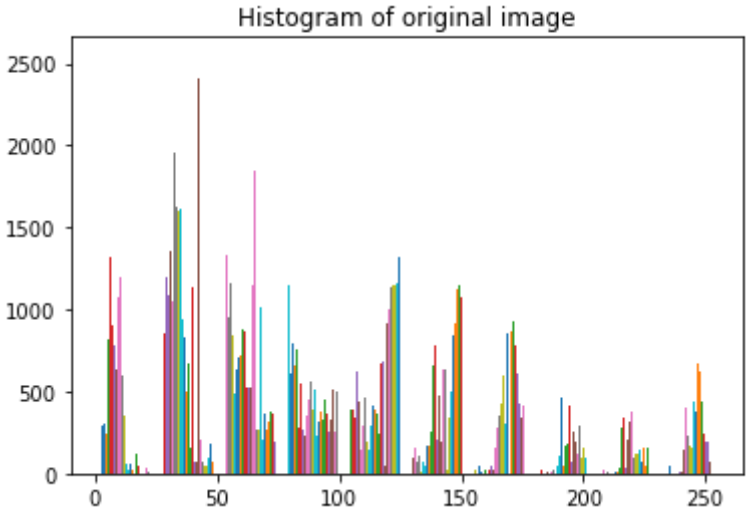
```
# Log Transformation
```

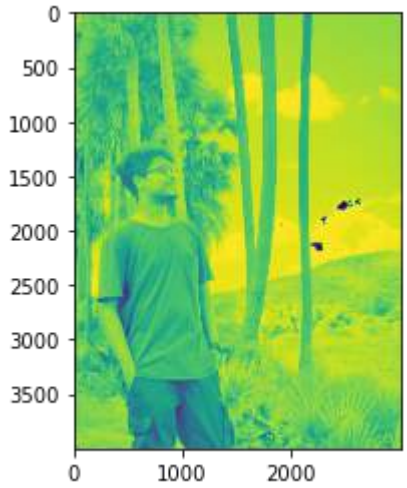
In [10]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
a=cv2.imread('D:/GOOGLE DRIVE - VIT/WIN SEM 2020-21/PYTHON VAC/Lecture6_20thMar/Andr
print('############## Log Transformation ################')
print('s=c*log(1+r)')
print('c=255 /log(1 + rmax(a))')
a=cv2.cvtColor(a,cv2.COLOR_BGR2GRAY)
#a=rgb2gray(img)
c=255 / np.log(1 + np.max(a)) #c=1.5
log_a = c * (np.log(a + 1))
log_a=np.array(log_a, dtype=np.uint8)
plt.figure(1)
plt.imshow(a)
plt.figure(2)
plt.hist(a)
plt.title("Histogram of original image")
plt.figure(3)
plt.imshow(log_a)
plt.figure(4)
plt.hist(log_a)
plt.title("Histogram of log transformed image")
```

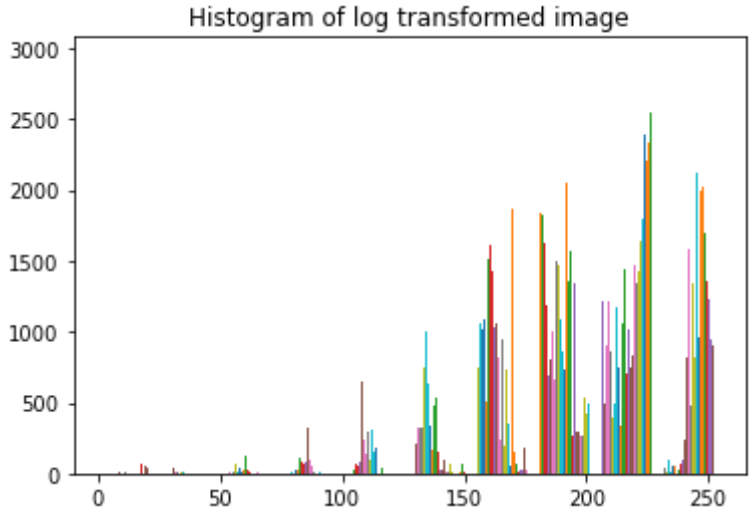
```
############## Log Transformation ################
s=c*log(1+r)
c=255 /log(1 + rmax(a))
<ipython-input-10-326c8572b819>:11: RuntimeWarning: divide by zero encountered in lo
g
  log_a = c * (np.log(a + 1))
```

Out[10]: Text(0.5, 1.0, 'Histogram of log transformed image')






0
500
1000
1500
2000
2500
3000
3500
0
1000
2000


Histogram of log transformed image
3000
2500
2000
1500
1000
500
0
0
50
100
150
200
250

In [ ]:

In [ ]:

In [ ]:

```
#VAP - PVT, Lecture-9
#27/03/21 - Saturday
#18BEC1278 - Andrew John
```

In [ ]:

```

```

In [ ]:

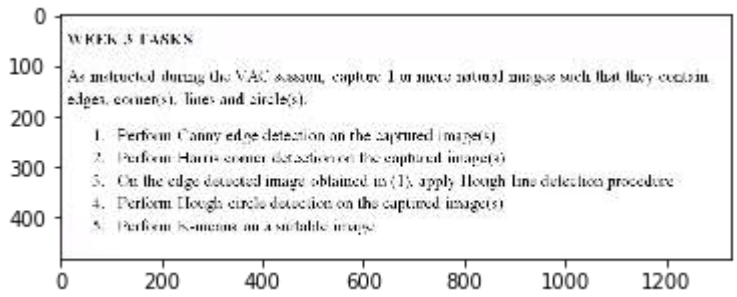
```
'''
1. Perform Canny Edge Detection on the Captured Image(s).
2. Perform Harris Corner Detection on the Captured Image(s).
3. On the Edge Detected image obtained in (1), apply Hough Line Detection procedure.
4. Perform Hough Circle Detection on the captured image(s).
5. Perform K-means on a suitable image.
'''
```

In [38]:

```
#GIVEN TASK
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-3/Task.jpg')
plt.figure(1)
plt.imshow(img)
```

Out[38]:

```
<matplotlib.image.AxesImage at 0x13902d50d88>
```



In [ ]:

```

```

In [ ]:

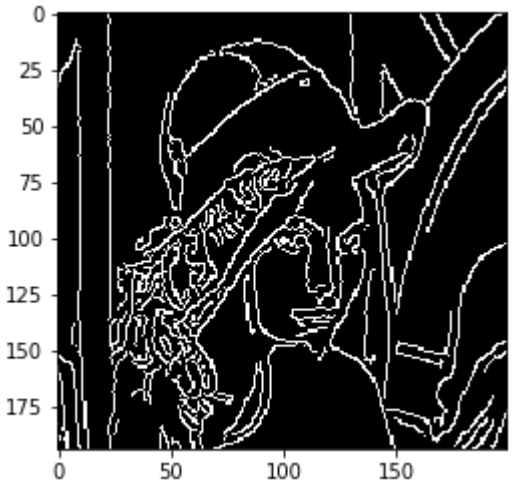
```
#CANNY EDGE DETECTOR
'''
The Canny edge detector is an edge detection operator
that uses a multi-stage algorithm to detect a wide range
of edges in images.
'''
```

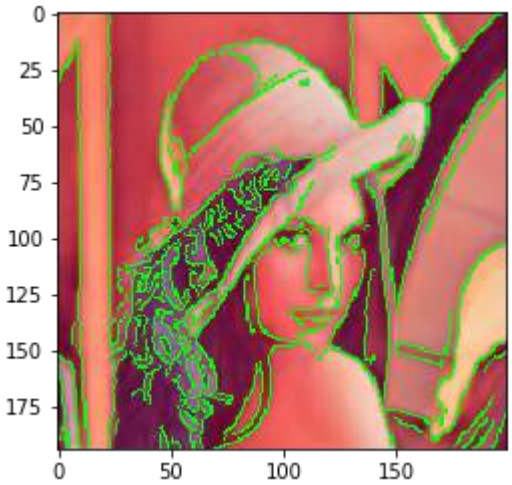
In [140]:

```
#Smoothness Included
import cv2
import matplotlib.pyplot as plt
a = cv2.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-3/lena.jpg')
th1=30
th2=3*th1
d=2
a_gauss=cv2.GaussianBlur(a, (2*d+1, 2*d+1), -1)[d:-d,d:-d]
img= cv2.cvtColor(a_gauss, cv2.COLOR_BGR2GRAY)
a_edge = cv2.Canny(img, th1, th2)
plt.figure(1)
plt.imshow(a_edge, cmap='gray')
a_gauss[a_edge != 0] = (0, 255, 0)
plt.figure(2)
plt.imshow(cv2.cvtColor(a_gauss, cv2.COLOR_BGR2RGB))
```

Out[140]:

```
<matplotlib.image.AxesImage at 0x13919f12f48>
```



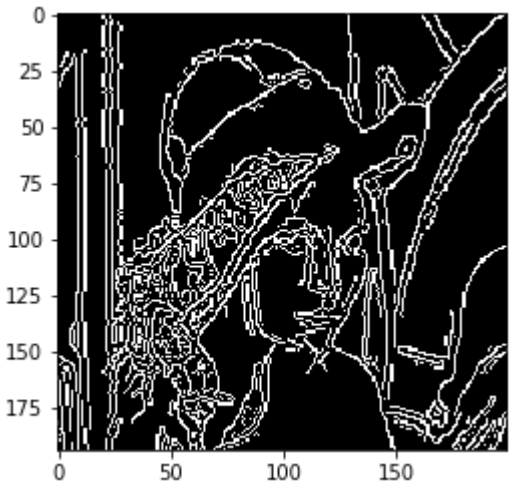


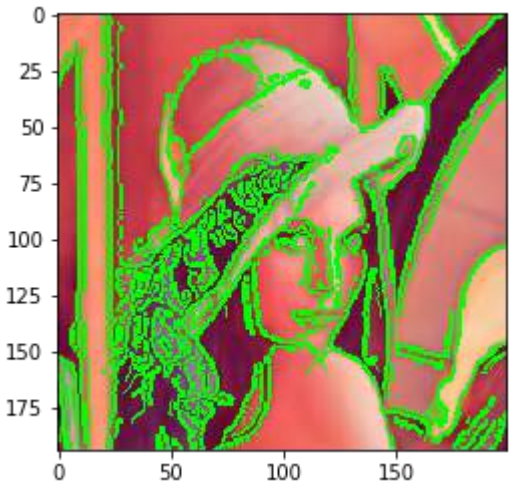
In [141]:

```
#No Smothness Included
import cv2
import matplotlib.pyplot as plt
a = cv2.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-3/lena.jpg')
th1=30
th2=3*th1
d=2
#a_gauss=cv2.GaussianBlur(a, (2*d+1, 2*d+1), -1)[d:-d,d:-d]
img= cv2.cvtColor(a_gauss, cv2.COLOR_BGR2GRAY)
a_edge = cv2.Canny(img, th1, th2)
plt.figure(1)
plt.imshow(a_edge, cmap='gray')
a_gauss[a_edge != 0] = (0, 255, 0)
plt.figure(2)
plt.imshow(cv2.cvtColor(a_gauss, cv2.COLOR_BGR2RGB))
```

Out[141]:

```
<matplotlib.image.AxesImage at 0x1391a1f44c8>
```



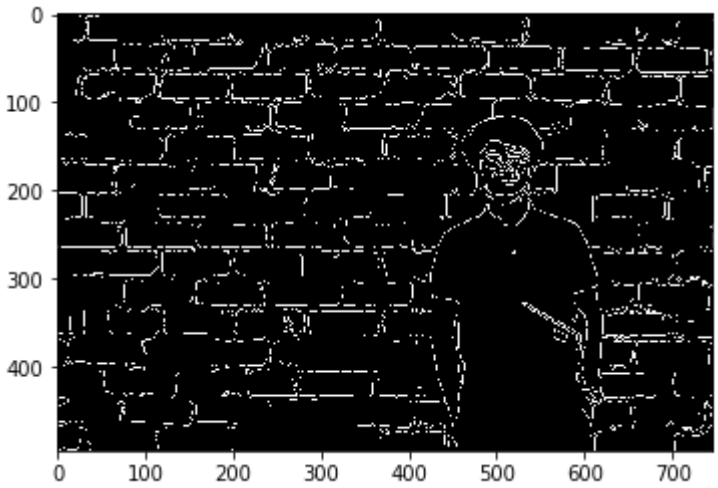


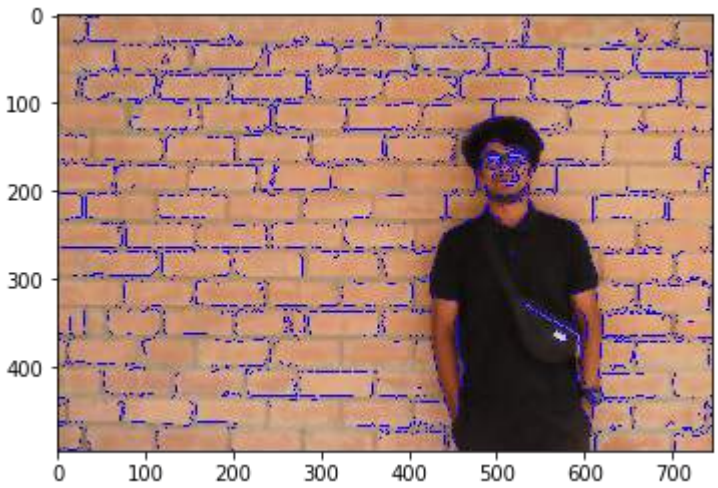
In [128]:

```
import cv2
import matplotlib.pyplot as plt
a = cv2.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-3/L93.jpeg')
th1=30
th2=3*th1
d=2
a_gauss=cv2.GaussianBlur(a, (2*d+1, 2*d+1), -1)[d:-d,d:-d]
img= cv2.cvtColor(a_gauss, cv2.COLOR_BGR2GRAY)
a_edge = cv2.Canny(img, th1, th2)
plt.figure(1)
plt.imshow(a_edge, cmap='gray')
a_gauss[a_edge != 0] = (255, 0, 0)
plt.figure(2)
plt.imshow(cv2.cvtColor(a_gauss, cv2.COLOR_BGR2RGB))
```

Out[128]:

```
<matplotlib.image.AxesImage at 0x1390aaaf3c8>
```



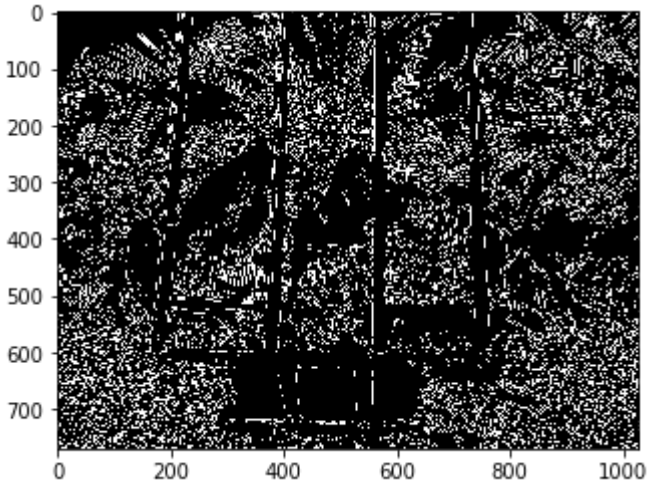


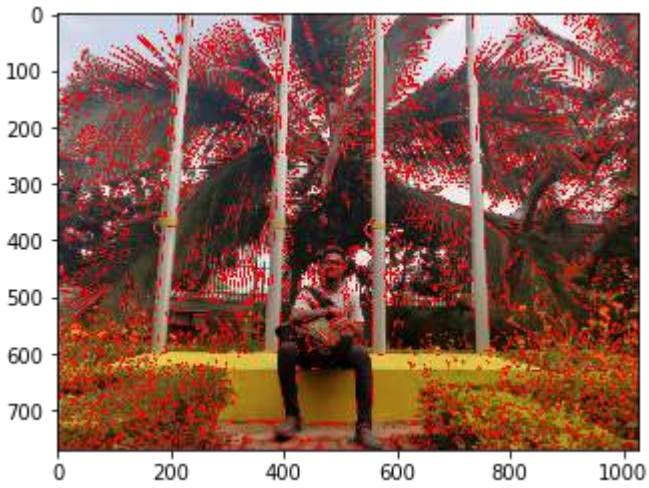
In [129]:

```
import cv2
import matplotlib.pyplot as plt
a = cv2.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-3/L91.jpeg')
th1=40
th2=3*th1
d=2
a_gauss=cv2.GaussianBlur(a, (2*d+1, 2*d+1), -1)[d:-d,d:-d]
img= cv2.cvtColor(a_gauss, cv2.COLOR_BGR2GRAY)
a_edge = cv2.Canny(img, th1, th2)
plt.figure(1)
plt.imshow(a_edge, cmap='gray')
a_gauss[a_edge != 0] = (0, 0, 255)
plt.figure(2)
plt.imshow(cv2.cvtColor(a_gauss, cv2.COLOR_BGR2RGB))
```

Out[129]:

```
<matplotlib.image.AxesImage at 0x1390d52a108>
```



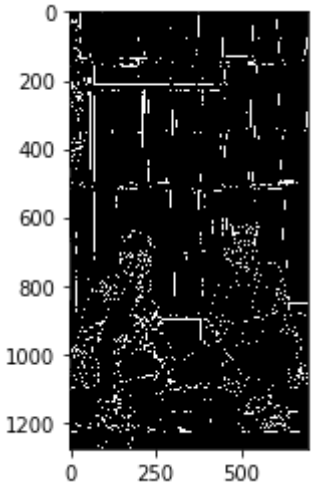


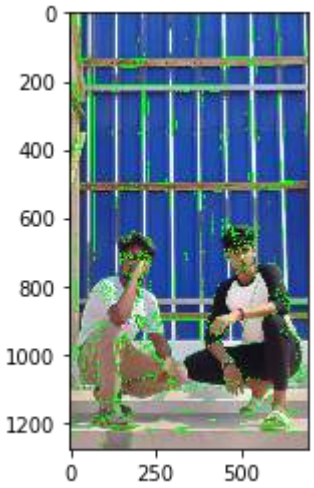
In [131]:

```
import cv2
import matplotlib.pyplot as plt
a = cv2.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-3/L96.jpeg')
th1=40
th2=3*th1
d=2
a_gauss=cv2.GaussianBlur(a, (2*d+1, 2*d+1), -1)[d:-d,d:-d]
img= cv2.cvtColor(a_gauss, cv2.COLOR_BGR2GRAY)
a_edge = cv2.Canny(img, th1, th2)
plt.figure(1)
plt.imshow(a_edge, cmap='gray')
a_gauss[a_edge != 0] = (0, 255, 0)
plt.figure(2)
plt.imshow(cv2.cvtColor(a_gauss, cv2.COLOR_BGR2RGB))
```

Out[131]:

```
<matplotlib.image.AxesImage at 0x1391962e388>
```





In [ ]:

```
#INFERENCE:
'''
Threshold value is image dependent.
As you increase the upper threshold value
more the image is smoothened .
'''
```

In [ ]:

```
```

In [ ]:

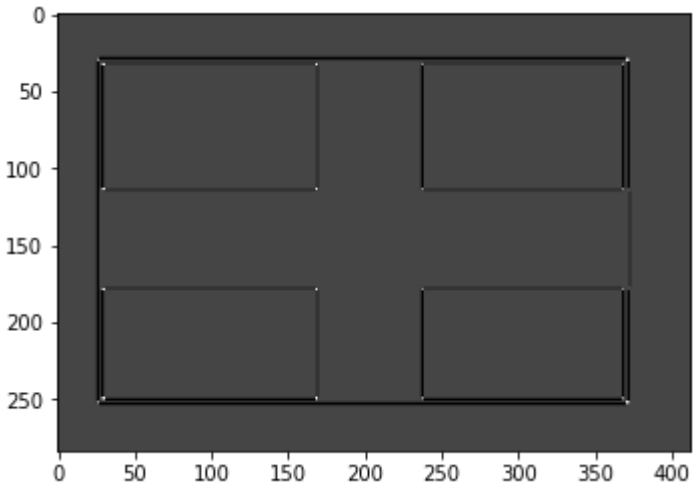
```
#HARRIS CORNER DETECTION
'''
Harris Corner Detector is a corner detection operator that is
commonly used in computer vision algorithms to
extract corners and infer features of an image.
'''
```

In [142]:

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
image = cv2.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-3/shape2.png')
gray=cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
gray = np.float32(gray)
dst = cv2.cornerHarris(gray,2,3,0.04)
#image[dst > 0.2*dst.max] = (0, 255, 0)
plt.imshow(dst,cmap='gray')
plt.figure(2)
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
```

Out[142]:

```
<matplotlib.image.AxesImage at 0x1391a1ae048>
```

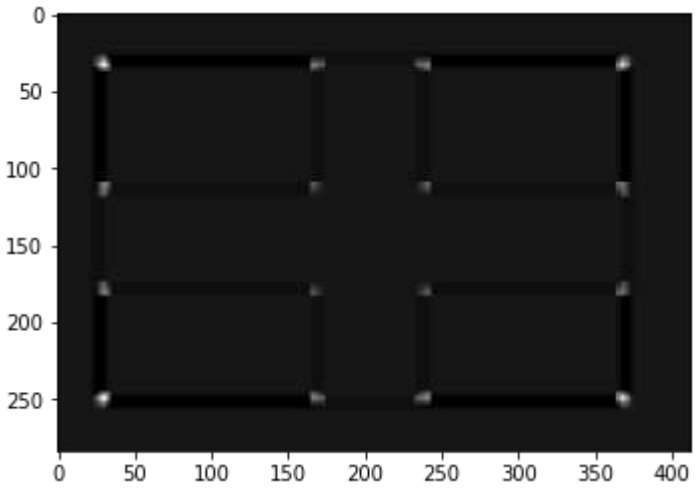


In [143]:

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
image = cv2.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-3/shape2.png')
gray=cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
gray = np.float32(gray)
dst = cv2.cornerHarris(gray,10,3,0.04)
#image[dst > 0.2*dst.max] = (0, 255, 0)
plt.imshow(dst,cmap='gray')
plt.figure(2)
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
```

Out[143]:

```
<matplotlib.image.AxesImage at 0x1391a1ff248>
```

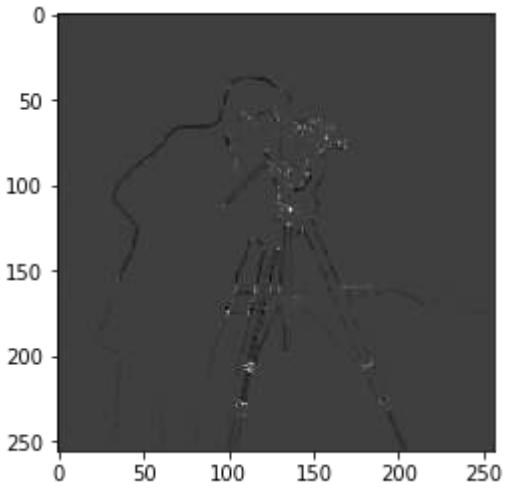


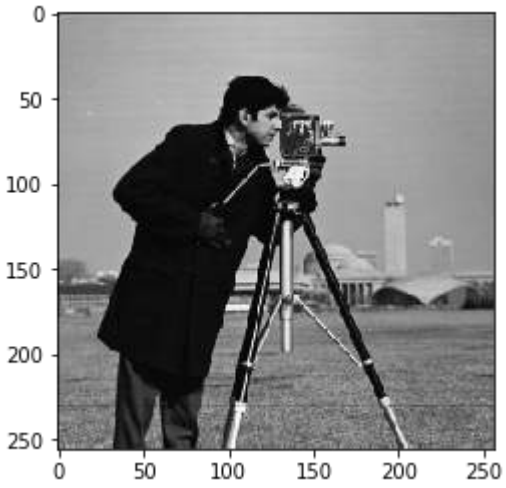
In [151]:

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
image = cv2.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-3/cameraman.png')
gray=cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
gray = np.float32(gray)
dst = cv2.cornerHarris(gray,2,3,0.04)
#image[dst > 0.2*dst.max] = (0, 255, 0)
plt.imshow(dst,cmap='gray')
plt.figure(2)
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
```

Out[151]:

```
<matplotlib.image.AxesImage at 0x1391afea708>
```



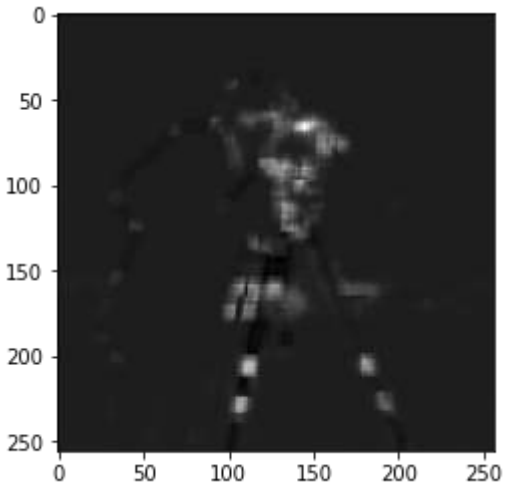


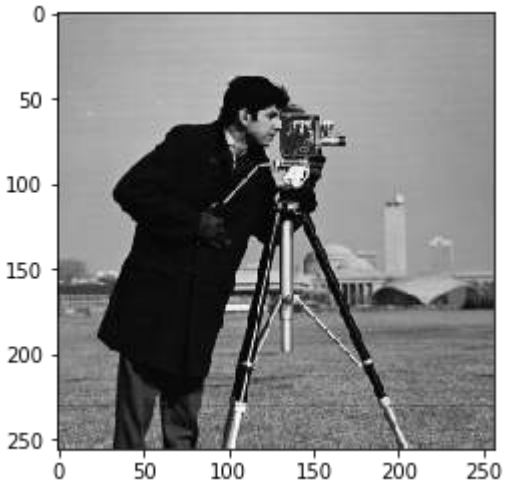
In [150]:

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
image = cv2.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-3/cameraman.png')
gray=cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
gray = np.float32(gray)
dst = cv2.cornerHarris(gray,10,3,0.04)
#image[dst > 0.2*dst.max] = (0, 255, 0)
plt.imshow(dst,cmap='gray')
plt.figure(2)
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
```

Out[150]:

```
<matplotlib.image.AxesImage at 0x1391aa907c8>
```



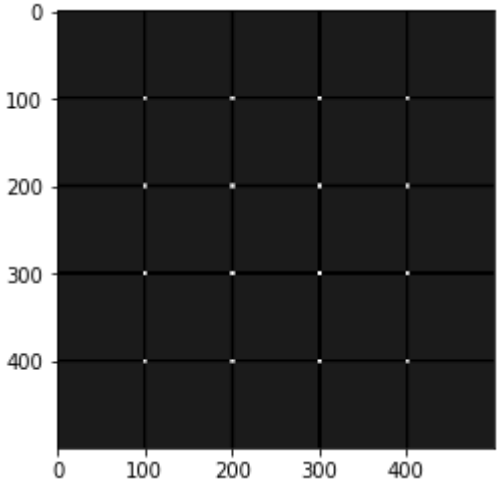


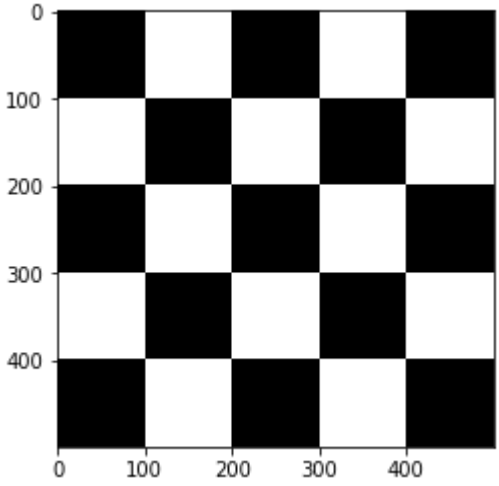
In [146]:

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
image = cv2.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-3/Checkerboard.png')
gray=cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
gray = np.float32(gray)
dst = cv2.cornerHarris(gray,4,3,0.04)
#image[dst > 0.2*dst.max] = (0, 255, 0)
plt.imshow(dst,cmap='gray')
plt.figure(2)
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
```

Out[146]:

```
<matplotlib.image.AxesImage at 0x1391a63fcc8>
```





In [ ]:

```
#INFERENCE
'''
When we upload the picture , through Harris corner detection it detects all the
edges of the image and finally the image is converted into RGB image where
only red, blue ,green colorss of the image are shown

Hence we have detected the corners of the image using Harris corner
detection.
''''''
```

In [ ]:

```
```

In [ ]:

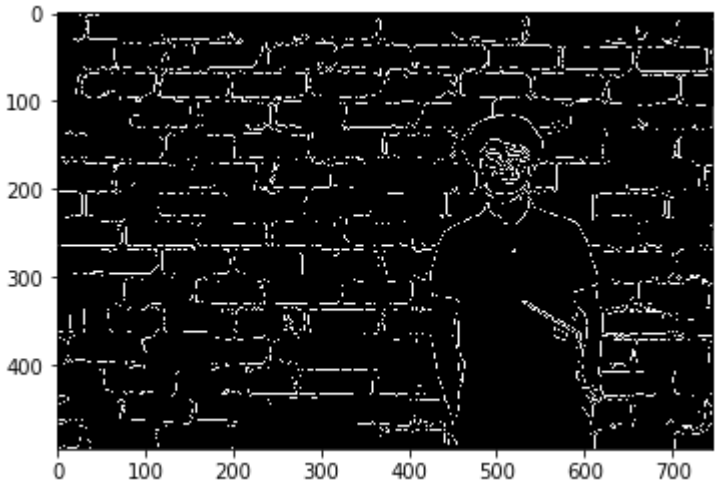
```
#HOUGH TRASNFORM - LINE DETECTION
'''
The Hough transform is a feature extraction technique used in image analysis,
computer vision, and digital image processing.[1] The purpose of the technique is to
find imperfect instances of objects within a certain class of shapes by a voting proced
ure.
This voting procedure is carried out in a parameter space, from which object
candidates are obtained as local maxima in a so-called accumulator space
that is explicitly constructed by the algorithm for computing the Hough transform.
'''
```

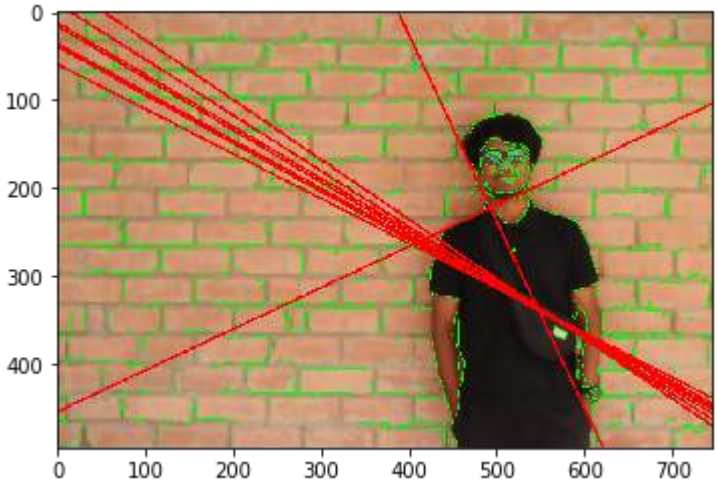
In [124]:

```
import cv2
import matplotlib.pyplot as plt
a = cv2.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-3/L93.jpeg')
th1=30
th2=3*th1
d=2
a_gauss=cv2.GaussianBlur(a, (2*d+1, 2*d+1), -1)[d:-d,d:-d]
img= cv2.cvtColor(a_gauss, cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(img,100,300,apertureSize = 3)
plt.figure(1)
plt.imshow(a_edge, cmap='gray')
a_gauss[a_edge != 0] = (0, 255, 0)

lines = cv2.HoughLines(edges,1,np.pi/100,40)

for i in range(lines.shape[0]):
    for rho,theta in lines[i]:
        a = np.cos(theta)
        b = np.sin(theta)
        x0 = a*rho
        y0 = b*rho
        x1 = int(x0 + 1000*(-b))
        y1 = int(y0 + 1000*(a))
        x2 = int(x0 - 1000*(-b))
        y2 = int(y0 - 1000*(a))

    cv2.line(a_gauss,(x1,y1),(x2,y2),(0,0,255),2)

plt.figure(2)
plt.imshow(cv2.cvtColor(a_gauss, cv2.COLOR_BGR2RGB))
```

Out[124]:

```
<matplotlib.image.AxesImage at 0x1390ab104c8>
```



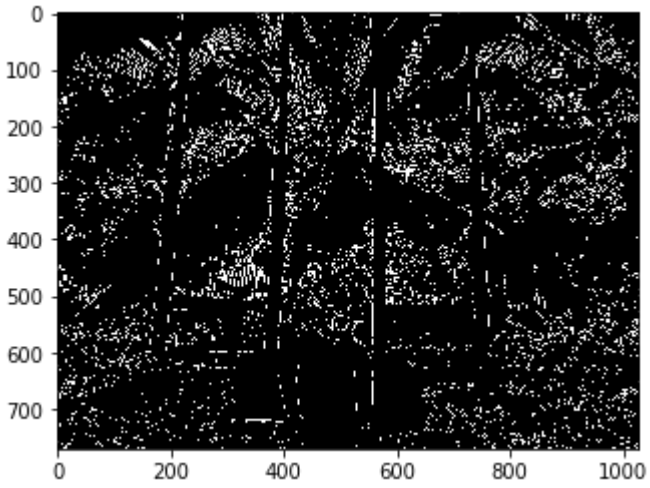


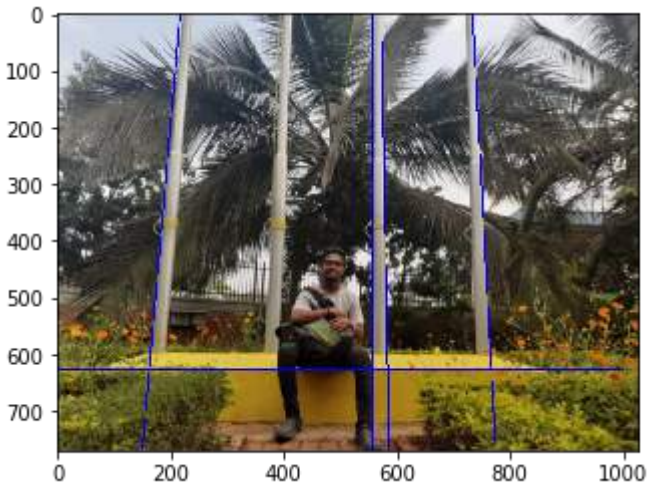
In [69]:

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

a = cv2.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-3/L91.jpeg')
d=2
a_gauss=cv2.GaussianBlur(a, (2*d+1, 2*d+1), -1)[d:-d,d:-d]
img= cv2.cvtColor(a_gauss, cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(img,150,150,apertureSize = 3)
plt.figure(1)
plt.imshow(cv2.cvtColor(edges, cv2.COLOR_BGR2RGB))

lines = cv2.HoughLines(edges,1,np.pi/180,200)

for i in range(lines.shape[0]):
    for rho,theta in lines[i]:
        a = np.cos(theta)
        b = np.sin(theta)
        x0 = a*rho
        y0 = b*rho
        x1 = int(x0 + 1000*(-b))
        y1 = int(y0 + 1000*(a))
        x2 = int(x0 - 1000*(-b))
        y2 = int(y0 - 1000*(a))

        cv2.line(a_gauss,(x1,y1),(x2,y2),(255,0,0),2)

plt.figure(2)
plt.imshow(cv2.cvtColor(a_gauss, cv2.COLOR_BGR2RGB))
```

Out[69]:

```
<matplotlib.image.AxesImage at 0x13908dc5548>
```



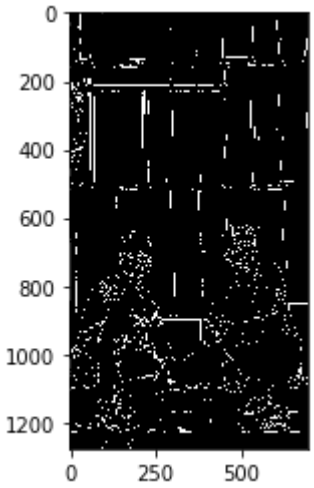


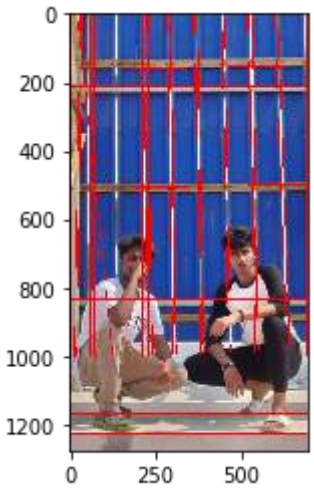
In [54]:

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

a = cv2.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-3/L96.jpeg')
d=2
a_gauss=cv2.GaussianBlur(a, (2*d+1, 2*d+1), -1)[d:-d,d:-d]
img= cv2.cvtColor(a_gauss, cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(img,50,150,apertureSize = 3)
plt.figure(1)
plt.imshow(cv2.cvtColor(edges, cv2.COLOR_BGR2RGB))

lines = cv2.HoughLines(edges,1,np.pi/180,200)

for i in range(lines.shape[0]):
    for rho,theta in lines[i]:
        a = np.cos(theta)
        b = np.sin(theta)
        x0 = a*rho
        y0 = b*rho
        x1 = int(x0 + 1000*(-b))
        y1 = int(y0 + 1000*(a))
        x2 = int(x0 - 1000*(-b))
        y2 = int(y0 - 1000*(a))

        cv2.line(a_gauss,(x1,y1),(x2,y2),(0,0,255),2)

plt.figure(2)
plt.imshow(cv2.cvtColor(a_gauss, cv2.COLOR_BGR2RGB))
```

Out[54]:

```
<matplotlib.image.AxesImage at 0x139071b5bc8>
```





In [ ]:

```
#INFERENCE
'''
As we increase the value of 'd' the image gets more blurred. This makes it
tough to detect the edges . As a result the red lines in the image gets reduced
whereas the image with 'd' value as 2 , the image is more clear and hence it
detects more straight lines as edges and there are more red lines in the output.
Hence we have applied the HOUGH transform and detected the lines
'''
```

In [ ]:

```

```

In [ ]:

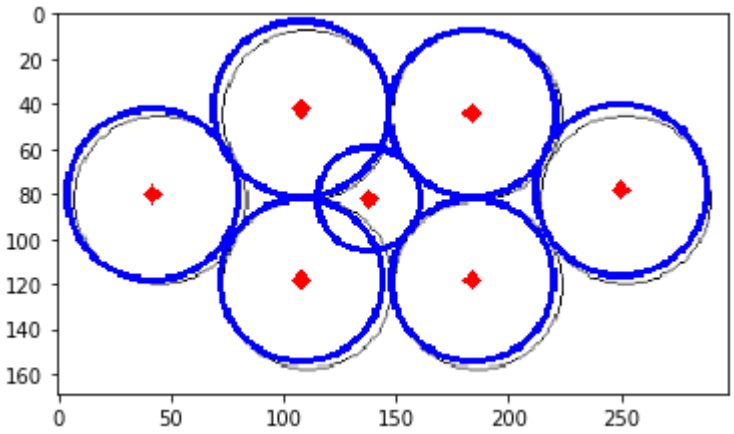
```
#HOUGH TRASNFORM - CIRCLE DETECTION
```

In [78]:

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

img = cv2.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-3/mulcircle.png')
d=2
gimg=cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
gimg = cv2.GaussianBlur(gimg, (2*d+1, 2*d+1), -1)[d:-d,d:-d]
#cimg = cv2.cvtColor(img,cv2.COLOR_GRAY2BGR)

circles = cv2.HoughCircles(gimg,cv2.HOUGH_GRADIENT,1,20,param1=50,param2=30,minRadius=0
,maxRadius=0)

circles = np.uint16(np.around(circles))
for i in circles[0,:]:
    # draw the outer circle
    cv2.circle(img,(i[0],i[1]),i[2],(255,0,0),2)
    # draw the center of the circle
    cv2.circle(img,(i[0],i[1]),2,(0,0,255),3)

plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
```

Out[78]:

```
<matplotlib.image.AxesImage at 0x1390d6979c8>
```

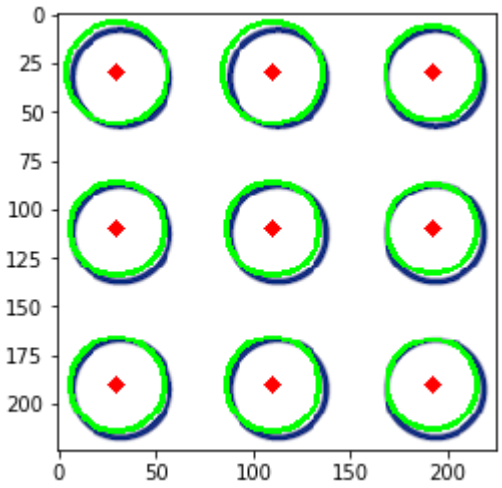


In [86]:

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

img = cv2.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-3/coins.png')
d=2
gimg=cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
gimg = cv2.GaussianBlur(gimg, (2*d+1, 2*d+1), -1)[d:-d,d:-d]
#cimg = cv2.cvtColor(img,cv2.COLOR_GRAY2BGR)

circles = cv2.HoughCircles(gimg,cv2.HOUGH_GRADIENT,1,20,param1=50,param2=30,minRadius=0
,maxRadius=0)

circles = np.uint16(np.around(circles))
for i in circles[0,:]:
    # draw the outer circle
    cv2.circle(img,(i[0],i[1]),i[2],(0,255,0),2)
    # draw the center of the circle
    cv2.circle(img,(i[0],i[1]),2,(0,0,255),3)

plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
```

Out[86]:

```
<matplotlib.image.AxesImage at 0x139125ffb08>
```

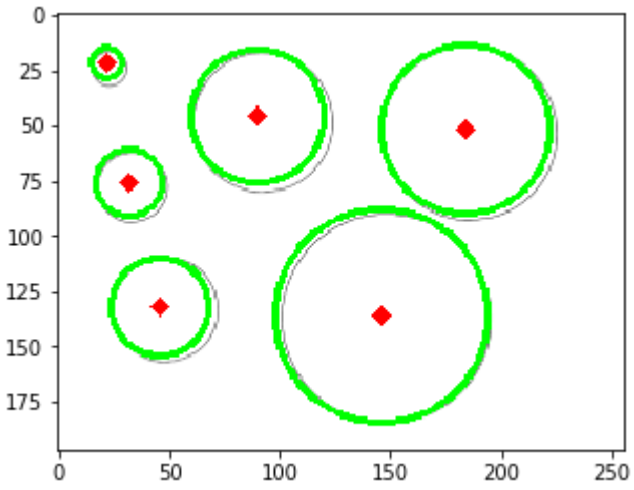


In [87]:

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

img = cv2.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-3/cir.png')
d=2
gimg=cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
gimg = cv2.GaussianBlur(gimg, (2*d+1, 2*d+1), -1)[d:-d,d:-d]
#cimg = cv2.cvtColor(img,cv2.COLOR_GRAY2BGR)

circles = cv2.HoughCircles(gimg,cv2.HOUGH_GRADIENT,1,20,param1=50,param2=30,minRadius=0
,maxRadius=0)

circles = np.uint16(np.around(circles))
for i in circles[0,:]:
    # draw the outer circle
    cv2.circle(img,(i[0],i[1]),i[2],(0,255,0),2)
    # draw the center of the circle
    cv2.circle(img,(i[0],i[1]),2,(0,0,255),3)

plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
```

Out[87]:

```
<matplotlib.image.AxesImage at 0x1391266b9c8>
```



In [ ]:

```
#INFERENCE
'''
Hough circles uses hough transform to detect the circles.
The center of the circle will be pointed by a red dot .
'''
```

In [ ]:

```

```

In [ ]:

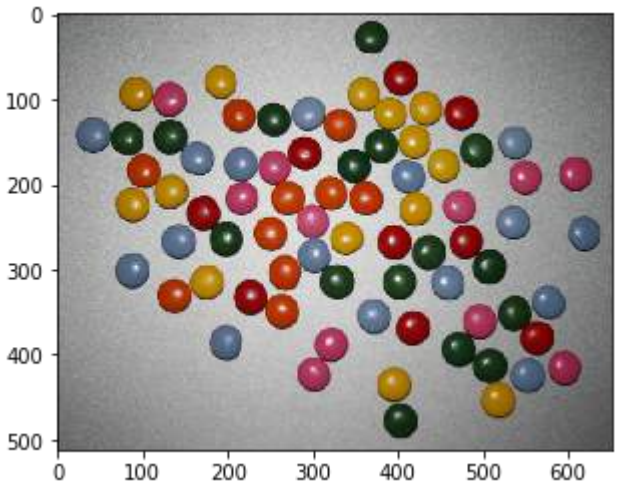
```
#k-Means Clustetring
'''
k-means clustering is a method of vector quantization, originally from
signal processing, that aims to partition n observations into k clusters
in which each observation belongs to the cluster with the nearest mean,
serving as a prototype of the cluster.
'''
#Here we use various combinations of RGB Pictures
```

In [88]:

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

img = cv2.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-3/gems.png')
img=np.float32(img)
plt.imshow(cv2.cvtColor(np.uint8(img),cv2.COLOR_BGR2RGB))
#img =cv2.cvtColor(orgimg, cv2.COLOR_BGR2GRAY)
```

Out[88]:

```
<matplotlib.image.AxesImage at 0x139126c7cc8>
```

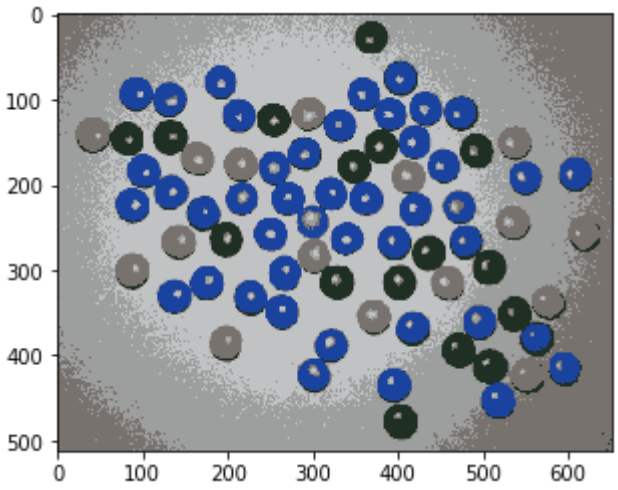


In [89]:

```
[m, n, k]=np.shape(img)

img_1d=img.reshape(m*n,k)

from sklearn.cluster import KMeans
#kmeans = KMeans(n_clusters=3, random_state=0).fit(img_1d)
kmeans = KMeans(n_clusters=5).fit(img_1d)
img2show = kmeans.cluster_centers_[kmeans.labels_]

img_cluster = img2show.reshape(m,n,k )
img_cluster=np.array(img_cluster, dtype=np.uint8)
plt.imshow(img_cluster)
```

Out[89]:

```
<matplotlib.image.AxesImage at 0x1391273e208>
```

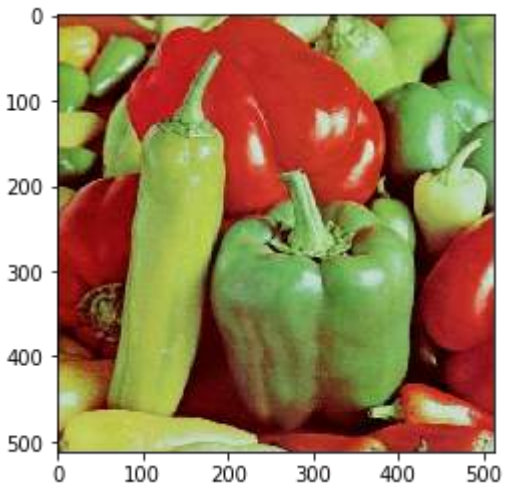


In [90]:

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

img = cv2.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-3/peppers_color.jpg')
img=np.float32(img)
plt.imshow(cv2.cvtColor(np.uint8(img),cv2.COLOR_BGR2RGB))
#img =cv2.cvtColor(orgimg, cv2.COLOR_BGR2GRAY)
```

Out[90]:

```
<matplotlib.image.AxesImage at 0x13907eeb208>
```

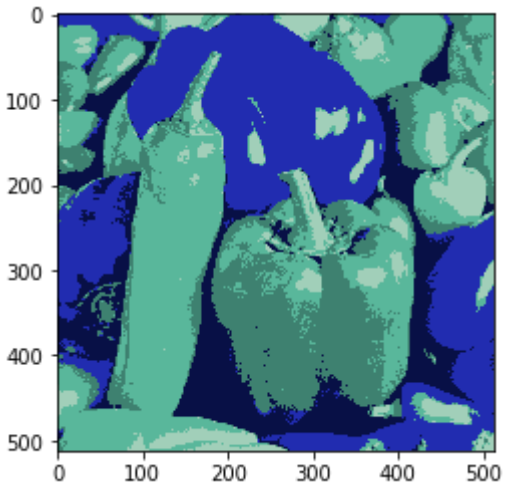


In [91]:

```
[m, n, k]=np.shape(img)

img_1d=img.reshape(m*n,k)

from sklearn.cluster import KMeans
#kmeans = KMeans(n_clusters=3, random_state=0).fit(img_1d)
kmeans = KMeans(n_clusters=5).fit(img_1d)
img2show = kmeans.cluster_centers_[kmeans.labels_]

img_cluster = img2show.reshape(m,n,k )
img_cluster=np.array(img_cluster, dtype=np.uint8)
plt.imshow(img_cluster)
```

Out[91]:

```
<matplotlib.image.AxesImage at 0x1390a80b888>
```

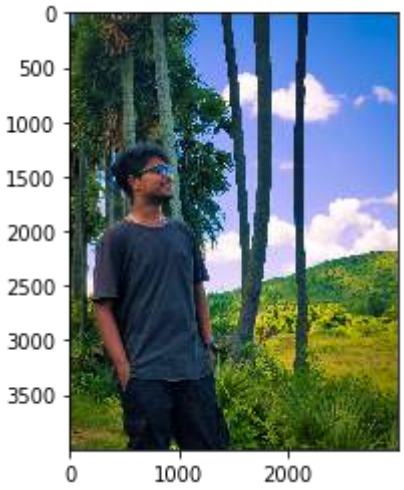


In [92]:

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

img = cv2.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-3/RGB1.jpg')
img=np.float32(img)
plt.imshow(cv2.cvtColor(np.uint8(img),cv2.COLOR_BGR2RGB))
#img =cv2.cvtColor(orgimg, cv2.COLOR_BGR2GRAY)
```

Out[92]:

```
<matplotlib.image.AxesImage at 0x1390a86ad48>
```

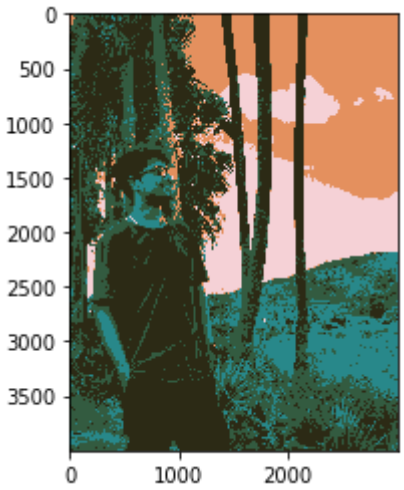


In [93]:

```
[m, n, k]=np.shape(img)

img_1d=img.reshape(m*n,k)

from sklearn.cluster import KMeans
#kmeans = KMeans(n_clusters=3, random_state=0).fit(img_1d)
kmeans = KMeans(n_clusters=5).fit(img_1d)
img2show = kmeans.cluster_centers_[kmeans.labels_]

img_cluster = img2show.reshape(m,n,k )
img_cluster=np.array(img_cluster, dtype=np.uint8)
plt.imshow(img_cluster)
```

Out[93]:

```
<matplotlib.image.AxesImage at 0x1390a8d5388>
```

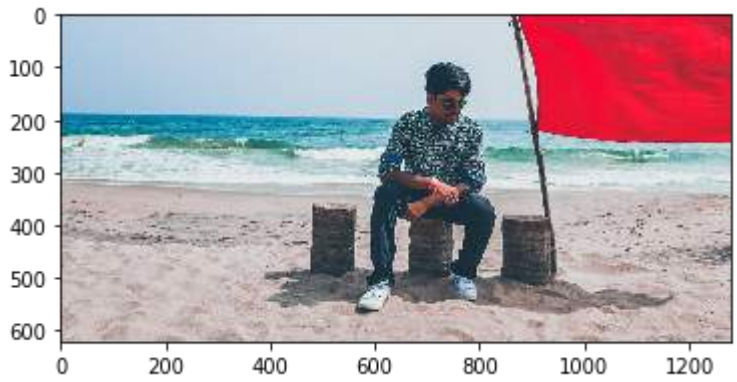


In [94]:

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

img = cv2.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-3/RGB2.jpg')
img=np.float32(img)
plt.imshow(cv2.cvtColor(np.uint8(img),cv2.COLOR_BGR2RGB))
#img =cv2.cvtColor(orgimg, cv2.COLOR_BGR2GRAY)
```

Out[94]:

```
<matplotlib.image.AxesImage at 0x1390a934a88>
```

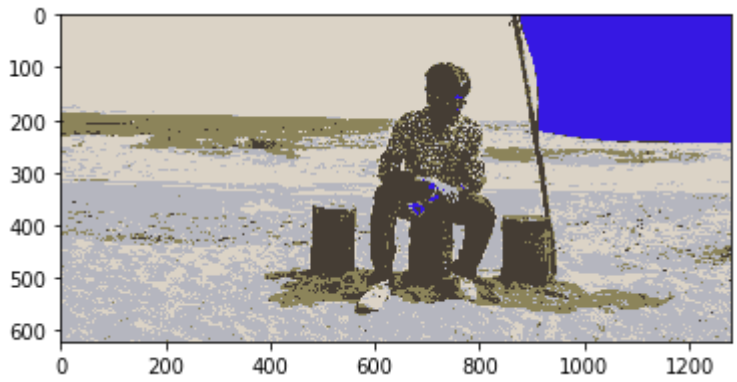


In [95]:

```
[m, n, k]=np.shape(img)

img_1d=img.reshape(m*n,k)

from sklearn.cluster import KMeans
#kmeans = KMeans(n_clusters=3, random_state=0).fit(img_1d)
kmeans = KMeans(n_clusters=5).fit(img_1d)
img2show = kmeans.cluster_centers_[kmeans.labels_]

img_cluster = img2show.reshape(m,n,k )
img_cluster=np.array(img_cluster, dtype=np.uint8)
plt.imshow(img_cluster)
```

Out[95]:

```
<matplotlib.image.AxesImage at 0x1390a9a4308>
```



In [96]:

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

img = cv2.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-3/RGB3.jpg')
img=np.float32(img)
plt.imshow(cv2.cvtColor(np.uint8(img),cv2.COLOR_BGR2RGB))
#img =cv2.cvtColor(orgimg, cv2.COLOR_BGR2GRAY)
```

Out[96]:

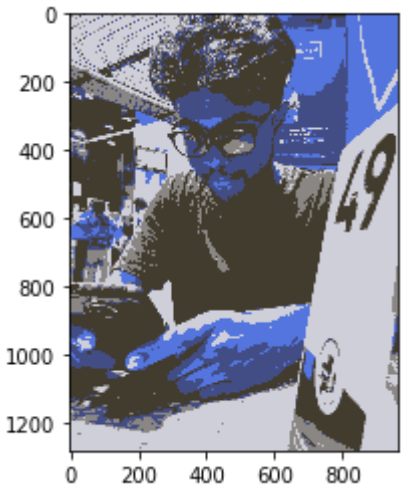
```
<matplotlib.image.AxesImage at 0x1390aa08e88>
```

In [97]:

```
[m, n, k]=np.shape(img)

img_1d=img.reshape(m*n,k)

from sklearn.cluster import KMeans
#kmeans = KMeans(n_clusters=3, random_state=0).fit(img_1d)
kmeans = KMeans(n_clusters=5).fit(img_1d)
img2show = kmeans.cluster_centers_[kmeans.labels_]

img_cluster = img2show.reshape(m,n,k )
img_cluster=np.array(img_cluster, dtype=np.uint8)
plt.imshow(img_cluster)
```

Out[97]:

```
<matplotlib.image.AxesImage at 0x139127c3408>
```



In [98]:

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

img = cv2.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-3/RGBb.jpg')
img=np.float32(img)
plt.imshow(cv2.cvtColor(np.uint8(img),cv2.COLOR_BGR2RGB))
#img =cv2.cvtColor(orgimg, cv2.COLOR_BGR2GRAY)
```

Out[98]:

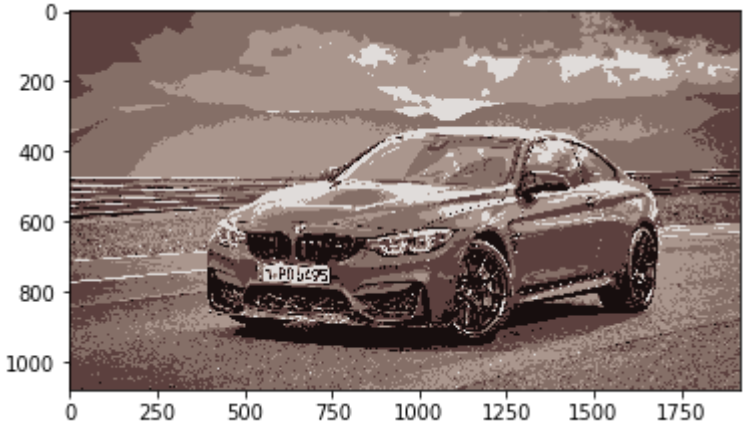
```
<matplotlib.image.AxesImage at 0x139135345c8>
```

In [99]:

```
[m, n, k]=np.shape(img)

img_1d=img.reshape(m*n,k)

from sklearn.cluster import KMeans
#kmeans = KMeans(n_clusters=3, random_state=0).fit(img_1d)
kmeans = KMeans(n_clusters=5).fit(img_1d)
img2show = kmeans.cluster_centers_[kmeans.labels_]

img_cluster = img2show.reshape(m,n,k )
img_cluster=np.array(img_cluster, dtype=np.uint8)
plt.imshow(img_cluster)
```

Out[99]:

```
<matplotlib.image.AxesImage at 0x13913599f88>
```

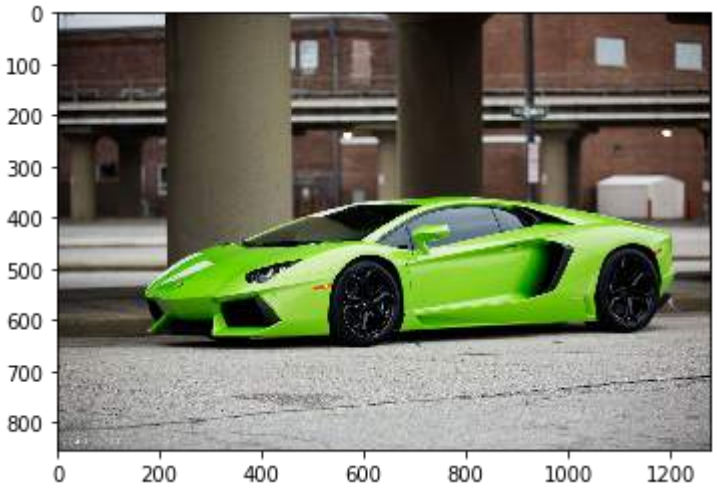


In [100]:

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

img = cv2.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-3/RGBg.jpg')
img=np.float32(img)
plt.imshow(cv2.cvtColor(np.uint8(img),cv2.COLOR_BGR2RGB))
#img =cv2.cvtColor(orgimg, cv2.COLOR_BGR2GRAY)
```

Out[100]:

```
<matplotlib.image.AxesImage at 0x139135f9b88>
```

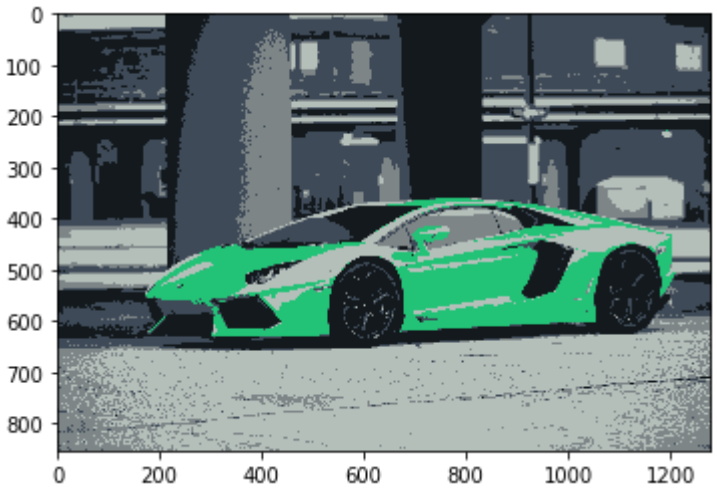


In [101]:

```
[m, n, k]=np.shape(img)

img_1d=img.reshape(m*n,k)

from sklearn.cluster import KMeans
#kmeans = KMeans(n_clusters=3, random_state=0).fit(img_1d)
kmeans = KMeans(n_clusters=5).fit(img_1d)
img2show = kmeans.cluster_centers_[kmeans.labels_]

img_cluster = img2show.reshape(m,n,k )
img_cluster=np.array(img_cluster, dtype=np.uint8)
plt.imshow(img_cluster)
```

Out[101]:

```
<matplotlib.image.AxesImage at 0x139137ad048>
```



In [102]:

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

img = cv2.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-3/RGBr.jpg')
img=np.float32(img)
plt.imshow(cv2.cvtColor(np.uint8(img),cv2.COLOR_BGR2RGB))
#img =cv2.cvtColor(orgimg, cv2.COLOR_BGR2GRAY)
```

Out[102]:

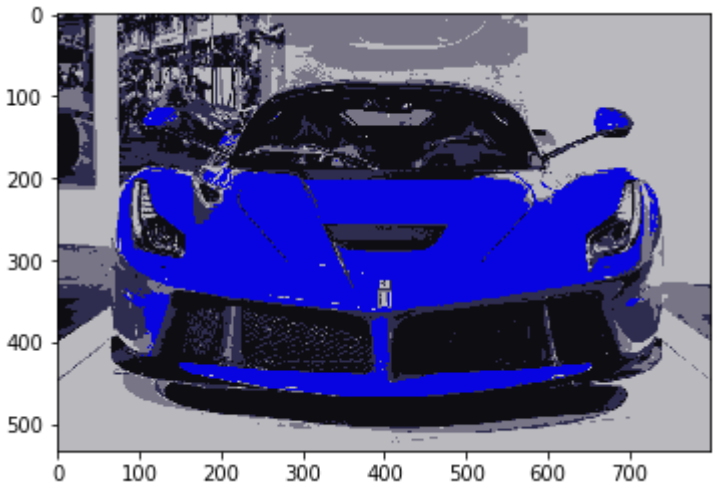
```
<matplotlib.image.AxesImage at 0x1391381ed48>
```

In [103]:

```
[m, n, k]=np.shape(img)

img_1d=img.reshape(m*n,k)

from sklearn.cluster import KMeans
#kmeans = KMeans(n_clusters=3, random_state=0).fit(img_1d)
kmeans = KMeans(n_clusters=5).fit(img_1d)
img2show = kmeans.cluster_centers_[kmeans.labels_]

img_cluster = img2show.reshape(m,n,k )
img_cluster=np.array(img_cluster, dtype=np.uint8)
plt.imshow(img_cluster)
```

Out[103]:

```
<matplotlib.image.AxesImage at 0x1391388e388>
```

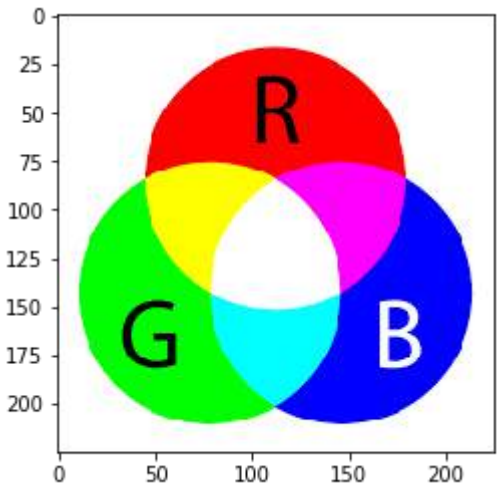


In [104]:

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

img = cv2.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-3/RGBrgb.jpg')
img=np.float32(img)
plt.imshow(cv2.cvtColor(np.uint8(img),cv2.COLOR_BGR2RGB))
#img =cv2.cvtColor(orgimg, cv2.COLOR_BGR2GRAY)
```

Out[104]:

```
<matplotlib.image.AxesImage at 0x13915880f88>
```

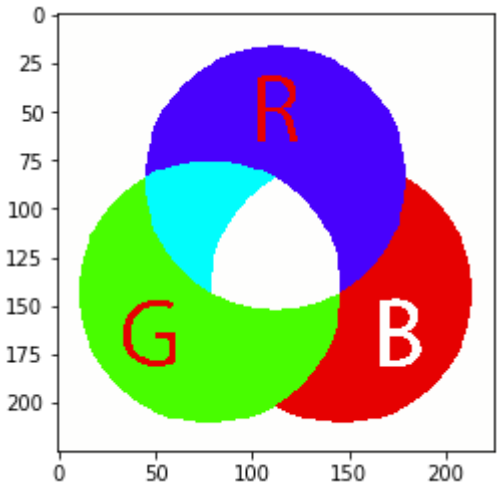


In [105]:

```
[m, n, k]=np.shape(img)

img_1d=img.reshape(m*n,k)

from sklearn.cluster import KMeans
#kmeans = KMeans(n_clusters=3, random_state=0).fit(img_1d)
kmeans = KMeans(n_clusters=5).fit(img_1d)
img2show = kmeans.cluster_centers_[kmeans.labels_]

img_cluster = img2show.reshape(m,n,k )
img_cluster=np.array(img_cluster, dtype=np.uint8)
plt.imshow(img_cluster)
```

Out[105]:

```
<matplotlib.image.AxesImage at 0x139162c92c8>
```

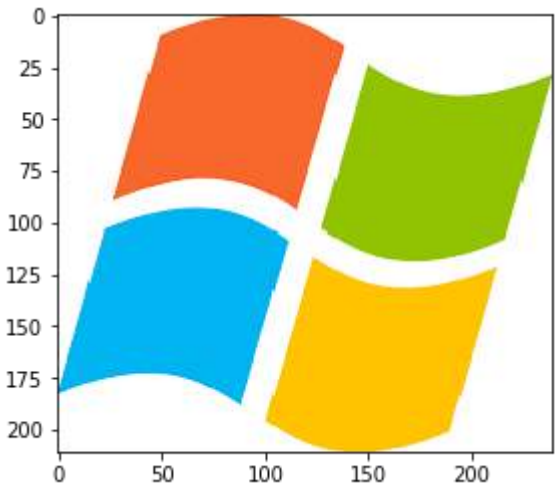


In [106]:

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

img = cv2.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-3/windows.png')
img=np.float32(img)
plt.imshow(cv2.cvtColor(np.uint8(img),cv2.COLOR_BGR2RGB))
#img =cv2.cvtColor(orgimg, cv2.COLOR_BGR2GRAY)
```

Out[106]:

```
<matplotlib.image.AxesImage at 0x13916332b48>
```

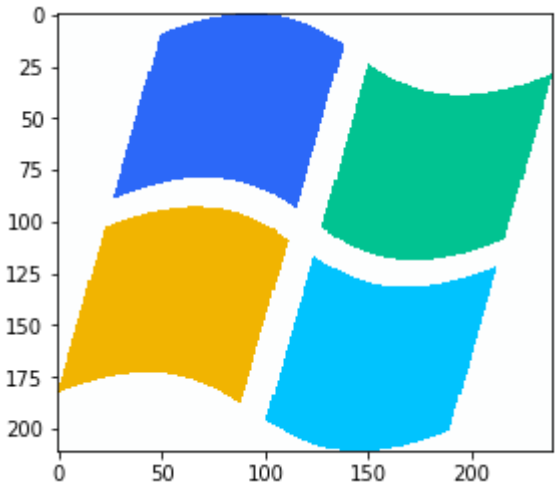


In [107]:

```
[m, n, k]=np.shape(img)

img_1d=img.reshape(m*n,k)

from sklearn.cluster import KMeans
#kmeans = KMeans(n_clusters=3, random_state=0).fit(img_1d)
kmeans = KMeans(n_clusters=5).fit(img_1d)
img2show = kmeans.cluster_centers_[kmeans.labels_]

img_cluster = img2show.reshape(m,n,k )
img_cluster=np.array(img_cluster, dtype=np.uint8)
plt.imshow(img_cluster)
```

Out[107]:

```
<matplotlib.image.AxesImage at 0x139163a2488>
```



In [ ]:

```
#INFERENCE
'''
When we upload images we note that only the red , blue ,green
color portions of the image are higlighted .
Hence we have performed k-means clustering on images and
observed the output.
'''
```

In [ ]:

Python for Vision Techniques Report

by

Andrew John 18BEC1278

A laboratory report submitted to

Dr. Sathiya Narayan & Dr. Annis Fathima

SCHOOL OF ELECTRONICS ENGINEERING

in complete fulfilment of the requirements for the value added course of

VAC: Python for Vision Techniques

in

B.Tech. Electronics and Communication Engineering

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

VIT CHENNAI

Vandalur - Kelambakkam Road

Chennai – 600127

April 2021

CONTENT

Final Task

18BEC1278_Final-Task_VAP_PVT

In [ ]:

```
#VAP - PVT, Final Task
#10/04/21 - Saturday
#18BEC1278 - Andrew John
```

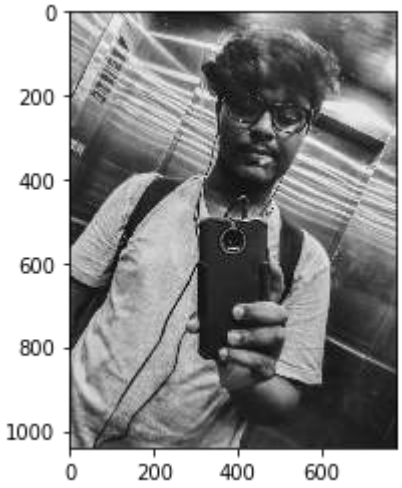
Task - 1:

1.Capture an image of you in an indoor environment and perform the following subtasks:

(i)Plot the Histogram of the Grayscale version of the image; and
(ii)Perform any one Smoothing technique (Mean Filtering, Gaussian Filtering or Median Filtering)

In [31]:

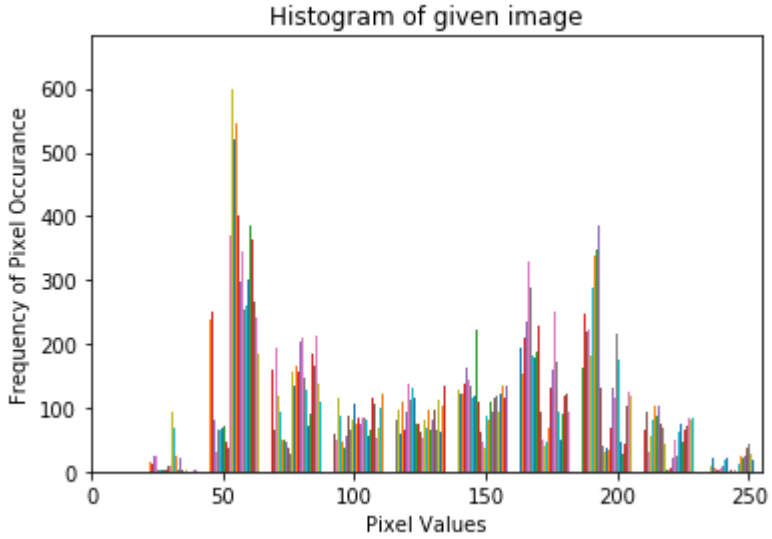
```
# Studying the Histogram with Original Picture
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-5/Final Images/Task-1/1.2.jpeg')
def rgb2gray(rgb):
    r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
    gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
    return gray
img_gray=rgb2gray(img)
plt.figure(1)
plt.imshow(img_gray,cmap='gray')
plt.figure(2)
plt.show()
plt.hist(img_gray)
plt.title("Histogram of given image")
plt.xlabel("Pixel Values")
plt.ylabel("Frequency of Pixel Occurance")
plt.xlim([0, 255])
```




```
<Figure size 432x288 with 0 Axes>
```

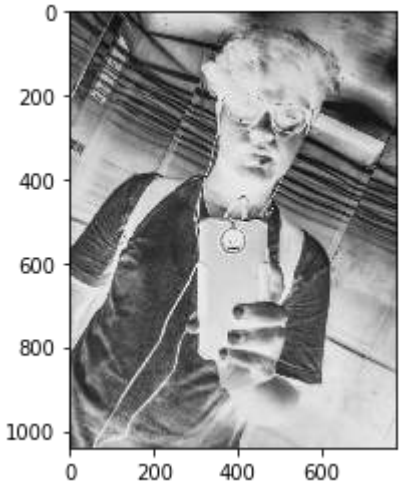
Out[31]:

```
(0, 255)
```



In [33]:

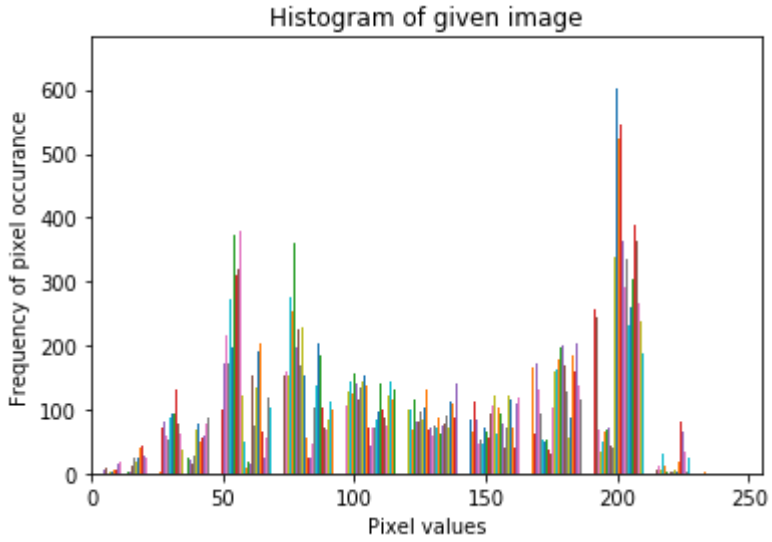
```
# Studying the Histogram with Negative Picture
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-5/Final Images/Task-1/1.2.jpeg')
def rgb2gray(rgb):
    r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
    gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
    return gray
img_gray=rgb2gray(img)
img_neg=255-img_gray
plt.figure(1)
plt.imshow(img_neg,cmap='gray')
plt.figure(2)
plt.show()
plt.hist(img_neg)
plt.title("Histogram of given image")
plt.xlabel("Pixel values")
plt.ylabel("Frequency of pixel occurance")
plt.xlim([0, 255])
```




```
<Figure size 432x288 with 0 Axes>
```

Out[33]:

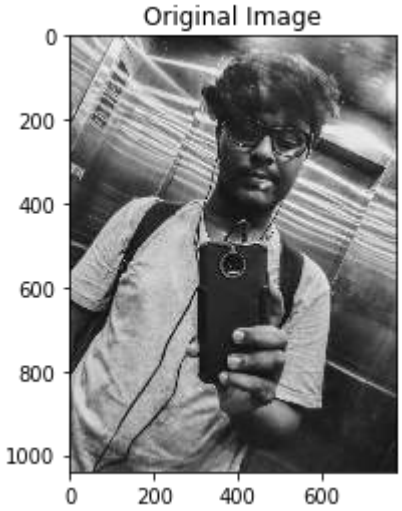
```
(0, 255)
```

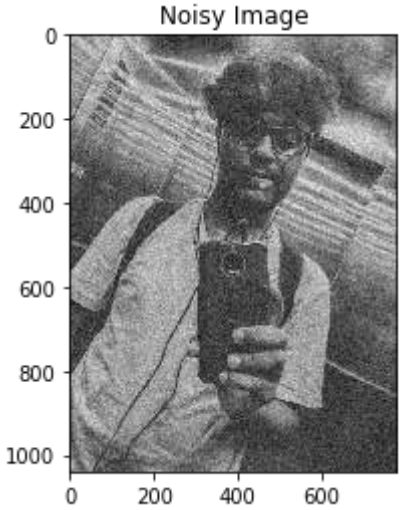


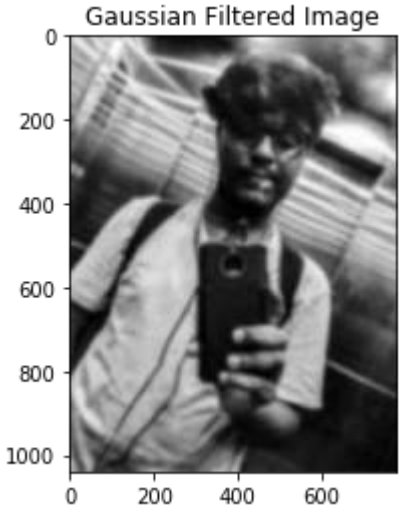
In [39]:

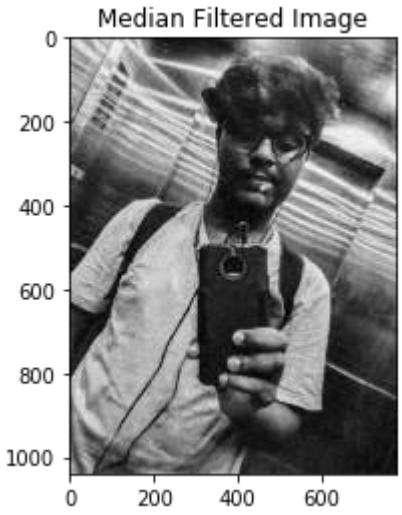
```
# Gaussian and Median Filters
# Reduced Noise
# Sigma = 5
# Size = 7
# Noise = 50
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from scipy import ndimage # Multi-dimensional image processing
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-5/Final Images/Task-1/1.2.jpeg')
def rgb2gray(rgb):
    r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
    gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
    return gray
img=rgb2gray(img)
img_noise = img + 20*np.random.randn(*img.shape)
blurred_img1=ndimage.gaussian_filter(img_noise, sigma=5) # Multidimensional Gaussian filter
blurred_img2=ndimage.median_filter(img_noise,7) # Calculate a multidimensional median filter
plt.figure(1)
plt.imshow(img,cmap='gray')
plt.title("Original Image")
plt.figure(2)
plt.imshow(img_noise,cmap='gray')
plt.title("Noisy Image")
plt.figure(3)
plt.imshow(blurred_img1,cmap='gray')
plt.title("Gaussian Filtered Image")
plt.figure(4)
plt.imshow(blurred_img2,cmap='gray')
plt.title("Median Filtered Image")
err1=img - blurred_img1
err2=img - blurred_img2
sqerr1 = np.power(err1,2)
sqerr2 = np.power(err2,2)
print(sqerr1.mean())
print(sqerr2.mean())
```

```
236.93222244890563
70.19879266090327
```





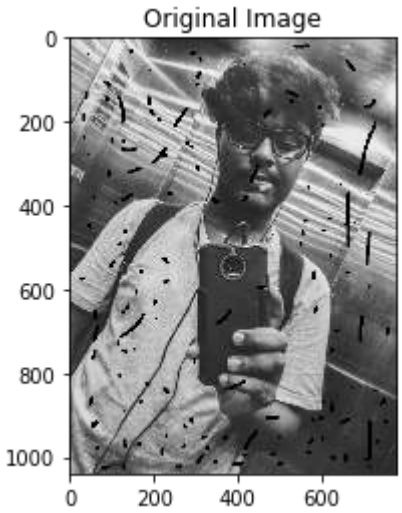


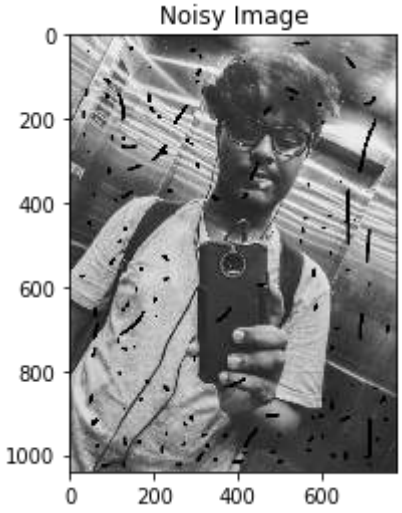

Median Filtered Image
0
200
400
600
800
1000
0
200
400
600

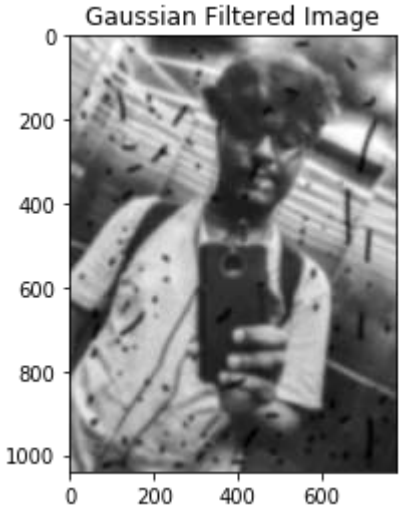
In [51]:

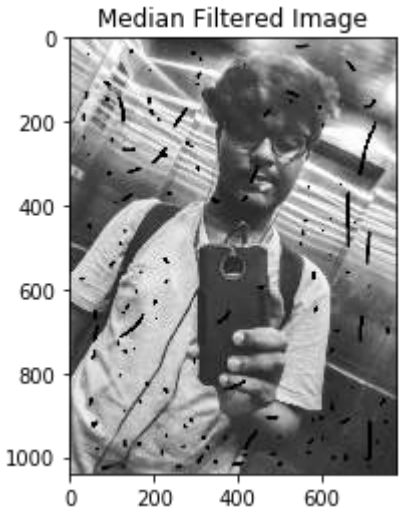
```
#Gaussian and Median Filters
#External Noise
#Sigma = 5
#Size = 7
#Noise = 0
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from scipy import ndimage # Multi-dimensional image processing
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-5/Final Images/Task-1/1.2_noise.png')
def rgb2gray(rgb):
    r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
    gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
    return gray
img=rgb2gray(img)
img_noise = img #+ 20*np.random.randn(*img.shape)
blurred_img1=ndimage.gaussian_filter(img_noise, sigma=5) # Multidimensional Gaussian filter
blurred_img2=ndimage.median_filter(img_noise,7) # Calculate a multidimensional median filter
plt.figure(1)
plt.imshow(img,cmap='gray')
plt.title("Original Image")
plt.figure(2)
plt.imshow(img_noise,cmap='gray')
plt.title("Noisy Image")
plt.figure(3)
plt.imshow(blurred_img1,cmap='gray')
plt.title("Gaussian Filtered Image")
plt.figure(4)
plt.imshow(blurred_img2,cmap='gray')
plt.title("Median Filtered Image")
err1=img - blurred_img1
err2=img - blurred_img2
sqerr1 = np.power(err1,2)
sqerr2 = np.power(err2,2)
print(sqerr1.mean())
print(sqerr2.mean())
```

```
0.00660491
0.0011490096
```









→ Inference:

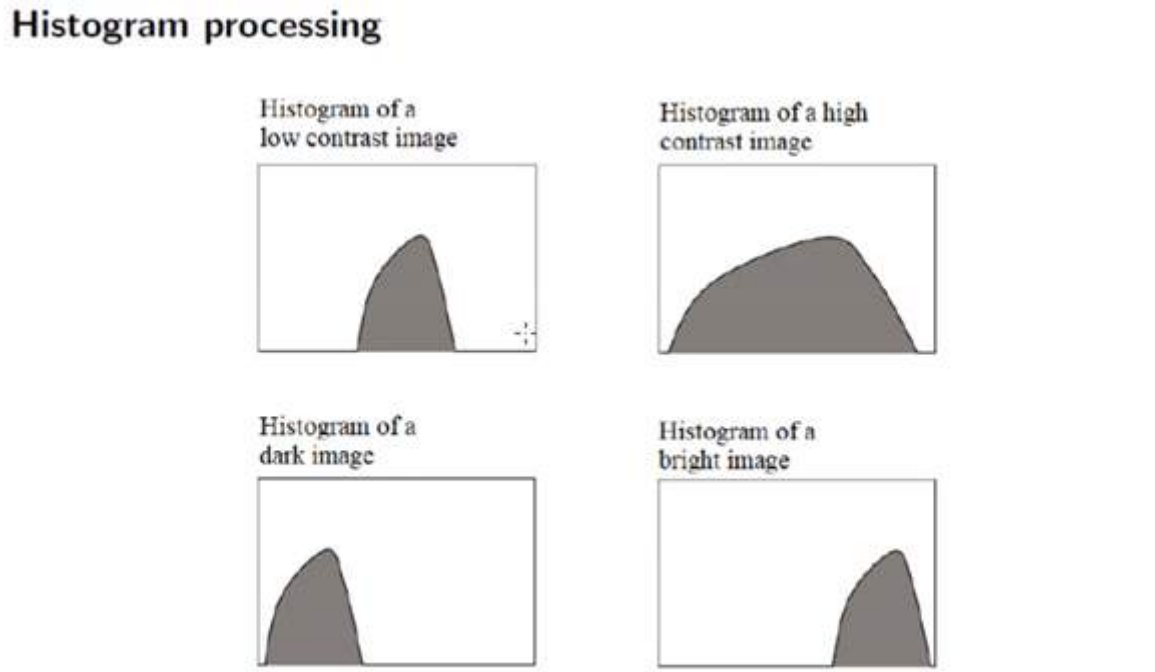


Figure 12: Image contrast and brightness assessment from image histogram.

1. In this experiment we have taken a image captured in indoor environment.
2. We have plotted the grayscale version of the image and then plotted its corresponding histogram.
3. We note that for the original image the histogram lies in the left plane whereas while plotting for negative image the histogram shift towards its right.
4. We have studied and seen that the locus of the graphs is almost similar to that of Ideal case.
5. We can observe that as we increase the sigma value the picture gets more and more blurred.
6. We can observe that as we increase the median size, they pictures gets smoothed but along with that we are starting to observe a lose in the important details of the image.
7. Caclucaltion of mean square error is helpful while denoising. The filtered image is more clear with lesser sigma value.

In [ ]:

Task - 2:

2.Create an image containing your registration number (similar to how we created images with names) and include some shapes in it (e.g. lines, rectangles or circles). Apply

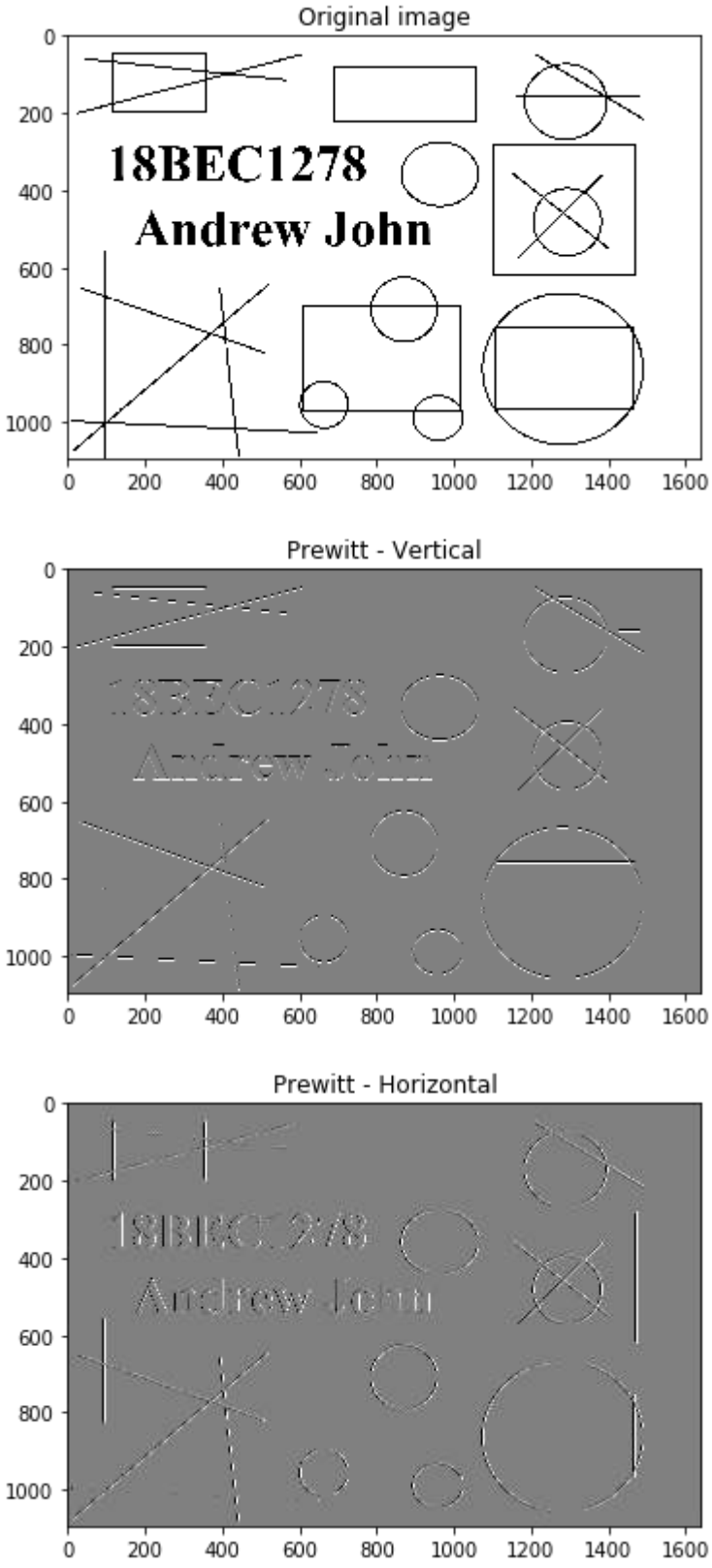
```
(i)Sobel and Prewitt operators
(ii)Line detection
(iii)Circle detection
(iv)Boundary extraction
```

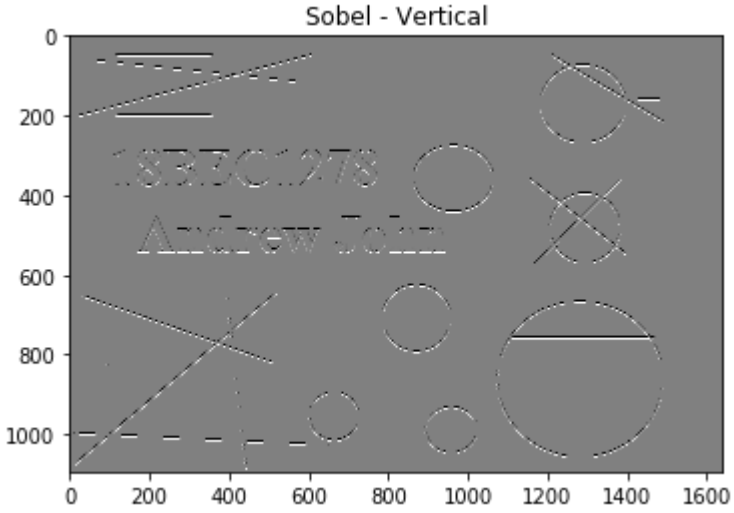
In [9]:

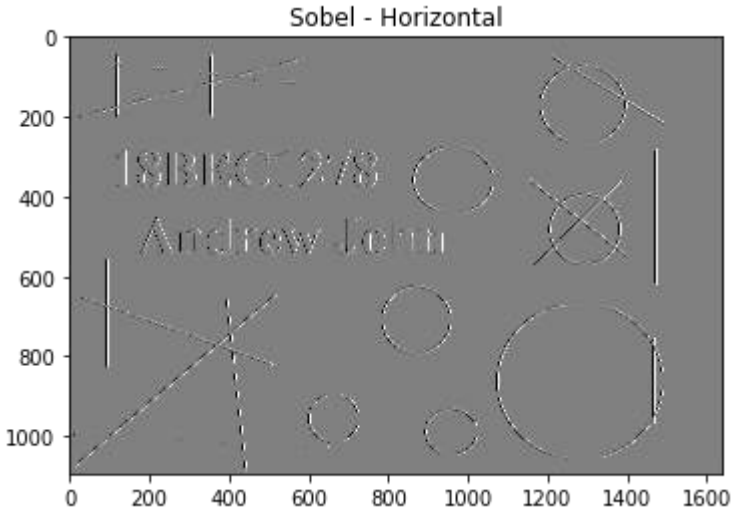
```
#SOBEL and PREWITT
#No External Noise
#Treating as GreyScale Image
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from scipy import ndimage # Multi-dimensional image processing
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-5/Final Images/Task-2/2.1.pn
g')
def rgb2gray(rgb):
    r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
    gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
    return gray
img=rgb2gray(img)
sharpened_img1=ndimage.prewitt(img,axis=0)  # Prewitt - Vertical
sharpened_img2=ndimage.prewitt(img, axis=1) # Prewitt - Horizontal
sharpened_img3=ndimage.sobel(img,axis=0)    # Sobel - Vertical
sharpened_img4=ndimage.sobel(img,axis=1)    # Sobel - Horizontal
plt.figure(1)
plt.imshow(img,cmap='gray')
plt.title("Original image")
plt.figure(2)
plt.imshow(sharpened_img1,cmap='gray')
plt.title("Prewitt - Vertical")
plt.figure(3)
plt.imshow(sharpened_img2,cmap='gray')
plt.title("Prewitt - Horizontal")
plt.figure(4)
plt.imshow(sharpened_img3,cmap='gray')
plt.title("Sobel - Vertical")
plt.figure(5)
plt.imshow(sharpened_img4,cmap='gray')
plt.title("Sobel - Horizontal")
```

Out[9]:

```
Text(0.5, 1.0, 'Sobel - Horizontal')
```




Sobel - Vertical
0
200
400
600
800
1000
0
200
400
600
800
1000
1200
1400
1600

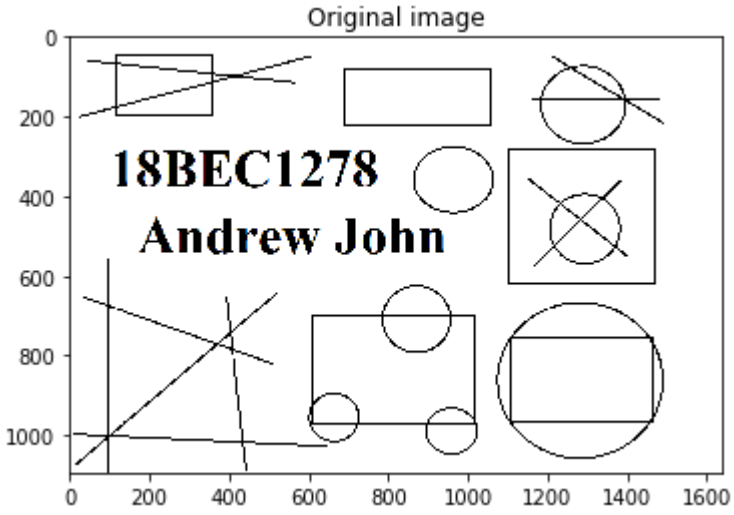

Sobel - Horizontal
0
200
400
600
800
1000
0
200
400
600
800
1000
1200
1400
1600

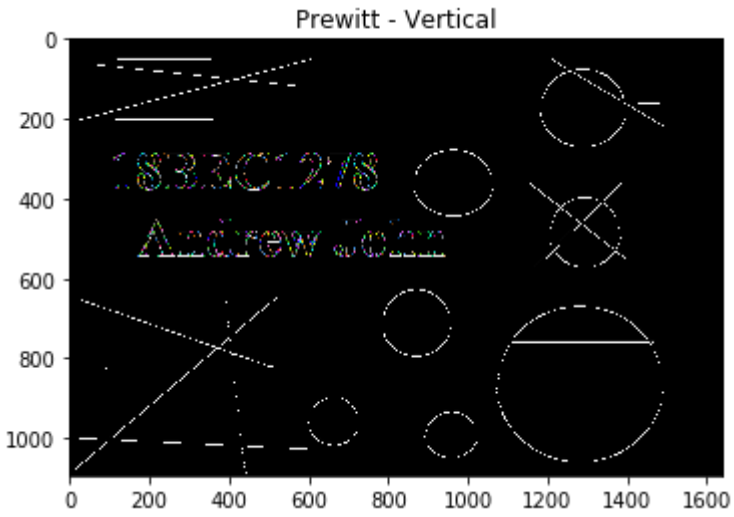
In [14]:

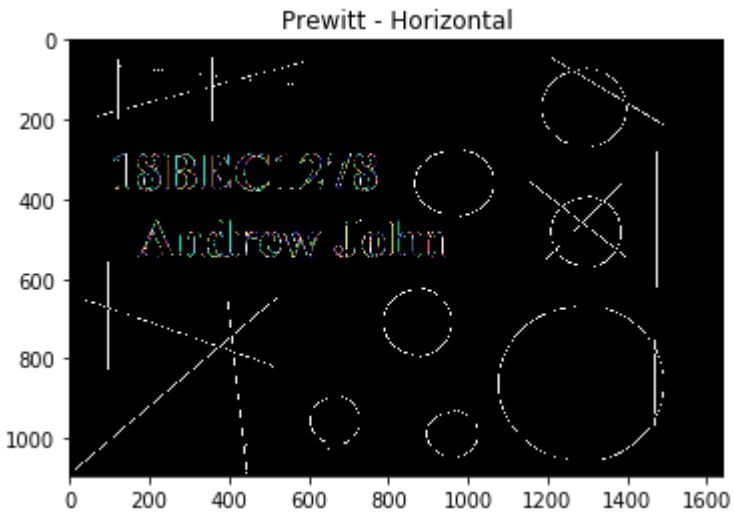
```
#SOBEL and PREWITT
#No External Noise
#Treating as B&W Image
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from scipy import ndimage # Multi-dimensional image processing
img=mpimg.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-5/Final Images/Task-2/2.1.pn
g',0)
'''def rgb2gray(rgb):
    r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
    gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
    return gray
img=rgb2gray(img)'''
sharpened_img1=ndimage.prewitt(img,axis=0)  # Prewitt - Vertical
sharpened_img2=ndimage.prewitt(img, axis=1) # Prewitt - Horizontal
sharpened_img3=ndimage.sobel(img,axis=0)    # Sobel - Vertical
sharpened_img4=ndimage.sobel(img,axis=1)    # Sobel - Horizontal
plt.figure(1)
plt.imshow(img,cmap='gray')
plt.title("Original image")
plt.figure(2)
plt.imshow(sharpened_img1,cmap='gray')
plt.title("Prewitt - Vertical")
plt.figure(3)
plt.imshow(sharpened_img2,cmap='gray')
plt.title("Prewitt - Horizontal")
plt.figure(4)
plt.imshow(sharpened_img3,cmap='gray')
plt.title("Sobel - Vertical")
plt.figure(5)
plt.imshow(sharpened_img4,cmap='gray')
plt.title("Sobel - Horizontal")
```

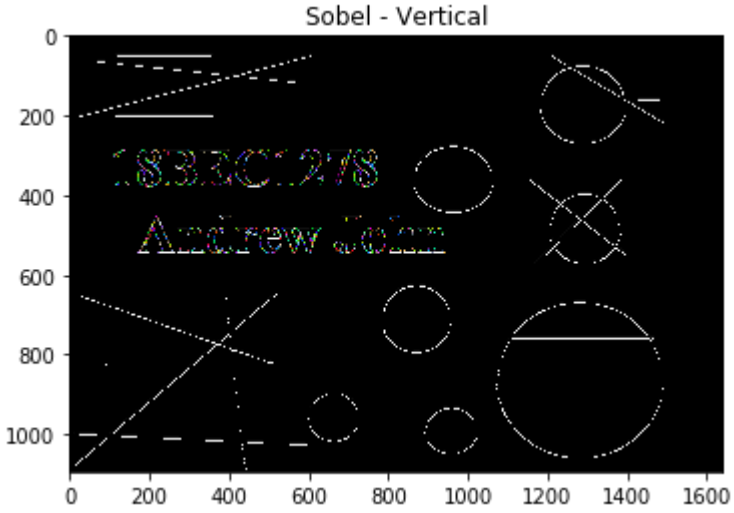
Out[14]:

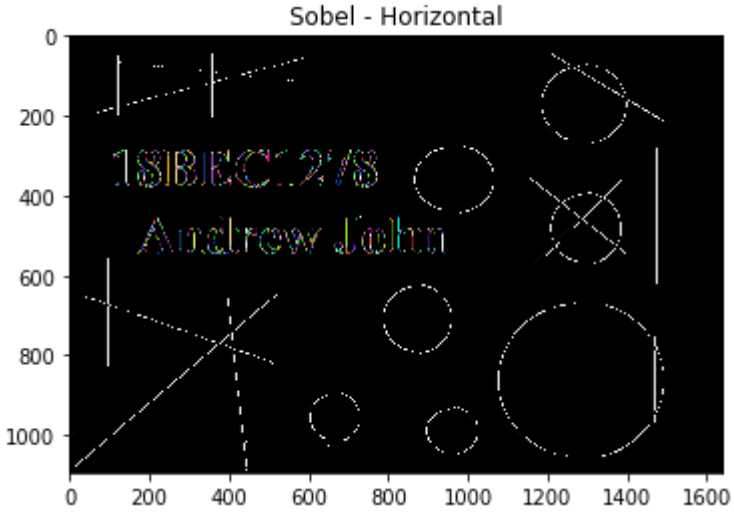
```
Text(0.5, 1.0, 'Sobel - Horizontal')
```








Sobel - Vertical
0
200
400
600
800
1000
0
200
400
600
800
1000
1200
1400
1600

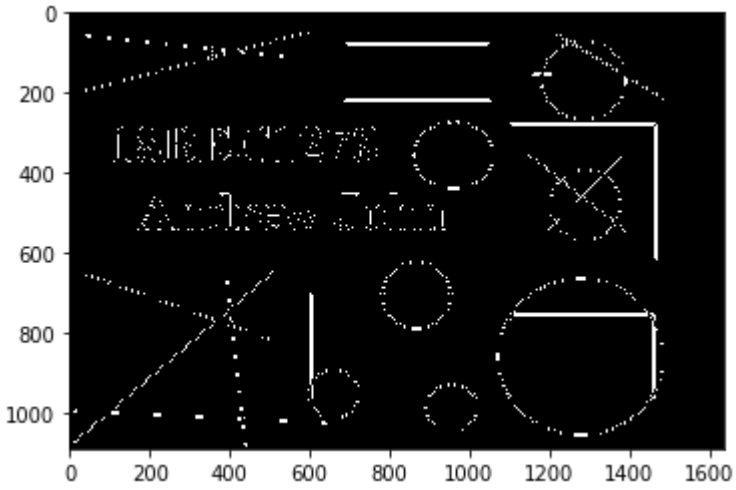

Sobel - Horizontal
0
200
400
600
800
1000
0
200
400
600
800
1000
1200
1400
1600

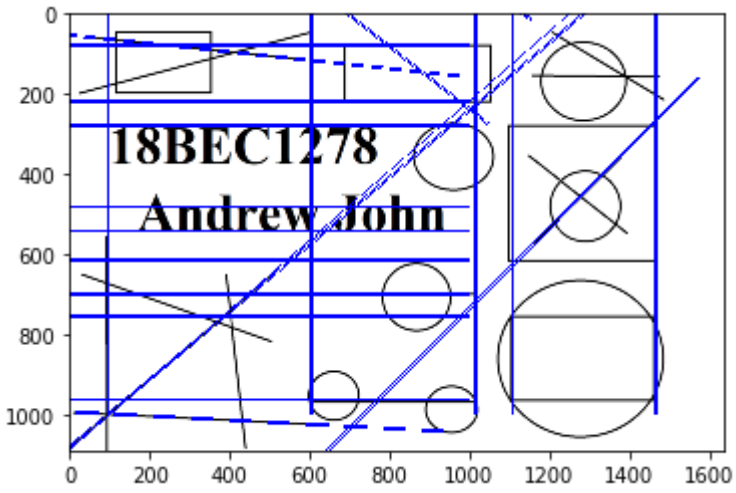
In [11]:

```
#HOUGH TRASNFORM - LINE DETECTION
import cv2
import numpy as np
import matplotlib.pyplot as plt

a = cv2.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-5/Final Images/Task-2/2.1.png')
d=2
a_gauss=cv2.GaussianBlur(a, (2*d+1, 2*d+1), -1)[d:-d,d:-d]
img= cv2.cvtColor(a_gauss, cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(img,150,150,apertureSize = 3)
plt.figure(1)
plt.imshow(cv2.cvtColor(edges, cv2.COLOR_BGR2RGB))

lines = cv2.HoughLines(edges,1,np.pi/180,200)

for i in range(lines.shape[0]):
    for rho,theta in lines[i]:
        a = np.cos(theta)
        b = np.sin(theta)
        x0 = a*rho
        y0 = b*rho
        x1 = int(x0 + 1000*(-b))
        y1 = int(y0 + 1000*(a))
        x2 = int(x0 - 1000*(-b))
        y2 = int(y0 - 1000*(a))

        cv2.line(a_gauss,(x1,y1),(x2,y2),(255,0,0),2)

plt.figure(2)
plt.imshow(cv2.cvtColor(a_gauss, cv2.COLOR_BGR2RGB))
```

Out[11]:

```
<matplotlib.image.AxesImage at 0x1d88a271888>
```



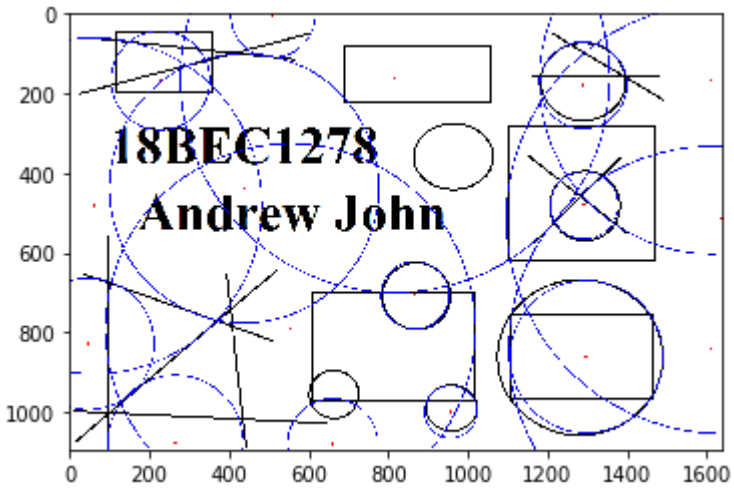


In [46]:

```
#HOUGH TRASNFORM - CIRCLE DETECTION
import cv2
import numpy as np
import matplotlib.pyplot as plt

img = cv2.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-5/Final Images/Task-2/2.1.png')
d=2
gimg=cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
gimg = cv2.GaussianBlur(gimg, (2*d+1, 2*d+1), -1)[d:-d,d:-d]
#cimg = cv2.cvtColor(img,cv2.COLOR_GRAY2BGR)

circles = cv2.HoughCircles(gimg,cv2.HOUGH_GRADIENT,0.2,300,param1=50,param2=30,minRadius=0,maxRadius=0)

circles = np.uint16(np.around(circles))
for i in circles[0,:]:
    # draw the outer circle
    cv2.circle(img,(i[0],i[1]),i[2],(255,0,0),2)
    # draw the center of the circle
    cv2.circle(img,(i[0],i[1]),2,(0,0,255),3)

plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
```

Out[46]:

```
<matplotlib.image.AxesImage at 0x1d882868b08>
```

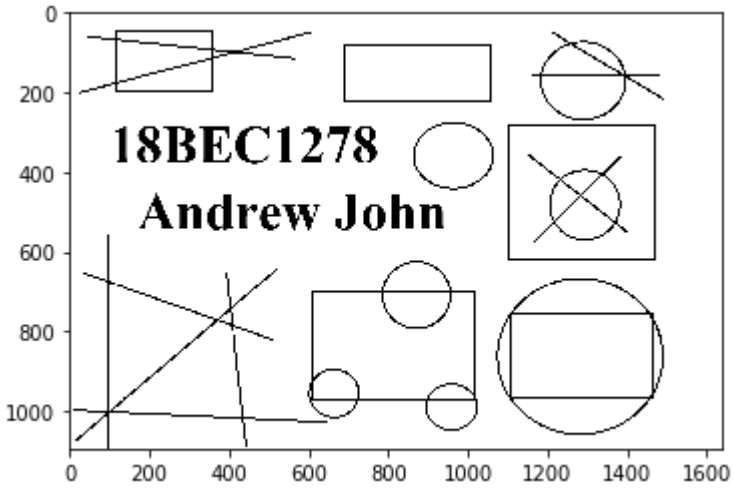


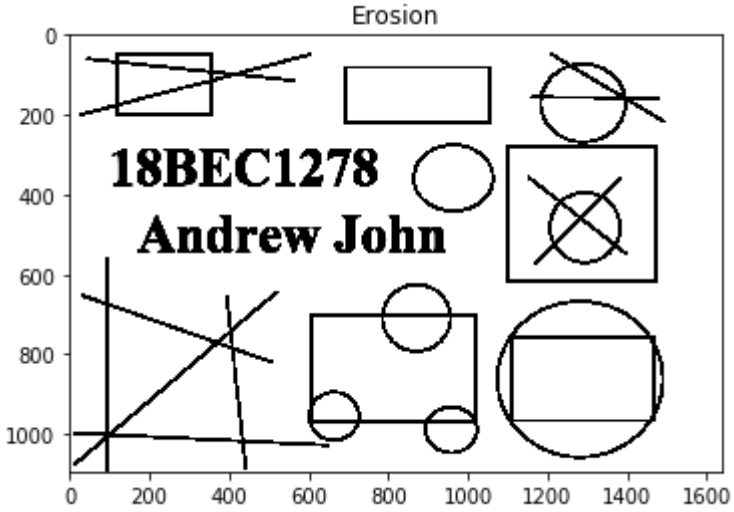
In [53]:

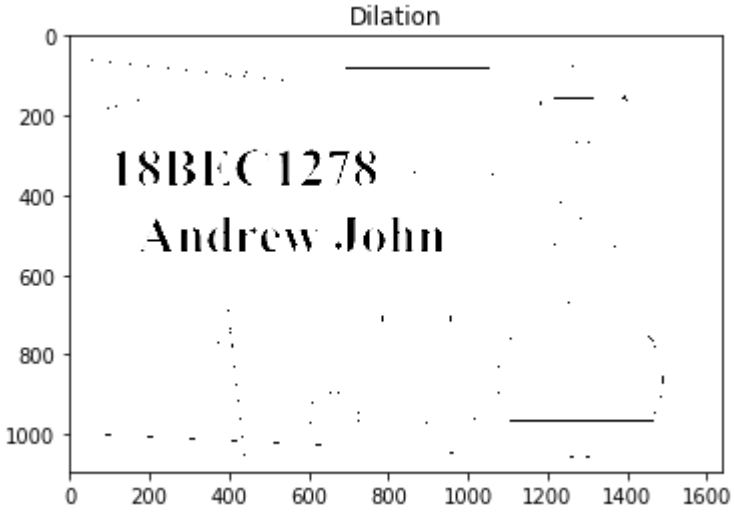
```
#MORPHOLOGICAL OPERATIONS ON IMAGES
import cv2
import numpy as np
import matplotlib.pyplot as plt

img = cv2.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-5/Final Images/Task-2/2.1.pn
g',0)
plt.figure()
plt.imshow(img,cmap='gray')

kernel = np.ones((5,5),np.uint8)
img_eroded = cv2.erode(img,kernel,iterations =1)
img_dilated = cv2.dilate(img,kernel,iterations = 1)
img_opened = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)
img_closed = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)

plt.figure()
plt.imshow(img_eroded,cmap='gray')
plt.title('Erosion')
plt.figure()
plt.imshow(img_dilated,cmap='gray')
plt.title('Dilation')
plt.figure()
plt.imshow(img_opened,cmap='gray')
plt.title('Opening')
plt.figure()
plt.imshow(img_closed,cmap='gray')
plt.title('Closing')
#BOUNDARY EXTRACTION
img_boundary1 = np.subtract(img,img_eroded)
plt.figure()
plt.imshow(img_boundary1,cmap='gray')
plt.title('img_boundary1')
img_boundary2 = np.subtract(img_dilated,img)
plt.figure()
plt.imshow(img_boundary2,cmap='gray')
plt.title('img_boundary2')
img_boundary3= np.subtract(img_dilated,img_eroded)
plt.figure()
plt.imshow(img_boundary3,cmap='gray')
plt.title('img_boundary3')
```

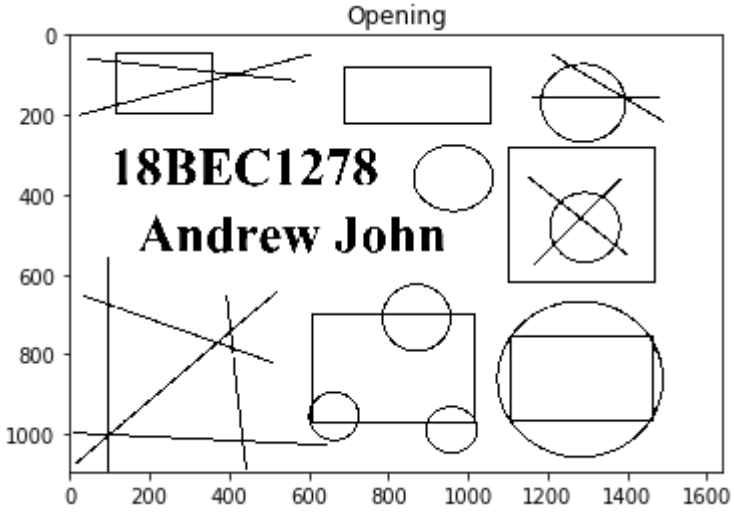
Out[53]:

```
Text(0.5, 1.0, 'img_boundary3')
```

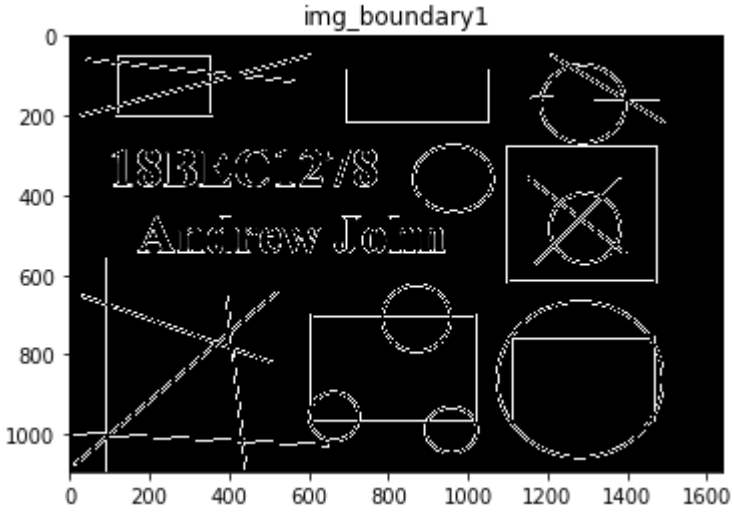


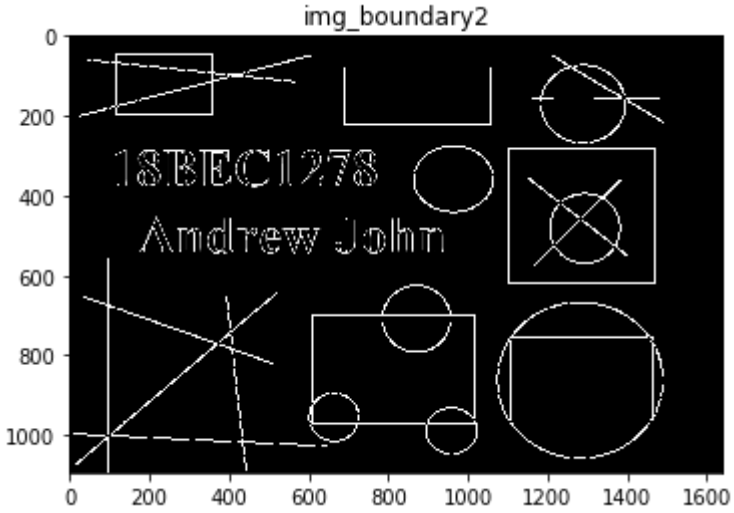


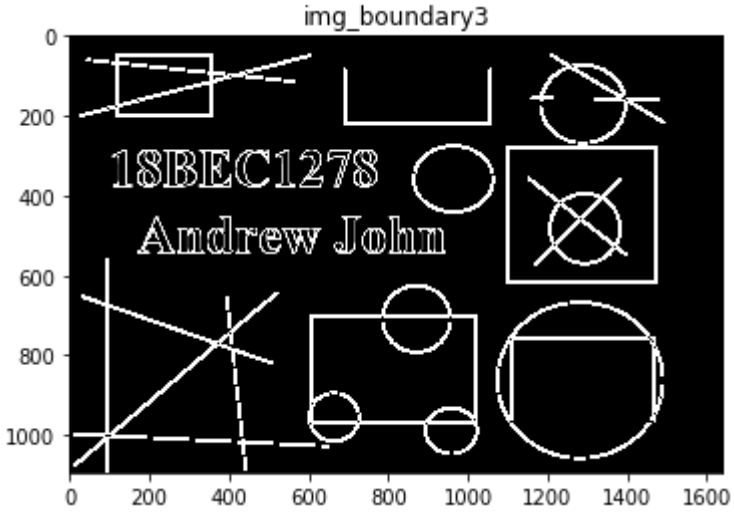

Dilation
18BEC1278
Andrew John


Opening
18BEC1278
Andrew John


Closing
18BEC1278
Andrew John


img_boundary1
18BEC1278
Andrew John


img_boundary2
18BEC1278
Andrew John

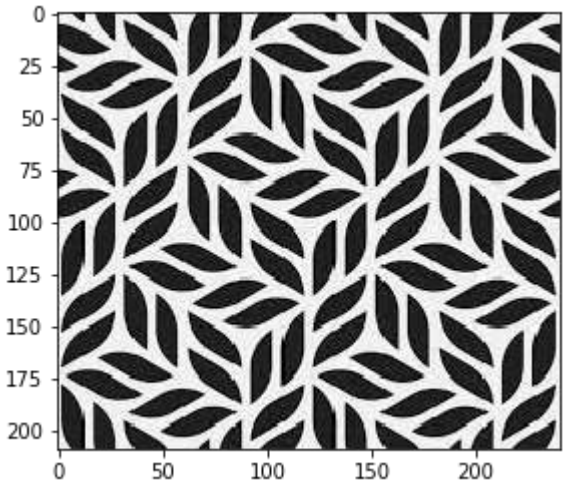

img_boundary3
18BEC1278
Andrew John

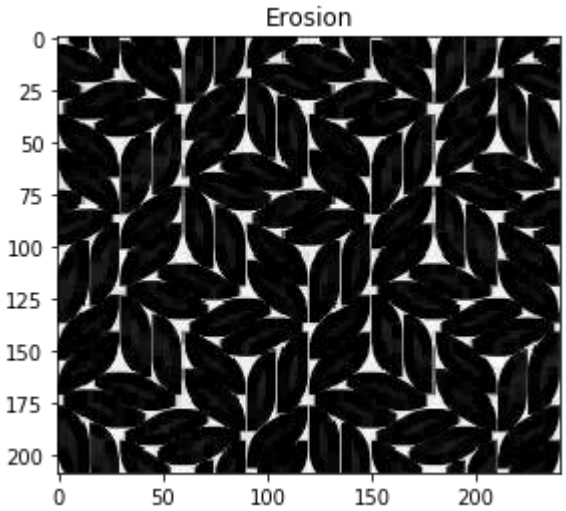
In [43]:

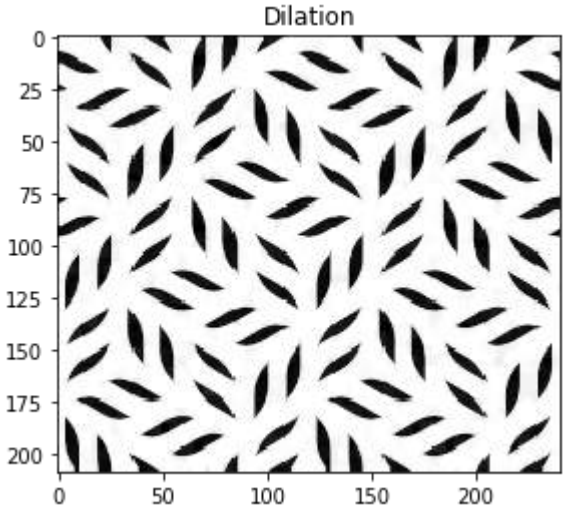
```
#MORPHOLOGICAL OPERATIONS ON IMAGES - with a Random Image
import cv2
import numpy as np
import matplotlib.pyplot as plt

img = cv2.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-5/Final Images/Task-2/2.2.jp
g',0)
plt.figure()
plt.imshow(img,cmap='gray')

kernel = np.ones((5,5),np.uint8)
img_eroded = cv2.erode(img,kernel,iterations =1)
img_dilated = cv2.dilate(img,kernel,iterations = 1)
img_opened = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)
img_closed = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)

plt.figure()
plt.imshow(img_eroded,cmap='gray')
plt.title('Erosion')
plt.figure()
plt.imshow(img_dilated,cmap='gray')
plt.title('Dilation')
plt.figure()
plt.imshow(img_opened,cmap='gray')
plt.title('Opening')
plt.figure()
plt.imshow(img_closed,cmap='gray')
plt.title('Closing')
#BOUNDARY EXTRACTION
img_boundary1 = np.subtract(img,img_eroded)
plt.figure()
plt.imshow(img_boundary1,cmap='gray')
plt.title('img_boundary1')
img_boundary2 = np.subtract(img_dilated,img)
plt.figure()
plt.imshow(img_boundary2,cmap='gray')
plt.title('img_boundary2')
img_boundary3= np.subtract(img_dilated,img_eroded)
plt.figure()
plt.imshow(img_boundary3,cmap='gray')
plt.title('img_boundary3')
```

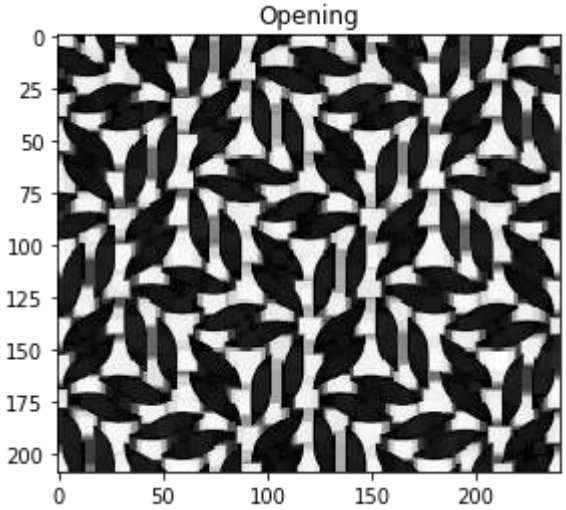
Out[43]:

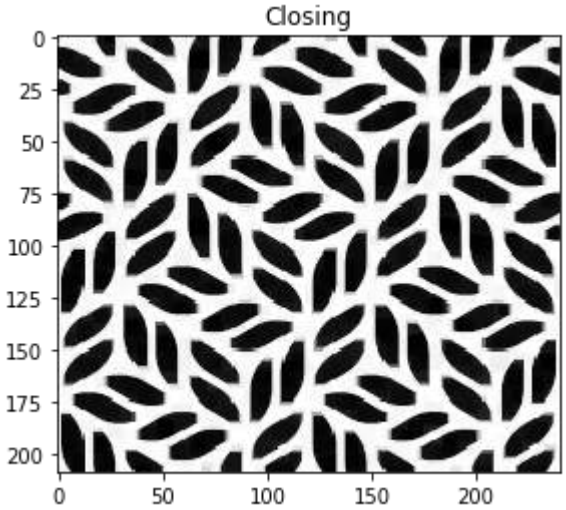
```
Text(0.5, 1.0, 'img_boundary3')
```

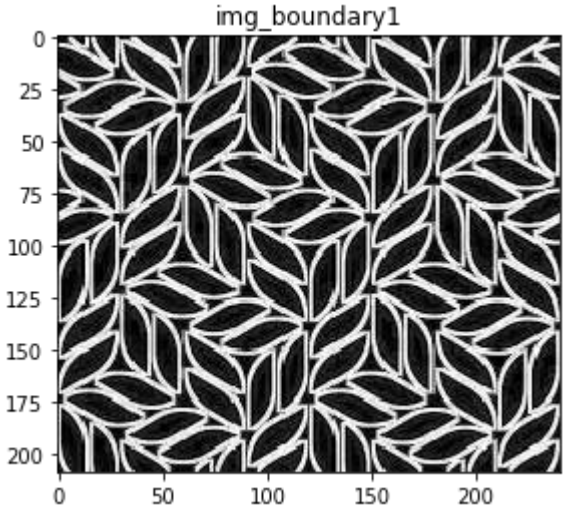


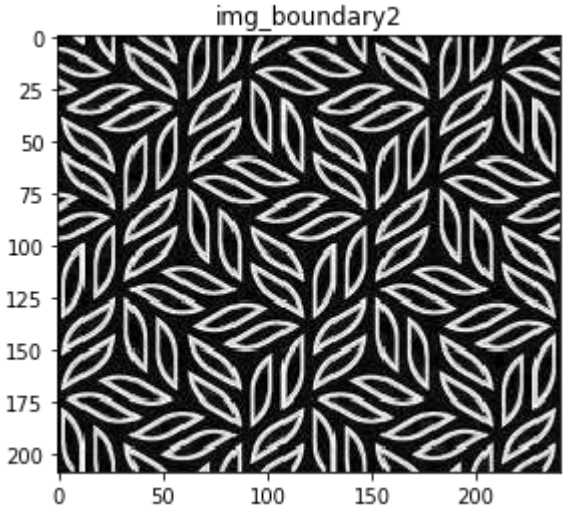


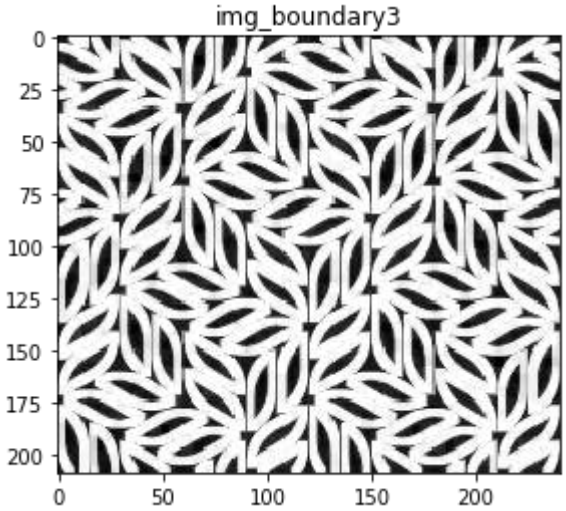

Dilation
0
25
50
75
100
125
150
175
200
0
50
100
150
200


Opening
0
25
50
75
100
125
150
175
200
0
50
100
150
200


Closing
0
25
50
75
100
125
150
175
200
0
50
100
150
200


img_boundary1
0
25
50
75
100
125
150
175
200
0
50
100
150
200


img_boundary2
0
25
50
75
100
125
150
175
200
0
50
100
150
200


img_boundary3
0
25
50
75
100
125
150
175
200
0
50
100
150
200

→ Inference:

1. In Prewitt vertical , horizontal edges are detected whereas in prewitt horizontal , vertical edges are detected. The borders are more clearly visible in horizontal one.
2. Hence edge detection or demarcation can be done using both sobel and prewitt operators.
3. As we increase the value of ‘d’ the image gets more blurred. This makes it tough to detect the edges.
4. As a result the red lines in the image gets reduced whereas the image with ‘d’ value as 2 , the image is more clear and hence it detects more straight lines as edges and there are more red lines in the output.
5. Hence we have applied the HOUGH transform and detected the lines
6. Hough circles uses hough transform to detect the circles. The center of the circle will be pointed by a red dot .

In [ ]:

```
```

Task - 3:

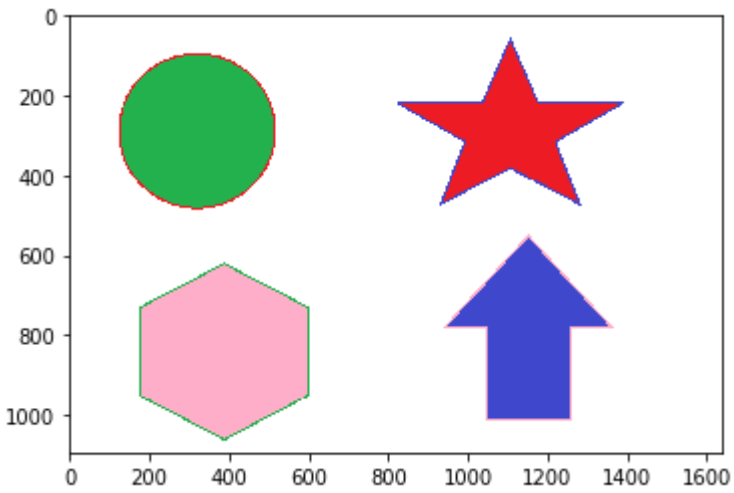
3. Create an image containing 4 different shapes each of different colors (red, green, blue, pink, etc.) and apply *Color-based Clustering* on it.

In [37]:

```
#k-Means Clustering
import numpy as np
import cv2
import matplotlib.pyplot as plt

img = cv2.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-5/Final Images/Task-3/3.2.png')
img=np.float32(img)
plt.imshow(cv2.cvtColor(np.uint8(img),cv2.COLOR_BGR2RGB))
#img =cv2.cvtColor(orgimg, cv2.COLOR_BGR2GRAY)
```

Out[37]:

```
<matplotlib.image.AxesImage at 0x1d88dde7f48>
```

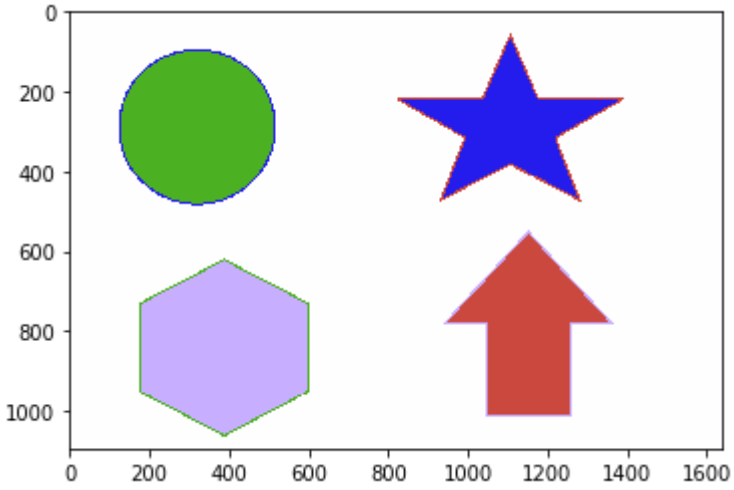


In [38]:

```
[m, n, k]=np.shape(img)

img_1d=img.reshape(m*n,k)

from sklearn.cluster import KMeans
#kmeans = KMeans(n_clusters=3, random_state=0).fit(img_1d)
kmeans = KMeans(n_clusters=5).fit(img_1d)
img2show = kmeans.cluster_centers_[kmeans.labels_]

img_cluster = img2show.reshape(m,n,k )
img_cluster=np.array(img_cluster, dtype=np.uint8)
plt.imshow(img_cluster)
```

Out[38]:

```
<matplotlib.image.AxesImage at 0x1d88e0b9ec8>
```

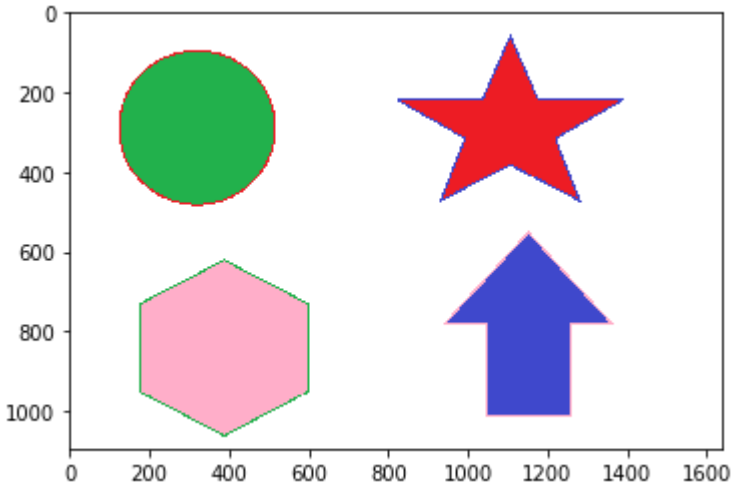


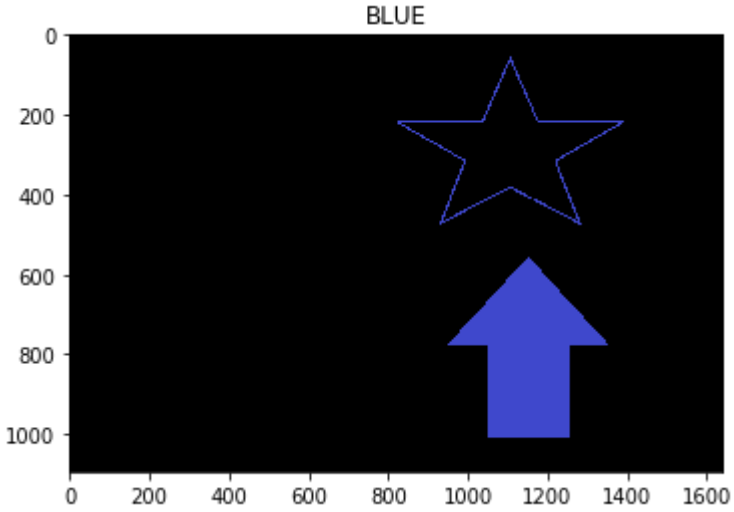
In [20]:

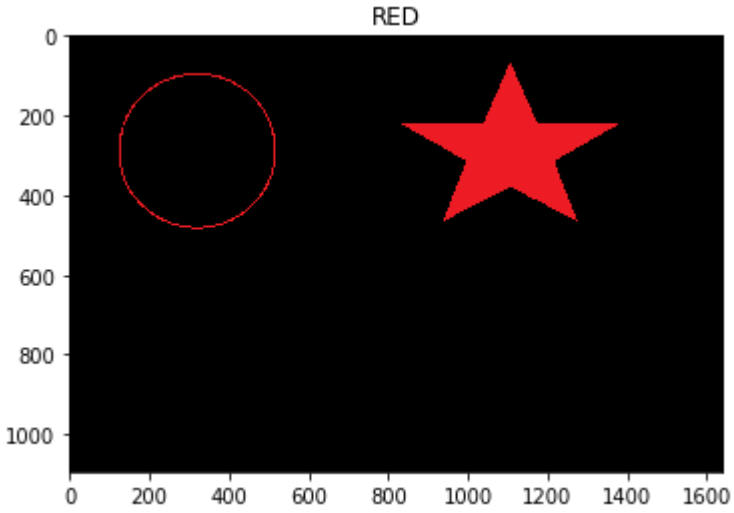
```
#Color Segmentation
import cv2
import numpy as np
import matplotlib.pyplot as plt

img = cv2.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-5/Final Images/Task-3/3.2.pn
g')
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

plt.figure()
plt.imshow(hsv)

lower_blue = np.array([0,20,50])
upper_blue = np.array([150,170,255])

lower_green = np.array([20, 100, 50])
upper_green = np.array([170, 200, 100])

lower_red = np.array([150, 0, 20])
upper_red = np.array([255, 130, 150])

lower_pink = np.array([150, 150, 150])
upper_pink = np.array([255, 200, 210])

# Threshold with inRange() get only specific colors
mask_blue = cv2.inRange(hsv, lower_blue, upper_blue)
mask_green = cv2.inRange(hsv, lower_green, upper_green)
mask_red = cv2.inRange(hsv, lower_red, upper_red)
mask_pink = cv2.inRange(hsv, lower_pink, upper_pink)

# Perform bitwise operation with the masks and original image
res_blue = cv2.bitwise_and(img,img, mask= mask_blue)
res_green = cv2.bitwise_and(img,img, mask= mask_green)
res_red = cv2.bitwise_and(img,img, mask= mask_red)
res_pink = cv2.bitwise_and(img,img, mask= mask_pink)

#hsv_red[res_red != 0] = hsv
res_red=cv2.cvtColor(res_red, cv2.COLOR_BGR2RGB)
res_blue=cv2.cvtColor(res_blue, cv2.COLOR_BGR2RGB)
res_pink=cv2.cvtColor(res_pink, cv2.COLOR_BGR2RGB)
#plt.imshow(cv2.cvtColor(hsv_red, cv2.COLOR_HSV2RGB))
plt.figure()
plt.imshow(res_blue)
plt.title('BLUE')
plt.figure()
plt.imshow(res_red)
plt.title('RED')
plt.figure()
plt.imshow(res_green)
plt.title('GREEN')
plt.figure()
plt.imshow(res_pink)
plt.title('PINK')
```

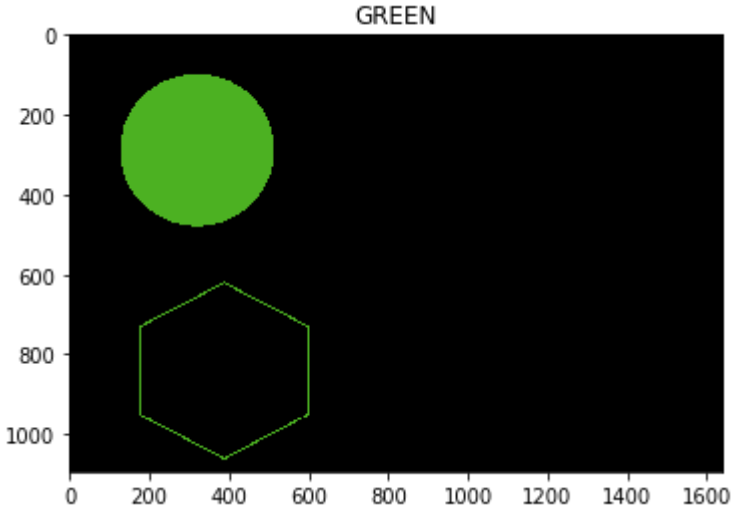
Out[20]:

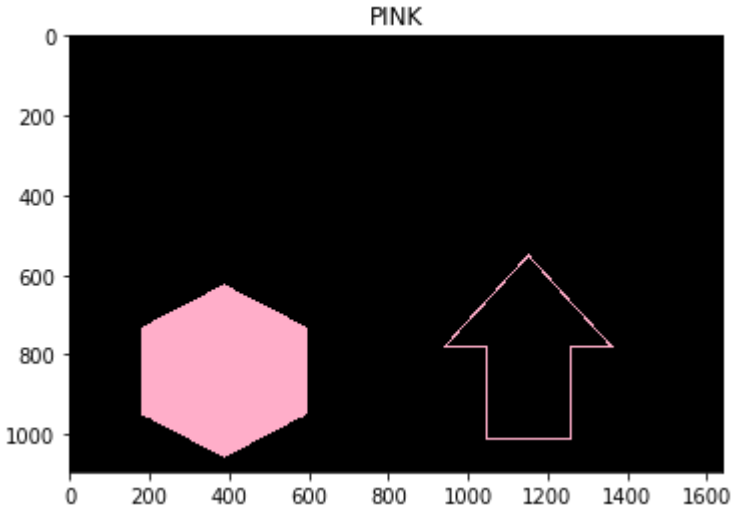
```
Text(0.5, 1.0, 'PINK')
```






RED
0
200
400
600
800
1000
0
200
400
600
800
1000
1200
1400
1600


GREEN
0
200
400
600
800
1000
0
200
400
600
800
1000
1200
1400
1600

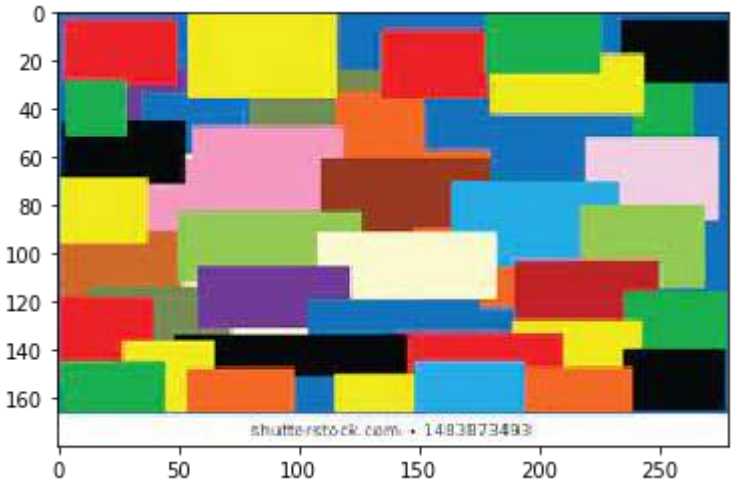

PINK
0
200
400
600
800
1000
0
200
400
600
800
1000
1200
1400
1600

In [56]:

```
#k-Means Clustering
import numpy as np
import cv2
import matplotlib.pyplot as plt

img = cv2.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-5/Final Images/Task-3/3.3.jp
g')
img=np.float32(img)
plt.imshow(cv2.cvtColor(np.uint8(img),cv2.COLOR_BGR2RGB))
#img =cv2.cvtColor(orgimg, cv2.COLOR_BGR2GRAY)
```

Out[56]:

```
<matplotlib.image.AxesImage at 0x1d883975608>
```

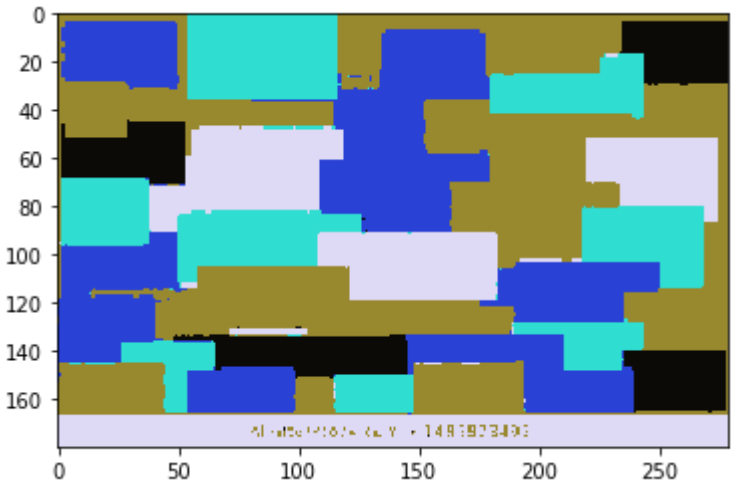


In [57]:

```
[m, n, k]=np.shape(img)

img_1d=img.reshape(m*n,k)

from sklearn.cluster import KMeans
#kmeans = KMeans(n_clusters=3, random_state=0).fit(img_1d)
kmeans = KMeans(n_clusters=5).fit(img_1d)
img2show = kmeans.cluster_centers_[kmeans.labels_]

img_cluster = img2show.reshape(m,n,k )
img_cluster=np.array(img_cluster, dtype=np.uint8)
plt.imshow(img_cluster)
```

Out[57]:

```
<matplotlib.image.AxesImage at 0x1d88e134d08>
```

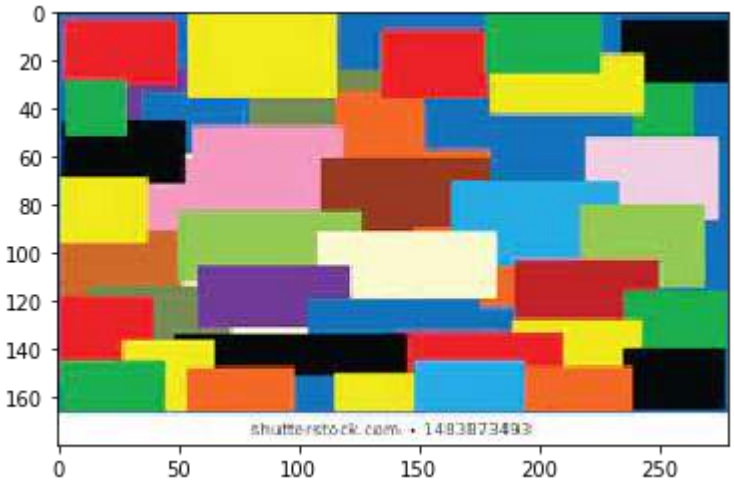


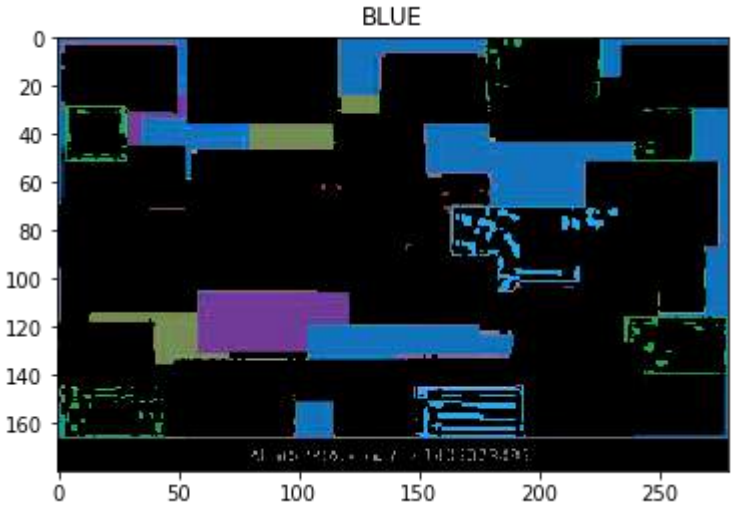
In [59]:

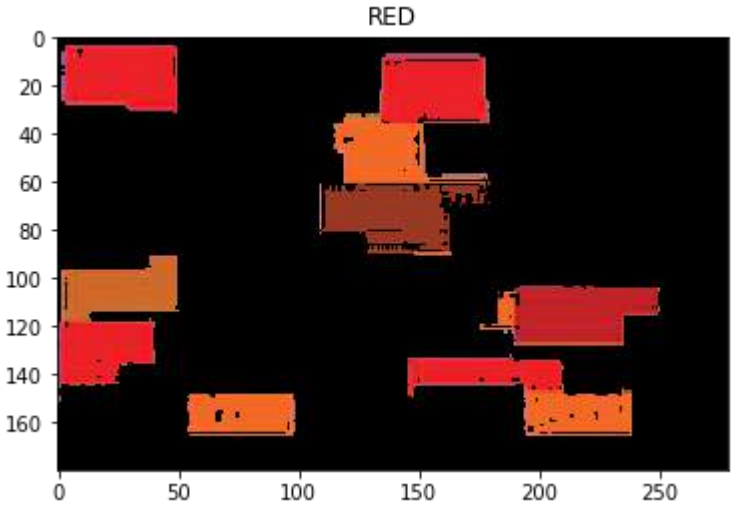
```
#Color Segmentation
import cv2
import numpy as np
import matplotlib.pyplot as plt

img = cv2.imread('C:/Users/1520a/Downloads/VAP - PVT/Week-5/Final Images/Task-3/3.3.jp
g')
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

plt.figure()
plt.imshow(hsv)

lower_blue = np.array([0,20,50])
upper_blue = np.array([150,170,255])

lower_green = np.array([20, 100, 50])
upper_green = np.array([170, 200, 100])

lower_red = np.array([150, 0, 20])
upper_red = np.array([255, 130, 150])

lower_pink = np.array([150, 150, 150])
upper_pink = np.array([255, 200, 210])

# Threshold with inRange() get only specific colors
mask_blue = cv2.inRange(hsv, lower_blue, upper_blue)
mask_green = cv2.inRange(hsv, lower_green, upper_green)
mask_red = cv2.inRange(hsv, lower_red, upper_red)
mask_pink = cv2.inRange(hsv, lower_pink, upper_pink)

# Perform bitwise operation with the masks and original image
res_blue = cv2.bitwise_and(img,img, mask= mask_blue)
res_green = cv2.bitwise_and(img,img, mask= mask_green)
res_red = cv2.bitwise_and(img,img, mask= mask_red)
res_pink = cv2.bitwise_and(img,img, mask= mask_pink)

#hsv_red[res_red != 0] = hsv
res_red=cv2.cvtColor(res_red, cv2.COLOR_BGR2RGB)
res_blue=cv2.cvtColor(res_blue, cv2.COLOR_BGR2RGB)
res_pink=cv2.cvtColor(res_pink, cv2.COLOR_BGR2RGB)
#plt.imshow(cv2.cvtColor(hsv_red, cv2.COLOR_HSV2RGB))
plt.figure()
plt.imshow(res_blue)
plt.title('BLUE')
plt.figure()
plt.imshow(res_red)
plt.title('RED')
plt.figure()
plt.imshow(res_green)
plt.title('GREEN')
plt.figure()
plt.imshow(res_pink)
plt.title('PINK')
```

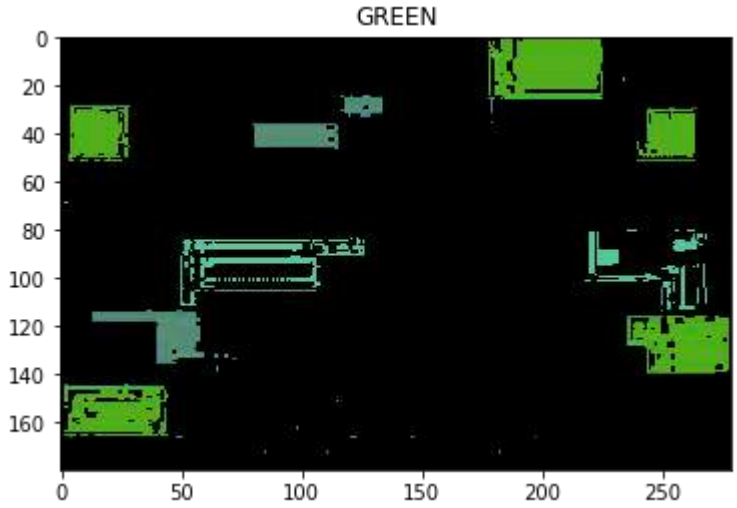
Out[59]:

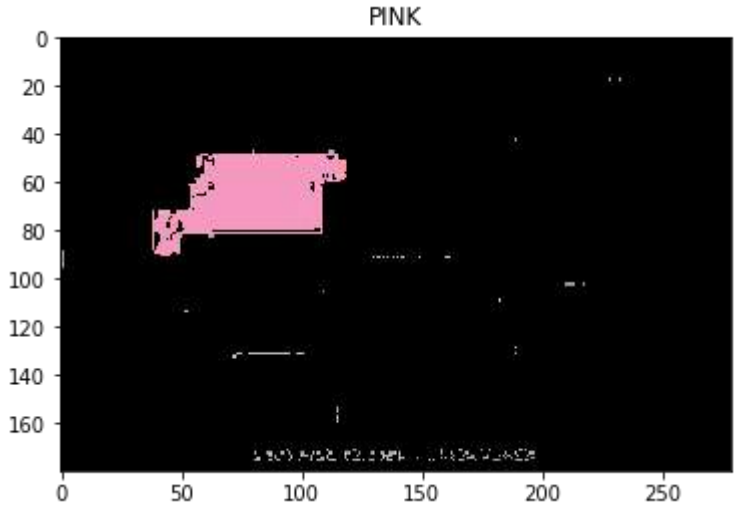
```
Text(0.5, 1.0, 'PINK')
```











→ Inference:

1. In this task I have taken a image consisting of different shapes filled with different colors.
2. Each time, for a particular color, the shape filled with that is getting segmented as the particular color is being highlighted and the magnitude of rest pixels are tending to 0(i.e., background getting black fully), except for the last case where the dominant color is “pink”.
3. When we upload images we note that only the red , blue ,green color portions of the image are higlighted .
4. Hence we have performed k-means clustering on images and observed the output.
5. Hence we have done color based clustering and segmented the green, red and pink colors separately from the original image.

In [ ]:

```
```