Computer Vision CS 559
ASSIGNMENT – 1

1. A) **Interchange Format**:
   - Images are simply data that has been recorded as numbers. Each square contains one pixel. The computer just enters a number to represent each square's color in order to store the image. The quality of the photographs will improve as the grid size increases. In order to transfer that picture data, we need some file formats that can be moved from one place or format to another. One such file format is the interchange format.
   - These interchange format, created to enable the sharing of image data between users and are adaptable to various gear and applications.
   - Image compression is a feature that all interchange formats must have.

   B) Examples for Interchange Format:
   - **JFIF**: JPEG File Interchange Format, The JPEG File Interchange Format is a simple file format that allows JPEG bitstreams to be transferred among several systems and applications.
   - **TIFF**: Tagged Image File Format, The Tagged Image File Format (TIFF) is a graphics container for raster images. It has excellent visuals with color depths that range from 1 to 24-bit and support both lossy and lossless compression.
   - **GIF**: Graphics Interchange Format, GIF is a raster file format made for simple graphics that are mostly found online. Each file may support 256 indexed colors and up to 8 bits per pixel. Additionally, rudimentary animations may be made with GIF files by combining pictures or frames.

   C) **PGM**: Portable Gray Map, Grayscale two-dimensional pictures are stored in portable gray map files. The image's pixels only include one or two bytes of data per pixel. PGM files can store tens of thousands of colors, ranging from pure black to white and every shade of gray in between.
   - PBM, PPM, and other portable file formats are only a few of the many available. Two-character signatures are used to identify them. The file type is determined by the PGM format's signature. For instance, the magic number in the signature is mostly P5 (grayscale RAW). The signature of the ordinary PGM format is P2(grayscale ASCII).
   - Example : P5 with colors

```
P5
4 2
7
B0   B1   B2    B3
B4   B5   B6    B7
```

   - Example: Image saying J

```
P2
6 7
15

0    0    0    0    13   0
0    0    0    0    13   0
0    0    0    0    13   0
0    0    0    0    13   0
0    13   0    0    13   0
0    13   0    0    13   0
0    0    13   13   0    0
```

D) **Image Printing**:

- The majority of printers output in binary, that is, they either print a black dot or nothing at all (white).
- By printing small, varying-sized black dots, newspaper images mimic a grayscale. Black dots and white backgrounds tend to blend together and appear as different shades of gray because of the tendency of the human visual system to average brightness across tiny regions.
- This procedure is known as halftoning, and there are numerous ways to put it into practice, one of them is **Patterning**
- **Patterning:**
  The easiest of the three methods for creating digital halftone pictures is patterning. In comparison to the original picture, it produces an image with a better spatial resolution. The output image has the same amount of halftone cells as pixels from the original image.
- This technique substitutes a pattern drawn from a binary typeface for each pixel.

E) **Error Diffusion:**

- Error diffusion is a sort of halftoning in which the quantization residual is transferred to nearby pixels that have not yet been processed. Although it has additional uses, its primary function is to transform a multi-level image into a binary image.
- Error diffusion, in contrast to many other halftoning techniques, is categorized as an area operation since what the algorithm accomplishes at one point affects what happens at other locations. Because of this, buffering is necessary, which makes parallel processing more difficult. These issues are not present in point operations like ordered dither.
- Starting with an 8-bit picture, we normally use a threshold of 128. The output's pixel at (x,y) is set to white if f(x,y) > 128 and to black otherwise. The mistake will be significant for pixels that are close to 128; in order to lessen the impact, this error is spread or diffused to the nearby pixels. The following technique for diffusing assumes traversal from top to bottom and left to right and distributes the mistake among 4 neighbors who are ahead of the pixel.
- We first establish a threshold value, which is often the average of pixel intensity in black and white.
  Threshold = (minimum + maximum)/2 # generally it is black and white

  The next step is to determine if the pixel value exceeds or falls short of the threshold. The matrix element turns black (which is 0) if the value is below the threshold, and the error is determined by subtracting the value from black from the value of the pixel.

  Error = f(x,y) – minimum # minimum = black

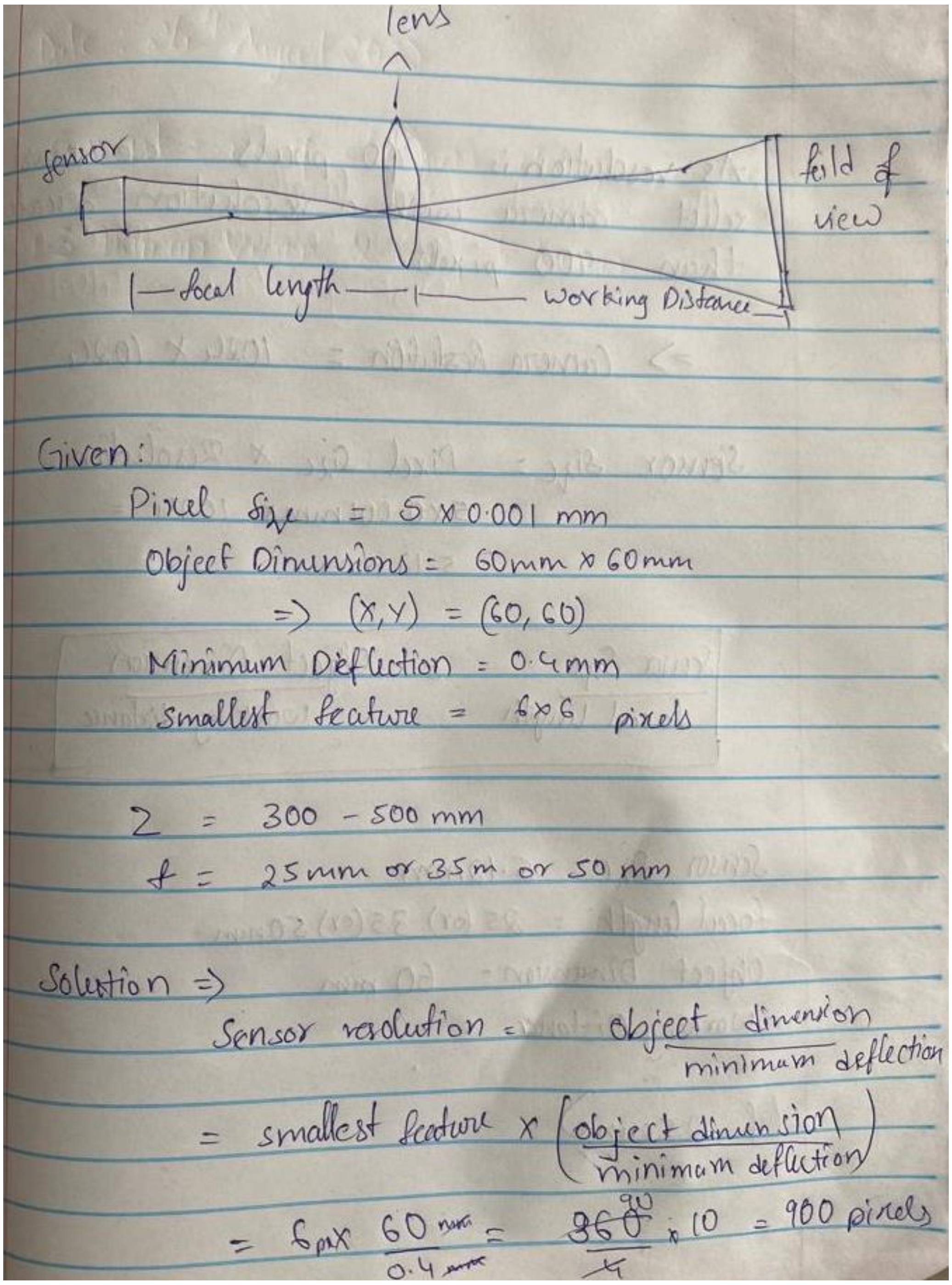
  If the value exceeds the threshold, the matrix element turns white, and the error is determined by subtracting the value of the pixel from white.

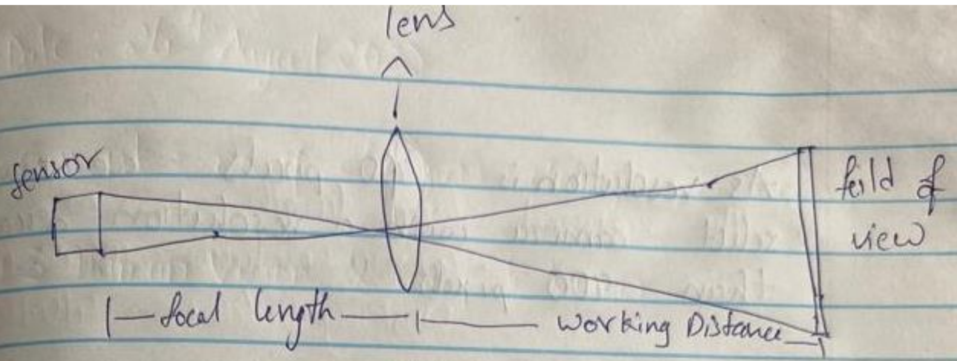
  Error = f(x,y) – maximum # maximum = white

  The error value is afterwards spread among the nearby components in such a way that,

  f(x+1,y) = f(x+1, y) + 7*error/16
  f(x-1, y+1) = f(x-1, y+1) + 3*error/16
  f(x, y+1) = f(x, y+1) + 5*error/16
  f(x+1,y+1) = f(x+1, y+1) + error/16

2.





Given:

Pixel size = 5 × 0.001 mm

Object Dimensions = 60mm × 60mm

=> $(X,Y) = (60, 60)$

Minimum Deflection = 0.4mm

smallest feature = 6×6 pixels

$Z = 300 - 500$ mm

$f = 25$mm or 35m or 50 mm

Solution =>

$$\text{Sensor resolution} = \frac{\text{object dimension}}{\text{minimum deflection}}$$

$$= \text{smallest feature} \times \left(\frac{\text{object dimension}}{\text{minimum deflection}}\right)$$

$$= 6\text{px} \times \frac{60\text{ mm}}{0.4\text{ mm}} = \frac{\overset{90}{\cancel{360}}}{\cancel{4}} \times 10 = 900 \text{ pixels}$$

As resolution is 900 pixels we must select camera with a resolution greater than 900 pixels

=> Camera Resolution = 1024 x 1024

Sensor Size = Pixel Size x Resolution
= 5 x 0.001 mm x 1024 =
= 5.12 mm

$$\boxed{\frac{\text{Sensor Size}}{\text{focal length}} = \frac{\text{Object Dimension}}{\text{working distance}}}$$

Sensor Size = 5.12mm
focal length = 25 (or) 35 (or) 50mm
Object Dimension = 60 mm
working Distance = ?

$f = 25mm$

$\Rightarrow Z = \frac{Xf}{x} = \frac{60 \times 25}{5.12} = 292.96$ mm

$Z = 292.96 < 300$

$f = 25$ mm is not scitable

$f = 35mm$

$\Rightarrow Z = \frac{Xf}{x} = \frac{60 \times 35}{5.12} = 410.16mm$

$Z = 410.16mm \in (300, 500)$

$f = 35$ is the required focal length.

$f = 50mm$

$\Rightarrow Z = \frac{Xf}{x} = \frac{60 \times 50}{5.12r} = 585.94$ mm

$Z = 585 > 500$

$f = 50mm$ is not seitable

∴ The required focal length = 35m

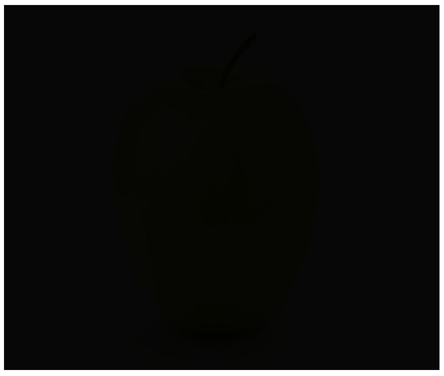
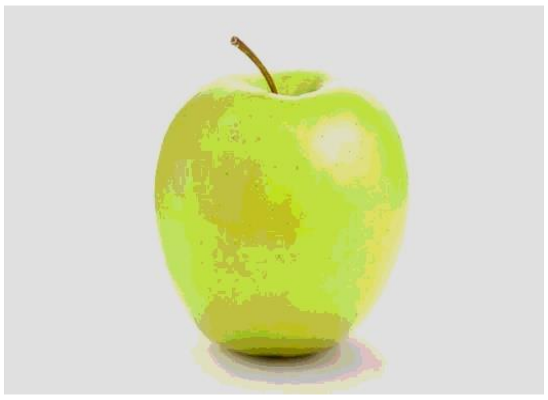
Camera Resolution = 1024 × 1024

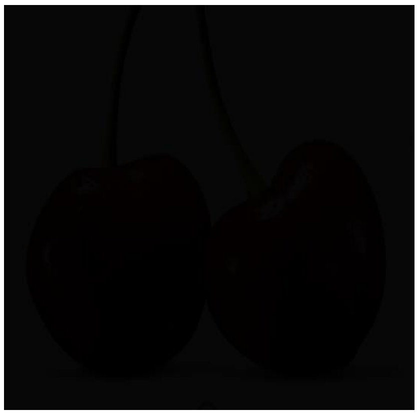
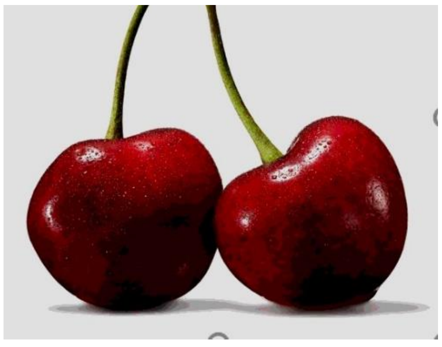
working Distance = 410.16 mm

3. **Storage Saving:**
   If c = 3, then we are essentially creating a picture that is 3/8th the size of the original image when converting an 8 bit image to a 3 bit image.

Program to Convert 8-bit image to 3-bit image:

When we read an image, it is stored as array, simulating the pixels.
When it’s a 8 bit image, the values range from 0 - 255.
So, dividing each pixel by 256 and multiplying it by 2**3 = 8 would range from 0 -7.
The result image is limited to very few colors as the range is from 0 – 7.

Code:

```
import cv2
import numpy as np

image_8bit = cv2.imread('8bit_cherry.jpg')  # reading image
image_8bit = np.uint8(image_8bit)
print("Max Value: {}, Image Size: {}, Shape: {}".format(image_8bit.max(),
image_8bit.size, image_8bit.shape))
image_3bit = (image_8bit/2**8)*2**3 # dividing each pixel value
image_3bit = np.uint8(image_3bit)  # typecasting to nd array
print("Max Value: {}, Image Size: {}, Shape: {}".format(image_3bit.max(),
image_3bit.size, image_3bit.shape))
upscale = image_3bit * 2**5  # as the 3 bit image is not visible, upscaleing the
image by multiplying whole array.
cv2.imshow('3bit_cherry.jpg', image_3bit)
cv2.imshow('3bit_cherry_upscaled.jpg',upscale )
cv2.imwrite('3bit_cherry.jpg', image_3bit)
cv2.imwrite('3bit_cherry_upscaled.jpg',upscale)
cv2.waitKey()
```

output:

```
Max Value: 255, Image Size: 679668, Shape: (418, 542, 3)
Max Value: 7, Image Size: 679668, Shape: (418, 542, 3)

Process finished with exit code 0
```

4. Program for keeping object of interest color and to turn the background into gray scale:

In order to keep the single color of interest highlighted
we need to grayscale the whole image and save it as a new image.

Convert the original image to hsv as working on colors using hue saturation values is easier.

Set the limits for the required color.
Lower and upper limits.

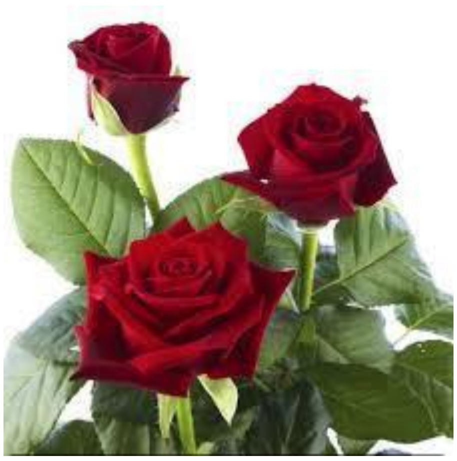
Create a new single colored image by bitwise and between original image and limits.
⇨ colored_image = cv2.bitwise_and(image, image, mask=limits)

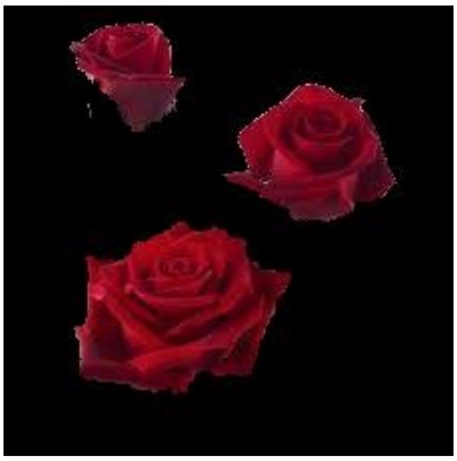
Create a grayscaled image subtracting the color part of the image.
⇨ gray_image = cv2.bitwise_and(grayscaled, grayscaled, mask=255-limits)

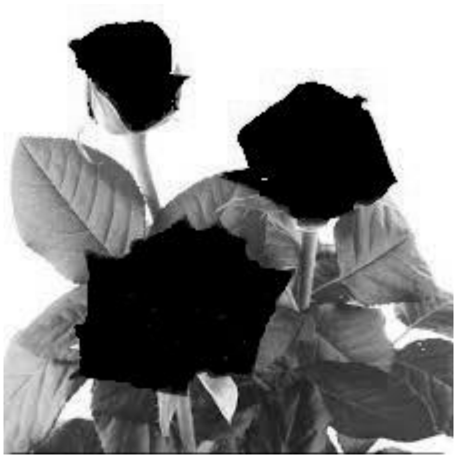
at last, combine both colored_image and gray_image
=> singleColorHighlighted = colored_image+gray_image

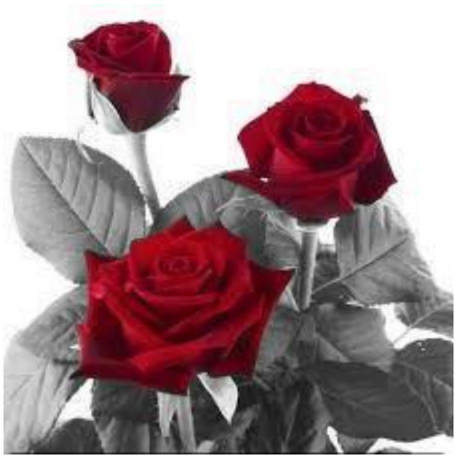
:

```
import numpy as np
import cv2

image = cv2.imread('red_rose.jpg')  # reading image using opencv library
grayscaled = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)  # converting the color
image to grayscaled image
grayscaled = cv2.merge([grayscaled, grayscaled, grayscaled])  # combining all
single channel images to make a multi-channel image
cv2.imshow('grayscaled',grayscaled)
hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)  # converting the color image
to hsv image

# setting limits to select the color
lower_limit = np.array([90, 70, 40])
upper_limit = np.array([255, 255, 255])
limits = cv2.inRange(hsv_image, lower_limit, upper_limit)

colored_image = cv2.bitwise_and(image, image, mask=limits) # selecting the image
in between limits of color image
gray_image = cv2.bitwise_and(grayscaled, grayscaled, mask=255-limits)
singleColorHighlighted = colored_image+gray_image

cv2.imshow('colored_output', colored_image)
cv2.imwrite('colored_output_red.jpg', colored_image)
cv2.imshow('gray_output', gray_image)
cv2.imwrite('gray_output_red.jpg', gray_image)
cv2.imshow('singleColorHighlighted', singleColorHighlighted)
cv2.imwrite('singleColorHighlighted_red.jpg', singleColorHighlighted)
cv2.waitKey()
```









CS 559 Fall 2022
Assignment 1 Grading Sheet

Sikakollu, Hareesh Chakravarthy

Red ID: 828571608

The following scores have been recorded for you. Please notify the lab assistant if you notice any discrepancies.

A1	A2	A3	A4	A5

Question 1	
Question 2	
Question 3	
Question 4	
Total Score	

Question 1 (20%)

Criteria	Possible scores	Earned	Comments
(a) to (e) each 4 points	20		

Question 2 (25%)

Criteria	Possible scores	Earned	Comments
(a) Procedure for selecting camera focal length	10		
(b) Procedure for selecting camera resolution	8		
(c) Completeness of reasoning	7		

Question 3 (25%)

Criteria	Possible scores	Earned	Comments
(a) Saving in terms of b and c	5		
(b) Program (code, etc.)	8		
Results	8		
Extra Credit for exceptionally well-done work	4		

Question 4 (30%)

Criteria	Possible	Earned	Comments
(a) Program for keeping object of interest color and to turn the background into gray scale.	12		
(b) Demonstrating your results with images	8		
(c) Report on features of your approach, program, etc.	4		
Extra Credit for exceptionally well-done work	6		