Introduction to Digital Image Processing **HW #1 Solutions**

1. Problem 2.1 (10 %)

The diameter, x, of the retinal image corresponding to the dot is obtained from similar triangles, as shown in Fig. P2.1. That is,

$$\frac{(d/2)}{0.2} = \frac{(x/2)}{0.017}$$

which gives $x = 0.085d$. From the discussion in Section 2.1.1, and taking some liberties of interpretation, we can think of the fovea as a square sensor array having on the order of 337,000 elements, which translates into an array of size 580×580 elements. Assuming equal spacing between elements, this gives 580 elements and 579 spaces on a line 1.5 mm long. The size of each element and each space is then $s = [(1.5\text{mm})/1,159] = 1.3 \times 10^{-6}$ m. If the size (on the fovea) of the imaged dot is less than the size of a single resolution element, we assume that the dot will be invisible to the eye. In other words, the eye will not detect a dot if its diameter, d, is such that $0.085(d) < 1.3 \times 10^{-6}$ m, or $d < 15.3 \times 10^{-6}$ m.

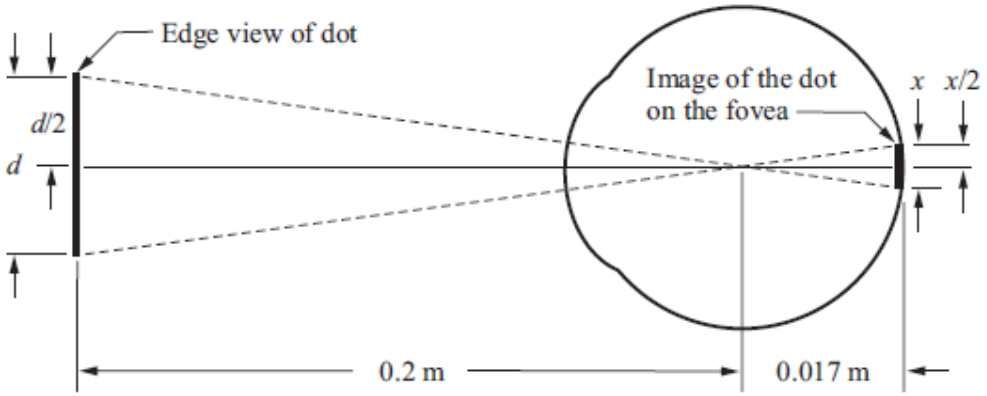


Figure P2.1

2. Problem 2.7 (10 %)

The image in question is given by

$$\begin{aligned} f(x,y) &= i(x,y)r(x,y) \\ &= 255\, e^{-[(x-x_0)^2+(y-y_0)^2]} \times 1.0 \\ &= 255\, e^{-[(x-x_0)^2+(y-y_0)^2]} \end{aligned}$$

A cross section of the image is shown in Fig. P2.7(a). If the intensity is quantized using m bits, then we have the situation shown in Fig. P2.7(b), where $\triangle G = (255+1)/2^m$. Since an abrupt change of 8 intensity levels is assumed to be detectable by the eye, it follows that $\triangle G = 8 = 256/2m$, or $m = 5$. In other words, 32, or fewer, intensity levels will produce visible false contouring.

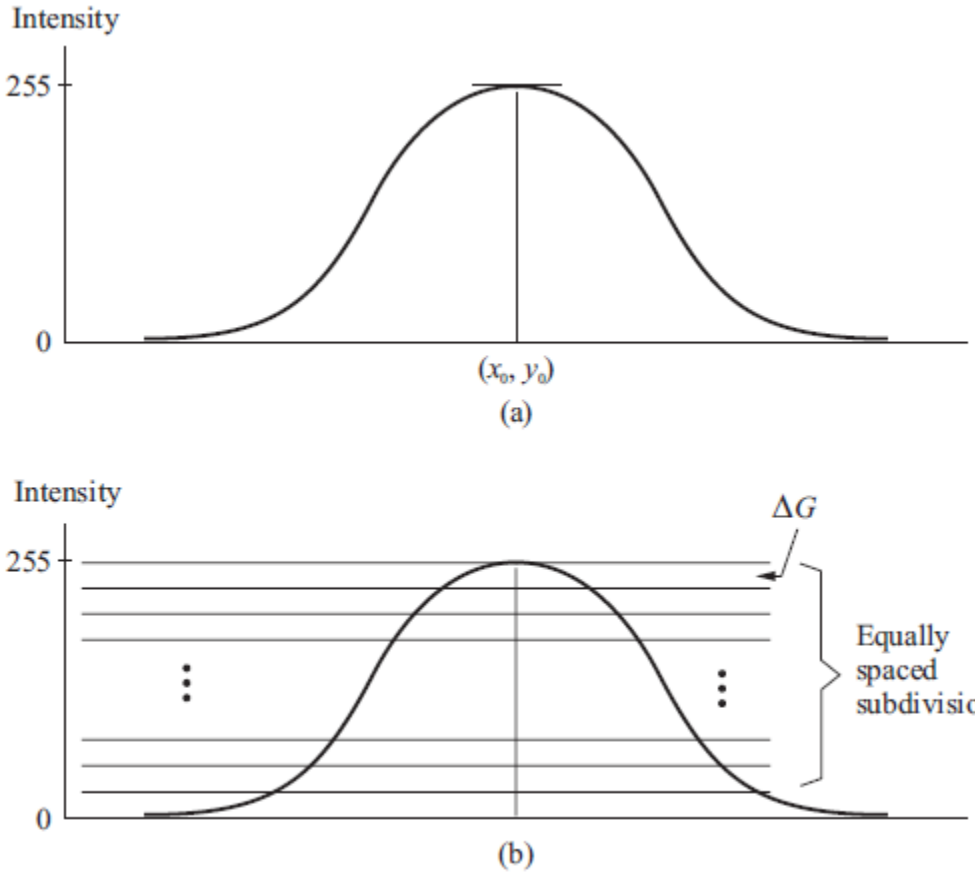


Figure P2.7

3. Problem 2.10 (10 %)

The width-to-height ratio is 16/9 and the resolution in the vertical direction is 1125 lines (or, what is the same thing, 1125 pixels in the vertical direction). It is given that the resolution in the horizontal direction is in the 16/9 proportion, so the resolution in the horizontal direction is $(1125)\times(16/9) = 2000$ pixels per line. The system "paints" a full 1125×2000, 8-bit image every 1/30 sec for each of the red, green, and blue component images. There are 7200 sec in two hours, so the total digital data generated in this time interval is $(1125)(2000)(8)(30)(3)(7200) = 1.166 \times 10^{13}$ bits, or 1.458×10^{12} bytes (i.e., about 1.5 terabytes). These figures show why image data compression (Chapter 8) is so important.

4. Problem 2.11 (10 %)

Let p and q be as shown in Fig. P2.11. Then, (a) S_1 and S_2 are not 4-connected because q is not in the set $N_4(p)$; (b) S_1 and S_2 are 8-connected because q is in the set $N_8(p)$; (c) S_1 and S_2 are m-connected because (i) q is in $N_D(p)$, and (ii) the set $N_4(p) \cap N_4(q)$ is empty.

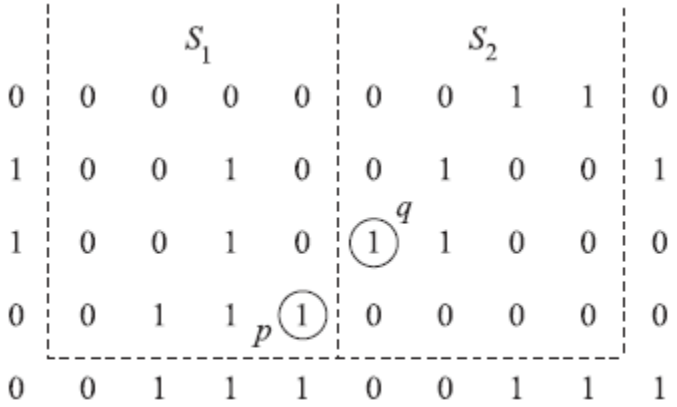


Figure P2.11

5. Problem 2.15 (10 %)

(a) When $V = \{0, 1\}$, 4-path does not exist between p and q because it is impossible to get from p to q by traveling along points that are both 4-adjacent and also have values from V. Figure P2.15(a) shows this condition; it is not possible to get to q. The shortest 8-path is shown in Fig. P2.15(b); its length is 4. The length of the shortest m- path (shown dashed) is 5. Both of these shortest paths are unique in this case.

(b) One possibility for the shortest 4-path when $V = \{1, 2\}$ is shown in Fig. P2.15(c); its length is 6. It is easily verified that another 4-path of the same length exists between p and q. One possibility for the shortest 8-path (it is not unique) is shown in Fig. P2.15(d); its length is 4. The length of a shortest m-path (shown dashed) is 6. This path is not unique.

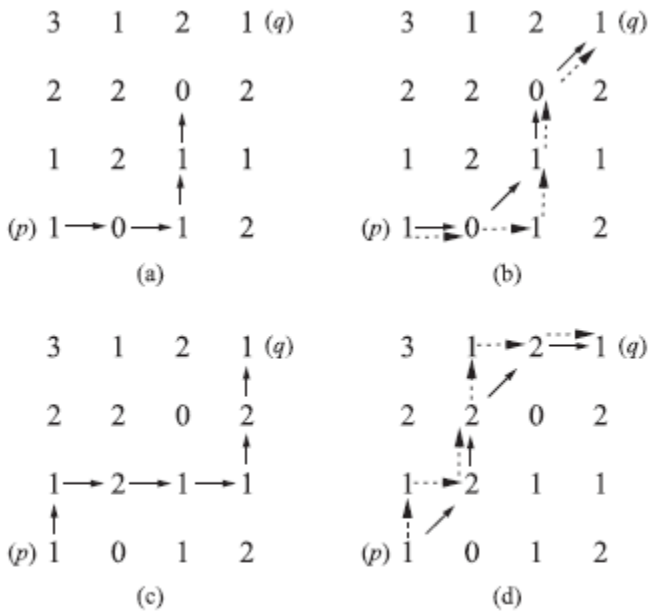


Figure P.2.15

6. Problem 2.19 (10 %)

The median, ζ, of a set of numbers is such that half the values in the set are below ζ and the other half are above it. A simple example will suffice to show that Eq. (2.6-1) is violated by the median operator. Let $S_1 = \{1, -2, 3\}$, $S_2 = \{4, 5, 6\}$, and $a = b = 1$. In this case H is the median operator. We then have $H(S_1 + S_2) = \text{median}\{5, 3, 9\} = 5$, where it is understood that $S_1 + S_2$ is the array sum of S_1 and S_2. Next, we compute $H(S_1) = \text{median}\{1, -2, 3\} = 1$ and $H(S_2) = \text{median}\{4, 5, 6\} = 5$. Then, because $H(aS_1 + bS_2) \neq aH(S_1) + bH(S_2)$, it follows that Eq. (2.6-1) is violated and the median is a nonlinear operator.

7. Problem 2.25 (10 %)

The Fourier transformation kernel is separable because

$$\begin{aligned} r(x, y, u, v) &= e^{-j2\pi(ux/M + vy/N)} \\ &= e^{-j2\pi(ux/M)} e^{-j2\pi(vy/N)} \\ &= r_1(x, u) r_2(y, v). \end{aligned}$$

It is symmetric because

$$r(x, y, u, v) = e^{-j2\pi\left(\frac{ux}{M} + \frac{vy}{N}\right)} = e^{-j2\pi\left(\frac{ux}{M}\right)} \times e^{-j2\pi\left(\frac{vy}{N}\right)}$$

when M=N, $r(x, y, u, v) = e^{-j2\pi\left(\frac{ux}{M}\right)} \times e^{-j2\pi\left(\frac{vy}{M}\right)} = r_1(x, u) r_1(y, v) = r_2(x, u) r_1(y, v)$

8. **Reducing the Number of Intensity Levels in an Image** (10 %)

(a) Write a computer program capable of reducing the number of intensity levels in an image from 256 to 2, in integer powers of 2. The desired number of intensity levels needs to be a variable input to your program.

(b) Download Fig. 2.21(a) from the course web site and duplicate the results shown in Fig. 2.21 of the book.

Sol:

(a)

(Matlab code)

```
clc;clear all;close all;
R=input('Bit of depth(1~8)=');%R為位元數
N = 2^R;
I = imread('Fig0221(a)(ctskull-256).tif');%讀圖
figure,imshow(grayslice(I,N),gray(N));%改變圖源階數
```

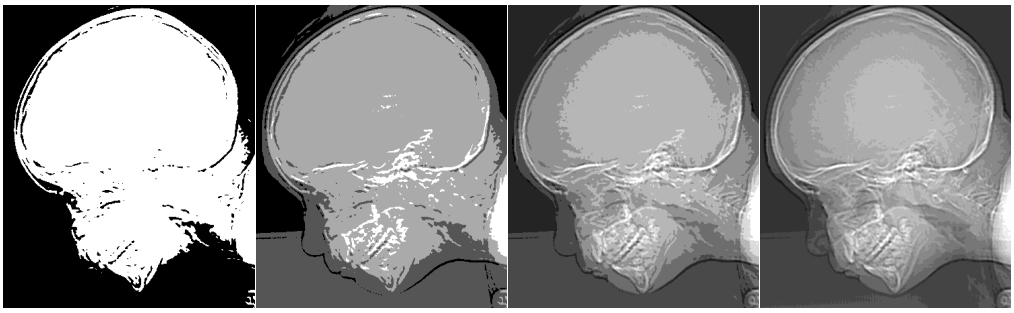
(Python code)

```
import cv2
import numpy as np
# Read input image
img = cv2.imread('Fig0221(a)(ctskull-256).tif', cv2.IMREAD_GRAYSCALE)
# Set the desired number of intensity levels (power of 2)
num_levels = int(input("Enter the number of intensity levels (2, 4, 8, 16, 32, 64, 128, 256): "))
assert num_levels in [2, 4, 8, 16, 32, 64, 128, 256], "Invalid number of intensity levels"
# Calculate the scaling factor
scale_factor = 256 // num_levels
# Perform intensity level reduction
if num_levels == 2:
    _, new_img = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)
else :
    new_img = np.zeros_like(img)
    for i in range(num_levels):
        low = i * scale_factor
        high = (i+1) * scale_factor
        mask = cv2.inRange(img, low, high-1)
        new_img[mask > 0] = int((i+0.5) * scale_factor)
if num_levels == 2:
    cv2.imwrite('output_image_2.png', new_img)
elif num_levels == 4:
    cv2.imwrite('output_image_4.png', new_img)
elif num_levels == 8:
```

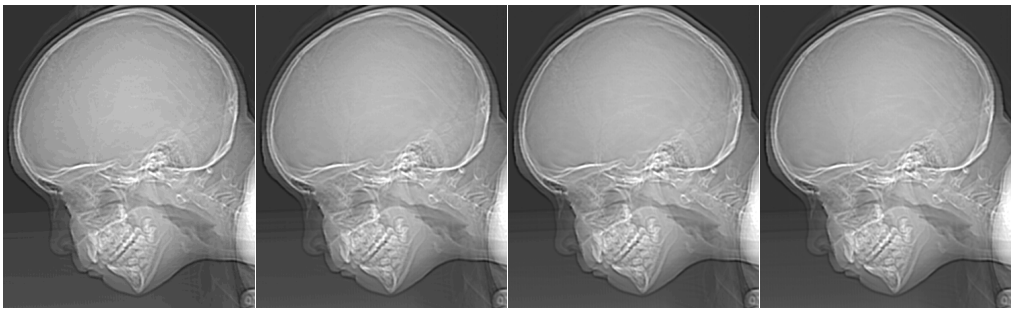
```
        cv2.imwrite('output_image_8.png', new_img)
elif num_levels == 16:
        cv2.imwrite('output_image_16.png', new_img)
elif num_levels == 32:
        cv2.imwrite('output_image_32.png', new_img)
elif num_levels == 64:
        cv2.imwrite('output_image_64.png', new_img)
elif num_levels == 128:
        cv2.imwrite('output_image_128.png', new_img)
elif num_levels == 256:
        cv2.imwrite('output_image_256.png', new_img)
else :
        print("Error: The number of intensity levels only 256")
```

(b)

2. 4. 8. 16.



32. 64. 128. 256.



9. **Zooming and Shrinking Images by Pixel Replication** (10 %)

(a) Write a computer program capable of zooming and shrinking an image by pixel replication. Assume that the desired zoom/shrink factors are integers.

(b) Download Fig. 2.20(a) from the course web site and use your program to shrink the image by a factor of 12.

(c) Use your program to zoom the image in (b) back to the resolution of the original. Explain the reasons for their differences.

Sol:

(a)

(Matlab code)

```
clc;clear all;close all;
I = imread('Fig0220(a)(chronometer 3692x2812  2pt25 inch 1250 dpi).tif');
F = input('desired zoom/shrink factors =');
info = imfinfo('Fig0220(a)(chronometer 3692x2812  2pt25 inch 1250 dpi).tif');
I2 = imresize(I, 1/F, 'nearest');
I3 = imresize(I2, [info.Height info.Width], 'nearest');
imwrite(I2,'shrinking_image.tif','Compression','none','resolution',[info.XResolution info.YResolution]);
imwrite(I3,'zooming_image.tif','Compression','none','resolution',[info.XResolution info.YResolution]);
```

(b)

(c)

Ans：圖像經過縮小再放大後，有些圖像的細節資訊已消失，因此圖像無法完全恢復，會比原圖像模糊。

(Python code)

```
import numpy as np
import cv2
# Load the input image
img = cv2.imread('Fig0220(a)(chronometer 3692x2812  2pt25 inch 1250 dpi).tif')
# Set the desired zoom/shrink factors
zoom_factor = 0  # zoom factor of 0 means no zooming
shrink_factor = 12  # shrink by a factor of 12
# Get the dimensions of the input image
h, w, c = img.shape
# Shrink by pixel replication
if shrink_factor:
```

```
    # Create a new output image with the desired dimensions
    out = np.zeros((h // shrink_factor, w // shrink_factor, c), dtype=np.uint8)
    # Copy each block of pixels in the input image to a single pixel in the output image
    for i in range(out.shape[0]):
        for j in range(out.shape[1]):
            for k in range(c):
                out[i, j, k] = img[i * shrink_factor, j * shrink_factor, k]
    # Display and save the output image
    cv2.imshow('Shrink Image', out)
    cv2.imwrite('shrink_image.jpg', out)
img = cv2.imread('shrink_image.jpg')
zoom_factor = 12
shrink_factor = 0
h, w, c = img.shape
if zoom_factor:
    out = np.zeros((h * zoom_factor, w * zoom_factor, c), dtype=np.uint8)
    for i in range(h):
        for j in range(w):
            for k in range(c):
                out[i * zoom_factor:(i + 1) * zoom_factor, j * zoom_factor:(j + 1) * zoom_factor, k] =
img[i, j, k]
    cv2.imshow('Zoomed Image', out)
    cv2.imwrite('zoomed_image.jpg', out)
# Wait for user input and then exit
cv2.waitKey(0)
cv2.destroyAllWindows()
```

10. **Image Rotation, Scaling, Translation and Intensity Interpolation** (10%)

(a) Write a computer program capable of rotating, scaling, and translating an image by specified degree, ratio, and pixels. The rotation degree, scaling ratio, and translating pixels need to be variable inputs to your program.

(b) Download Fig. 2.36 (a) from the course web site and rotate the image 23° clockwise, scale it to 2/3 of original size, and shift it by 18 and 22 pixels in x and y directions, respectively. Show the results using three interpolation approaches mentioned in the textbook. Please also zoom in the results to compare the differences as shown in Figs. 2.36 (b) – (d).

Sol:

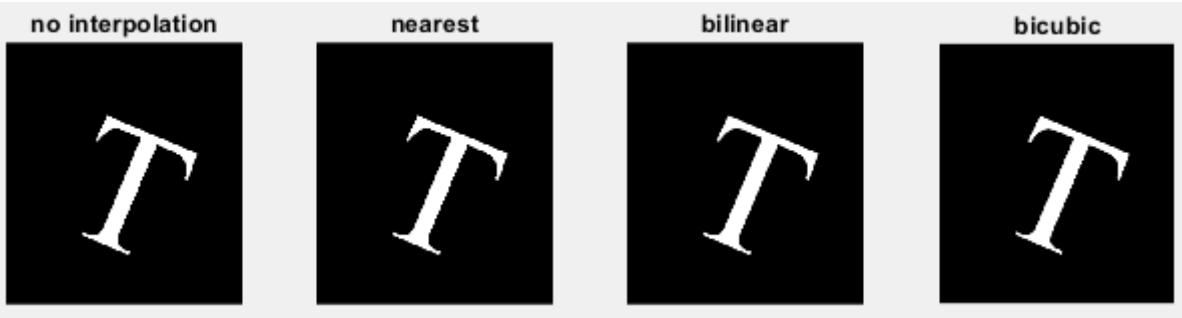
(a)

(Matlab code)

```
Clc;clear;close all
img = imread(' Fig0236(a)(letter_T) ');
```

```
scale = input('Scaling ratio = ');
translate_x = input('Translate x-coordinate pixels = ');
translate_y = input('Translate y-coordinate pixels = ');
degree = input('Rotation degree = ');
 % Resize & Translate
 S_img = imresize(img, scale);
 T_img = imtranslate (S_img, [translate_x, translate_y]);
 % Rotate & interpolation
 R_img = imrotate(T_img, degree);
 R_img_near = imrotate(T_img, degree,'nearest');
 R_img_bili = imrotate(T_img, degree,'bilinear');
 R_img_bicu = imrotate(T_img, degree,'bicubic');
 % Show the result
 subplot(1,4,1);
 imshow(R_img)
 title('no interpolation');
 subplot(1,4,2);
 imshow(R_img_near)
 title('nearest');
 subplot(1,4,3);
 imshow(R_img_bili)
 title('bilinear');
 subplot(1,4,4);
 imshow(R_img_bicu)
 title('bicubic');
```

(b)



(Python code)

```
import numpy as np
import cv2
# Load the image
img = cv2.imread('Fig0236(a)(letter_T).tif')
# Get user input for scaling ratio, translation in x and y directions, and rotation angle
```

```
scale = float(input('Enter scaling ratio: '))
tx = int(input('Enter translation in x direction (in pixels): '))
ty = int(input('Enter translation in y direction (in pixels): '))
angle = float(input('Enter rotation angle (in degrees): '))
# Compute the center of the image
(h, w) = img.shape[:2]
center = (w // 2, h // 2)
# Define the rotation matrix
M = cv2.getRotationMatrix2D(center, angle, scale)
# Apply the rotation to the image
rotated = cv2.warpAffine(img, M, (w, h))
# Define the translation matrix
T = np.float32([[1, 0, tx], [0, 1, ty]])
# Apply the translation to the rotated image
translated = cv2.warpAffine(rotated, T, (w, h))
# Resize the translated image to 2/3 of the original size using different interpolation methods
resized_nearest = cv2.resize(translated, None, fx=scale, fy=scale, interpolation=cv2.INTER_NEAREST)
resized_linear = cv2.resize(translated, None, fx=scale, fy=scale, interpolation=cv2.INTER_LINEAR)
resized_cubic = cv2.resize(translated, None, fx=scale, fy=scale, interpolation=cv2.INTER_CUBIC)
# Show the results
cv2.imshow('no interpolation', translated)
cv2.imshow('Resized Image (Nearest Interpolation)', resized_nearest)
cv2.imshow('Resized Image (Bilinear Interpolation)', resized_linear)
cv2.imshow('Resized Image (Bicubic Interpolation)', resized_cubic)
cv2.waitKey(0)
cv2.destroyAllWindows()
```