ECE 302-003, Project #1
Due date: Friday 11/3/2023, 11:59pm;

`https://engineering.purdue.edu/~chihw/23ECE302F/23ECE302F.html`

Project: Synthesize a grayscale picture from multiple binary pictures generated by single-photon detectors.

Authors: The current version of this project is based on latest the revision of Pin-Wen Su and Prof. Chih-Chun Wang in fall 2023. This project was originally designed and created by Prof. Stanley Chan and Kent Gauen.

• **A short introduction about why we are interested in this project:**

This project studies a single-photon image sensor. The goal is to demonstrate that by repeatedly using a single-photon sensor (over time), one can recreate the original image as if we are using a traditional CMOS sensor. Some background information: single-photon sensors, as the name indicated, are designed for extremely low light environments. You can Google the term for more information.

• **The following process will be performed on each pixel separately:**

Specifically, the goal is to capture the image of an object as a 1000x750 grayscale image file (i.e., totally we have 750K pixels). The following describes the light-capturing process for *each pixel* and the process needs to be repeated for all 750K pixels.

Suppose for a given pixel, the average photon arrival rate is α during each measurement period. Obviously, the brighter the subject, the larger the average photon arrival rate α. The challenge is that the sensor (of each pixel) does not know the *average* arrival rate α, and can only count how many photons it has *actually* received in each measurement period, which is a random variable.

First, recall that photons arrive according to a Poisson distribution, i.e., the probability of receiving k photons is

$$P(Y = k) = \frac{\alpha^k e^{-\alpha}}{k!},$$

where α is the (unknown) underlying photon arrival rate. Because the hardware is a "single-photon" detector, when photons arrive at the sensor/detector, the detector generates a binary response "1" when one or more photons are detected, and "0" when no photon is detected.

• **Basic analysis:** Please write down your answers for the following questions (a)–(e) when submitting the results of your project.

(a) Let B be the random variable denoting the response of the single-photon detector. That is,

$$B = \begin{cases} 1, & Y \geq 1, \\ 0, & Y = 0, \end{cases}$$

where Y is the number of photons received. Find the pmf of B. Obviously, your answer would be a function of the unknown average arrival rate α.

(b) Suppose we have repeated the measurement for T times. That is, it is similar to taking a "burst" of T pictures of a single pixel. (It is worth emphasizing again that all our discussion is only for one pixel. The same process needs to be applied to each individual pixel separately.)

We also assume that the T measurements are independent and we denote the T measurement results by B_1, B_2, to B_T, respectively, where each B_t is either 0 or 1. It is convenient to further group the T measurements as a T-dimensional vector $\vec{B} = (B_1, \cdots, B_T)$

For example, if $T = 4$ and $\vec{B} = (1, 0, 1, 1)$, it means that for this particular pixel, we take $T = 4$ independent measurements. In the second measurement, the sensor did not receive any photon at all. But for the first, third, and fourth measurements, the sensor received at least one photon. Note that the sensor could have received > 1 photons. Nonetheless, because of the hardware limitation, it can only tell whether no photon has arrived or whether there are ≥ 1 photons.

What is the probability $P(\vec{B} = (1, 0, 1, 1))$? Your answer should be a function of α since we do not know the α value.

(c) Do you know how to find the probability $P(\vec{B} = (1, 0, 1, 0, 0, 0, 1, 0))$ in terms of the α value? Do you know how to find the probability $P(\vec{B} = (0, 0, 1, 1, 0, 1, 1, 1))$ in terms of the α value? Please repeat this questions several times for different 8-dimensional vector values until you are fully confident how to find the $P(\vec{B} = \vec{b})$ for any arbitrarily given 8-dimensional vector $\vec{b}$.

(d) For arbitrary T and arbitrary vector $\vec{b}$, find the closed-form expression of the probability $P(\vec{B} = \vec{b})$. Make sure you know how to answer this question for arbitrarily given α, T, and $\vec{b}$ values. You should not proceed unless you know how to solve this question.

(e) Suppose we receive $\vec{B} = \vec{b}$ for some T-dimensional $\vec{b}$ vector. Recall that your answer of the previous question depends on the α value, i.e., $P(\vec{B} = \vec{b}) = f(\alpha)$ for some function $f(\alpha)$. If I received a T-dimensional vector $\vec{B} = \vec{b}$, then the "most likely α" should be the α value that maximizes the $f(\alpha)$ value.

Question: Prove that the following choice of α^* will maximize the $f(\alpha)$ value found in question (d).

$$\alpha^* = -\log\left(1 - \frac{\sum_{t=1}^{T} b_t}{T}\right). \tag{1}$$

Hint: you can use basic calculus to find the α^* value that maximizes $f(\alpha)$.

• **The actual MATLAB/Python project:** Please complete the following Deliverables #1 to #9 when submitting the results of your project.

The previous sub-questions have paved the mathematical foundation of how to complete the following project. Here are the detailed steps of this project and the corresponding deliverables.

1. Download an arbitrary image from the Internet. Depending on the memory size of your PC, you may want to start with a small picture, say 1000 by 750, or some comparably-sized ones.

   Deliverable #1: Please print/display the image you have downloaded from the Internet.

2. Please use the downloaded image and turn it into a grayscale image, which can be achieved by the reference Python program Lines 8 – 12 given in the end of this project description.

   Deliverable #2: Please save the output of the grayscale image. Print (copy/paste) that grayscale image as your answer for this question.

3. Please turn the grayscale image into a grayscale array with values between 0.0 and 6.0, which can be achieved by the reference Python program Lines 14 – 21.

   Also in the given code, Line 24 of that program uses the grayscale array as the α values of the Poisson distributions (each entry of the array is the α value for one pixel). Line 24 then uses the computer to "simulate" the Poisson distributions and generates the actual number of photon arrivals for each pixel.

   Please write down a short code of yourself to convert the number of arrivals for each pixel to a binary image based on the above idea of single-photon detectors. And save the binary image into a file. (Please refer to Lines 26 – 36 of the Python code below.)

   Repeat the above process for $T = 3$ times. Namely, generate 3 arrays of photon arrivals and their corresponding binary images by Lines 24 – 36.

   Deliverable #3: Print (copy/paste) all three binary images you have created thus far.

   Deliverable #4: Answer the following question: Are the three binary images pixel-by-pixel identical? It is a yes/no question. Please also add a few sentences justifying your answer.

4. Repeat the above process for $T = 50$ and you will thus have 50 binary images. **The main task at hand is how we can use the 50 binary images to synthesize the original picture.** We will provide two different methods: One is a heuristic one and one is a probability-based one. You should compare the performance of these two competing methods by yourself.

5. **Method #1:** Because we have $T = 50$ binary images, we can take the pixel-by-pixel average of these $T = 50$ binary images directly. Namely, for each pixel, we compute the average of $T = 50$ pixels (one from each binary image) by

$$\mathsf{avg.b} = \frac{\sum_{t=1}^{T} b_t}{T} \tag{2}$$

Note that each average avg.b is in the range of $[0, 1]$ because the value of each b_t is either 0 or 1. Since each pixel of a grayscale image has a range [0,255], we set the actual pixel value to be

$$\mathsf{pixel.value} = \mathsf{avg.b} \cdot 255 \tag{3}$$

so that the pixel.value is now distributed[1] between 0 to 255. Repeat this averaging process for all pixels. (Please refer to Lines 39 – 50 of the Python code below.)

Deliverable #5: Display (copy/paste) the result of yours generated by this simple average-based method.

6. **Method #2:** We apply the following process to each pixel separately. Specifically, we use the method previously discussed in Eq. (1) to reconstruct the best α^* value for each pixel.

Please write down a short code to find the best α^* value for each pixel using the formula in Eq. (1). Then Lines 56 – 66 convert the α^* values back to a grayscale image.

Deliverable #6: Display (copy/paste) the result of yours when using this probability-based method.

7. Repeat the steps of Deliverables #5 and #6 except that this time we choose $T = 1000$. Namely, we like to combine 1000 binary images (instead of 50 binary images) this time.

Deliverable #7: Please display (copy/paste) your grayscale image generated by Method #1 with $T = 1000$.

Deliverable #8: Please display (copy/paste) your grayscale image generated by Method #2 with $T = 1000$.

Deliverable #9: Please attach your complete program code for this experiment of $T = 1000$.

[1] In fact, because each pixel value is usually an integer. A more precise formula is $\mathsf{pixel.value} = \lfloor \mathsf{avg.b} \cdot 255 \rfloor$ where $\lfloor \cdot \rfloor$ is the *floor* function.

A Python reference code is attached below.

```
import numpy as np
import numpy.random as npr
from PIL import Image

kappa = 6.0
T = 50 # the number of measurements

# Load the image and convert to grayscale
img = Image.open('original.jpg').convert('L')

# Save the original image in grayscale
img.save('original_grayscale.jpg')

# Convert image to numpy array
img = np.asarray(img)

# Convert to float {0.0,1.0,2.0,...,255.0}
img = img.astype(np.float32)

# Convert to values between 0 and kappa
img /= (255.0/kappa)

# Generate 1 photon image from the original using poisson distr.
photon_arrivals = npr.poisson(img, (1, img.shape[0], img.shape[1]))

# Please convert to binary images: B=0 if Y=0 and B=1 if Y>0
measurement_one = xxxxxxx

# Convert to range 0-255
binaryimg = measurement_one * 255.0

# Convert to uint8
binaryimg = binaryimg.astype(np.uint8)

# Save the binary images
Image.fromarray(binaryimg[0]).save('binary_image.jpg')


# Method 1
# Please compute the mean of the T binary images
avgb = xxxxxxx

# Convert to range 0-255
pixelv1 = avgb * 255.0

# Convert to uint8
pixelv1 = pixelv1.astype(np.uint8)

# Save the reconstructed image by Method 1
Image.fromarray(pixelv1).save('method1.jpg')
```

```
51
52 # Method 2
53 # Please use Eq. (1) to estimate alpha*
54 est_alpha = xxxxxxx
55
56 # Convert to range 0-255
57 est_alpha *= (255.0/kappa)
58
59 # Clip the pixel value within the range 0-255
60 est_alpha = np.clip(est_alpha, 0, 255)
61
62 # Convert to uint8
63 est_alpha = est_alpha.astype(np.uint8)
64
65 # Save the reconstructed image by Method 2
66 Image.fromarray(est_alpha).save('method2.jpg')
```