Image Processing

HW1

Lian Natour 207300443

Mohammad Mhneha 315649814

Problem1:

a) **Physical Considerations** are Sensor Size, a larger image sensor can support more pixels, which allows for higher resolution without sacrificing individual pixel size, smaller sensors may lead to smaller pixels, which can introduce noise and reduce image quality, especially in low light.
Also, Lens Quality, Higher megapixels require lenses capable of capturing fine details to avoid wasted resolution. And Light Sensitivity, more pixels on a small sensor can result in smaller individual pixels, which might struggle in low light.
Hardware Considerations are Processing Power Cameras or computers need powerful processors to handle large image files generated by high-resolution sensors or rendering software also, memory and storage, higher resolution images require more storage space, necessitating robust memory and storage solutions. And battery life, Capturing and processing high-resolution images consumes more energy, so hardware must account for power limitations.
Software Considerations is Image Processing Algorithms. Software must handle compression, noise reduction, and other post-processing effectively for high megapixel images.

b) The considerations for deciding how strong the quantization of an image should be including several factors. One important factor is the computer's processing power, as weaker (finer) quantization requires more processing to handle the extra detail. Limited memory is another consideration; older devices often had small storage capacity, so stronger quantization was used to save space by reducing the amount of data stored. The capabilities of the display are also important. older screens could not show a wide range of colors or brightness levels, so it made no sense to use finer quantization if the screen couldn't display the extra detail.

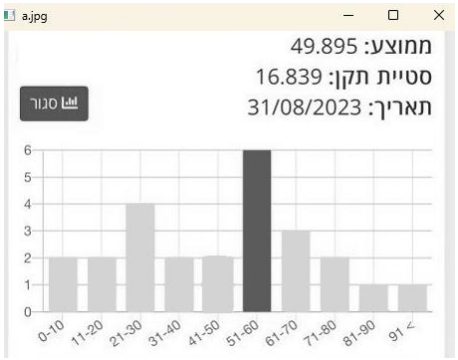
Problem 2:

a) For the function $\sin(\pi kx)$, the frequency of the wave is $f=k/2$ (measured in cycles per cm). The wavelength is the reciprocal of the frequency, which gives $\lambda=2/k$. Also, about the hint the variable A is not relevant in this question 😊

b)Based on part (a), the frequency of the sine wave is $f=k/2$. For A=2, the sampling interval is 2A=4 cm, meaning we take one sample every 4 cm. Thus, the sampling frequency is $f_{sample}=1/4f$. According to the Nyquist theorem, in order to accurately restore the signal without errors, the sampling frequency must satisfy $f_{sample}\geq 2f$. Substituting $f=k/2$, we get $1/4\geq k$, meaning $k\leq 0.25$. For A=0.25, we sample twice within a

1 cm unit (because 2A=0.5), giving a sampling frequency of fsample=2*f*. Again, applying the Nyquist condition fsample≥2f, we find 2≥k, meaning k≤2. This ensures that the signal can be fully restored without aliasing.

Problem 3:
a)we ran the code section a was already done and we got the first image in gray as it shows :



b)we updated the code to make sure that we can read the images of the digits

c)In this section, I implemented and tested the compare_hist function to identify digits in the histogram image. The function uses Earth Mover's Distance (EMD) to compare histograms within sliding windows of the histogram against target digit templates. To optimize the search, I defined a restricted search region (:130, 15:50) that focuses on the area where digits are expected to appear. The function iterates over the digit templates (0 to 9) and checks if they are present in the histogram, printing the results for each digit. Additionally, I visualized the search region using OpenCV to ensure it includes the relevant digits. The function successfully identified the digit '6' in the first histogram, validating the implementation.

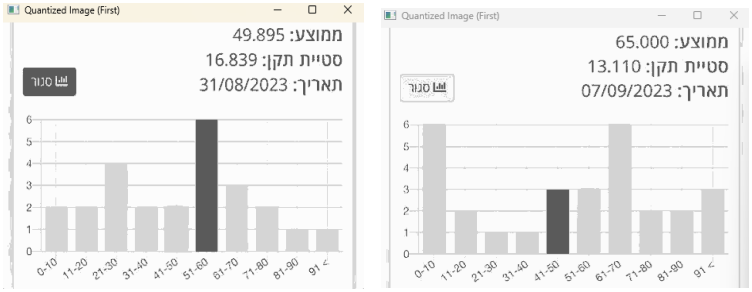
d)In this section, I extended the functionality of the compare_hist function to identify the highest digit present in each histogram image. For each histogram, I tested the digits from 9 to 0 using the compare_hist function. Starting with the largest digit ensures efficiency, as no smaller digit can be detected once a higher digit is found. When the first digit is successfully detected, the process stops, and the detected digit is stored and printed. This process is repeated for all histogram images in the data folder. For example, if the digit 6 is detected in a histogram, the function skips testing digits 5 to 0. This method ensures both accuracy and computational efficiency.
Here is the code and the output:

```
C:\Users\liann\Desktop\uni\Semó_Winter\ImageProcessing\Hws\Hw1\.venv\Scripts\python.exe C:\Users\liann\Desktop\uni\Semó_Winter\ImageProcessing\Hws\Hw1\TranscribeHistogram.py
Histogram a.jpg detected the number: 6
Histogram b.jpg detected the number: 6
Histogram c.jpg detected the number: 1
Histogram d.jpg detected the number: 6
Histogram e.jpg detected the number: 5
Histogram f.jpg detected the number: 4
Histogram g.jpg detected the number: 9

Process finished with exit code 0
```

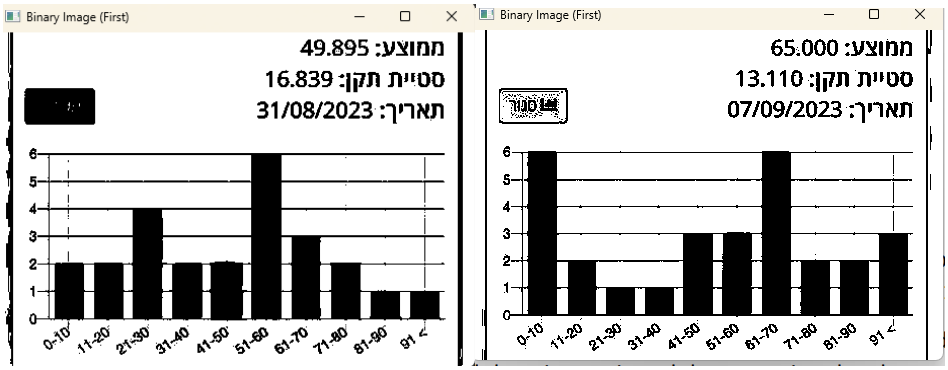
```python
#====================================
#section d
# Iterate over all histogram images in 'images'
for img_id, src_image in enumerate(images):
    detected_number = None  # Variable to store the detected number
    for digit_id in range(9, -1, -1):  # Test digits from 9 to 0
        target_image = numbers[digit_id]
        is_present = compare_hist(src_image, target_image)
        if is_present:
            detected_number = digit_id  # Store the detected number
            break  # Stop once the first number is found

    # Print the result for the current histogram
    print(f"Histogram {names[img_id]} detected the number: {detected_number}")
#====================================
```

e)To simplify the histogram images, I first reduced the number of gray levels to 3 This number was chosen after testing various values, as it struck the ideal balance between simplicity and preserving the necessary details. With 3 gray levels, the bars remained visually distinct from the background, simplifying further processing. Using fewer levels, like 2, resulted in the loss of important details in the bars, while using more levels, like 4 or higher, added unnecessary complexity without improving clarity. After quantizing the images, I applied a threshold value of 240 to convert the images to black and white. This value was selected because it clearly separated the bars (black) from the background (white), ensuring no loss of bar details. Lower threshold values, such as 220, sometimes merged parts of the bars with the background, while higher values, such as 250, included extra noise.

Here we changed the gray color:



and we turned the image to black and white:



f) In this section, I calculated the heights of all 10 bars in each histogram using the modified (binary) images from section (e). For each image, I created a function get_bar_heights that iterates through all bins (0-9) and measures the height of each bar in pixels using the get_bar_height function. The resulting heights for all bars in an image were stored in a list.This process was repeated for all histograms, and the heights were stored in a master list, all_bar_heights, for further use.I also did a small change in get_bar_height so the black is zero.

```
C:\Users\liann\Desktop\uni\Semó_Winter\ImageProcessing\Hws\Hw1\.venv\Scripts\python.exe C:\Users\liann\Desktop\uni\Semó_Winter\ImageProcessing\Hws\Hw1\TranscribeHistogram.py
Bar heights for Histogram a.jpg: [48, 48, 102, 48, 49, 156, 75, 48, 20, 20]
Bar heights for Histogram b.jpg: [154, 45, 18, 18, 72, 73, 154, 45, 45, 72]
Bar heights for Histogram c.jpg: [0, 0, 0, 0, 0, 0, 157, 157, 157, 157]
Bar heights for Histogram d.jpg: [19, 0, 46, 73, 100, 73, 127, 127, 154, 46]
Bar heights for Histogram e.jpg: [58, 26, 26, 91, 59, 156, 26, 26, 59, 91]
Bar heights for Histogram f.jpg: [33, 0, 33, 33, 33, 156, 33, 33, 74, 33]
Bar heights for Histogram g.jpg: [12, 12, 12, 48, 12, 30, 157, 48, 48, 0]
```

g) we calculated the actual number of students in each bin using the formula you gave us . This scaled the relative bar heights to real student counts. The results were transcribed and printed for all histograms.

```
C:\Users\liann\Desktop\uni\Semó_Winter\ImageProcessing\Hws\Hw1\.venv\Scripts\python.exe C:\Users\liann\Desktop\uni\Semó_Winter\ImageProcessing\Hws\Hw1\TranscribeHistogram.py
Histogram a.jpg gave 2,2,4,2,2,6,3,2,1,1
Histogram b.jpg gave 6,2,1,1,3,3,6,2,2,3
Histogram c.jpg gave 0,0,0,0,0,0,1,1,1,1
Histogram d.jpg gave 1,0,2,3,4,3,5,5,6,2
Histogram e.jpg gave 2,1,1,3,2,5,1,1,2,3
Histogram f.jpg gave 1,0,1,1,1,4,1,1,2,1
Histogram g.jpg gave 1,1,1,3,1,2,9,3,3,0

Process finished with exit code 0
```