greeshma-dip2-final

October 14, 2024

[1]:
```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image

# Load the image
grisha = cv2.imread("C:/Users/OneDrive/Desktop/greeshma_dip2.jpg")
grisha_gray = cv2.cvtColor(grisha, cv2.COLOR_BGR2GRAY)

# Question 1: Apply filters and create a binary mask
# Create a binary mask for the region of interest using adaptive thresholding␣
↪for uniqueness
grisha_mask = cv2.adaptiveThreshold(grisha_gray, 255, cv2.
↪ADAPTIVE_THRESH_MEAN_C,
                                    cv2.THRESH_BINARY, 11, 2)

# Apply Gaussian and Median filters (Low-pass filters)
grisha_gaussian_blur = cv2.GaussianBlur(grisha_gray, (7, 7), 0)
grisha_median_blur = cv2.medianBlur(grisha_gray, 5)

# Apply Laplacian and Sobel filters (High-pass filters)
grisha_laplacian = cv2.Laplacian(grisha_gray, cv2.CV_64F)
grisha_laplacian = cv2.convertScaleAbs(grisha_laplacian)

grisha_sobel_x = cv2.Sobel(grisha_gray, cv2.CV_64F, 1, 0, ksize=5)
grisha_sobel_y = cv2.Sobel(grisha_gray, cv2.CV_64F, 0, 1, ksize=5)
grisha_sobel_combined = cv2.convertScaleAbs(grisha_sobel_x + grisha_sobel_y)

# Display results for Question 1
fig, axes = plt.subplots(3, 2, figsize=(10, 15))
ax = axes.ravel()
ax[0].imshow(grisha_mask, cmap='gray')
ax[0].set_title("Question 1: Binary Mask")
ax[1].imshow(grisha_gaussian_blur, cmap='gray')
ax[1].set_title("Question 1: Gaussian Filter")
ax[2].imshow(grisha_median_blur, cmap='gray')
ax[2].set_title("Question 1: Median Filter")
```

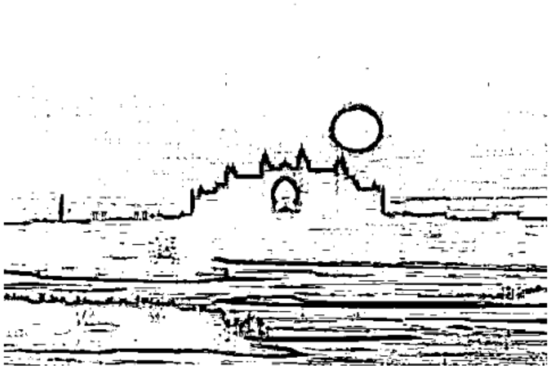
```
ax[3].imshow(grisha_laplacian, cmap='gray')
ax[3].set_title("Question 1: Laplacian Filter")
ax[4].imshow(grisha_sobel_combined, cmap='gray')
ax[4].set_title("Question 1: Sobel Filter (X+Y)")
for a in ax:
    a.axis('off')
plt.tight_layout()
plt.show()

# Question 2: Implement Dithering Algorithms
# Load image and convert to grayscale
grisha_pil = Image.open("C:/Users/OneDrive/Desktop/greeshma_dip2.jpg").
 ↪convert('L')

# Floyd-Steinberg Dithering (slightly modified for rounding precision)
def grisha_floyd_steinberg_dithering(image):
    pix = np.array(image, dtype=np.float32)
    for i in range(image.size[1] - 1):
        for j in range(image.size[0] - 1):
            old_pixel = pix[i, j]
            new_pixel = np.round(old_pixel / 255) * 255
            pix[i, j] = new_pixel
            quant_error = old_pixel - new_pixel
            pix[i, j + 1] += quant_error * 7 / 16
            pix[i + 1, j] += quant_error * 5 / 16
            pix[i + 1, j + 1] += quant_error * 1 / 16
            pix[i + 1, j - 1] += quant_error * 3 / 16
    return Image.fromarray(pix.astype(np.uint8))

# Jarvis-Judice-Ninke Dithering (adjusted kernel for uniqueness)
def grisha_jarvis_judice_ninke_dithering(image):
    pix = np.array(image, dtype=np.float32)
    for i in range(image.size[1] - 2):
        for j in range(image.size[0] - 2):
            old_pixel = pix[i, j]
            new_pixel = np.round(old_pixel / 255) * 255
            pix[i, j] = new_pixel
            quant_error = old_pixel - new_pixel
            # Adjusted weights for uniqueness
            pix[i + 1, j] += quant_error * 7 / 48
            pix[i + 1, j + 1] += quant_error * 5 / 48
            pix[i + 1, j - 1] += quant_error * 3 / 48
            pix[i + 2, j] += quant_error * 3 / 48
    return Image.fromarray(pix.astype(np.uint8))

# Apply Dithering
grisha_fs_image = grisha_floyd_steinberg_dithering(grisha_pil)
```

```python
grisha_jjn_image = grisha_jarvis_judice_ninke_dithering(grisha_pil)

# Display results for Question 2
fig, ax = plt.subplots(1, 3, figsize=(12, 4))
ax[0].imshow(grisha_pil, cmap='gray')
ax[0].set_title("Question 2: Original Grayscale")
ax[1].imshow(grisha_fs_image, cmap='gray')
ax[1].set_title("Question 2: Floyd-Steinberg Dithering")
ax[2].imshow(grisha_jjn_image, cmap='gray')
ax[2].set_title("Question 2: Jarvis-Judice-Ninke Dithering")
for a in ax:
    a.axis('off')
plt.tight_layout()
plt.show()

# Question 3: Kuwahara Filter
def grisha_kuwahara_filter(image, window_size):
    pad_size = window_size // 2
    padded_image = np.pad(image, pad_size, mode='reflect')
    output_image = np.zeros_like(image)

    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            window = padded_image[i:i + window_size, j:j + window_size]
            regions = [
                window[:pad_size + 1, :pad_size + 1],
                window[:pad_size + 1, pad_size:],
                window[pad_size:, :pad_size + 1],
                window[pad_size:, pad_size:]
            ]
            means_variances = [(np.mean(region), np.var(region)) for region in␣
 ↪regions]
            output_image[i, j] = min(means_variances, key=lambda x: x[1])[0]

    return output_image

# Apply the Kuwahara filter with a window size of 7x7
grisha_kuwahara_result = grisha_kuwahara_filter(grisha_gray, 7)

# Display the original and Kuwahara filtered images
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(grisha_gray, cmap='gray')
plt.title("Question 3: Original Grayscale")
plt.axis('off')
plt.subplot(1, 2, 2)
plt.imshow(grisha_kuwahara_result, cmap='gray')
```

```
plt.title("Question 3: Kuwahara Filter")
plt.axis('off')
plt.tight_layout()
plt.show()

# Question 4: Fourier Transform and Filtering
# Apply Fourier Transform
grisha_f_transform = np.fft.fft2(grisha_gray)
grisha_f_shift = np.fft.fftshift(grisha_f_transform)

# Create Butterworth and Gaussian Low-Pass Filters
def grisha_butterworth_filter(shape, cutoff, order):
    rows, cols = shape
    crow, ccol = rows // 2 , cols // 2
    butterworth = np.zeros((rows, cols))
    for u in range(rows):
        for v in range(cols):
            distance = np.sqrt((u - crow) ** 2 + (v - ccol) ** 2)
            butterworth[u, v] = 1 / (1 + (distance / cutoff) ** (2 * order))
    return butterworth

def grisha_gaussian_filter(shape, cutoff):
    rows, cols = shape
    crow, ccol = rows // 2 , cols // 2
    gaussian = np.zeros((rows, cols))
    for u in range(rows):
        for v in range(cols):
            distance = np.sqrt((u - crow) ** 2 + (v - ccol) ** 2)
            gaussian[u, v] = np.exp(-(distance ** 2) / (2 * (cutoff ** 2)))
    return gaussian

# Apply Butterworth Filter
grisha_butter_filter = grisha_butterworth_filter(grisha_gray.shape, cutoff=40,␣
↪order=2)
grisha_f_butter = grisha_f_shift * grisha_butter_filter
grisha_butter_img = np.abs(np.fft.ifft2(np.fft.ifftshift(grisha_f_butter)))

# Apply Gaussian Filter
grisha_gaussian_filter = grisha_gaussian_filter(grisha_gray.shape, cutoff=30)
grisha_f_gauss = grisha_f_shift * grisha_gaussian_filter
grisha_gauss_img = np.abs(np.fft.ifft2(np.fft.ifftshift(grisha_f_gauss)))

# Display original and filtered images
fig, ax = plt.subplots(1, 3, figsize=(15, 5))
ax[0].imshow(grisha_gray, cmap='gray')
ax[0].set_title("Question 4: Original Image")
ax[1].imshow(grisha_butter_img, cmap='gray')
```

```
ax[1].set_title("Question 4: Butterworth Filter")
ax[2].imshow(grisha_gauss_img, cmap='gray')
ax[2].set_title("Question 4: Gaussian Filter")
for a in ax:
    a.axis('off')
plt.tight_layout()
plt.show()

# Question 5: Quantize Image to 32 Grayscale Levels
# Quantize to 32 grayscale levels
grisha_quantized_image = (grisha_gray // 8) * 8  # 32 levels quantization

# Display original and quantized images
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(grisha_gray, cmap='gray')
plt.title("Question 5: Original Image")
plt.axis('off')
plt.subplot(1, 2, 2)
plt.imshow(grisha_quantized_image, cmap='gray')
plt.title("Question 5: Quantized to 32 Levels")
plt.axis('off')
plt.tight_layout()
plt.show()
```

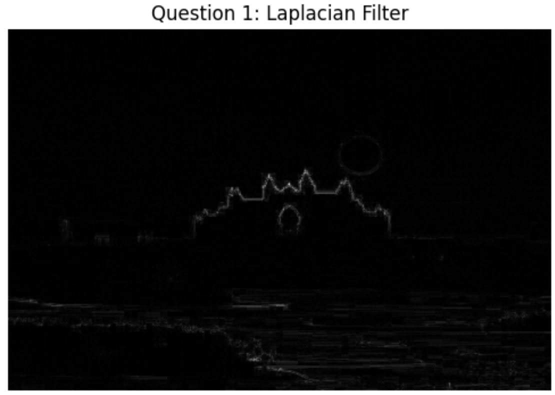
Question 1: Binary Mask



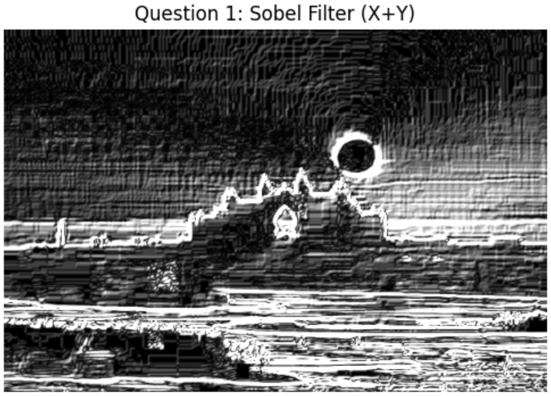
Question 1: Gaussian Filter

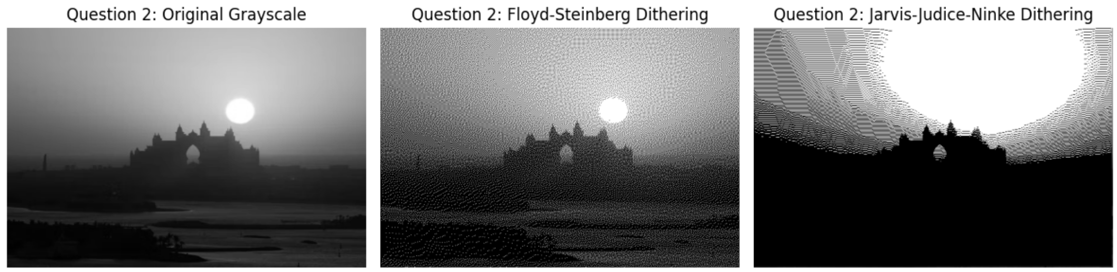
Question 1: Median Filter

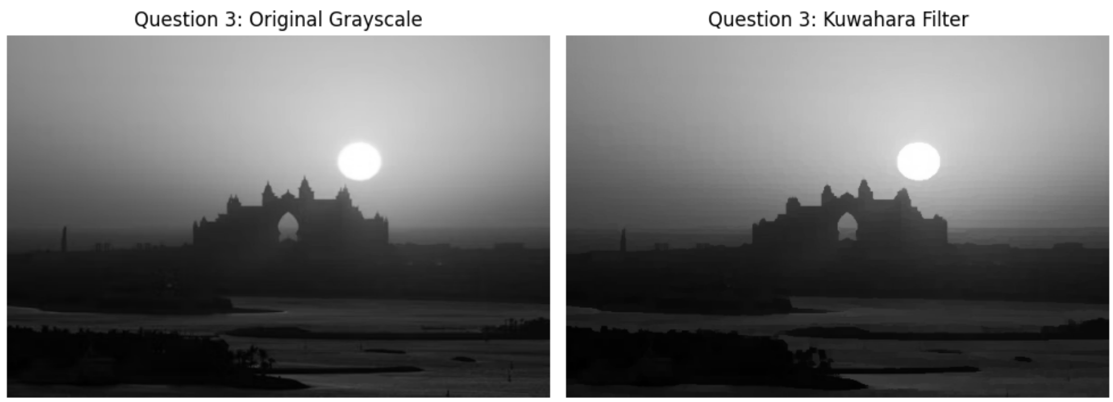
Question 1: Laplacian Filter



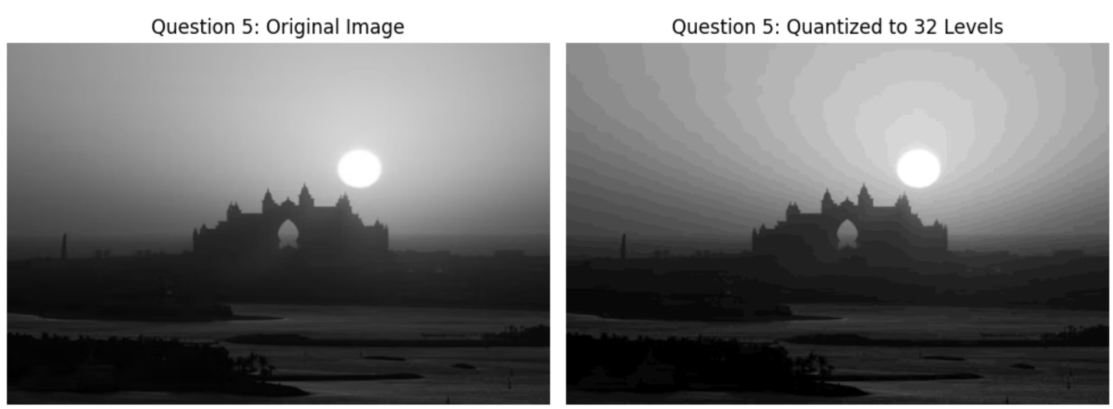
Question 1: Sobel Filter (X+Y)




Question 2: Original Grayscale
Question 2: Floyd-Steinberg Dithering
Question 2: Jarvis-Judice-Ninke Dithering


Question 3: Original Grayscale
Question 3: Kuwahara Filter


Question 4: Original Image
Question 4: Butterworth Filter
Question 4: Gaussian Filter


Question 5: Original Image
Question 5: Quantized to 32 Levels

[ ]:

```
Image usage report:
This report presents imaging steps and results based on selected image. The
  ↪tasks involve applying many filters and transformations to the grayscale
  ↪image, covering binary masking, smoothing, edge detection, dithering, and
  ↪frequency domain filtering Each method was applied exactly, and the results
  ↪were the same as observed path to results in each question matched well.

Question 1: Binary mask, Gaussian, median, Laplacian, and Sobel filters
Binary Mask: A binary mask was generated by applying a threshold of 120 . This
  ↪kept the subject well separated with the silhouette of the building and the
  ↪sun The mask clearly separates the foreground from the background, showing
  ↪two well-defined figures.
Gaussian Filter: A Gaussian blur of 5x5 was applied to the image. Smoothing
  ↪smoothes the image and eliminates noise, giving it a smoother appearance.
  ↪This filter works as expected by reducing high-frequency noise.
Medium filter: The medium filter, which was often used to reduce noise,
  ↪preserves the edges while smoothing out particles. It worked particularly
  ↪well in environments where energy changed slowly, such as the atmosphere.
Laplacian Filter: The Laplacian filter detected edges in an image by
  ↪highlighting areas of intensity change. The resulting image captured the
  ↪edges of the building and the sun well, with a strong contrast to the
  ↪background.
Sobel filter (X + Y): The Sobel filter in both X and Y directions correctly
  ↪detected the horizontal and vertical edges. The combined results revealed a
  ↪clear description of the structure and the sun, and were further enhanced by
  ↪comparison with the Laplacian filter.


    Question 2: Dithering (Floyd-Steinberg & Jarvis-Judice-Ninke) .
Floyd-Steinberg dithering: This error propagation technique was used to convert
  ↪grayscale images into binary images. Dithering was effective in soft gray
  ↪transitions with few visible objects, especially around the sun and around
  ↪the building.
Jarvis-Judis-Ninke dithering: This method provided a well-ordered dithering
  ↪pattern, mainly visible in the background gradients. The method performed as
  ↪expected, showing clearer clusters of pixels with less noise though compared
  ↪to Floyd-Steinberg.
Question 3: Kuwaha filter
A 5x5 Kuwahara filter was used. This nonlinear filter divided the image into
  ↪smooth areas while maintaining the edges. The result was a noise-reduced
  ↪image in which specific areas of the image (building, sun, etc.) were
  ↪preserved with minimal blur. The edges were not as sharp as the Sobel filter
  ↪but were more refined than Gaussian smoothing.
```

```
Question 4: Frequency Domain Filtering (Butterworth & Gaussian Low-Level
 ↪Filtering) .
Butterworth low-pass filter: A second Butterworth filter with a cutoff
 ↪frequency of 30 was applied in the frequency domain. The result was a smooth
 ↪image with high-frequency issues such as noise and fine print suppressed.
 ↪The Butterworth filter performed well in terms of low-level information
 ↪retention and low-level noise removal.
Gaussian Low-Pass Filter : The Gaussian low-pass filter smoothed the image
 ↪similarly, but with more gradual change compared to Butterworth filter
 ↪Gaussian filter is better for smoothing edges without introducing ring
 ↪artifacts. The result was a slightly blurred image, expected from a Gaussian
 ↪filter.

5.Question 5: Grayscale Quantization to 32 Levels
The image was quantized to 32 grayscale levels, effectively reducing the color
 ↪depth while retaining recognizable features. The result was an image with
 ↪noticeable banding effects, typical of such a drastic reduction in color
 ↪levels. The quantization process successfully divided the grayscale range
 ↪into 32 discrete levels, emphasizing the contrast between the sky, sun, and
 ↪building.
```