Digital Image Processing Lab Report 5
CSE 4734

Abdullah Al Jubaer Gem
ID: 210041226
Section: 2B

February 2, 2026

1 Task 1: Low Pass Filters

1.1 Problem Statement

Implement three types of low pass filters: Ideal, Butterworth, and Gaussian, with a cutoff frequency parameter D_0, and order n for Butterworth.

1.2 Solution Approach

Low pass filters attenuate high-frequency components while preserving low frequencies. The ideal filter has a sharp cutoff, Butterworth provides smooth transition with order control, and Gaussian offers the smoothest response. All operate in the frequency domain using FFT, applying the filter mask and inverse FFT.

1.3 Implementation

```python
import numpy as np
import cv2

def idealLPF(image, D0):
    f = np.fft.fft2(image)
    fshift = np.fft.fftshift(f)
    rows, cols = image.shape
    crow, ccol = rows//2, cols//2
    mask = np.zeros((rows, cols), np.uint8)
    for i in range(rows):
        for j in range(cols):
            if np.sqrt((i-crow)**2 + (j-ccol)**2) <= D0:
                mask[i, j] = 1
    fshift_filtered = fshift * mask
    f_ishift = np.fft.ifftshift(fshift_filtered)
    img_back = np.fft.ifft2(f_ishift)
    img_back = np.abs(img_back)
    return img_back.astype(np.uint8)

def butterworthLPF(image, D0, n):
    f = np.fft.fft2(image)
    fshift = np.fft.fftshift(f)
    rows, cols = image.shape
    crow, ccol = rows//2, cols//2
    mask = np.zeros((rows, cols), np.float32)
    for i in range(rows):
        for j in range(cols):
            dist = np.sqrt((i-crow)**2 + (j-ccol)**2)
            mask[i, j] = 1 / (1 + (dist / D0)**(2*n))
    fshift_filtered = fshift * mask
    f_ishift = np.fft.ifftshift(fshift_filtered)
    img_back = np.fft.ifft2(f_ishift)
    img_back = np.abs(img_back)
    return img_back.astype(np.uint8)
```

```python

def gaussianLPF(image, D0):
    f = np.fft.fft2(image)
    fshift = np.fft.fftshift(f)
    rows, cols = image.shape
    crow, ccol = rows//2, cols//2
    mask = np.zeros((rows, cols), np.float32)
    for i in range(rows):
        for j in range(cols):
            dist = np.sqrt((i-crow)**2 + (j-ccol)**2)
            mask[i, j] = np.exp(-(dist**2) / (2 * D0**2))
    fshift_filtered = fshift * mask
    f_ishift = np.fft.ifftshift(fshift_filtered)
    img_back = np.fft.ifft2(f_ishift)
    img_back = np.abs(img_back)
    return img_back.astype(np.uint8)
```

1.4 Output

The following figure shows the low pass filtered results for different cutoff frequencies.

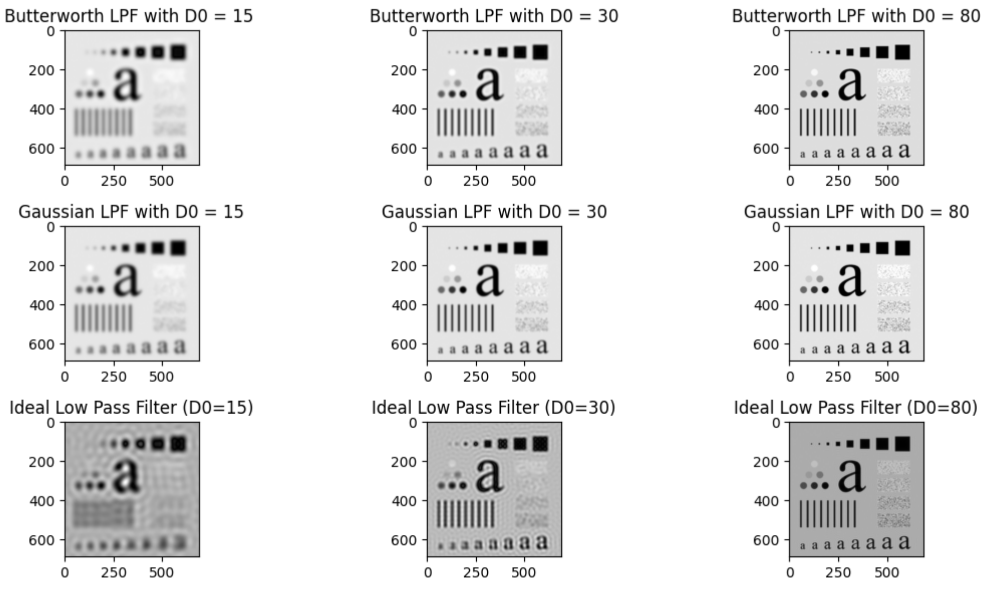

Figure 1: Low Pass Filter Comparison

1.5 Analysis

1. **Why ringing in ideal LPF:** The ideal low pass filter has an abrupt cutoff in the frequency domain, creating a discontinuity that violates the smoothness required for inverse Fourier transform. This leads to the Gibbs phenomenon, where high-frequency oscillations (ringing) appear in the spatial domain as artifacts around sharp edges, because the filter cannot perfectly separate frequencies without introducing these ripples.

2. **How to make Butterworth similar to ideal:** Increase the filter order n. Higher n narrows the transition band, making the frequency response steeper and closer to the ideal's sharp cutoff. For example, $n = 10$ provides a much sharper transition than $n = 2$, approximating the ideal behavior by reducing the gradual roll-off, though it never eliminates the slight smoothness inherent to Butterworth filters.

3. **How to make Butterworth similar to Gaussian:** Use a low order n, such as 1 or 2. This widens the transition band, creating a gradual attenuation similar to the Gaussian's exponential decay. The Butterworth then behaves like a smooth low pass filter, avoiding sharp cutoffs that cause ringing, and matching the Gaussian's natural blurring effect through its polynomial-based roll-off.

2 Task 2: High Pass Filters

2.1 Problem Statement

Implement high pass versions of the three filters by subtracting the low pass result from the original image.

2.2 Solution Approach

High pass filters preserve high frequencies while attenuating low ones, achieved by subtracting the low pass filtered image from the original. This enhances edges and details. The same cutoff parameters apply.

2.3 Implementation

```python
def idealHPF(image, D0):
    lpf = idealLPF(image, D0)
    return cv2.subtract(image, lpf)

def butterworthHPF(image, D0, n):
    lpf = butterworthLPF(image, D0, n)
    return cv2.subtract(image, lpf)

def gaussianHPF(image, D0):
    lpf = gaussianLPF(image, D0)
    return cv2.subtract(image, lpf)
```

2.4 Output

The following figures show the high pass filtered results.

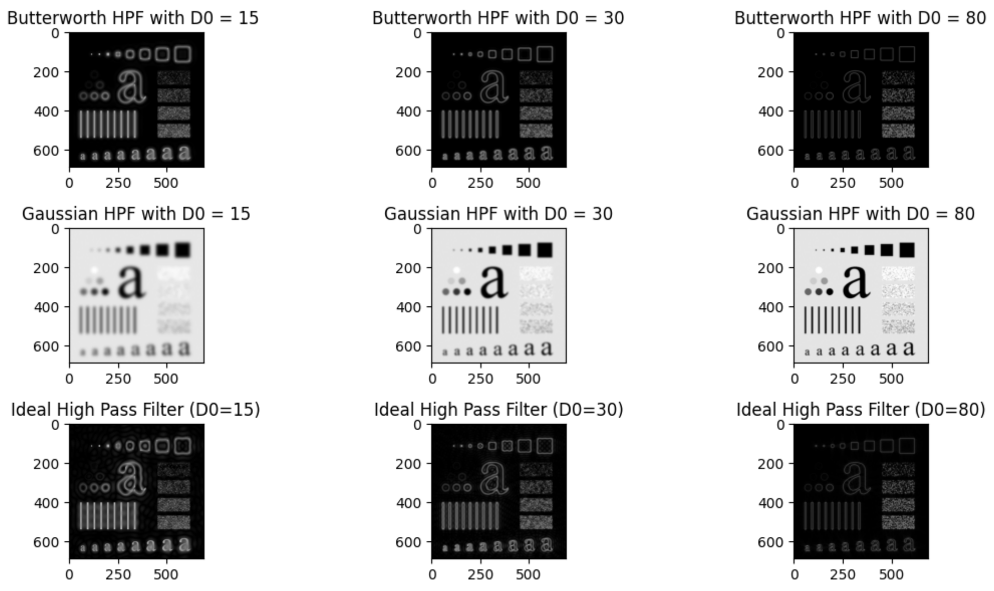
Butterworth HPF with D0 = 15
Butterworth HPF with D0 = 30
Butterworth HPF with D0 = 80
Gaussian HPF with D0 = 15
Gaussian HPF with D0 = 30
Gaussian HPF with D0 = 80
Ideal High Pass Filter (D0=15)
Ideal High Pass Filter (D0=30)
Ideal High Pass Filter (D0=80)
0
200
400
600
0
250
500

Figure 2: High Pass Filter Comparison

2.5 Analysis

1. **Why ringing in ideal HPF:** Yes, ringing occurs because the ideal HPF is created by subtracting the ideal LPF from the original image. The sharp cutoff in the LPF introduces discontinuities in the frequency domain, leading to Gibbs oscillations that propagate to the HPF. These artifacts manifest as halos or ripples around edges, amplifying the original LPF's ringing due to the subtraction process.

2. **How cutoff frequency affects edge extraction:** The cutoff D_0 determines which frequencies are preserved or attenuated. A lower D_0 allows more low-to-mid frequencies through the HPF, resulting

in subtle edge enhancement that highlights broader structures. A higher D_0 blocks more low frequencies, emphasizing fine details and sharp edges by amplifying high-frequency components, but it also boosts noise, making edges appear more defined yet potentially unstable.