import numpy as np

import matplotlib.pyplot as plt

# Simple find\_peaks function

def simple\_find\_peaks(signal):

peaks = []

for i in range(1, len(signal) - 1):

if signal[i] > signal[i - 1] and signal[i] > signal[i + 1]:

peaks.append(i)

return peaks

# Linear interpolation based on given peaks

def simple\_linear\_interpolate(signal, peaks):

interpolated = np.zeros\_like(signal)

for i in range(1, len(peaks)):

start\_idx = peaks[i - 1]

end\_idx = peaks[i]

start\_val = signal[start\_idx]

end\_val = signal[end\_idx]

# Linear interpolation between peaks

step = (end\_val - start\_val) / (end\_idx - start\_idx)

for j in range(start\_idx, end\_idx + 1):

interpolated[j] = start\_val + step \* (j - start\_idx)

return interpolated

# Simple thresholding function

def simple\_thresholding(imf, threshold):

return np.where(np.abs(imf) > threshold, imf, 0)

# A simple decomposition function

def simple\_lmd\_decomposition(signal):

peaks = simple\_find\_peaks(signal)

interpolated\_signal = simple\_linear\_interpolate(signal, peaks)

imf = signal - interpolated\_signal

return imf

# Reconstructs a signal from a list of IMFs

def simple\_reconstruct\_signal(imfs):

return np.sum(imfs, axis=0)

# Apply median filter (simplified)

def simple\_median\_filter(signal, filter\_size):

filtered\_signal = np.copy(signal)

half\_size = filter\_size // 2

for i in range(half\_size, len(signal) - half\_size):

filtered\_signal[i] = np.median(signal[i - half\_size:i + half\_size + 1])

return filtered\_signal

# Denoising using iterative LMD and thresholding

def simple\_iterative\_lmd\_thresholding(signal, max\_iterations, threshold):

residue = np.copy(signal)

for \_ in range(max\_iterations):

imf = simple\_lmd\_decomposition(residue)

denoised\_imf = simple\_thresholding(imf, threshold)

residue -= denoised\_imf

return simple\_reconstruct\_signal([denoised\_imf])

# Combined denoising method

def simple\_combined\_denoising(signal, max\_iterations, lmd\_threshold, median\_filter\_size):

# Iterative LMD with thresholding

denoised\_signal\_lmd = simple\_iterative\_lmd\_thresholding(signal, max\_iterations, lmd\_threshold)

# Apply median filter

denoised\_signal\_median = simple\_median\_filter(signal, median\_filter\_size)

# Combine both results

denoised\_signal\_combined = (denoised\_signal\_lmd + denoised\_signal\_median) / 2

return denoised\_signal\_combined

# Example usage with LiDAR dataset

if \_name\_ == "\_main\_":

# Load LiDAR dataset from a .txt file

dataset\_file = "0000000000.txt" # Update this with your file path

lidar\_data = np.loadtxt(dataset\_file)

# Extract time and signal values

time = lidar\_data[:, 0] # Assuming the first column represents time

signal = lidar\_data[:, 1] # Assuming the second column represents signal

max\_iterations = 5

lmd\_threshold = 15

median\_filter\_size = 21

# Denoise the signal

denoised\_signal\_combined = simple\_combined\_denoising(signal, max\_iterations, lmd\_threshold, median\_filter\_size)

# Plot the original signal and the denoised signal in two plots

plt.figure(figsize=(12, 8))

# Plot the original signal

plt.subplot(2, 1, 1)

plt.plot(time, signal, label='Original Signal', color='red')

plt.xlabel('Time')

plt.ylabel('Amplitude')

plt.title('Original LiDAR Signal')

plt.legend()

# Plot the denoised signal

plt.subplot(2, 1, 2)

plt.plot(time, denoised\_signal\_combined, label='Denoised Signal', color='green')

plt.xlabel('Time')

plt.ylabel('Amplitude')

plt.title('Denoised LiDAR Signal')

plt.legend()

plt.tight\_layout()

plt.show()

