ECE278C Imaging Systems

Lab 5

March 11, 2021

Student:	Ivan Arevalo
Perm Number:	5613567
Email:	ifa@ucsb.com

Department of Electrical and Computer Engineering, UCSB

0 Introduction

The main objective of this report is to reconstruct the image from GPR gathered data.

1 Single Coherent Point Source

Figure 1 shows the scattered wave field pattern for a single point source at the origin. The point source scattered pattern is mathematically described by Green's function:

$$h(x, y) = \frac{1}{\sqrt{j\lambda_0 r}} exp \frac{j2\pi r}{\lambda_0})$$

$$r = \sqrt{x^2 + y^2}$$

Figure 2 shows the magnitude spectrum of a coherent point source scattered pattern. We can observe it takes its highest values along a ring of radius $\frac{1}{\lambda}$. We can confirm that this magnitude spectrum agrees with the spectrum derived from the differential equation associated with wave propagation also known a Helmholtz equation.

$$H(f_x, f_y) = \delta\left(|f| - \frac{1}{\lambda_0}\right)$$

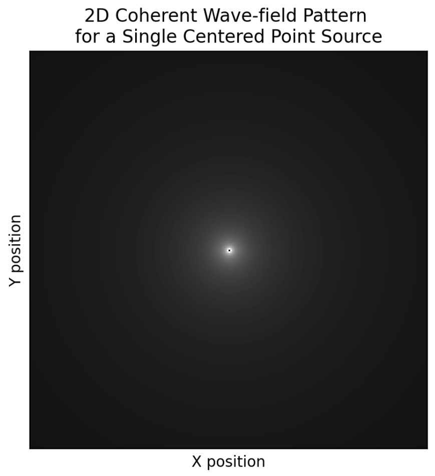

Figure 1: Scattered Wave-field

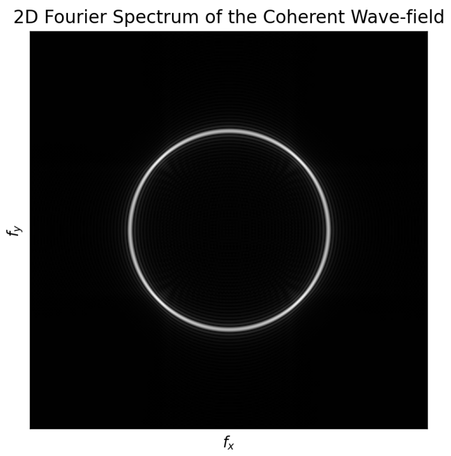

Figure 2: Magnitude of Spectrum

2 Image reconstruction

We now reconstruct our image by back-propagation as in Lab 4. The main difference now is that we have real recorded data at a receiver. The data set was taken over the walkway pavement in front of the Broida Hall at UCSB. The ground-penetrating radar imaging unit scanned along a linear path and took data at 200 spatial positions. The spatial spacing between the data-collection positions is 0.0213 m (2.13 cm).

At each data-collection position, the system illuminates the walkway pavement with microwaves in the step-frequency mode, stepping through 128 frequencies with a constant increment, from 0.976 GHz to 2.00 GHz. The relative permittivity ϵ_r is approximately 6.0.

For simplicity, we'll use the phase only version of green's theorem. Furthermore, we need to adjust the wavelength used in the calculation with the relative permittivity provided.

$$\lambda_i = \frac{C_0}{f_i\sqrt{\epsilon_r}}$$

Where C_0 is the speed of light. Now we just need to iteratively reconstruct and add the images from each calculated wavelength at each receiver location. Figure 3 shows the final reconstructed image.

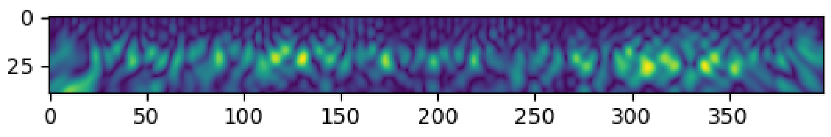

Figure 3: Reconstructed Image

3 Summary

Observing the reconstruction video https://youtu.be/USsWiHK-k7g, we can observe how the resolution in the y-direction gets better as we add reconstructed images from each of the 128 wavelengths. We had previously observed this in Lab 4, but we now implement it with real GPR gathered data.

4 Code

```
    import numpy as np
import matplotlib.pyplot as plt
import multiprocessing
import time
import os
from Lab1.wave_field import Wave_Field
from Lab2.image_reconstruction import LinearReceiver
from scipy.io import loadmat

def wave_field_tf(wavelength, amplitude, sample_x, sample_y, x, y):
    r = np.sqrt((x - sample_x) ** 2 + (y - sample_y) ** 2)

    return amplitude * np.exp(1j * 2 * np.pi * r / wavelength)

def generate_wave_field_pattern(wavelength, amplitude, sample_coordinate, spacing, xlim,
    ylim):

    x = np.arange(xlim[0], xlim[1] + spacing, spacing)
    y = np.arange(ylim[0], ylim[1] + spacing, spacing)

    X, Y = np.meshgrid(x, y)
    v_wave_field_tf = np.vectorize(wave_field_tf, cache=False)
    wave_field_pattern = v_wave_field_tf(wavelength, amplitude, sample_coordinate[0],
        sample_coordinate[1], X, np.rot90(Y, 2)) # Y because of numpy indexing start from
        top left.

    return np.nan_to_num(wave_field_pattern)


def main(wavelength, sample_spacing, GPR_data):
    rad = 100*sample_spacing - sample_spacing/2
    receiver_coordinates = np.linspace(-rad, rad, 200)

    first_flag = True
    for i, sample_coord in enumerate(receiver_coordinates):
        if first_flag:
            reconstruction = generate_wave_field_pattern(wavelength, GPR_data[i],
                (sample_coord, 0), sample_spacing/2, (-rad, rad), (-0.4, 0))
            first_flag = False
        else:
            reconstruction += generate_wave_field_pattern(wavelength, GPR_data[i],
                (sample_coord, 0), sample_spacing/2, (-rad, rad), (-0.4, 0))
    return reconstruction


if __name__ == "__main__":

    # Spacing in m, Wavelength in Ghz

    # frequencies = np.linspace(0.976,2,128) * 1 * 10**9
    frequencies = loadmat('gpr_data.mat')['f'].astype(np.float).squeeze()
    permitivity = 6

    velocity = 3*(10**8) / np.sqrt(permitivity)
    wavelengths = velocity / frequencies

    print(wavelengths)
    #
    start_time = time.time()
    #
    GPR_data = loadmat('gpr_data.mat')['F']
    # GPR_data = loadmat('gpr_data.mat')

    print(GPR_data.shape)

    first_w_flag = True
    for i in range(wavelengths.size):
        if first_w_flag:
            final_im = main(wavelengths[i], 0.0213, GPR_data[i, :])
            first_w_flag = False
        else:
            final_im += main(wavelengths[i], 0.0213, GPR_data[i, :])

        print(f"Iteration : {i}")

        if (i+1)%32 == 0:

            plt.imshow(abs(final_im))
            plt.savefig(f'{i}.png')
    plt.imshow(abs(final_im))
    plt.show()
```

```
    import numpy as np
import matplotlib.pyplot as plt
import multiprocessing
import time
import os
from Lab1.wave_field import Wave_Field
from Lab2.image_reconstruction import LinearReceiver
from scipy.io import loadmat


# def reconstruction(data):
def reconstruction(wavelength, sampling_spacing, receiver_samples):
    # wavelength, sampling_spacing, receiver_samples = data[0], data[1], data[2]

    sample_spacing = sampling_spacing
    radius = 100 * sample_spacing - sample_spacing/2
    receiver = LinearReceiver(y_int=0, xlim=(-radius, radius), slope=0)


    source_WF = Wave_Field(radius=radius, sample_spacing=sample_spacing, phase_only=True)
    receiver.get_receiver_coordinates(source_WF)
    source_WF.sampled_wavefield = receiver_samples

    # Reconstruct image with given wavelength
    # pattern_tittle
    ax = receiver.reconstruct_image(source_WF, wavelength, ylim= (-0.6, 0),
                                pattern_title=f"Reconstructed Source Region\n for
                                    $\lambda = {round(wavelength,3)}\lambda_0$")


    # Save image png and data npy matrix
    data_dir = os.path.join(os.path.dirname(__file__), 'data')
    # plt.savefig("{}/{}_lambda.png".format(os.path.join(data_dir,
        'reconstructed_images'), wavelength))
    np.save("{}/{}_lambda".format(os.path.join(data_dir, 'reconstructed_image_data'),
        wavelength),
           receiver.reconstructed_wavefield.composite_wave_field_pattern)

    plt.close('all')

    '''
    # # Reconstruct spectrum
    ax = receiver.reconstruct_spectrum(title=f"Wave Field Spectrum\n for $\lambda =
        {round(wavelength,3)}\lambda_0$")
    plt.savefig("{}/{}_lambda.png".format(os.path.join(data_dir,
        'reconstructed_spectrum'), wavelength))
    np.save("{}/{}_lambda".format(os.path.join(data_dir, 'reconstructed_spectrum_data'),
        wavelength),
           receiver.reconstructed_wavefield.wave_field_spectrum)
    '''


def multi_frequency_reconstruction(wavelengths, sample_spacing, receiver_samples):
    with multiprocessing.Pool() as pool:
        pool.map(reconstruction, wavelengths, sample_spacing, receiver_samples)


def sort_float(dir):

    x = os.path.splitext(dir)[0][:5]

    try:
        return float(x)
    except:
        pass
```

```
def superimpose_reconstructions(directory, reconstruction_snapshots=None,
    video_name='reconstructed_image_video', FPS=4):
    first_im_flag = True
    superimposed_reconstructions = np.zeros((10, 10)) # Final reconstruction image
        initialization

    for i, file in enumerate(sorted(os.listdir(directory))):
        if file.endswith(".npy"):
            vid_size = np.load(os.path.join(directory, file)).shape
            print('should only have 1 of these')
            break


    width = vid_size[1]
    height = vid_size[0]

    fourcc = VideoWriter_fourcc(*'MP42')
    reconstructed_video = VideoWriter(os.path.join(os.path.dirname(directory),
        f"{video_name}.avi"), fourcc, float(FPS), (width, height), 0)

    for i, file in enumerate(sorted(os.listdir(directory), key=lambda x:
        float(os.path.splitext(x)[0][:5]))):
    # for i, file in enumerate(sorted(os.listdir(directory))):
        if file.endswith(".npy"):
            print(os.path.join(directory, file))
            reconstruction = np.load(os.path.join(directory, file))
            if first_im_flag:
                superimposed_reconstructions = reconstruction
                first_im_flag = False
            else:
                superimposed_reconstructions += reconstruction

            if reconstruction_snapshots:
                if i+1 in reconstruction_snapshots:
                    wavelength = round(float(file.split('_')[0]), 3)
                    # fig, ax = plt.figure()
                    plt.imshow(np.abs(superimposed_reconstructions),
                        cmap=plt.get_cmap('hot'))
                    title = f"Reconstruction for $\lambda = {wavelength}\lambda_0$"
                    plt.title(title)
                    plt.savefig("{}/super_{}_lambda.png".format(directory,wavelength))

            # superimposed_reconstructions =
                (abs(superimposed_reconstructions)/np.max(superimposed_reconstructions))*255

            reconstructed_video.write((255*(np.abs(superimposed_reconstructions)/np.max(np.abs(superimpos

    reconstructed_video.release()

    plt.imshow(np.abs(superimposed_reconstructions))
    plt.title(video_name)
    plt.show()
```

```
if __name__ == "__main__":

    # Spacing in m, Wavelength in Ghz

    # frequencies = np.linspace(0.976,2,128) * 1 * 10**9
    frequencies = loadmat('gpr_data.mat')['f'].astype(np.float).squeeze()
    permitivity = 6

    velocity = 3*(10**8) / np.sqrt(permitivity)
    wavelengths = velocity / frequencies

    print(wavelengths)
    #
    start_time = time.time()
    #
    GPR_data = loadmat('gpr_data.mat')['F']
    # GPR_data = loadmat('gpr_data.mat')

    print(GPR_data.shape)

    # #
    for i in range(wavelengths.size):
       reconstruction(wavelengths[i], 0.0213, GPR_data[i, :])


    from cv2 import VideoWriter, VideoWriter_fourcc # Multiprocessing module raises
        error if imported before running
    root_dir = os.path.join(os.path.dirname(__file__), 'data/reconstructed_image_data')
    superimpose_reconstructions(root_dir, reconstruction_snapshots=[30,60,90,127],
        video_name='reconstructed_image')
    # root_dir = os.path.join(os.path.dirname(__file__),
        'data/reconstructed_spectrum_data')
    # superimpose_reconstructions(root_dir, (512,512),
        reconstruction_snapshots=[10,20,30,40], video_name='reconstructed_spectrum')
    # duration = time.time() - start_time
    # print(f"Duration {duration} seconds")
```