Introduction to Digital Image Processing

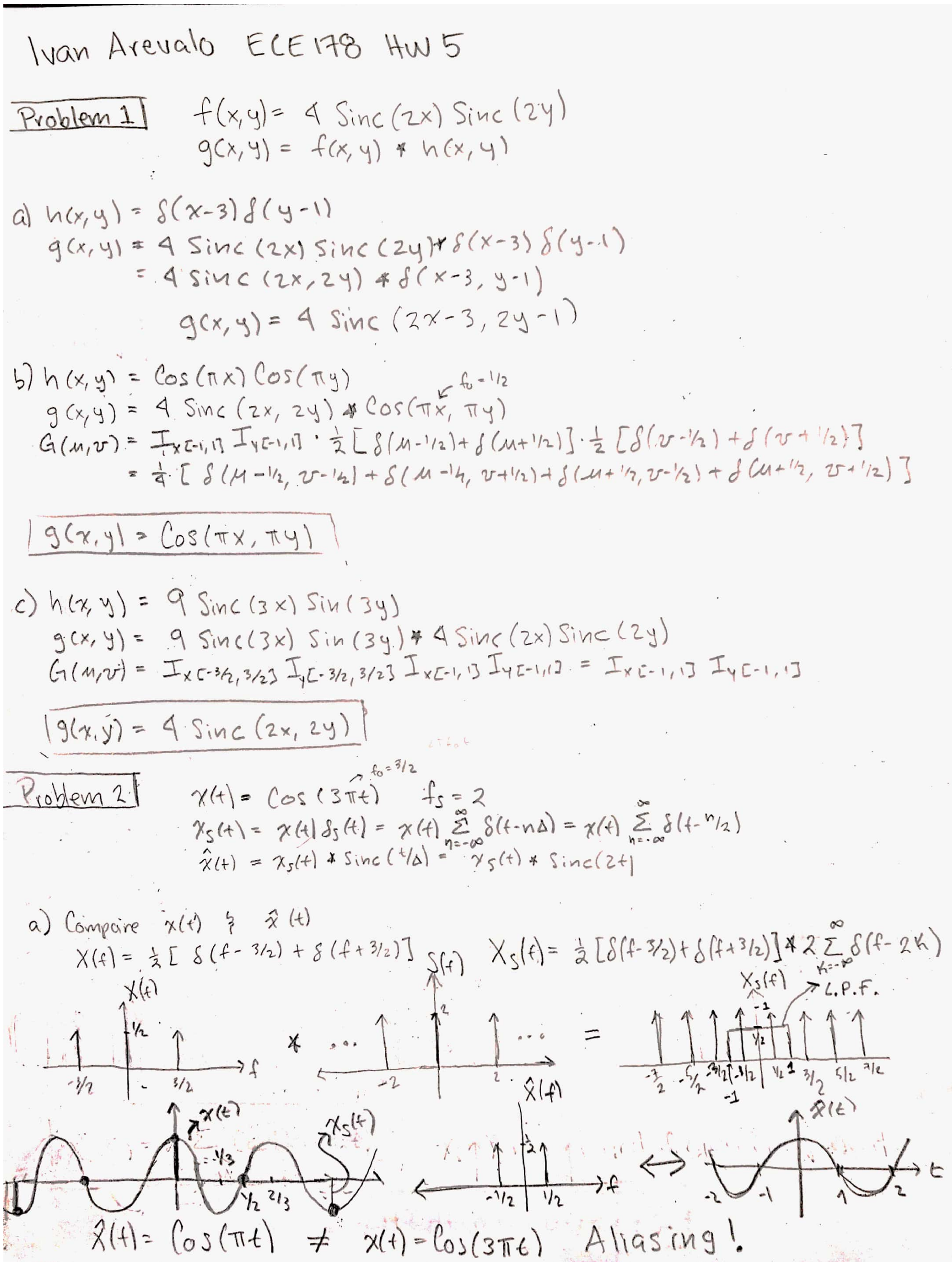
Homework 5

November 14, 2020

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

Department of Electrical and Computer Engineering, UCSB

Problem 1, 2, 3



Ivan Arevalo ECE 178 HW 5

Problem 1 $f(x,y) = 4\,\text{Sinc}(2x)\,\text{Sinc}(2y)$

$g(x,y) = f(x,y) * h(x,y)$

a) $h(x,y) = \delta(x-3)\delta(y-1)$

$g(x,y) = 4\,\text{Sinc}(2x)\,\text{Sinc}(2y) * \delta(x-3)\delta(y-1)$

$= 4\,\text{sinc}(2x, 2y) * \delta(x-3, y-1)$

$g(x,y) = 4\,\text{sinc}(2x-3, 2y-1)$

b) $h(x,y) = \text{Cos}(\pi x)\,\text{Cos}(\pi y)$

$f_0 = 1/2$

$g(x,y) = 4\,\text{Sinc}(2x, 2y) * \text{Cos}(\pi x, \pi y)$

$G(u,v) = I_{x[-1,1]} I_{y[-1,1]} \cdot \frac{1}{2}[\delta(u-1/2) + \delta(u+1/2)] \cdot \frac{1}{2}[\delta(v-1/2) + \delta(v+1/2)]$

$= \frac{1}{4}[\delta(u-1/2, v-1/2) + \delta(u-1/2, v+1/2) + \delta(u+1/2, v-1/2) + \delta(u+1/2, v+1/2)]$

$\boxed{g(x,y) = \text{Cos}(\pi x, \pi y)}$

c) $h(x,y) = 9\,\text{Sinc}(3x)\,\text{Sin}(3y)$

$g(x,y) = 9\,\text{Sinc}(3x)\,\text{Sin}(3y) * 4\,\text{Sinc}(2x)\,\text{Sinc}(2y)$

$G(u,v) = I_{x[-3/2,3/2]} I_{y[-3/2,3/2]} I_{x[-1,1]} I_{y[-1,1]} = I_{x[-1,1]} I_{y[-1,1]}$

$\boxed{g(x,y) = 4\,\text{Sinc}(2x, 2y)}$

Problem 2 $x(t) = \text{Cos}(3\pi t)$ $f_0 = 3/2$ $f_s = 2$

$x_s(t) = x(t)\delta_s(t) = x(t)\sum_{n=-\infty}^{\infty}\delta(t-n\Delta) = x(t)\sum_{n=-\infty}^{\infty}\delta(t-n/2)$

$\hat{x}(t) = x_s(t) * \text{Sinc}(t/\Delta) = x_s(t) * \text{Sinc}(2t)$

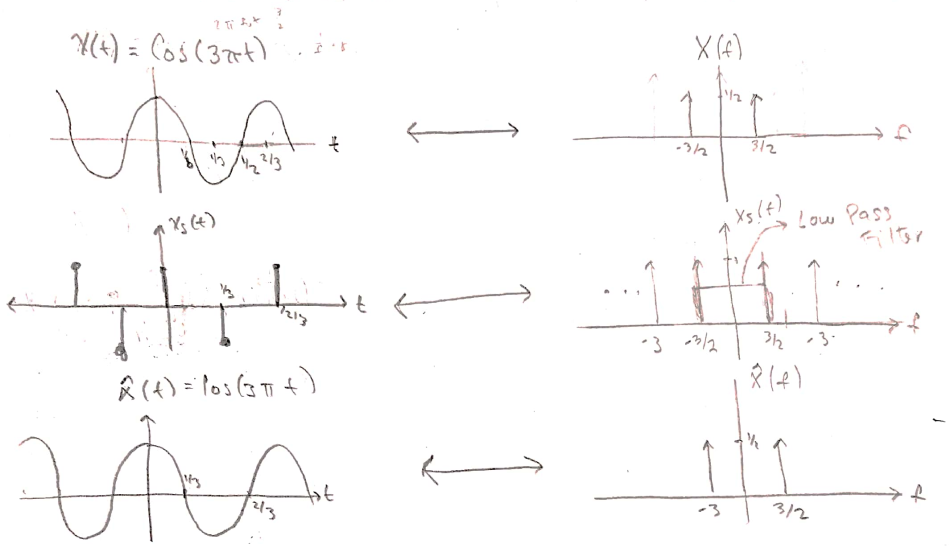
a) Compare $x(t)$ & $\hat{x}(t)$

$X(f) = \frac{1}{2}[\delta(f-3/2) + \delta(f+3/2)]$ $S(f)$ $X_s(f) = \frac{1}{2}[\delta(f-3/2) + \delta(f+3/2)] * 2\sum_{k=-\infty}^{\infty}\delta(f-2k)$

$\hat{x}(t) = \text{Cos}(\pi t) \neq x(t) = \text{Cos}(3\pi t)$ Aliasing!

Figure 1: Caption

b) The appropiate sampling frequency is Nyquist whis 2 times the highest frequency present in the signal. For this case f_s should be 3.

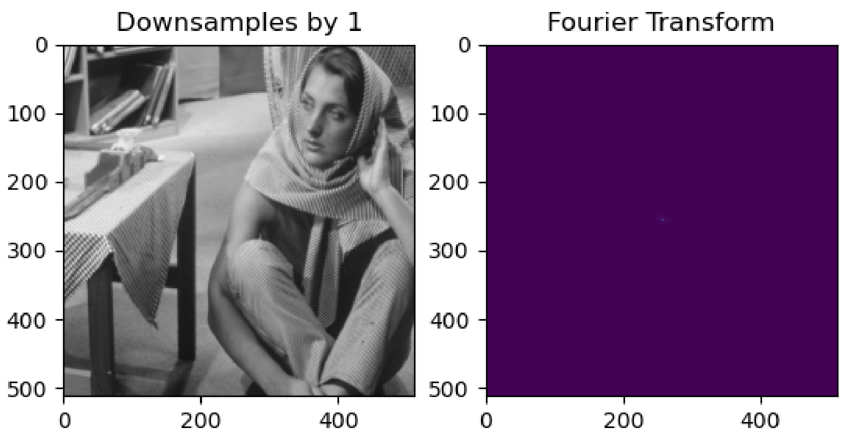


Problem 3) We can see that the downsampled image displays patterns in the clothes that are not present in the original image. We can predict that when we downsample, the new sampling rate is not spatially close enough to capture the high freq exponentials present in the image. When this occurs, we notice alising from other frequencies in our signal. hence the different patters present in the downsampled images.

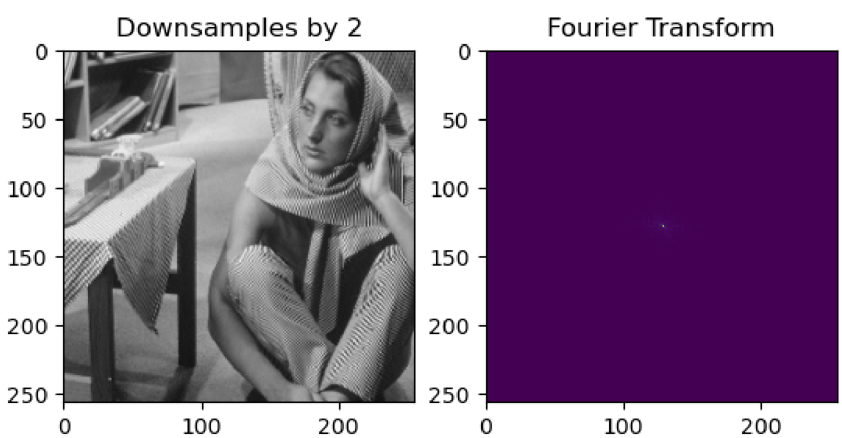
Figure 2: Caption

Fourier Transform is difficult to see but most frequencies are centered around DC with a really high value at 0,0.

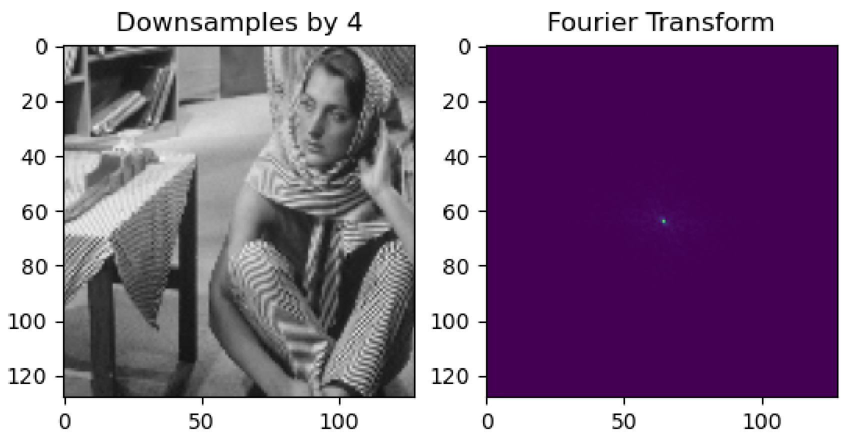
Problem 3

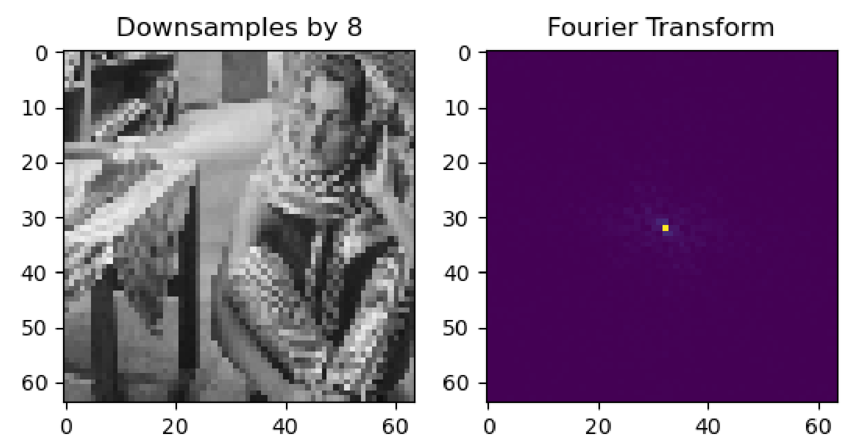


Downsamples by 1
Fourier Transform



Downsamples by 2
Fourier Transform





Problem 4

Video attached.

Code Aliasing

```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
import os

def plot_downsampled_image(image, sampling_factor):
    # plt.imshow(img, cmap='gray', vmin=0, vmax=255)
    img_pxl = np.asarray(img)[::sampling_factor, ::sampling_factor]
    fft_img = np.fft.fft2(img_pxl)
    fig = plt.figure(sampling_factor)
    ax1 = fig.add_subplot(1, 2, 1)
    ax1.set_title("Downsamples by {}".format(sampling_factor))
    ax1.imshow(img_pxl, cmap='gray', vmin=0, vmax=255)
    ax2 = fig.add_subplot(1, 2, 2)
    ax2.set_title('Fourier Transform')
    ax2.imshow(np.fft.fftshift(np.abs(fft_img)))
    fig.savefig("{}/downsampled/downsampled_{}.png".format(os.getcwd(), sampling_factor))

# Downsample and display each image FFT Pair
if not os.path.exists("{}/downsampled".format(os.getcwd())):
    os.makedirs("{}/downsampled".format(os.getcwd()))
img = Image.open("barbara.png")
plot_downsampled_image(img, 1)
plot_downsampled_image(img, 2)
plot_downsampled_image(img, 4)
plot_downsampled_image(img, 8)
plt.show()
```

FT reconstruction

```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
import cv2
import glob
import os
from moviepy.editor import *


def generate_fourier_basis(fy, fx, image):
    x_vec = np.arange(0, image.shape[1])
    y_vec = np.arange(0, image.shape[0])
    ex = np.exp(1j * 2 * np.pi * fx/x_vec.size * x_vec)
    ey = np.exp(1j * 2 * np.pi * fy/y_vec.size * y_vec)
    fourier_basis = np.outer(ey, ex)
    return fourier_basis


def generate_reconstruction_video():
```

```python
    current_directory = os.getcwd()
    img_array = []

    for filename in sorted(glob.glob("{}/video_frames/*.jpg".format(current_directory))):
        img = cv2.imread(filename)
        img_array.append(img)

    # creating a Image sequence clip with fps = 2
    clip = ImageSequenceClip(img_array, fps=10)
    clip.write_videofile("FT_reconstruction.mp4")


# Main
img = Image.open("goldhill.png")
img_pxl = np.asarray(img)[::1,::1] # Can downsample for testing
fft_img = np.fft.fft2(img_pxl)
high_energy_fft_idx =
    np.dstack(np.unravel_index(np.argsort(abs(fft_img).ravel())[-1:0:-1],
    (fft_img.shape[1], fft_img.shape[0]))).squeeze()
reconstructed_image = np.zeros(fft_img.shape, dtype='complex128')

# Create video frames folder
if not os.path.exists("{}/video_frames".format(os.getcwd())):
    os.makedirs("{}/video_frames".format(os.getcwd()))

frame_num = 1

for i, idx in enumerate(high_energy_fft_idx):

    # Update reconstructed image
    fourier_basis = generate_fourier_basis(idx[0], idx[1], img_pxl)
    fourier_phase = np.exp(1j*np.angle(fft_img[idx[0], idx[1]]))
    reconstructed_image += 1/(img_pxl.size) * abs(fft_img[idx[0], idx[1]]) *
        fourier_basis * fourier_phase
    if i <= 100 or (i < 500 and i % 5 == 0) or (i < 5000 and i % 100 == 0) or (i < 50000
        and i % 500 == 0) or (i % 5000 == 0):
        # Save video frame
        fig = plt.figure()
        ax1 = fig.add_subplot(1, 2, 1)
        ax1.set_title('Reconstructed Image')
        ax1.imshow(np.real(reconstructed_image), cmap='gray')
        ax1.tick_params(left=False, bottom=False, labelleft=False, labelbottom=False)

        ax2 = fig.add_subplot(1, 2, 2)
        ax2.set_title("{}th Highest Energy FT Basis: ({}, {})".format(i, idx[0], idx[1]))
        ax2.imshow(np.real(fourier_basis), cmap='gray')
        ax2.tick_params(left=False, bottom=False, labelleft=False, labelbottom=False)

        fig.savefig("video_frames/frame{:010d}.jpg".format(frame_num))

        frame_num += 1
        plt.close()

generate_reconstruction_video()
```

```
plt.imshow(np.real(reconstructed_image), cmap='gray')
plt.show()
```