CSE 166: Image Processing, Spring 2022 – Assignment 3

Instructor: Ben Ochoa

- Due On: **Monday, April 25, 2022, 11:59 PM (Pacific Time)**.

Instructions

Please answer the questions below using Python in the attached Jupyter notebook and follow the guidelines below:

- This assignment must be completed **individually**. For more details, please follow the Academic Integrity Policy and Collaboration Policy on Canvas.
- This assignment contains both math and programming problems.
- All the solutions must be written in this Jupyter notebook.
- After finishing the assignment in the notebook, please export the notebook as a PDF and submit both the notebook and the PDF (i.e. the `.ipynb` and the `.pdf` files) on Gradescope. Please assign the pages of your PDF submission to the corresponding problems.
- You may use basic algebra packages (e.g. `NumPy`, `SciPy`, etc) but you are not allowed to use the packages that directly solve the problems. Feel free to ask the instructor and the teaching assistants if you are unsure about the packages to use.
- It is highly recommended that you begin working on this assignment early.

Late Policy: Assignments submitted late will receive a 15% grade reduction for each 12 hours late (i.e., 30% per day). Assignments will not be accepted 72 hours after the due date. If you require an extension (for personal reasons only) to a due date, you must request one as far in advance as possible. Extensions requested close to or after the due date will only be granted for clear emergencies or clearly unforeseeable circumstances.

Problem 1: Textbook problems (11 points)

a) Problem 4.3 (1 point)

your answer here

b) Problem 4.4 (1 point)

your answer here

c) Problem 4.9 (2 points)

your answer here

d) Problem 4.27 (1 point)

your answer here

e) Problem 4.33 (2 points)

your answer here

f) Problem 4.40 (3 points)

your answer here

g) Problem 4.45 (1 point)

your answer here

Problem 2: Programming: The Fourier transform and filtering in the frequency domain (35 points)

Part 1: The Fourier transform pair (10 points)

- Complete the function **dft2d** that computes the 2D discrete Fourier transform of an array. The function should take as input a 2D array and return as output the 2D array of complex numbers corresponding to the discrete Fourier transform.
- Print the execution time of your **dft2d** function on an array of size 100×100. Compare the execution time of your **dft2d** function with the numpy.fft.fft2 function. Why do you think the execution times are different?
- Complete the function **dft_ift** which, in order,
  - Calls the **dft2d** function with the image to compute the discrete Fourier transform (DFT) $F(u, v)$ (shifted such that the zero-frequency component $F(0, 0)$ is centered) of the input image $f(x, y)$
  - Calculates the magnitude $||F(u, v)||$ and phase $\phi(u, v)$ of $F(u, v)$
  - Calculates the DFT $G(u, v) = ||F(u, v)||e^{j\phi(u,v)}$

- Computes the inverse discrete Fourier transform (IDFT) $g(x, y)$ of $G(u, v)$ (after inverting the centering shift)
- Returns the real part of g(x,y), magnitude $\|F(u, v)\|$ and phase $\phi(u, v)$ of $F(u, v)$

Note: You may use Python built-in functions to perform the intermediate steps.

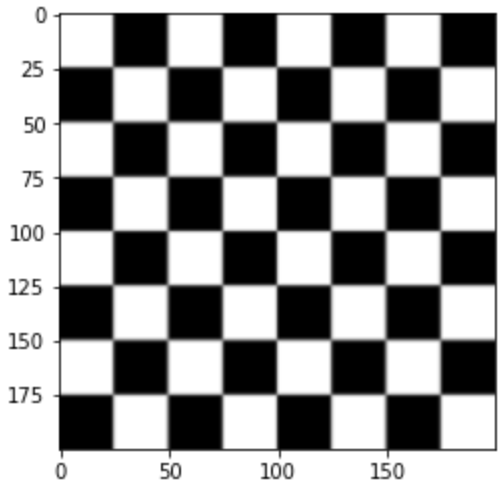
- Call the **dft_ift** function with the checkerboard image. Display the input and the output image. Additionally, display figures of $\log\|F(u, v)\|$ and $\phi(u, v)$. What are the row and column indices of the $F(0, 0)$ component before and after the centering shift?

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from skimage import data
import math
import cv2
import time
```

In [ ]:
```
# Read and display image
img = data.checkerboard()

plt.imshow(img, cmap='gray')
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x7f4ee47435d0>
```



In [ ]:
```
# function that computes the 2D discrete Fourier transform of an array.
def dft2d(arr):
  """
  arr: input array (H,W)

  returns:
  F: 2D discrete Fourier transform of the input array (H,W)
  """
  # your code here

  return F
```

In [ ]:
```
"""
Call the function dft2d with a 100*100 array and print the execution time
Compare the execution time of your dft2d function with the numpy.fft.fft2 function
"""
# your code here
```

Out[ ]:
```
'\nCall the function dft2d with a 100*100 array and print the execution time\nCompare th
e execution time of your dft2d function with the numpy.fft.fft2 function\n'
```

Why do you think the execution times are different?

your answer here

In [ ]:
```
# see part 1 above to read complete function description
def dft_ift(img):
  """
  img: input image (H,W)

  returns:
  img_g_real: real part of g(x,y) (H,W)
  f_mag: magnitude ||F(u,v)|| (H,W)
  f_phase: phase of F(u,v) (H,W)
  """
  # your code here

  return img_g_real, f_mag, f_phase
```

In [ ]:
```
"""
Call the dft_ift function with the checkerboard image. Display the input and the output
Additionally, display figures of ||F(u,v)|| and phase(u,v).
"""
# your code here
```

Out[ ]:
```
'\nCall the dft_ift function with the checkerboard image. Display the input and the outp
ut image.\nAdditionally, display figures of ||F(u,v)|| and phase(u,v).\n'
```

What are the row and column indices of the $F(0,0)$ component before and after the centering shift?

your answer here

Part 2: The convolution theorem (15 points)

The objective of this problem is to show that the output image

$$g(x,y) = f(x,y) \bigstar h(x,y) = \Im^{-1}\{F(u,v)H(u,v)\}$$

where $F(u,v)$ and $H(u,v)$ are the DFTs of the input image $f(x,y)$ and kernel $h(x,y)$, respectively.

- Complete the function **filter_in_frequency_domain** that applies a filter to an image in the frequency domain. The function inputs are a grayscale image and a kernel, and the function output is the filtered (floating point) image corresponding to the input image. The inputs and output are in the spatial domain. Zero padding must be used to mitigate wraparound error. The calculated output image must be the same size as the input image.
  - You may use the Python functions np.fft.fft2, np.fft.ifft2, np.fft.fftshift, np.fft.ifftshift
- Complete the function **conv_theorem** that applies the filter to the image $f(x,y)$ in the frequency and spatial domain. The function takes as input an image and a kernel, and returns the output images filtered in the frequency and spatial domain.
  - You must call the function **filter_in_frequency_domain** to apply the filter in the frequency domain.
  - You may use the function cv2.filter2D to apply the filter in the spatial domain.

- Call the function **conv_theorem** with the Laplacian kernel $h(x, y)$ and the camera image $f(x, y)$, where $$ h(x,y) = \begin{bmatrix}

  1 & 1 & 1\\
  1 & -8 & 1\\
  1 & 1 & 1
  \end{bmatrix} $$

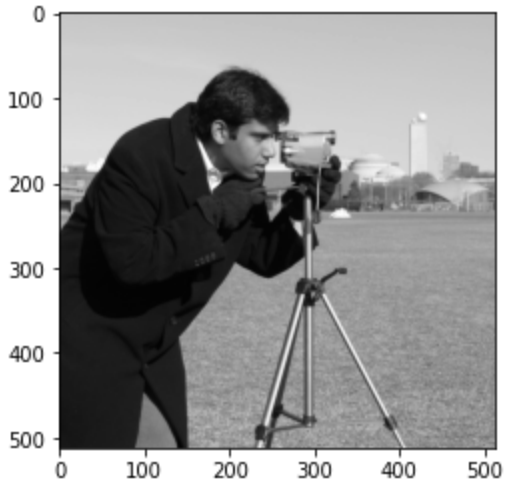
  Display the input image along with both resulting filtered images.

- Briefly discuss your results. Why would it be beneficial to implement filtering in the frequency domain? (Optional: subtract a filtered image from the input image to yield a sharpened image.)

In [ ]:
```
# Read and display the image
img = data.camera()

plt.imshow(img, cmap='gray')
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x7f2cc2e35310>
```



In [ ]:
```
# function that applies a filter to an image in the frequency domain
def filter_in_frequency_domain(img, kernel):
  """
  img: input image (H, W)
  kernel: filter (kH, kW)

  returns:
  f_img: image filtered in frequency domain (H, W)
  """
  # your code here

  return f_img
```

In [ ]:
```
# function that applies the filter to the image in the frequency and spatial domain.
def conv_theorem(img, kernel):
  """
  img: input image (H,W)
  kernel: kernel (kH, kW)

  returns:
  f_img: image filtered in frequency domain (H, W)
  sp_img: image filtered in spatial domain (H, W)
  """
  # your code here

  return f_img, sp_img
```

In [ ]:
```
"""
Call the function conv_theorem with the Laplacian kernel and the camera image.
Display the input image along with both resulting filtered images.
"""
# your code here
```

Out[ ]:
```
'\nCall the function conv_theorem with the Laplacian kernel and the camera image.   \nDi
splay the input image along with both resulting filtered images.\n'
```

Briefly discuss your results. Why would it be beneficial to implement filtering in the frequency domain? (Optional: subtract a filtered image from the input image to yield a sharpened image.)

your answer here

Part 3: Lowpass filtering (10 points)

1. Complete the function **ideal_lpf** that applies an ideal lowpass filter in the frequency domain. The function inputs are an image and a radius, and the function output is the image with ideal low-pass filter applied in the frequency domain.
2. Complete the function **gaussian_lpf** that applies a Gaussian lowpass filter in the frequency domain. The function inputs are an image and a variance, and the function output is the image with Gaussian low-pass filter applied in the frequency domain.
3. Call the function **ideal_lpf** with the camera image and for three different radii. One of the function call must have radius $D_0 = 50$. Display the original image and all three ideal lowpass filtered images and briefly discuss your results.
4. Call the function **gaussian_lpf** with the camera image and for three different standard deviations (σ). Display the original image and all three Gaussian lowpass filtered images. Briefly discuss the difference between your Gaussian lowpass filtering results and your ideal lowpass filtering results.

In [ ]:
```
# function that applies ideal low-pass filter in the frequency domain
def ideal_lpf(img, radius):
  """
  img: input image (H,W)
  radius: Radius for the ideal lowpass filter

  returns:
  out: output image after low-pass filter has been applied in the frequency domain
  """
  # your code here

  return out
```

In [1]:
```
# function that applies gaussian low-pass filter in the frequency domain
def gaussian_lpf(img, std):
  """
  img: input image (H,W)
  std: standard deviation for the gaussian lowpass filter

  returns:
  out: output image after low-pass filter has been applied in the frequency domain
  """
  # your code here
```

```
  return out
```

In [ ]:
```
"""
Call the function ideal_lpf with the camera image and for three different radii.
One of the function call must have radius = 50
Display the original image and all three ideal lowpass filtered images.
"""
img = data.camera()
# your code here
```

Briefly discuss your results

your answer here

In [ ]:
```
"""
Call the function **gaussian_lpf** with the camera image and for three different varianc
Display the original image and all three Gaussian lowpass filtered images.
"""
# your code here
```

Out[ ]:
```
'\nCall the function **gaussian_lpf** with the camera image and for three different vari
ance.\nDisplay the original image and all three Gaussian lowpass filtered images. \n'
```

Briefly discuss the difference between your Gaussian lowpass filtering results and your ideal lowpass filtering results.

your answer here