CSE 166: Image Processing, Spring 2022 – Assignment 4

Instructor: Ben Ochoa

- Due On: **Wednesday, May 4, 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 (5 points)

a) Problem 5.10 (2 points)

your answer here

b) Problem 5.12 (2 points)

your answer here

c) Problem 7.7 (1 point)

your answer here

Problem 2: Programming: Image Restoration and Color Image Processing (30 points)

Part 1: Spatial Image Restoration (10 points)

- Complete the function **adaptive_local_noise_reduction** that computes an estimate of the original image $\hat{f}(x,y)$ given a noisy image $g(x,y)$, the overall noise variance σ_η^2, and the filter window size using the adaptive expression

$$\hat{f}(x,y) = g(x,y) - \frac{\sigma_\eta^2}{\sigma_{S_{xy}}^2}\left(g(x,y) - \bar{z}_{S_{xy}}\right)$$

  and enforce the assumption $\sigma_\eta^2 \leq \sigma_{S_{xy}}^2$ (see the last paragraph of page 384), where the local mean $\bar{z}_{S_{xy}}$ and local variance $\sigma_{S_{xy}}^2$ are calculated over the filter window centered at (x,y).

- Apply Gaussian noise with variance 0.1 using the function **skimage.util.random_noise** for the camera image. Then, call the **adaptive_local_noise_reduction** with the noisy image and a window size of $(7,7)$.

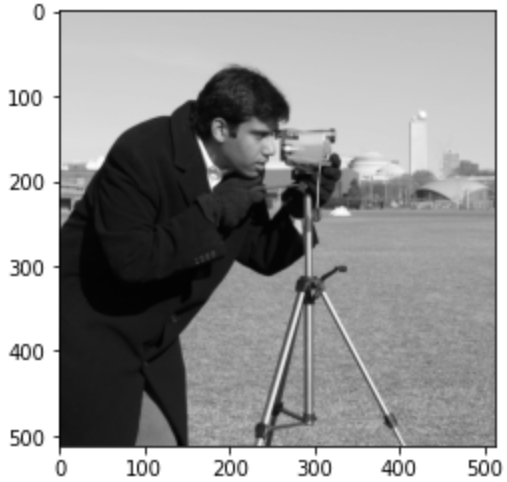
- Display the input image, noisy image, and restored image.

- Briefly discuss how well the restored image approximates the original image and whether or not a local noise reduction filter is suitable for applying to this type of noise. What are the trade offs of using adaptive local noise reduction versus other noise reduction techniques such as an arithmetic mean filter? What are the advantages and disadvantages of using adaptive local noise reduction?

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from skimage import data
import skimage
import cv2
import  scipy.signal
import math
```

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

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

Out[ ]:
```
<matplotlib.image.AxesImage at 0x7f2110cc7f90>
```



In [ ]:
```python
# function that computes an estimate of the original image for a given noisy image
def adaptive_local_noise_reduction(noisy_img, var, window_size):
  """
  noisy_img: image with noise added (H,W)
  var: overall noise variance
  window_size: filter window (n,n)

  returns:
  fhat_img: estimate of the image after noise removal (H,W)
  """
  # your code here

  return fhat_img
```

In [ ]:
```python
"""
Apply Gaussian noise with a variance of 0.1 for the camera image.
Then, call the adaptive_local_noise_reduction with the noisy image and a window size of

Display the input image, noisy image, and restored image.
"""
# your code here
```

Out[ ]:
```
'\nApply Gaussian noise with a variance of 0.1 for the camera image. \nThen, call the ad
aptive_local_noise_reduction with the noisy image and a window size of (7,7).\n  \nDispl
ay the input image, noisy image, and restored image.  \n'
```

Briefly discuss how well the restored image approximates the original image and whether or not a local noise reduction filter is suitable for applying to this type of noise. What are the trade offs of using adaptive local noise reduction versus other noise reduction techniques such as an arithmetic mean filter? What are the advantages and disadvantages of using adaptive local noise reduction?

your answer here

Part 2: Inverse Filtering (10 points)

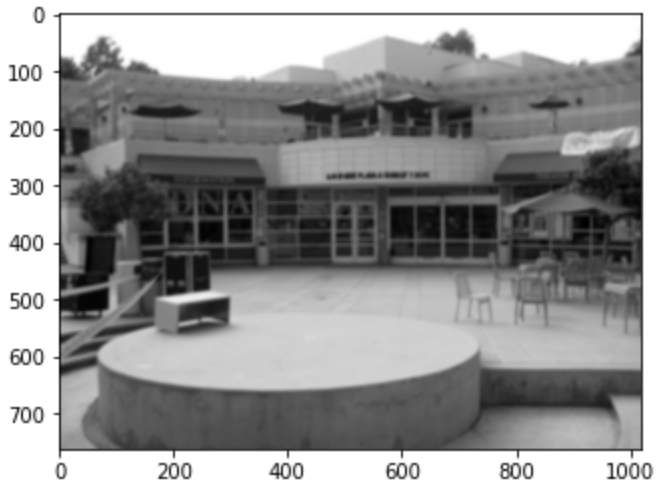
The degraded image $d(x, y)$ is the result of convolving an input image with the degradation function

$$h(x,y) = \frac{1}{239}\begin{bmatrix} 3 & 7 & 7 & 7 & 8 & 9 & 5 \\ 4 & 6 & 2 & 5 & 8 & 8 & 3 \\ 8 & 5 & 4 & 4 & 6 & 5 & 8 \\ 1 & 5 & 6 & 5 & 4 & 6 & 2 \\ 1 & 3 & 8 & 3 & 8 & 6 & 3 \\ 2 & 7 & 1 & 5 & 5 & 2 & 2 \\ 6 & 2 & 9 & 5 & 4 & 3 & 3 \end{bmatrix}$$

- Complete the function **inverse_filter** that performs inverse filtering in the frequency domain. The function takes as input the degraded image $d(x,y)$, degradation function $h(x,y)$ and a threhold value t, and returns the estimated image after inverse filtering.
  - Once you obtain the estimated function $\hat{F}$ in the frequency domain, set values in $\hat{F}$ at coordinates (u,v) to 0, if the magnitude of H at the same coordinates (u,v) is less than the threshold value t. Then, map $\hat{F}$ to the estimated image $\hat{f}$ in the spatial domain.
  - You can use scipy.signal.convolve2d for performing convolution and numpy functions for performing 2D Fourier transform related operations.
- Call the function **inverse_filter** with the q2_input.png image (provided to you in the assignment folder), the degradation function $h(x,y)$ described above and a threhold $t = 0.01$.
- Display the degraded image and the estimated image after inverse filtering.
- Discuss how well the inverse filtering restored the original image. How well would inverse filtering work if noise was also applied to the image?

In [3]:
```
# Read and display the image
degraded_img = plt.imread('q2_input.png')[:,:,0]

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

Out[3]:
```
<matplotlib.image.AxesImage at 0x7fa1c229ab10>
```



In [ ]:
```
# function that performs inversing filtering in the frequency domain
def inverse_filter(degraded_img, h, t):
  """
  degraded_img: degraded image (H,W)
  h: degradation function (kH, kW)
  t: threshold of H to determine corresponding coordinates in Fhat to set to zero

  returns:
  fhat_img: estimated image after inverse filtering (H,W)
```

```
    """
    # your code here

    return fhat_img
```

In [ ]:
```
"""
Call the function inverse_filter with the q2_input.png image (provided to you in
 the assignment folder) and the degradation function h(x,y) desribed above.

Display the degraded image and the estimated image after inverse filtering.
"""
# your code here
```

Out[ ]:
```
'\nCall the function inverse_filter with the q2_input.png image (provided to you in\n th
e assignment folder) and the degradation function h(x,y) desribed above. \n  \nDisplay t
he degraded image and the estimated image after inverse filtering.\n'
```

Discuss how well the inverse filtering restored the original image. How well would inverse filtering work if noise was also applied to the image?

your answer here

Part 3: Color Histogram Equalization (10 points)

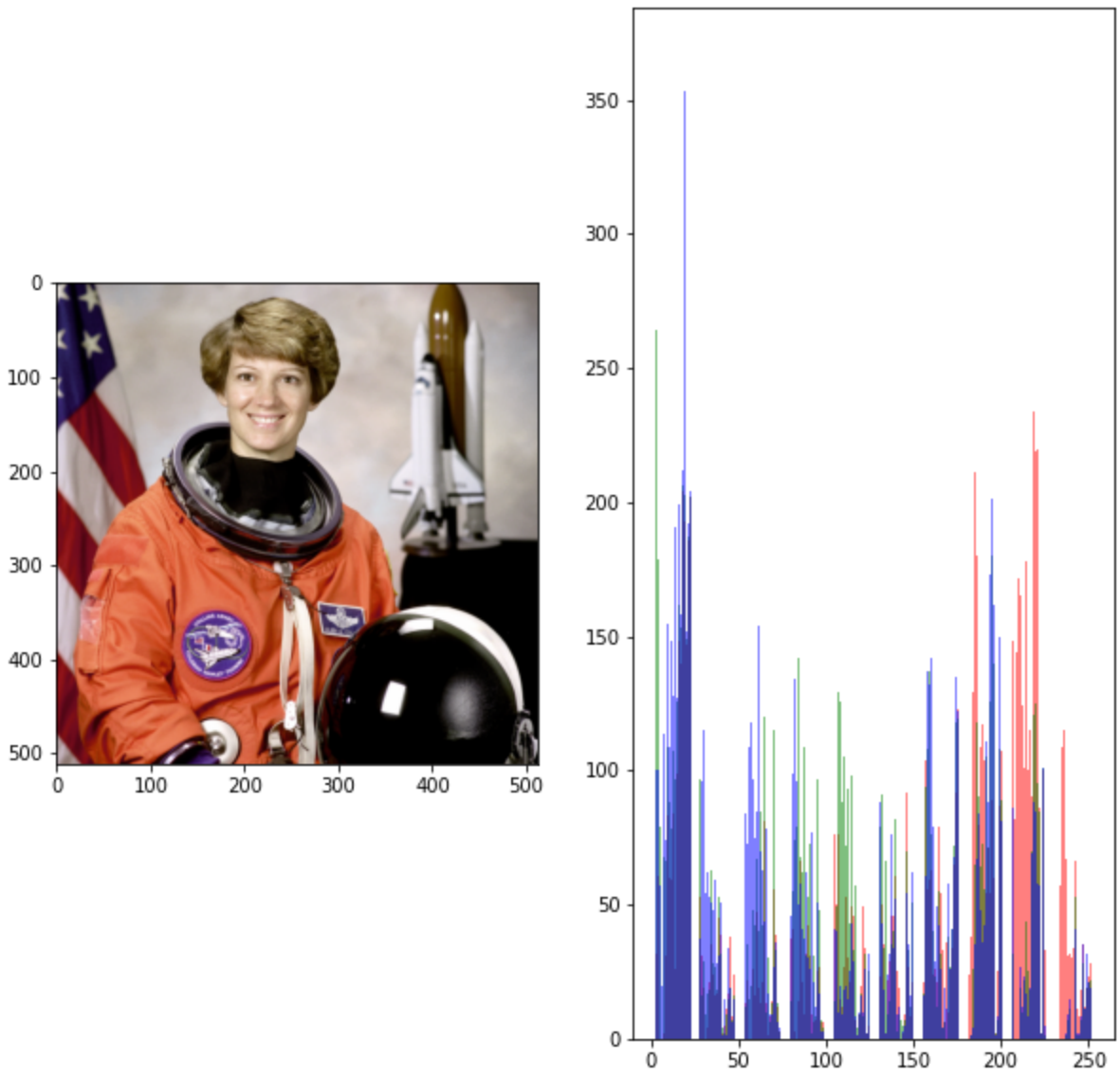
- Complete the function **histeq_1** that histogram equalizes each color channel **independently** (i.e., independent of the other channels). The function takes as input an rgb image and returns the histogram equalized image.
- Complete the function **rgb2hsi** that converts an RGB image to an HSI image. The function should take as input an RGB image and output the corresponding HSI image.
- Complete the function **hsi2rgb** that converts an HSI image to an RGB image. The function should take as input an HSI image and output the corresponding RGB image.
- Complete the function **histeq_2** that
  - Takes as input an RGB image
  - Calls your function **rgb2hsi** to convert the image from RGB to HSI
  - Uses the function skimage.exposure.equalize_hist to histogram equalize the **I channel only**
  - Calls your function **hsi2rgb** to convert the image from HSI to RGB
  - Returns the output image
- Call the function **histeq_1** and **histeq_2** with the astronaut image
- Display the original image, output image of histeq_1 and output image of histeq_2.
- Display the **histograms** for the original image, output image of histeq_1 and output image of histeq_2. In each histogram, the color of each channel should be drawn with its corresponding color and transparency value of 0.5 (Check the histogram of the input image for reference).
- Briefly discuss the qualitative differences between the output images. Briefly discuss the quantitative differences, namely the range of intensities in each channel, between the output images.

In [ ]:
```
# Read and display image along with its histogram
img = data.astronaut()

fig, ax = plt.subplots(1,2, figsize=(10,10))
```

```
ax[0].imshow(img)

ax[1].hist(img[:,:,0], color=['r']*512, alpha=0.5)
ax[1].hist(img[:,:,1], color=['g']*512, alpha=0.5)
ax[1].hist(img[:,:,2], color=['b']*512, alpha=0.5)
```

Out[ ]:

```
(array([[127.,  37.,   3., ...,  39., 144.,   0.],
        [108.,  55.,  15., ...,  35., 147.,   0.],
        [ 87.,  78.,  27., ...,  38., 142.,   0.],
        ...,
        [217.,   1.,   2., ...,  11.,   3.,   1.],
        [219.,   2.,   2., ...,  11.,   4.,   0.],
        [222.,   3.,   1., ...,   6.,   1.,   0.]]),
 array([  0. ,  25.5,  51. ,  76.5, 102. , 127.5, 153. , 178.5, 204. ,
        229.5, 255. ]),
 <a list of 512 Lists of Patches objects>)
```



In [ ]:

```
# function that histogram equalizes each color channel independently
def histeq_1(img):
  """
  img: input image in RGB format (H,W,3)

  returns:
  out: output image after histogram equalization (H,W,3)
  """
  # your code here

  return out
```

In [ ]:
```
# function that conver rgb image to hsi image
def rgb2hsi(rgb):
  """
  rgb: image in RGB format (H,W,3)

  returns:
  hsi: image in HSI format (H,W,3)
  """
  # your code here

  return hsi
```

In [ ]:
```
# function that converts hsi image to rgb image
def hsi2rgb(hsi):
  """
  hsi: image in HSI format (H,W,3)

  returns:
  rgb: image in RGB format (H,W,3)
  """
  # your code here

  return rgb
```

In [ ]:
```
# see above for full function description
def histeq_2(img):
  """
  img: image in RGB format (H,W,3)

  returns:
  out: output image in RGB format after histogram equalization (H,W,3)
  """
  # your code here

  return out
```

In [ ]:
```
"""
Call the function histeq_1 and histeq_2 with the astronaut image
Display the original image, output image of histeq_1 and output image of histeq_2.
"""
# your code here
```

Out[ ]:
```
'\nCall the function histeq_1 and histeq_2 with the astronaut image\nDisplay the origina
l image, output image of histeq_1 and output image of histeq_2.\n'
```

In [ ]:
```
"""
Display the **histograms** for the original image, output image of histeq_1 and output i
In each histogram, the color of each channel should be drawn with its corresponding colo
(Check the histogram of the input image for reference)
"""
```

Out[ ]:
```
'\nDisplay the **histograms** for the original image, output image of histeq_1 and outpu
t image of histeq_2. \nIn each histogram, the color of each channel should be drawn with
its corresponding color and transparency value of 0.5 \n(Check the histogram of the inpu
t image for reference) \n'
```

Briefly discuss the qualitative differences between the output images. Briefly discuss the quantitative differences, namely the range of intensities in each channel, between the output images.

your answer here