CSE 166: Image Processing, Spring 2022 – Assignment 7

Instructor: Ben Ochoa

- Due On: **Wednesday, June 1, 2022, 11:59 PM**.

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 (7 points)

a) Problem 10.6 (1 point)

your answer here

b) Problem 10.10 (2 points)

your answer here

c) Problem 10.26 (2 points)

your answer here

d) Problem 10.41 (2 points)

your answer here

Problem 2: Programming (30 points)

Part 1: Edge Detection (10 points)

- Complete the function **edge_detection** that takes an image as input and performs the following steps:
  - Normalizes the image to [0,1] range
  - Applies Gaussian smoothing with standard deviation 0.5
  - Computes the gradient magnitude and angle images
  - Applies nonmaximal suppression to the gradient magnitude image. For non-maximum suppression, discretize the angle into 8 directions (or bins), where each bin accounts for 45 degrees. For example, first bin would range from [-22.5,22.5] degrees, second bin would range from [22.5,67.5] degrees, and so on. Please refer to Lecture 15, slide 21 for a graphical description.
  - Returns the gradient magnitude image, angle image and gradient image after non-maximal suppression
- Call the function **edge_detection** function with the coins image. Display the input image and all three output images from your **edge_detection** function.

Note: You may use scipy.ndimage.gaussian_filter and scipy.signal.convolve2d functions for performing gaussian filtering and convolution operations, respectively.

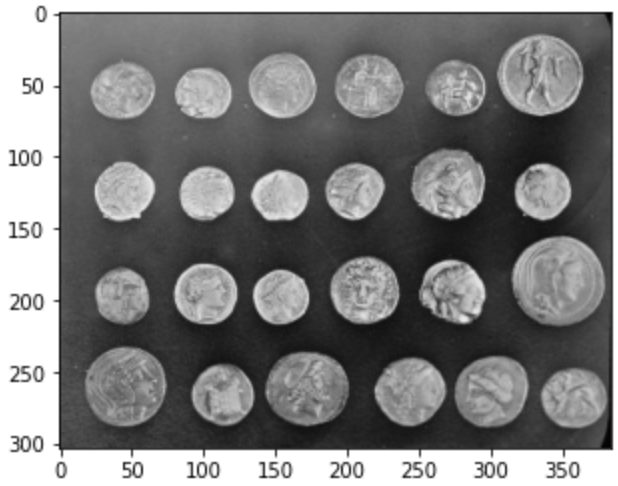
- Briefly discuss the resulting images and how you might implement edge linking techniques using these outputs. Explain your reasoning.

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from skimage import data
from scipy.ndimage import gaussian_filter
from scipy.signal import convolve2d
```

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

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

Out[ ]:
```
<matplotlib.image.AxesImage at 0x7fd258064b10>
```



In [ ]:
```python
# function that performs edge detection
def edge_detection(img):
  """
  img: input image (H,W)

  returns:
  gmag: gradient magnitude image (H,W)
  ang: angle image (H,W)
  gmag_nms: gradient magnitude image after non-maximal suppression (H,W)
  """
  # your code here

  return gmag, ang, gmag_nms
```

In [ ]:
```python
"""
Call the function edge_detection function with the coins image.
Display the input image and all three output images from your edge_detection function.
"""
# your code here
```

Out[ ]:
```
'\nCall the function edge_detection function with the coins image. \nDisplay the input i
mage and all three output images from your edge_detection function.\n'
```

Briefly discuss the resulting images and how you might implement edge linking techniques using these outputs. Explain your reasoning.

your answer here

Part 2: Region segmentation using k-means clustering (20 points)

- Complete the funciton **region_segmentation** that performs region-based segmentation using k-means clustering. The function that takes an input image, number of clusters k, nonnegative convergence threshold T, and maximum number of iterations N as inputs and performs the following steps:
  - Normalizes the input image to [0,1] range.
  - Randomly initializes a set of means
  - Then iteratively assigns samples to clusters using Euclidean distance followed by cluster mean m_i updates. The script must perform a test for convergence

$$\sum_{i=1}^{k} \|m_i^{(t)} - m_i^{(t-1)}\| < T \quad (1)$$

where T is the specified nonnegative convergence threshold T, but must also stop iterating after a specified maximum number of iterations N.

- Returns the output image with region-based segmentation, and the number of iterations t.
- Hint: The output image must assign each pixel to the mean m_i value corresponding to its cluster

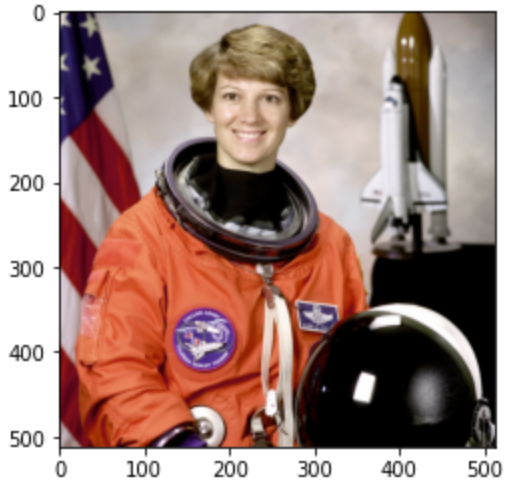
- Call the function **region_segmentation** with the astronaut image, $N = 100$, $T = 0.001$ and two different values of $k = 4$ and $k = 8$. For each value of k, run the **region_segmentation** function twice.
  - Note: You may get different outputs for each run since the means are initiatized randomly.
- Display the input image, along with output images for $k = 4$ and $k = 8$ (4 output images in total: 2 runs * 2 values of k). Also, print the number of iterations t taken by your function for different values of k.
- Briefly describe the differences between the two images and how k impacts the result.

Note: Your function may take a long time to converge if you do not vectorize your code. Note: You may use the function np.random.rand for generating random numbers

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

plt.imshow(img)
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x7fd24b65d710>
```



In [ ]:
```
# function that performs region-based segmenation using k-means clustering
def region_segmentation(img, k, N, T):
  """
  img: input image(H,W)
  k: number of clusters (integer)
  N: maximum number of iterations (integer)
  T: non-negative convergence threshold (floating-point)

  returns:
  out: output image after region-based segmentation (H,W)
  t: number of iterations
```

```
    """
    # your code here

    return out, t
```

In [1]:
```
"""
Call the function region_segmentation with the astronaut image, N = 100, T=0.001 and two
Display the input image, along with output images for k = 4 and k = 8 (4 output images i
Also, print the number of iterations t taken by your function for different values of k.
"""
# your code here
```

Out[1]:
```
'\nCall the function region_segmentation with the astronaut image, N = 100, T=0.001 and
two different values of k=4 and k=8. For each value of k, run the region_segmentation fu
nction twice.\nDisplay the input image, along with output images for k = 4 and k = 8 (4
output images in total: 2 runs * 2 values of k). \nAlso, print the number of iterations
t taken by your function for different values of k.\n'
```

Briefly describe the differences between the two images and how k impacts the result.

your answer here