EEE/AME 515 **HOMEWORK ASSIGNMENT 1** **Spring 2021**

Due on Tuesday, September 12th at 11:59 pm
Answer all problems following the instructions. Please submit a PDF report with pictures of the output for each step followed by relevant code snippets/screenshots that executed the function.

1 Front Page of the Report

You must use either Latex or Microsoft Word or Jupyter notebooks to typeset your reports. Throughout this homework assignment, we will use font color blue for the parts of the assignment you need to have a written response for. In addition, please present relevant code and figures to explain your method to others. Think about this like a job, you need to explain to your supervisor/boss what you did without dumping a whole bunch of code, answering questions in comments (big NO), and using labeled figures with captions.

Please include a cover page with your name, ASU ID, and indicate which problems have been solved, and which problems need extensions of one month. Please provide a justification for each extension (i.e. "I solved problems a, b, but having difficulty with the getting real data for part c").

2 Coding Assignments

Problem 1. *Setting up Coding Environment*
We will utilize the Python programming language along with OpenCV bindings to do most of the image processing required in the homework assignments. In addition, we will utilize several helper libraries for visualizing images and displaying output. Problem 1 is not graded but rather provided to make sure you install OpenCV successfully.

1. Install Anaconda Python `https://docs.anaconda.com/anaconda/install/`. Make sure you follow the instructions specific to your operating system. I used the graphical interface to install, although you can also use the command line version if you are comfortable with that. To test your Python is working, create a file **helloworld.py**, and write the following line:

```
print('Hello World!')
```

   To run the code, you can open a terminal shell and run the following command:

```
python helloworld.py
```

   and should get the terminal output of "Hello, World!". If you run into errors, check your installation of Anaconda Python (Google is your friend for debugging errors! and Stack-Overflow).

2. Install OpenCV via Anaconda using the following command:

```
conda install -c conda-forge opencv
```

If this command does not work, Google how to install OpenCV for your Anaconda distribution (or search for OpenCV on the package finder)

3. Install Matplotlib (a useful library for visualizing graphical output)

```
conda install matplotlib
```

4. Make sure you can import all the libraries. Write the following file **testimport.py** with the following lines:

```
import cv2
import matplotlib
import ipdb
```

If you get no errors while running this, then you have successfully installed these packages. **NOTE:** If you are stuck on any step, Google is your friend! Just type "How do I install X on a Mac/Windows/Linux?" and experiment till you find something that works!

5. I recommend using an IDE such as Spyder `https://www.spyder-ide.org/` or PyCharm `https://www.jetbrains.com/pycharm/` to prototype your code. Spyder is built into Anaconda Python, just use the package installer in Anaconda to install it.

Problem 2. *Messing around with OpenCV and Images*
The point of this problem is to mess around with images, and explore the OpenCV framework.

1. First import all the dependencies you will need as follows:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import ipdb
```

For every section, you should block it off with a set of comments as follows:

```
########## Section 2.X --<insert description>########################
# load image
<your code here>
# display image
<your code here>
###################################################
```

This makes it easy to read, and legible.

2. **Loading and displaying images:**
a. Pick an image you really like to display, and use Matplotlib to do so. Comands such as *plt.imshow* and *plt.show* are helpful here. Notice how the colors seem inverted, so you can use the function *cv2.cvtColor* to change that.

b. Zoom in on a particular part of the image by cropping the image. Display the image. In the report, explain what you are looking at in this part of the image. Comment about the textures, the colors being displayed, etc. The point of this question is to get you looking deeply at images and making observations.

3. **Resizing images:**
   a. Read in another image to be used for image sampling.

   b. Downsample the image by 10x in width and height. Use the *cv2.resize* command. If you have problems with this command, look at examples online and use the keyword "None" for the optional parameters. Display the downsampled image.

   c. Using the resize command, upsample the same downsampled image from part b by 10x back to its original resolution. This time, try two different interpolation methods: nearest neighbor and bicubic. Display them in the report. a

   d. Calculate the absolute difference between the ground truth image and the two upsampled images with the two methods (find the appropriate OpenCV command, should be one line of code for each image). Write out the difference images for both methods. Sum all the pixels in the difference image for the two methods, and report the number. Which method caused less error in upsampling? Explain when you might use one method over the other (you should have use cases for using both methods, not just using one method all the time).

Problem 3. *(Convolution as Matrix Multiplication)*

2D Convolution: We will implement 2D convolution as a matrix multiplication. In fact, this is used often in machine learning algorithms such as convolutional neural networks where one needs to perform several convolutions for each layer/filter bank of the algorithm. We will now implement 2D convolution as a matrix operation. Define h as follows:

$$h = \begin{bmatrix} 1 & 0 & -1 \\ 1 & \mathbf{0} & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

.

(a) First use image

$$I = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

. Assume zero-padding and no flipping of the convolution kernal (i.e. the center of the 3x3 kernal is the pivot point, and since the filter is symmetric, there is no reason for flipping). First, vectorize $\vec{I}$ where each row of the image is stacked vertically into a column vector, i.e. $\vec{I} = [1, 2, 3, 4, 5, 6, 7, 8, 9]^T$. Then write by hand a matrix H such that $H * \vec{I}$ is the vectorized output image. H should be a 25 x 9 matrix in this case. The output should be a $25x1$ vector, taht corresponds to a 5x5 output image that is row stacked. This part should be calculated by hand on paper.

(b) Now write a function *conv2dmatrix* that takes inputs $[image, H]$ and outputs $[image * H, time, error]$. This code should compute the same result as (a) which you did calculate numerically.

(c) Generalize your code so the image is any image of the user's choosing, and the new function can still compute the convolution as a matrix multiplication. The output should be an edge filtered image.

Problem 4. *(Fourier Domain Fun)*

(a) Read in a picture of your choice, and make it grayscale. Plot the Fourier transform (magnitude and phase) of the picture.

(b) Implement both a low-pass, high-pass filter, and a diagonal bandpass filter (of your choice) on the picture in the frequency domain. Make sure to plot the Fourier magnitudes of the image before and after filtering. Show the resulting filtered images.

(c) Phase Swapping: Select two images, compute their Fourier transform, and display their magnitude and phase images. Then swap their phase images, perform the inverse Fourier transform, and reconstruct the images (display these). Comment on how this looks perceptually. Try to modify the phase differently than just swapping them, and explain what observation you made.

(d) Hybrid Images: Take two images (of your choice), and try the hybrid images approach from the paper from Olivia et al `http://olivalab.mit.edu/publications/OlivaTorralb_Hybrid_Siggraph06.pdf`. I will show the best results in class!

Problem 5. *(Multiresolution Blending using Gaussian/Laplacian Pyramids)*

In this problem, You are going to practice performing multiresolution blending. Use the paper by Burt and Adelson (`http://ai.stanford.edu/~kosecka/burt-adelson-spline83.pdf`) as reference for this assignment.

(a) First pick two or more images that you wish to use for blending. Please be creative here in your choice of images and why you want to blend them. In the report, please explain your choice of images and the type of blending you want to do, and why you felt multiresolution blending would work for this problem.

(b) First write a code to generate the Gaussian and Laplacian pyramids of an image. You should confirm that your Laplacian pyramid can be collapsed back into the original image. You can use existing OpenCV commands as you want, but please show that your method is working properly.

(c) Make mask(s) that will act as the transition between your images (binary masks).

(d) Direct Blending: Try to directly blend the images by using $I = (1 - M) * I_1 + M * I_2$, where M is the mask and $I_{1,2}$ are the two images. If you have multiple images, you can use this formula multiple times as you stitch images together. Display this image.

(e) Alpha Blending: Try to blur the mask edge with a Gaussian $G(M)$, and then $I = (1-G(M)) * I_1 + G(M) * I_2$ (i.e. alpha blending or feathering). How does this look (does it look like one fruit?)

(f) Multiresolution blending: Write a function *multiblend* that takes in as inputs [$image1, image2$] and outputs [$blendedimage$]. Construct a Gaussian pyramid of the Mask, and Laplacian pyramids of the images (depth is your choice, but keep it fixed for all pyramids). Blend the images using the multiresolution blending algorithm shown in class. Try to play around with parameters, etc until you find a blending you like. In your report, discuss which of the three blendings (direct, alpha, multiresolution) worked the best for your imaging examples, and speculate why that might be the case.

Problem 6. *Pinspeck Cameras*
We are going to explore the anti-pinhole camera or pinspeck camera, and show some applications.

a. Read the paper: "Accidental Pinhole and Pinspeck Cameras" by Antonio Torralba and William T. Freeman `https://people.csail.mit.edu/torralba/research/accidentalcameras/`.

b. Write code to load in the movie "livingroom.mov" provided in the assignment. The OpenCV function *cv2.VideoCapture* will help read in movies and convert them to frames.

c. Print out the difference video, achieveing something similar to the video "livingroomoutput.avi". Talk about the choice of reference frame you use for subtraction (Torralba/Freeman discuss this in the paper, mention which strategy you use.)

d. Isolate a portion of the video that corresponds to the pinspeck image. See if you can match the scene outside the window that looks like "outdoorimage.png".

e. Using your own camera/phone, try to recreate a pinspeck image with a window or aperture. Show your original background image and image with occluder, the difference image, and the ground truth image outside the window. Note I'm not looking for perfection here, just a good effort at capturing a pinspeck image. You can see if you can acquire RAW images from your cell phone, or borrow a DSLR camera from a friend or the AME department (ask Dr. J for more info) if you want to improve the quality.

3 Grading and Report

The assignment report is very important for the grading of this assignment. We will return reports that we are not satisfied with the presentation, this is to help you practice improving your written communication skills. Grading breakdown is as follows:

- Problem 2: 10%
- Problem 3: 15%
- Problem 4: 20%
- Problem 5: 20%
- Problem 6: 20%

- Presentation (are answers clearly defined and easy to find, is code snippets and figures utilized well for the report): 15%

Please only upload the PDF of the final report (we do not want any code). Also acknowledge any web sources, classmate help that you used in this assignment in an acknowledgements section.