

# Problem Set 1: Images as Functions

## *(really, arrays or matrices of numbers)*

### Description

This problem set is really just to make sure you can load an image, manipulate the values, produce some output, and submit the code along with the report.

To start off this problem set, we will be setting up Python with three libraries: NumPy, SciPy and OpenCV. Get started on this as soon as possible by looking at our [installation tutorial](#). If you have trouble with the tutorial, please post any errors / questions on Piazza. Due to the variety of past experiences with Python, and the diversity of computing platforms out there, we expect that this will be a breeze for some, but harder for others. We expect that everyone will help each other set up and get started on this problem set.

So, please, if you are having problems, look for information on Piazza, as someone may have resolved it, if not, post a question with detailed specifics of your problem. Again, please help each other set up. We want to emphasize the goal of this problem set is to get you all setup so we can do more in later weeks.

### What to submit

Create a folder named ps1 with the following structure and contents:

ps1/

- input/ - directory containing input images, videos or other data supplied with the problem set
- output/ - directory containing output images and other generated files

Note: Output images must be stored with following mandatory naming convention:

**ps<problem set #>-<question #>-<part>-<counter>.png**

Example: ps1-1-a-1.png (see question 1-a)

PNG format is easier to manipulate as it is lossless, hence we recommend using it over JPEG.

- ps1.py - your main Python code for this problem set
- ps1\_report.pdf - A PDF file that shows all your output for the problem set, including images labeled appropriately (by filename, e.g. ps1-1-a-1.png) so it is clear which section they are for and the small number of written responses necessary to answer some of the questions (as indicated). Also, for each main section, if it is not obvious how to run your code please provide brief but clear instructions (no need to include your entire code in the report).
- \*.py - Any other supporting files, including Python modules, etc.

Zip it as ps1.zip, and submit on T-Square.

## Questions

### 1. Input images

- a. Find two interesting images to use. They should be color, rectangular in shape (NOT square). Pick one that is wide and one tall.

You might find some classic vision examples [here](#). Or take your own. Make sure the image width or height does not exceed 512 pixels.

Output: Store the two images as ps1-1-a-1.png and ps1-1-a-2.png inside the output folder

### 2. Color planes

- a. Swap the red and blue pixels of image 1

Output: Store as ps1-2-a-1.png in the output folder

- b. Create a monochrome image (img1\_green) by selecting the green channel of image 1

Output: ps1-2-b-1.png

- c. Create a monochrome image (img1\_red) by selecting the red channel of image 1

Output: ps1-2-c-1.png

- d. Which looks more like what you'd expect a monochrome image to look like? Would you expect a computer vision algorithm to work on one better than the other?

Output: Text response in report ps1\_report.pdf

### 3. Replacement of pixels (Note: For this, use the *better* channel from 2-b/2-c as monochrome versions.)

- a. Take the ~~inner~~ center square region of 100x100 pixels of monochrome version of image 1 and insert them into the center of a monochrome version of image 2

Output: Store the new image created as ps1-3-a-1.png

### 4. Arithmetic and Geometric operations

- a. What is the min and max of the pixel values of img1\_green? What is the mean? What is the standard deviation? And how did you compute these?

Output: Text response, with code snippets

- b. Subtract the mean from all pixels, then divide by standard deviation, then multiply by 10 (if your image is 0 to 255) or by 0.05 (if your image ranges from 0.0 to 1.0). Now add the mean back in.

Output: ps1-4-b-1.png

- c. Shift img1\_green to the left by 2 pixels.

Output: ps1-4-c-1.png

- d. Subtract the shifted version of img1\_green from the original img1\_green, and save the difference image.

Output: ps1-4-d-1.png (make sure that the values are legal when you write the image so that you can see all relative differences), text response: What do negative pixel values mean anyways?

### 5. Noise

- a. Take the original colored image (image 1) and start adding Gaussian noise to the pixels in the green channel. Increase sigma until the noise is somewhat visible.

Output: ps1-5-a-1.png, text response: What is the value of sigma you had to use?

- b. Now, instead add that amount of noise to the blue channel.

Output: ps1-5-b-1.png

- c. Which looks better? Why?

Output: Text response