Lab 5: Image Processing

1 Objectives

- **Be comfortable using OpenCV and Matplotlib**
- **Be able to open, write, and edit image files**

2 Resources

- **OpenCV Documentation:** `https://docs.opencv.org/3.4.1/d6/d00/tutorial_py_root.html`
- **Basic/Core OpenCV Operations:** `https://docs.opencv.org/3.4.1/d3/df2/tutorial_py_basic_ops.html`
- **Image Processing using OpenCV:** `https://docs.opencv.org/3.4.1/d2/d96/tutorial_py_table_of_contents_imgproc.html`

3 Homework Questions

3.1 Color Filtering (20 points)

1. **Implement the function calculate_spectrum(String) that takes the file path to an image as input and outputs a 3-element tuple representing the total color-intensity within each of the BGR layers in that image (i.e. (sum_blue, sum_green, sum_red)). Intensity refers to the value between 0 and 255 that each color channel can be assigned for a given pixel. Total intensity means the sum of every pixel's intensity value. Provided is a test picture test_image.png that contains a 100x600 purely blue region, a 100x600 purely green region, and 400x600 purely red region. You are encouraged to supply your own images for additional testing.**
2. **Implement the function get_dominant_color(String) that takes the file path to an image as input and outputs an integer indicating which (BGR) color channel has the highest sum intensity of the three. The output should be either 0, 1 or 2 corresponding to blue, green, or red, respectively. Break any ties with the lowest index channel. (e.g., if green(1) and red(2) are tied for dominance, output 1; if blue, green, and red are all tied for dominance, output 0). Consider how you could use the calculate_spectrum() function to help.**
3. **The numpy.item() method is much faster at accessing pixel data than normal python syntax. Implement the function measure_runtime_numpy(String) and also the function measure_runtime_python(String). These should measure how long it takes to run the sample functions touch_numpy(str) and touch_python(String), respectively, on test_image.png. Use either the time library's time.time() function or OpenCV's cv2.getTickCount() and cv2.getTickFrequency() functions to do this calculation. How many seconds did it take your computer to run touch_numpy(String)? How many seconds did it take your computer to run touch_python(String)? Note: Individual differences between computers mean that there is no single "correct" answer, besides that measure_runtime_numpy(String) should be faster than measure_runtime_python(String).**

3.2 Blur (20 points)

1. **Implement function blur_image(String), that takes an already loaded OpenCV image as an input. This function should apply a gaussian blur to the image, save it as Lastname_2a.png, and output the blurred values . Replace Lastname with your last name. The image generated**

should be included in your writeup you will submit along with your code. Use (5,5) as the specified kernel size when calculating Gaussian blur. Two sample images, BlurMe.png and Blurred.png, have been provided to help with testing your code.

2. Implement function findoutlines(String), that takes the file path to an image as input. The input image is expected to contain a few arbitrarily colored shapes against a black-ish background. This function should output an image of those shapes outlined visibly in white and should save it as Lastname_2b.png. Replace Lastname with your last name. Aside from the outlines, the generated image should be identical to the original image. Your function should work even in the presence of mild noise. Consider how to use blur_image(String) to combat this noise. Note: The phrase "black-ish background" means that the background will be dark enough to see both the shapes and outline, but will not be uniformly (0,0,0) intensity. Two sample images, BlueSquare.png, and OutlinedShape.png, have been provided to help with testing your code.

3.3 Autostereogram (10 points)

Given the stereo.py file shown in class, use as much of this code as you would like to create your own autostereogram using YOUR OWN pattern and depth map. You may write and add as many functions in addition to or instead of the functions included in stereo.py. When we run this file, it should output the pattern image, the depth map image, and the autostereogram.

3.4 Image Segmentation (20 points)

Given the segmentation.py file shown in class, use as much of this code as you would like to segment the five tiger jpeg images. Be sure to change the color values when thresholding the bit masks. Begin with tiger1.jpeg, and then evaluate how well your segmentation performs on the other four images. Do this for each of the five images. In a separate write-up document, explain any changes you have to make to each image in order to achieve better segmentation. For example, the values you use for tiger1.jpeg may not work for tiger5.jpeg, etc.

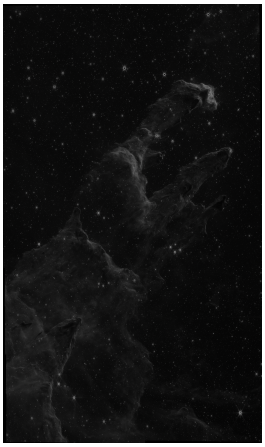
3.5 Astronomy Image Processing (20 points)

As discussed in class, image processing is used extensively in many of the pictures we see from space. NASA's James Webb Space Telescope sees the universe in infrared light, allowing us to see intimate details of star formations and the faint light of some of the galaxies formed more than 13 billion years ago. Biologically, we don't have the capability to see these images the way that Webb or Hubble would be able to. The brilliant color images you see are depictions of different bands of infrared light. But the actual images received are grayscale (see Figure 1). With the right software packages or editing tools, anyone can do their own image processing. All of these space photos are available for free to download (`https://mast.stsci.edu/portal/Mashup/Clients/Mast/Portal.html`). These images are in their own format called .fits.

Watch this video tutorial to see how it's done: `https://www.youtube.com/watch?v=UC6UQqmnOfA`. This tutorial is for the Southern Ring Nebula, but the same methods can be applied for any image. So, your job is to re-create the NASA Webb image of the Pillars of Creation. (When searching for target in MAST, look up M16 or eagle nebula.) You DO NOT NEED to use Python for this. Photoshop, or GIMP, or any other image editing software is fine for this. Python has a number of libraries that can be used to work on fits files and perform processing for astronomy, but that is beyond the scope of this assignment. Be warned that the files are large, and the six files you will need to download for this are slightly larger than 9 GB.

You do not need to re-create the image perfectly, but try to get as close as possible to the publicly released image. Then, have fun and change up the image however you like. You do not need to show any code for this, but you will need to upload the photos in their editable format (i.e., Photoshop PSD or XCF in GIMP). In the pdf write-up, you will show the six NIRCAM images downloaded from MAST, a side by side comparison of your re-creation of the Pillars of Creation and your stylized version versus the official one (in other words, there will be three

pictures, your re-creation, your "fun" one, and the official one). Also add any discussion on what you did to create your images.



(a) Near infrared camera image of Pillars of Creation

(b) Full colorized image of Pillars of Creation, JWST 2022

Figure 1: Comparison of single NIRCAM filter image with full colorized image.

4 Submitting Homework to ELMS

List all assumptions, and use comments to notate your code whenever possible. Clarity and style will matter. Your TF should be able to read your code and understand everything. You will be submitting three python files containing your code/functions for 3.1 and 3.2, a stereogram file, and a segmentation file. You will submit a pdf document containing your answers for 3.1.3, 3.4, and 3.5. And finally, you will be submitting two images you created for 3.5 in PSD or XCF format.

1. **lab5.py**
2. **stereo.py**
3. **segmentation.py**
4. **lab5_writeup.pdf**
5. **The two images created for 3.5 in PSD or XCF format, saved as image1 and image2**