Problem Set 4: Filtering, Template Matching, Derivative, and Gradient Images
Computational Perception and Artificial Intelligence

Description:

The tools of Gaussian filters, SSD and Correlation algorithms, and derivative images form the foundation for in depth image analysis and preparation for perception of lines, shapes, and stereo. This problem set will work through these tools in preparation for line and circle detection in the next unit. Note that with this problem set you will be saving images into the output folder located in ps04 directory. You will submit your work in a zipped folder called ps04.zip emailed to Mr. Michaud.

Setup:

A. Download and unzip the following folder into your Perception Lastname directory:

http://www.nebomusic.net/perception/ps04.zip

B. Open the `ps04.py` file with IDLE and place your name in the comments as indicated in the code.

C. For this problem set, you will need to use comments and functions to delineate each question. Use `ps02.py` and `ps03.py` as a guide to how to set up your code. Remember, you will get partial credit for attempts at the functions. Make sure you use comments!

D. You will also have an `output` folder in the ps04 directory. Several questions on this problem set require you to use the `cv2.imwrite(path, array)` function to save the image data to this folder. Make sure the file names are correct.

Questions:

1. Find a 512 x 512 pixel image. Save this image into your input folder. Import this image as a color array `img`.

2. Convert `img` to a grayscale image called `img_g` and then convert to the datatype of floating point numbers. (`np.float`). You may use `cv2.cvtColor()` function to make the grayscale image.

3. Create a Gaussian blur of `img_g` with a block size of 15 and a sigma of 2. Save this image as `img2_blurred.png` in the `output` folder in the `ps04` directory.

4. Create a 2D numpy array of 5 x 5 with the value of 0.04 in each element of the array. The datatype should be float. Name this array `box_filter`.

5. Use the `cv2.filter2D()` function and create a box filter blur with `img_g`. Save this blurred image as `img2_boxBlur.png` in the `output` folder.

SSD and Correlation Matching:

6. Take a 2 different group pictures with at least 3 individuals. (The same individuals in each picture). Make sure the individuals change position within the pictures. Reduce this picture to a pixel size under 1000 x 1000 pixels. (You may use Photoshop or other editing software). Save this picture as `group_image01.jpg` and `group_image02.jpg` in your input folder.

7. Import the group pictures into python as image arrays. Name these arrays `img_group1` and `img_group2`. Convert these arrays to grayscale and `np.float` datatypes.

8. Take 3 individual headshots of the 3 individuals in your pictures from step 1. Reduce the size of these images to under 512 pixels. Save these images as `patch01.jpg, patch02.jpg, patch03.jpg` in your inputs folder.

9. Import the individual pictures of the 3 individuals into python as grayscale images named `patch1, patch2, patch3`. Convert these arrays to grayscale and `np.float` datatypes.

10. The SSD algorithm using corner matching is as follows:

$$SSD(r,c) = \sum_{j=0}^{patchRows-1} \sum_{i=0}^{patchColumns-1} [patch(j,i) - img(r+j, c+i)]^2$$

Implement the SSD algorithm in a function called `matchSSD(img, patch)`. The function should return an SSD map (2D array of values with the lowest value representing the location of the corner of the best match). ***You may not use cv2.matchTemplate() function to implement this solution!***

11. Use the function `matchSSD(img, patch)` from question 3 match at least one of your patches with the group picture. Use the `np.where()` function to identify the lowest value and position (r, c) from the correlation map. With this data, use `cv2.rectangle()` and draw a red rectangle around the best match found by your function on the group picture array of your choice. Save this picture showing the match as `my_match.png` into the `output` folder.

12. Using the 3 image patches from step 2, use the `cv2.matchTemplate()` function and try to match the 3 faces to the images `group_image01.jpg` and `group_image02.jpg`. This might not work perfectly the first time. Adjust the sizes of the patches as needed to find the best match. The output should be an image with the 3 faces marked with a different colored rectangle around each face. Save this image as template_match.png into the output folder.

Derivative Images

13. We will now create x and y derivative images from your `img`. Define the following functions in python:

```
def getDeltaX(img):
    return img_deltaX # np.float

def getDeltaY(img):
    return img_deltaY # np.float
```

14. Using the `getDeltaX(img)` and `getDeltaY(img)` functions, write a function to return the gradient vector of an image. The Gradient vector should have the format (velocity, direction) and return numpy arrays of r x c for velocity and direction. The incoming img will be a grayscale image.

```
def getGradient(img):
    return distances, thetas
```

15. Use `getGradient(img)` function to get a gradient distance image from the `img_g` array you created in step 2. Call the gradient distance array `img_gradient` and save it as `img_gradient.png` in the output folder.