Problem Set 5: Derivatives, Gradients, and Edges
Computational Perception and Artificial Intelligence

Description:

In this problem set you will implement an edge detection algorithm using image derivatives with respect to *X* and *Y*. While Open CV provides several edge detection algorithms (such as Canny . . .), we will create our own in order to gain insight into how the rate of change in two directions *(X, Y)* can combine for direction and magnitude. Note that for this problem set you will create your own folder `ps05` with the needed directories input and output. You will write one Python module called `ps05.py` in the style and structure of our previous problem sets. Work will be submitted in a zipped folder called `ps05.zip` and emailed to Mr. Michaud.

Setup:

A. Create a directory `ps05`.

B. Inside the `ps05` directory, create two folders: `input` and `output`.

C. Create a python file `ps05.py`. Make sure the top lines of the file have your name, date, and honor pledge as shown below:

```
# Name:
# Date:

# Honor Pledge Below:
```

D. Remember that your code should have an exit sequence.

Questions:

1. Select Picture with dimensions smaller than 512 x 512 pixels. Save this to the `input` folder. Import this image into your code as a grayscale image array called `img`.

2. Use the Open CV2 Gaussian Blur function to blur your `img` image. Name this blurred image `img_noise` in your code. Save the image as `img_noise.png` in the `output` folder. (You might have to convert and normalize the image in another file before you save it)

Writing images to files

`cv2.imwrite(pathToFile, imageArray)`

<u>Example:</u>

- Convert image array to 0-255 np.uint8 datatype

```
img = img * 255
img = np.uint8(img)
```

- Use the cv2.imwrite(path, image) to write to file

```
cv2.imwrite('output\name.png', img)
```

3. Implement functions to map the gradient distance in each pixel. You may use the functions from the previous problem set. (You may copy and paste these functions over from PS04).

```
def getDeltaX(img):
    return img_deltaX

def getDeltaY(img):
    return img_deltaY

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

4. Using your gradient functions from #3, implement the algorithm to return an edge image. Name this function `getEdgeImage(img)`

```
def getEdgeImage(img):
     return E
```

Algorithm: Edge Image

Given: Grayscale image I with dimensions m x n

-Initialize a threshold value
-Initialize D and T from gradient distances and theta from I
-Normalize the D values
-Identify the indexes of D that are greater than a threshold
-Initialize E as 2D float matrix with zeros m x n = size of I
-Set the E values at indexes to be 1

-Return E

5. Use your `getEdgeImage()` function and generate an Edge Image from your blurred image from steps one and two. Name this image `img_myedge`. Save the image as `img_myedge.png` in the `output` folder.

6. Use the python open CV2 function Canny Edge and compare your results. Name this image `img_cannyedge`. Save the image as `img_cannyedge.png` in the `output` folder.

Note: You have to use a uint8 depth on the image (Unsigned 8 bit image) for the `cv2.Canny()` function. You also might need to experiment with the parameters minVal and maxVal (2nd and 3rd parameters) to get the best result.

Example:

```
image_copy = np.uint8(image)
img_edge_canny = cv2.Canny(image_copy, 100, 200)
```

7. Write a one paragraph response describing the differences between your Edge Detection Algorithm and the Open CV Canny Edge Algorithm. Include the two edge detection images `img_myedge.png` and `img_cannyedge.png` in your paragraph response. Save the paragraph response as `ps05report.docx` in the output folder.