CIS581: Computer Vision and Computational Photography
Project 1A: Canny Edge Detection
Due: Sept. 25, 2018 at 3:00 pm

Instructions

- This is an **individual** project. 'Individual' means each student must hand in their **own** answers, and each student must write their **own** code in the homework. It is admissible for students to collaborate in solving problems. To help you actually learn the material, what you write down must be your own work, not copied from any other individual. You **must** also list the names of students (maximum two) you collaborated with.
- You **must** submit your code online on Canvas. We recommend that you can include a README.txt file to help us execute your code correctly. Please place your **code** and **result images** into the top level of a single folder named `<Pennkey>_Project1.zip`
- This handout provides two versions of the code: MATLAB and Python. You are free to select **either one of them** for this project. A test script will be provided shortly which will call your functions to ensure that they will run inside the grading script.
- **Start early!** If you get stuck, please post your questions on Piazza or come to office hours!

Overview

The project focuses on understanding image convolution and edge detection. The final goal of this project is to compute the Canny Edges for any RGB image.

The main code structure is provided in **cannyEdge.m** or **cannyEdge.py** file:

E = cannyEdge(I)

- (INPUT) **I:** *H* x *W* x *3* matrix representing the RGB image, where H, W are the height and width of the input image respectfully.
- (OUTPUT) **E:** *H* x *W* binary matrix representing the canny edge map, where a '1' is an Edge position while a '0' is a Non-Edge pixel.

Your task is to implement three basic functions **findDerivatives.m/py**, **nonMaxSup.m/py** and **edgeLink.m/py** which correspond to the steps described in the lecture notes:

1. Apply Gaussian smoothing and compute local edge gradient magnitude as well as orientation.
2. Seek local maximum edge pixel in corresponding orientation.
3. Continue search in the edge orientation of detected edge points.

1 Gradient Computation

Goal Implement Gaussian smoothing, compute magnitude and orientation of derivatives for the image.

[Mag, Magx, Magy, Ori] = findDerivaties(I_gray)

- (INPUT) **I_gray:** *H* x *W* matrix representing the grayscale image.

- (OUTPUT) **Mag:** H x W matrix represents the magnitude of derivatives.
- (OUTPUT) **Magx:** H x W matrix represents the magnitude of derivatives along x-axis.
- (OUTPUT) **Magy:** H x W matrix represents the magnitude of derivatives along y-axis.
- (OUTPUT) **Ori:** H x W matrix represents the orientation of derivatives.

2 Detect Local Maximum

Goal Find local maximum edge pixel using **non-maximum suppression** along the line of the gradient.

M = nonMaxSup(Mag, Ori)

- (INPUT) **Mag:** H x W matrix representing the magnitude of derivatives.
- (INPUT) **Ori:** H x W matrix represents the orientation of derivatives.
- (OUTPUT) **M:** H x W binary matrix represents the edge map after non-maximum suppression.

3 Edge Linking

Goal Use **hysteresis** to link edges based on high and low magnitude thresholds.

E = edgeLink(M, Mag, Ori)

- (INPUT) **M** H x W binary matrix represents the edge map after non-maximum suppression.
- (INPUT) **Mag:** H x W matrix represents the magnitude of derivatives.
- (INPUT) **Ori:** H x W matrix represents the orientation of derivatives.
- (OUTPUT) **E:** H x W binary matrix represents the final Canny Edge map.

4 Starter Code Clarifications

- Please do not modify the main file **cannyEdge**.
- **Utils File:** If you choose to complete this project via Python, we provide one more functional file called **utils.py** to contribute your implementation. It contains functions to visualize output image, build Gaussian distribution, and convert the image color space. Please feel free to import these functions in your code but **do not modify** them.
- **Helper File:** You are allowed and encouraged to come up with your own helper functions in **helpers.m/py** file such as local thresholding or line segmentation. Of course, you are not allowed to use packages that are not in the list.

5 Test and Submission

- Good test images can be found in the Berkeley Segmentation Dataset.
  We will also provide a few images with the test script in the Dataset folder. Test your code with these images before submitting the code.
- If you use new images, resize them to the size of the images given to you before running them through your code.
- We have provided a test script for MATLAB and Python for this project. Extract the contents of the test script to the same directory as your functions and run `Test_script.m/py` in MATLAB/Python. This does not test the correctness of your code itself, only whether the output is in the expected format. Additionally, when grading, we'll be calling your functions in the same manner, so make sure they work as you'd expect on the sample in the test script.

- Collect all your source code files and test images into a folder named as `<Pennkey>_Project1`. Zip this folder and submit it to Canvas. Any break in this rule will lead to a failure in the test script. Only submit codes pertaining to your language of implementation. For example: If you choose to do the project in Python, do not submit the MATLAB folder containing the MATLAB starter codes.

6 Extra Credit (Optional)

- In this part of the project, you are going to run the Canny Edge Detection algorithm in a short video. The challenging part of this project is to find a way to automatically set the threshold values for each frame in the video. The testing videos are in the folder named "challenging videos".
- Put all the code, output video (edge detection) and a README file into the same folder. The README file should describe how to run your code and what techniques you used. Please name the folder as `<Pennkey>_Project1ExtraCredit` and submit it to canvas.