CSL461: Digital Image Analysis

Lab 2: Fundamentals

Aim:

- To help understand the concepts of pixel neighborhoods and connected pixel sets and piece-wise intensity transformations

Let's get started!

- Create a directory structure to hold your work for this course and all the subsequent labs:
  - Suggestion: `CSL461/Lab2`

Neighbors & Connectivity

- In class we discussed different neighbors of a pixel (refer to Fundamentals2 Slides on Moodle):
  - 4 – Neighbors (Vertical & Horizontal)
  - Diagonal Neighbors
  - 8 – Neighbors (Vertical, Horizontal & Diagonal)
- We also discussed the idea of connectivity:
  - Two neighboring pixels x, y are said to be connected if $\left| Ix - Iy \right| < T$ where T is a fixed threshold and Ix, Iy are the intensity values of pixels x, y respectively.
- Create a function called `ConnectedSet` to find all the pixels connected to a given pixel `s`. It can be implemented by maintaining:
  - A list of pixels `B` which are known to be connected to `s`, but have not yet been searched
  - A segmentation image `Y` which is equal to 1 for pixels which are known to be connected to `s` and is zero otherwise.
- Create a test script (`TestConnected.m`) that:
  1. Prints out the image `img22gd2.tif`
  2. Prints out of the image showing the connected set for s = (67, 45), and T = 2.
  3. Prints out of the image showing the connected set for s = (67, 45), and T = 1.
  4. Prints out of the image showing the connected set for s = (67, 45), and T = 3.
  5. Use `subplot` to print all the above images in the same figure with relevant titles!!
- Use the `ConnectedSet` function to extract all the connected sets in the image `img22gd2.tif`. You can do this by indexing through the image in raster order and applying the `ConnectedSet` subroutine at each pixel that does not yet belong to a connected set. Note that for a small threshold, the size of most connected sets for this image will be small, resulting in a large number of connected sets in the segmentation.

- Generate a segmentation of the image consisting of connected sets containing greater than 100 pixels. Number each of these large connected sets sequentially starting at 1. All remaining connected sets should be labeled as 0. There will be fewer than 255 large connected sets, so you can store the label for each pixel as a 2-D unsigned character array. Save this 2-D array as a monochrome TIFF image, `segmentation.tif`.
- To view your segmentation clearly, you will need to scramble the colormap to provide contrast between the distinct regions. You can do this in Matlab with the following commands:

```
x=imread('segmentation.tif');
N=max(x(:));
image(x)
colormap(rand(N,3))
axis('image')
```

- Print this color segmentation of the image (*you can experiment with and choose T!)*
- Create a test script (`TestSegmentation.m`) that showcases your segmentation work

Intensity Transformation

- We have discussed intensity transformation functions in detail in class and you have gained necessary implementation skills from Lab1.
- Now, let's apply them to a real life challenge!
- Although not very common among us Indians, red-eye effect is a common phenomenon in flash photography where pupils in the eyes appear red!
- Automatic Redeye Detection and Correction Algorithms are now very common in most cameras and cell phones!!
- Due to the inherent complexity, we will not deal with automatic detection aspect now, but will only work on correction.
- Four sample images with red eyes are provided with this lab. You can read these images using the command `A = imread('filename.jpg');` It will return a matrix of dimensions: `X x Y x 3` where the third dimension has the R,G,B components.
- Come up with your own red-eye correction algorithm (`RedEyeRemoval.m`) using piece-wise intensity transformations! (Don't worry if other parts of the image with red color get distorted!)
- Two important aspect of this algorithm are (1) Figuring out the range of intensities to modify (2) Deciding on what the modified intensity values should be!
- Your function should take an input image and display/output the corrected image.
- Create a Test script (`TestRedEye.m`) that tests your algorithm on all the sample images!

Submitting your work:

- All source files and class files as one tar-gzipped archive.
  - When unzipped, it should create a directory with your ID. Example: **P2008CS1001–L1** (NO OTHER FORMAT IS ACCEPTABLE!!! Case sensitive!!!)
  - ***Negative marks if the TA has to manually change this to run his/her scripts!!***
- Source / class files should include the following: (Case-Sensitive file names!!)
  - `ConnectedSet.m`
  - `TestConnected.m`
  - `TestSegmentation.m`
  - `RedEyeRemoval.m`
  - `TestRedEye.m`
- ***Negative marks for any problems/errors in running your programs***
- Submit/Upload it to Moodle