ECE558 Project04-Final

Due 11pm, 12/06/2019

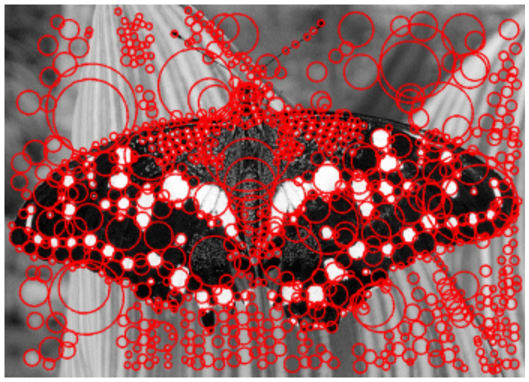
How to submit your solutions: put source code folder [your_unityid_code], your report (word or pdf) and results images (.png) in a folder named [your_unityid_project04] (e.g., twu19_project04), and then compress it as a zip file (e.g., twu19_project04.zip). Submit the zip file through **moodle**.

Important Note: No late days for the final project due to the deadline of submitting the final grades to the university office. We will **NOT** accept any replacement of submission after deadline, even if you can show the time stamp of the replacement is earlier than the deadline. So, **please double-check if you submit correct files**.

Team: You can form a team with no more than 3 members. It is also a good time to learn from each other and work together to improve the codes in project02/03 to be re-used /updated in this project. You can share and merge your codes in project02/03 within the team ONLY.

Project description: As illustrated by the below two images, the objective of this project is to implement a Laplacian blob detector.

An input image


A blob detection result

Algorithm outline:

- Generate a Laplacian of Gaussian filter.
- Build a Laplacian scale space (core function to be implemented on your own, you need to reuse your codes in project3 and build on top it), starting with some initial scale and going for n iterations:

    - Filter image (core function to be implemented on your own, you need to reuse your codes in project2 and improved versions in project3 if had) with scale-normalized Laplacian at current scale.
    - Save square of Laplacian response for current level of scale space.
    - Increase scale by a factor k.
- Perform non-maximum suppression (core function to be implemented on your own, new for project4) in scale space.
- Display resulting circles at their characteristic scales.

Test images: four images are provided and the testing results are required. Note that your output may look different depending on your threshold, range of scales, and other implementation details. In addition to the images provided, **also run your code on at least four images of your own choosing (required)**.

Requirement: Your report need to provide details of how you elaborate the algorithm outline and how you implement them. Your code should be self-contained: Given an input RGB image, it will generate the detection results (see the example above). Your code can be run without extra packages unless clear installation instructions are provided step-by-step. You need to implement the entire algorithm independently without built-in functions used for core components, except for the image I/O and displaying functions. E.g. You can reuse your convolution code.

5-point Bonus: It will be given to the top-5 projects. The projects will be ranked using the following criteria:

- The code is clean and self-contained. It can run without extra packages unless clear installation scripts are provided. Basically, we need to 1-click code to test the results.
- The code is fast. We will compare the average runtime for the four provided images.
- The report gives details of how you implement the code, how you address different issues.
- If you are not interested in competing for the 5 points, please make it clear in your report.
- If you are indeed interested in this, it may be a good opportunity to reshape all the codes you have implemented in this class. Be proud of your achievement by running your code faster in a cleaner way!