COMP9517: Computer Vision

2022 Term 2

Assignment Specification

Maximum Marks Achievable: 10

This assignment is **worth 10% of the total course marks**.

The assignment files should be submitted online.
Instructions for submission will be posted closer to the deadline.
Deadline for submission is Week 4, Tuesday 21 June 2022, 18:00:00.

Deliverables: You must submit the following items:

1. A report explaining the approach you have taken in Tasks 1, 2, and 3. The report can be written in Word, LaTeX, or a similar text processor, but must be submitted as a PDF and include your name and zID on the first page. It must also show the sample input images and the intermediate and final output images obtained. No templates are provided, but the report must be no more than 5 pages (A4), and must be single column, use 11 points Times New Roman or similar font, and have 2.5 cm margins.
2. Output images as separate files (in .png or similar lossless format).
3. Source code in the form of a Jupyter notebook (.ipynb).

Submission: Submit all deliverables in a single zip-file. Instructions for submission will be posted closer to the deadline. To avoid confusion for the tutor/marker, who may handle many submissions, put your zID in all file names (e.g. **zID_Report.pdf** for the report and **zID_Notebook.ipynb** for the notebook).

Software: You are required to use OpenCV 3+ with Python 3+ and submit your code as a Jupyter notebook (see deliverables above and requirements below).

Objectives: This assignment is to familiarise you with basic image processing methods. It also introduces you to common image processing and analysis tasks using OpenCV.

Learning Outcomes: After completing this assignment, you will have learned how to:

1. Open and read image files.
2. Display and write image files.
3. Perform mathematical operations on images.
4. Apply basic image filtering operations.

5. Perform image adjustment and restoration.

Description: Many computer vision applications use image processing techniques to suppress artifacts and enhance relevant features to prepare images for quantitative analysis.

The goal of this assignment is to write image processing code that can open a digital image, perform a sequence of pixel manipulation operations (step-by-step as listed under **Tasks** below), in order to remove background shading.

Below is an image (on the left) from an astronomy project where the goal is to count and measure the stars. However, these measurements are impeded by the haze of the Milkyway in the background. Thus, the shading needs to be removed (as on the right).

Tasks: This assignment consists of three tasks, described below. Each task depends on the previous one, so you need to do them in the given order.

<u>Task 1 (4 marks): Background Estimation</u>

Open the given image **Milkyway.png**. Like the example shown above, it has background shading patterns that need to be removed. The first step to get rid of the shading is to make an accurate estimate of the background of the image.

Write an algorithm that performs the following two steps:

1. Create a new image, which we will call image A, with the same size (number of pixel rows and columns) as the input image, which we call image I.
2. Go through the pixels of image I one by one, and for each pixel (x,y) find the **minimum** gray value in a neighbourhood (see below) centred around that pixel, and then write that minimum value in the corresponding pixel location (x,y) in A.

The resulting image A is called the **min-filtered** image of input image I.

The neighbourhood is a square of size N x N pixels, where N is a free parameter of the algorithm and typically has an odd value (3, 5, 7, …), with the pixel (x,y) under consideration being the centre pixel. Manually try different values of N and mention in your report what is the smallest value of N that causes the stars in I to visually disappear altogether in image A. Also explain why this value of N, and not smaller values, causes the stars to disappear, and what is the effect of taking larger values than this smallest value.

The min-filtering causes the gray values in A to be smaller than the actual background values in I, so a correction is needed. Extend your algorithm with two more steps:

3. Create another new image, which we call image B here, of the same size as I and A.
4. Go through the pixels of A one by one, and for each pixel (x,y) find the **maximum** gray value in a neighbourhood of the same size (N x N pixels) centred around that pixel, and then write that maximum gray value to (x,y) in B.

The resulting image B is called the **max-filtered** image of the image A.

In your report, include image B computed from **Milkyway.png**.

<u>Task 2 (1 mark): Background Subtraction</u>

Now that your algorithm can estimate the background B of an image I, removing the shading artifacts from I can be done simply by subtracting B from I pixel by pixel, resulting in the output image O. Extend your algorithm to perform this subtraction.

In your report, include image O computed from **Milkyway.png**.

<u>Task 3 (5 marks): Algorithm Extensions</u>

Open the given image **Cells.png**. It is a microscopy image showing many cells (the dark blobs) from a biology project. Like **Milkyway.png**, the image has background shading patterns (also dark) that need to be removed. There are three main differences between the two images:

- The sizes of the images are different.
- The sizes of the objects (stars versus cells) in the images are different.
- The objects in **Cells.png** are dark and the background is bright (except for the dark shading patterns), whereas in **Milkyway.png** the objects are bright and the background is dark (except for the bright shading patterns).

Further extend your algorithm as follows:

5. Ensure your algorithm can deal with input images of arbitrary size. This is a matter of retrieving the number of rows R and columns C in the beginning of your code and using R and C as the limits in the loops when going over the pixels.
6. Put the neighbourhood size parameter N at the beginning of your code so that users can

easily find it and adjust the value as needed.

7. Add a parameter M (or rather a flag) at the beginning of your code which determines the order of the min-filtering and max-filtering operations. Specifically:
   * If the user sets M = 0, the algorithm should first perform min-filtering (image I to A), then max-filtering (image A to B), and then pixelwise subtraction (O = I – B).
   * If the user sets M = 1, the algorithm should first perform max-filtering (image I to A), then min-filtering (image A to B), and then pixelwise subtraction (O = I – B + 255).

In your report, explain why you must use the M = 0 (and not M = 1) variant of your algorithm to make it work for **Milkyway.png**, and conversely why you must use the M = 1 (and not M = 0) variant to make it work for **Cells.png**. Also explain why the M = 1 variant requires adding 255 (for 8-bit images) to each subtracted pixel value in image O (see above).

Furthermore, mention what is a good value of N for **Cells.png** and why. And include images B and O computed from **Cells.png** in your report.

Coding Requirements

For all three tasks, implement all mentioned image processing operations yourself. That is, **do not use library functions** from OpenCV (such as `cv2.dilate()`, `cv2.erode()`, `cv2.morphologyEx()`), the Python Imaging Library (PIL) (such as `MinFilter()`, `MaxFilter()`), and Scikit-Image Library (such as `white_tophat()`, `black_tophat()`). Using these functions instead of your own implementation will result in deduction of points.

In your Jupyter notebook, the input image should be readable from the location specified as an argument, and all output images should be displayed in the notebook environment. In other words, all cells in your notebook should have been executed so that the tutor/marker does not have to execute the notebook again to see the results.

Released: 7 June 2022