CPS843/CP8307 Introduction to Computer Vision, Winter 2016: Assignment 4
Due: Sunday, April 17th, 11:59pm

All programming questions are to be completed in MATLAB. CPS843 students can work in groups of two, if they choose, and CP8307 students are to work individually. For those working in groups, only submit one copy and clearly indicate the group members. Include a file that will step through the answers to the entire assignment, and name this file `a4_script.m`.

Do not share code with those outside your group or use code from the web (other than the code instructed to use). We will be checking for copying and reserve the right to give offenders (both copier and source) a zero on this assignment and pursue academic sanctions.

1 Convolution (15 points)

In MATLAB, if you place the command `tic` before your script and `toc` after your script, MATLAB will return the total execution time.

1. **[5 points]** Use `conv2` to convolve an image by a 2D Gaussian kernel in the spatial domain. Plot the execution time to perform convolution on a 640×480 image using the following dimensions for the Gaussian kernel: 3×3, 5×5, 7×7, 13×13, 21×21, 31×31, 41×41, 51×51 and 71×71. Use MATLAB's `plot` function to plot the kernel size (horizontal dimension) vs. the execution time (vertical dimension).
2. **[5 points]** Perform convolution in the frequency domain (i.e., DFT, multiply and inverse DFT), plot the time taken to convolve a 640×480 image with a 2D Gaussian kernel with the following dimensions: 3×3, 5×5, 7×7, 13×13, 21×21, 31×31, 41×41, 51×51 and 71×71. Overlay this plot onto the spatial convolution plot from the previous question using MATLAB's `hold on` function.
3. **[5 points]** Describe the plots you just generated. What conclusions can you draw between spatial- and frequency-based convolution?

2 Hybrid Image (20 points)

1. **[20 points]** For this question you will create a *Hybrid image* as described in the lecture, e.g., Marilyn Monroe + Albert Einstein. The images used to generate this illusion are of your own choosing. To optimize the appeal of the illusion, you may have to: (i) adjust the alignment of the images using photo editing software (or MATLAB) and (ii) adjust the lowpass and highpass cutoff frequencies by varying the standard deviations of the Gaussians used to generate the respective filters. All filtering and image compositing should be performed in the frequency domain. Beyond technical correctness, marks will be awarded for the appeal of the illusion, so be creative.

3 Seam carving (20 points + 10 points bonus)

1. **[20 points]** Seam carving is a procedure to resize images in a manner that preserves "important" image content. A video demo is available on YouTube. The general steps for seam carving are as follows:

(a) Compute the energy image, E, for the input image, e.g., the sum of the gradient magnitude images computed for each of the three colour channels of the input image.

(b) Create a scoring matrix, M, with spatial image dimensions matching those of the input image.

(c) Set the values of the first row of the scoring matrix, S, to match those of the energy image, E.

(d) Set the values of every entry in the scoring matrix to the energy value at that position and the minimum value in any of the neighbouring cells above it in the seam, i.e.,

$$M(x,y) = E(x,y) + \min\Big(M(x-1,y-1), M(x,y-1), M(x+1,y-1)\Big), \tag{1}$$

where $M(x,y)$ is the cost of the lowest cost seam crossing through that point. This minimization procedure is an instance of *dynamic programming*.

(e) Find the minimum value in the bottom row of the scoring matrix. The corresponding position of the minimal value is the bottom of the optimal seam.

(f) Using $M(x,y)$, trace back up the seam by following the smallest value in any of the neighbouring positions above.

(g) Remove the seam from the image.

(h) To reach a desired resized image, you will have to repeat the above procedure. Note that you will have to recompute the energy matrix (and scoring matrix) each time to take into account changes in the resized image.

Your task is to write a MATLAB function that takes in an image and the desired new resolution by removing the necessary horizontal and vertical seams and returns the resized image. You can implement the seam carving routine with a **single** helper function that removes all the necessary vertical (horizontal) seams. You first call the helper function to remove the vertical (horizontal) seams, transpose the result of this function and then call the seam removal function again with the transposed image as the input to remove the horizontal (vertical) seams. In addition to submitting your code, submit resizing outputs for the `Ryerson` image to 640×480 and 720×320. Also submit an image of your own and its resized output.

2. **[10 points]** (Bonus): Implement image expansion by inserting seams, see the original paper for details.

4 Optical flow estimation (35 points)

In this part, you will implement the Lucas-Kanade optical flow algorithm that computes the pixelwise motion between two images in a sequence. Compute the optical flow fields for the three image sets labeled `synth, sphere and corridor`. Before running your code on the images, you should first convert your images to grayscale and map the intensity values to the range [0, 1].

1. **[20 points]** Recall from lecture, to compute the optical flow at a pixel, compute the spatial derivatives (in the first frame), I_x and I_y, compute the temporal derivative, I_t, and then over a window centred around each pixel solve the following:

$$\begin{pmatrix} \sum\sum I_x I_x & \sum\sum I_x I_y \\ \sum\sum I_x I_y & \sum\sum I_y I_y \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = -\begin{pmatrix} \sum\sum I_x I_t \\ \sum\sum I_y I_t \end{pmatrix}. \tag{2}$$

Write a MATLAB function, call it `myFlow`, that takes as input two images, $img1$ and $img2$, the window length used to compute flow around a point, and a threshold, τ. The function should return three images, u and v, that contain the horizontal and vertical components of the estimated optical flow, respectively, and a binary map that indicates whether the flow is valid.

To compute spatial derivatives, use the five-point derivative of Gaussian convolution filter `(1/12)*[-1 8 0 -8 1]` (make sure the filter is flipped correctly); the image origin is located in the top-left corner of the image, with the positive direction of the x and y axes running to the right and down, respectively. To compute the temporal derivative, apply Gaussian filtering with a small σ value (e.g., 3×3 filter with $\sigma = 1$) to both images and then subtract the first image from the second image. Since Lucas-Kanade only works for small displacements (roughly a pixel or less), you may have to resize the input images (use MATLAB's *imresize*) to get a reasonable flow field. *Hint: The (partial) derivatives can be computed once by applying the filters across the entire image. Further, to efficiently compute the component-wise summations in (2), you can apply a smoothing filter (e.g., box filter, Gaussian, etc.) on the image containing the product of the gradients.*

Recall, the optical flow estimate is only valid in regions where the 2×2 matrix on the left side of (2) is invertible. Matrices of this type are invertible when their smallest eigenvalue is not zero, or in practice, greater than some threshold, τ, e.g., $\tau = 0.01$. At image points where the flow is not computable, set the flow value to zero.

2. **[5 points]** Visualize the flow fields using the function `flowToColor`. Play around with the window size and explain what effect this parameter has on the result.

3. **[10 points]** Another way to visualize the accuracy of the computed flow field is to warp *img*2 with the computed optical flow field and compare the result with *img*1. Write a function, call it `myWarp`, that takes *img*2 and the estimated flow, u and v, as input and outputs the (back)warped image. If the images are identical except for a translation and the estimated flow is correct then the warped *img*2 will be identical to *img*1 (ignoring discretization artifacts). *Hint: Use MATLAB's functions* `interp2` *(try bicubic and bilinear interpolation) and* `meshgrid`*. Be aware that* `interp2` *may return NaNs. In particular, this may occur around the image boundaries, since data is missing to perform the interpolation. Make sure your code handles this situation in a reasonable way.*

   Visualize the difference between the warped *img*2 and *img*1 by: (i) take the difference between the two images and display their absolute value output (use an appropriate scale factor for `imshow`) and (ii) using `imshow` display *img*1 and the warped *img*2 consecutively in a loop for a few iterations, the output should appear approximately stationary. When running `imshow` in a loop, you will need to invoke the function `draw now` to force MATLAB to render the new image to the screen.