EECS 4422/5323 Computer Vision, Winter 2023
Assignment 4 - Optical Flow

All programming questions are to be completed in Python. You will be working individually on this assignment. Include a file that will step through the answers to the entire assignment, and name this file `a4_script.ipynb`.

Do not discuss or share code with others or use code from the web (other than the code instructed to use). We will be checking for plagiarism using MOSS and reserve the right to give offenders (both copier and source) a zero on this assignment and pursue academic sanctions. Finally, we may ask you to explain the details about the answers in your submission. If you cannot explain how you arrived at the answer, you will receive a zero on that part of the assignment.

1 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:

$$\begin{pmatrix} \sum\sum I_x I_x & \sum\sum I_x I_y \\ \sum\sum I_x I_y & \sum\sum I_y I_y \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = - \begin{pmatrix} \sum\sum I_x I_t \\ \sum\sum I_y I_t \end{pmatrix}. \tag{1}$$

   Write a Python 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 tap (central differences) derivative 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 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 (1), 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 (1) 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.

   Visualize the flow fields using the function `flowToColor`. Play around with the window size and explain what effect this parameter has on the result.

2. **[5 Points]** Another way to visualize the accuracy of the computed flow field is to warp $img2$ with the computed optical flow field and compare the result with $img1$. Write a function, call it `myWarp`, that takes $img2$ 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 $img2$ will be identical to $img1$ (ignoring discretization artifacts). *Hint: Use `scipy.interpolate.interp2d()` (try cubic and linear interpolation) and `numpy.meshgrid()`. Make sure your code handles boundary conditions in a reasonable way.*

Visualize the difference between the warped $img2$ and $img1$ by: (i) take the difference between the two images and display their absolute value output (use an appropriate scale factor for `plt.imshow()`) and (ii) using `matplotlib` display $img1$ and the warped $img2$ consecutively in a loop for a few iterations, the output should appear approximately stationary. When running `matplotlib` plots in a loop, you will need to invoke `matplotlib's flush_events` function to force Python to render the new image to the existing plot. You can find an example usage of this function at the following link.

3. **[10 Points]** In this question you will implement the Kanade-Lucas-Tomasi (KLT) tracker. The steps to implement are as follows:

   - Detect a set of keypoints in the initial frame using the Harris Corner detector. Here, you can use the pseudocode outlined in the lecture as a starting point.
   - Select 20 random keypoints in the initial frame and track them from one frame to the next; for each keypoint, use a window size of 15×15. The tracking step consists of computing the optical flow vector for each keypoint and then shifting the window, i.e., $x_i^{t+1} = x_i^t + u$ and $y_i^{t+1} = y_i^t + v$, where i denotes the keypoint index and t the frame. This step is to be repeated for each frame using the estimated window position from the previous tracking step.
   - Discard any keypoints if their predicted location moves out of the frame or is near the image borders. Since the displacement values (u, v) are generally not integer value, you will need to use interpolation for subpixel values; see `scipy.interpolate.interp2d()`.
   - Display the image sequence and overlay the 2D path of the keypoints using line segments.
   - Display a separate image of the first frame and overlay a plot of the keypoints which have moved out of the frame at some point in the sequence.

   For this question, you will use the images from the `Hotel Sequence`.