HVCA - Exercise 2

April 3, 2017

Deadline: Wednesday, 3/5/2017 at 09:00.

Please submit the exercise in a single pdf file including all the matlab code you wrote and your answers to all questions.

In this exercise you will implement the regularized (or Bayesian) Lucas Kanade algorithm for motion estimation. That is, you will find the velocity vector v that solves the following equation:

$$Av = b$$

with:

$$A = \begin{pmatrix} \sum I_x^2 + \lambda & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 + \lambda \end{pmatrix}$$

and

$$B = -\begin{pmatrix} \sum I_x I_t \\ \sum I_y I_t \end{pmatrix}$$

We will implement the algorithm in its iterated form, where in each iteration we use as an initial guess the velocity computed in the previous iteration (this is equivalent to developing the Taylor series around that vector). To initialize the algorithm, we will compute a single iteration of the algorithm on a pair of blurred and downsampled images, where we expect a better guess, especially in complex, natural images.

1. Write a function named `ImageDerivatives.m` that receives two images as inputs (the first and the second images in a sequence) and returns three values: I_x, I_y and I_t, those being the derivatives of the images across the X dimension, the Y dimension, and time. Use the following guidelines:

(a) The derivatives are obtained by convolving the two images with a kernel. Use the following kernels:

$$K_y = \frac{1}{4}\begin{pmatrix} -1 & -1 \\ 1 & 1 \end{pmatrix}, \; K_x = \frac{1}{4}\begin{pmatrix} 1 & -1 \\ 1 & -1 \end{pmatrix}, \; K_t = \frac{1}{4}\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}.$$

(b) I_x is the sum of the convolution of each image with the appropriate kernel (K_x)[1]

(c) I_y is the sum of the convolution of each image with the appropriate kernel (K_y) .

(d) I_t is the subtraction of the convolution of the first image with K_t from the convolution of the second image with K_t.

(e) You may use Matlab's `conv2` function. If you do, use the '`same`' option that returns a matrix the same size as the original image, discarding the borders which we would have eliminated anyway to avoid edge artifacts.

2. Write a function named `LK_alg.m` which implements the iterative algorithm. The function should have six arguments: I_1, I_2 , λ, *mask*, *v_initial*, *num_iterations*, and a single output v. I_1, I_2 are the images, λ is the algorithm's parameter which in the Bayesian formulation is the ratio between the noise variance and the prior variance, *mask* is a binary matrix the same size as the image that defines the pixels over which the summation should be made, *v_initial* is the initial guess for the velocity, and *num_iterations* is the number of iterations to be computed. Use the following guidelines:

   (a) In each iteration we use the previous velocity estimate *v_prev* to compute I_2^w , a warped version of I_2 that has been translated by that velocity. We then calculate the velocity v between I_1 and I_2^w, and we sum that velocity to *v_prev* to obtain the new estimate.

   (b) Use the provided function `warp` that receives as arguments I_2 and v and returns as outputs I_2^w and *warpMask*. *warpMask* defines which pixels of I_2^w are valid, since the translation needs some pixels that are not available. Multiply term-by-term *warpMask* with the current mask to obtain the new mask.

3. Write a function named `blur_dowsample.m` which receives as argument an image I and returns an image that has been blurred and downsampled. To blur an image, convolve it with a Gaussian kernel. You may use the provided kernel in `GaussKernel.mat` or create your own. If you use the `conv2` function remember to use the '`same`' option. Downsample the image by taking every second pixel in both dimensions. In Matlab, you may use ':' (the colon operator).

[1] The reason why we use the sum of derivatives of *both* images is that this provides a better approximation of the images (by using the average of derivatives of I_1 and I_2).

4. Write a function named `Full_LK_alg.m` which receives as arguments I_1, I_2 , λ, *mask*, *num_iterations* , and returns the estimated velocity v. You should first blur and downsample both images, get a *v_blurred* estimate using a single iteration of the algorithm (with a $\begin{pmatrix} 0 & 0 \end{pmatrix}$ initial velocity guess), and the use 2**v_blurred* as the initial estimate for the iterative algorithm (this time with the original images). The 2 factor is required because of the downsampling.

5. Load the two frames from the flower garden sequence. View these frames using the provided function `mymovie.m`. What is moving faster, the tree or the flowers? Now run your function on subimages that contain the flowers and on subimages that contain the tree with λ= 0. What are the velocities calculated by your algorithm? In Matlab, use: `im=double(imread('imagename'));` to read an image.

6. Use the provided function `GausSpot.m` to create pairs of frames of a 128x128 Gaussian spot that moves with different velocities in the X axis. First, create a wide Gaussian spot ($\sigma = 10$) and plot the estimation error (the L_2 difference between the velocities calculated by your algorithm versus the true velocities) as a function of the true velocity's speed. Use a single iteration of the algorithm. How does the error vary as a function of the speed? Repeat the experiment with a narrow spot ($\sigma = 1$). Discuss your results.

7. Use the provided function `rhombusMovie.m` to generate a thin rhombus and a fat rhombus. Plot them. Calculate and plot the IOC and VA predictions for these two stimuli assuming both lines are moving to the right with velocity 1. Run your algorithm with $\lambda = 0.01$ for these two stimuli with varying degrees of contrast (between 0 and 1). What happens to the algorithm output as the contrast is lowered? Try with different numbers of iterations. Compare these results to your own percept by looking at the rhombus demo in the course's moodle.

8. This question is purely theoretical. We define x as a random variable which is normally distributed such that $x \sim N(\mu_p, \sigma_p^2)$. We wish to measure x but we can only use a noisy measurement tool, which gives us a noisy measuremnt y that is distributed such that $y|x \sim N(x, \sigma_y^2)$. Show (full proof including all justifications) that the MAP estimate (Maximum APosteriori - the most probable estimate for x given the measurement y) for x given y is:

$$\hat{x} = \arg\max_x p(x|y) = \frac{\frac{1}{\sigma_p^2}\mu_p + \frac{1}{\sigma_y^2}y}{\frac{1}{\sigma_p^2} + \frac{1}{\sigma_y^2}}$$