CS 223b Homework 2 (Due January Wednesday, 30th) *J. Košecká*

e-mail your answers to cs223b@gmail.com

1. **Optical Flow.** The image brightness constancy constraint captures the relationship between the optical flow $\mathbf{u}$ at the image location $\mathbf{x}$ and the spatial and temporal derivatives of the image brightness:

$$\frac{\partial I}{\partial x}u_x + \frac{\partial I}{\partial y}u_y + \frac{\partial I}{\partial t} = (\nabla I)^T\mathbf{u} + \frac{\partial I}{\partial t} = 0$$

   (a) Assuming that $\mathbf{u} = [u_x, u_y]^T$ is constant for all pixels in the neighborhood $W(\mathbf{x})$, find such $\mathbf{u}$ that minimizes the constraint in the least-square sense. Write down the formulation and the solution of this minimization problem. This exercise is to work out the details of the optical flow algorithm outlined in the lecture. At the end of the exercise you will arrive at simplified version of the optical flow/tracking algorithm discussed in the OpenCV review session. (Steps: 1. Write down the objective function by summing up the brightness constancy errors for all pixels in some window W. 2. Take a derivative of that function with respect to u_x and u_y. 3. Set the derivative to 0.)

   (b) The formulation leads to a solution of a linear system of equations $G\mathbf{u} = -\mathbf{b}$. Discuss the cases when the solution is undetermined (i.e. G is singular).

   (c) Implement in Matlab this locally constant flow algorithm. Starting sample code and Yosemite sequence is available on the http://ai.stanford.edu/∼kosecka/assignments.html. Experiment with the size of the window W. Test the program on a Yosemite Sequence and one other sequence of your choice. Submit the code and figures of the computed flow fields. For plotting the flow fields you can use the function `QUIVER` in Matlab.

Solution:

   (a) Sum up the brightness constancy errors for all pixels in windwo W to obtain the objective function:

$$E_b(\mathbf{u}) = \sum_{W(x,y)} [(\nabla I(x,y,t))^T\mathbf{u}(x,y) + I_t(x,y,t)]^2$$

   To minimize the function,take the derivative with respect to u_x and u_y and set it to 0.

$$\begin{aligned}
\nabla E_b(\mathbf{u}) &= 2\sum_{W(x,y)} \nabla I(\nabla I^T\mathbf{u} + I_t) \\
&= 2\sum_{W(x,y)} [\begin{pmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{pmatrix}\mathbf{u} + \begin{pmatrix} I_xI_t \\ I_yI_t \end{pmatrix}] \\
&= 0
\end{aligned}$$

   We have:

$$\begin{aligned}
\begin{pmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{pmatrix}\mathbf{u} + \begin{pmatrix} I_xI_t \\ I_yI_t \end{pmatrix} &= 0 \\
G\mathbf{u} + \mathbf{b} &= 0 \qquad (1)
\end{aligned}$$

(b) If G is invertible, the solution to Equation1 above is $\mathbf{u} = G^{-1}\mathbf{b}$. The solution is undetermined when G is singular. This occurs when either I_x or I_y or both are equal to zero. In the case where both I_x and I_y are zero, rank(G)=0 and we have the blank wall problem. In the case where one of I_x and I_y is zero, i.e. intensity variation in a window W varies only along one dimension, rank(G)=1 and we have the aperture problem.

2. **Corner detector.** In this problem you will use Matlab implementation of the Harris corner detector which you can download from http://cs223b.stanford.edu/homework/ass2/1/harris corners.m. This includes comments on the input parameters and how to run the detector. Your task is to run this corner detector on two sample images on the images provided in http://cs223b.stanford.edu/homework/ass2/1/, visualize the results and answer some questions below. Here is a set of Matlab commands demonstrating its use.
>> `im = imread('lincoln.png');`
>> `corners = harris_corner(img, 7, 1.5);`
>> `imshow(img); hold on;`
>> `plot(corners(:, 1), corners(:, 2), 'r+')`

This reads the image from file lincoln.png, runs the corner detector using a gaussian kernel of width7 pixels and standard deviation 1.5. The last three commands display the image and the detected corners superimposed on it. Run the corner detector on the image grid-1.png and try to tweak the parameters so that all (or as many as you can manage) corners of the black squares are detected and the number of spurious corners is minimized.

(a) For fixed parameter values, run the detector on grid-1.png and grid-rotated.png. The latterimage is a rotated copy of the former. If we rotate the input image, do the detected corner positions rotate by the same amount ? Justify your answerbased on your observations.

(b) If we scale down the input image, are all the detected corner positions scaled accordingly ? You can test this experimentally by comparing the corner detection on the images grid-1.png, grid-2.png, grid-3.png. Each image in this sequenceis half the size of its predecessor. Justify your answer based on your observations.

Solution:

(a) The Harris corner detector is rotation invariant; rotating the image results in almost all the corners being rotated by the same amount. The rotated image may have more corners which were not present in the original, and vice versa. This happens because the two images see slightly different parts of the scene or due to resampling.

(b) The Harris corner detector is not scale invariant, this limitation was discussed in the lecture and was one of the reasons SIFT features were developed.

3. **Correspondence.** Use the Harris corner code from the previous exercise, select the features in the first image and find the corresponding points in the second image. Carry out the experiments on the stereo pair of images of the images of the house (house.zip) - link provided at http://ai.stanford.edu/∼kosecka/assignments.html. Implement the correspondence using

(a) SSD (sum-of-squared-differences) similarity measure

(b) NCC (normalized cross-correlation) measure.

Submit the code (of SSD and NCC experiment), the image with the overlayed result of the feature detector and the image pair with corresponding points in two views (to visualize the result make a new image putting the two images side-by-side and connect the corresponding features by plotting lines originating in one view

and finishing in another. This is done easiest in Matlab in 3 lines of code). Function appendimages.m to make a composite image is available on the web site.

Note: For problems 2 and 3 you are welcome to use other implementation of the Harris corner you find on-line or in OpenCV. Just be sure that you run in on the images provided.

4. **Image Transformations.** Given an image of your choice, apply following linear transformations to the image (every pixel in the images). The goal of the exercise is to deal with the fact that the transformed pixel values will not be in the grid and have to be interpolated in order to generate the transformed image.

   (a) scaling along both x and y axis

   (b) rotation by 45^o

   (c) simultaneous scaling and rotation.

   To do so write a function `imOut = warpLinear(A,imIn)` which takes the 2×2 matrix A and the original image and returns a new image after the transformation. Submit the code and three images obtained after applying the transformation and matrix form of the transformations used. *Hint: Use the Matlab function* `interp2` *and* `meshgrid`.