Digital Video Processing (EE392J)
Department of Electrical Engineering
Stanford University

Problem Set No. 2

Issued: January 23, 2002

Due: January 30, 2002 (in class)

Reading: Sections 6.4 through 6.11

Problem 1. Briefly answer Problem 6.1. Please limit your answer to one paragraph or one table.

Problem 2. Consider the signal $s_c(x, y, t)$ defined over the entire 3-D space (x, y, t), where $s_c(x, y, t)$ is generated from one image by translational motion with uniform (constant) velocities of v_x and v_y along the horizontal and vertical directions, respectively. Suppose $s_c(x, y, 0) = x + y + xy$.

(a) Determine $s_c(x, y, t)$.

(b) The optical flow equation (or optical flow constraint) is based on the assumption that the intensity of a point in a video signal remains constant along its motion trajectory. Show that the optical flow equation (below) is satisfied for all points (x, y, t) of the signal $s_c(x, y, t)$.

$$v_x(x, y, t)\frac{\partial s_c(x, y, t)}{\partial x} + v_y(x, y, t)\frac{\partial s_c(x, y, t)}{\partial y} + \frac{\partial s_c(x, y, t)}{\partial t} = 0 \tag{1}$$

(c) What is the spatio-temporal frequency spectrum of $s_c(x, y, t)$? Provide both a mathematical expression (in terms of $s_c(x, y, 0)$ or its Fourier transform) as well as a geometrical interpretation.

Problem 3. Block-matching motion estimation algorithms use a variety of search strategies to find the best matching block, where these strategies trade off motion estimation accuracy for reduced computational complexity. In this problem assume that the sum of absolute differences (SAD) criterion is used to determine the best match and that computing the SAD between two sets of N pixels, $\sum_{i=1 \, to \, N} |a_i - b_i|$ requires N operations. Assume the blocksize is 16×16 pixels, the frame size is $N_1 \times N_2$ pixels (N_1 and N_2 are integer multiples of 16), and the search range is +/-15 pixels horizontally and vertically. Also assume that the number of operations required for a 16×16 block on the boundary of the frame is equal to the number of operations required for a 16×16 block in the interior of the frame.

(a) Determine the number of operations required to compute the motion between the luminance component of each frame when performing *full search* over all integer motion vectors in the search range.

(b) Determine the number of operations required to compute the motion between the luminance component of each frame when performing *4-step search* (log search) over all integer motion vectors in the search range.

(c) Determine the number of operations required for (a) and (b) in the case of CIF video ($N_1 = 352$, $N_2 = 240$, 30 frames/sec) and HDTV video ($N_1 = 1280$, $N_2 = 720$, 60 frames/sec).

Problem 4. **Block-Matching Motion Estimation**

The goal of this Matlab-based exercise is to develop a block-based motion estimation algorithm to compute the motion between a pair of images. The images are available on the Problem Set webpage off of the course webpage www.stanford.edu/class/ee392j/ . A summary of useful Matlab operations and commands, as well as suggestions for writing your programs, are given on pages 3 and 4 of this handout. It is strongly recommended that you read these suggestions.

(a) Read the two images into Matlab and perform a color space conversion from RGB to YIQ using the color space conversion equation 1.4.3 given on page 18 of the text. Display and turn in separate images for each of the Y, I, and Q components (you many find *imagesc* and *subplot* commands useful). Comment on the information contained in each color component.

(b) In the remainder of this problem we estimate the motion using only the luminance component of each frame. Why is computing motion using only the luminance component usually sufficient for estimating motion for a color video sequence? When does this approach fail?

(c) Design a block-matching motion estimation algorithm that partitions the current frame into nonoverlapping 16×16-pixel blocks, and for each block determine the best-matching block in the previous frame based on the sum of absolute error (SAE) criterion. Determine the best-matching block by performing a full-search over all integer motion vectors within the search range, where the search range is +16/-15 pixels in the horizontal and vertical directions of the previous frame. You may limit your algorithm to only estimate the motion of the 16×16-pixel blocks that are in the interior of the image, i.e. you do not have to estimate the motion of the 16×16-pixel blocks on the boundary of the image. Plot the estimated 2-D motion vector field using *quiver* plot, and turn in a copy of this plot.

(d) Compute the motion-compensated (MC) prediction of the current frame by using the computed motion vectors to appropriately translate the blocks from the previous frame. Turn in an image of the MC-prediction. How does the MC-prediction compare to the original current frame? What is the peak signal-to-noise ratio (PSNR) (equation 1.5.6 in the text) between the MC-prediction and the original current frame (only use pixels in interior blocks to compute the PSNR)?

(e) Instead of using full search to determine the best matching block, you can use another method, for example a four-step logarithmic search. Briefly describe one advantage and one disadvantage of using four-step logarithmic search instead of full search. Note that this part of the problem only requires a written answer, specifically we are not asking you to redesign your motion estimation algorithm using four-step log search.

Include in your problem set solutions copies of your Matlab programs. The programs should include sufficient comments to enable one to easily follow your program.

Save copies of your programs as they may be useful for your final project.

Some Useful Matlab Tips

The following contains some Matlab tips that may be useful for people who are new to Matlab as well as Matlab experts who are new to processing images. A list of matlab topics for which help is available can be found by typing **help** at the Matlab prompt. Help for a particular command can be found by typing **help command**, and M-files can be searched for a particular keyword with **lookfor**.

Some useful matlab commands:

diary To save text of Matlab session into a file

whos To view current variables in a session

imread To read in an image file. You may need to convert the input image to **double**.

imwrite To write out an image file

fread To read binary data, e.g. an image in raw format

colormap(gray) To set the colormap to gray for viewing monochrome images

image To display an image, e.g. *image(A)* displays the image contained in the matrix A

imagesc Scales the image to use the full dynamic range and displays the scaled image

movie To create and display a sequence of frames

quiver To plot velocity vectors as arrows

subplot To plot multiple plots in one figure window (also useful when printing)

axis equal To set the horizontal and vertical axis on a figure to have equal units

hold To hold the current plot and its properties so that later plot commands overlay on the current plot

save To save current variables to a file for later use

load To load a file containing Matlab variables

Additional notes:

1. Matlab indexing begins with 1 (as opposed to 0 in C/C++).

2. It is very useful to use **script**s or **function**s to perform various tasks.

3. In a Matlab figure, the default placement of the origin is in the bottom left corner, however it is often beneficial to change this to the upper left corner. See **axis xy** and **axis ij**.

4. Matlab can easily run out of memory if many images are stored in memory. Be careful! *uint8* can be useful, however it is often best to save your important variables, exit Matlab, then restart Matlab and reload your variables.

5. The matrix/vector data structures of Matlab enable many complicated mathematical equations to be expressed very concisely and computed very efficiently. A little thinking before beginning to program can be very beneficial! A simple example: given two 16×16-pixel blocks represented by matrices A and B, the sum of absolute difference between them can be computed by either

```
SAD = 0;
for i = 1:16,
   for j = 1:16,
      SAD = SAD + abs(A(i,j)-B(i,j));
   end
end
```

or by

```
SAD = sum(sum(abs(A - B)));
```

Note that the second approach exploits Matlab's use of matrices and vectors, and *does not require any for loops*. This representation is concise, easy to read, and very easy to debug (as compared to the case with two for loops). Furthermore, since Matlab is not very efficient at using control structures, such as for loops, the matrix approach is much faster.

6. Motion estimation is very computationally intensive, so don't be surprised if Matlab appears to be frozen when computing the motion. Therefore, it is a good idea within your motion estimation algorithm to periodically print to the screen an indication of where your program is in the processing, e.g. whenever you start performing block matching for a new block, print the index of the block. Furthermore, as you are initially developing and debugging your algorithm and Matlab code, it is good practice to start by processing only a small portion of each frame, e.g. one 16×16-pixel block or four 16×16-pixel blocks. Once you are sure that your algorithm works correctly on these small and quickly solved problems, you can run it on the entire frame.

7. The images can be loaded into Matlab in the following manner.

```
frame1 = double(imread('frame1','tif'));
```

imread reads into Matlab the image frame1.tif which is in TIFF format and of type uint8, and **double** converts it to double floating point format for conventional Matlab processing. frame1 is 240x352x3 (RGB color planes). Convert it to YIQ, and display the Y component using **image**. You may need to change the colormap, e.g. **colormap(gray(256))**.