STONY BROOK UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CSE 391/591: COMPUTATIONAL PHOTOGRAPHY AND INTRO. TO COMPUTER VISION

Handed out: October 2, 2008 **Due on:** October 16, 2008

Problem Set #2

Submit the homework by email to cse391@cs.sunysb.edu

Problem 1: *Implement and experiment with a Skin-Color Tracker.*

This could be one possible first step in a vision based body tracking system. The input is a video recording. The output is a sequence of binary images of the same dimension. Your program will set the pixels of the output image sequence to 1 for skin regions, and set to 0 everywhere else. Here are example matlab functions you can use. If you don't fully understand the matlab functions, use the matlab help feature. (for instance: **`help aviread, help aviinfo`**).

Here is the "recipe" :

- In matlab you can load a sequence of frames with the **`aviread`** function. For example load the first 20 frames with:

  ```
  mov = aviread('subject01.avi',[1:20]);
  ```

  (If your source is a quicktime movie, just convert it to an AVI)

- You can "digout" the first frame from the **`mov`** structure with: **`im = mov(1).cdata;`**
- You can display that image with: **`image(im); axis image;`**
- Now select a "training set" of skin pixels. You can do that with: **`skin_mask = roipoly;`** (and select with the mouse a skin region)
- To get a list of pixel indices of the mask area use: **`skin_inds = find(skin_mask>0);`**
- For better performance mark several training sets across different subjects.
- Use all those labeled skin pixels to estimate a multivariate Gaussian (as discussed in class and the first online paper describes). First build a matrix that contains the RGB values of the labeled skin pixels:

  ```
  skin_R = double(im(:,:,1));
  skin_G = double(im(:,:,2));
  skin_B = double(im(:,:,3));

  % skin data
  skin_data_rgb = [skin_R(skin_inds),skin_G(skin_inds),skin_B(skin_inds)];
  % the entire image data
  all_data_rgb = [skin_R(:),skin_G(:),skin_B(:)];
  ```

  To calculate the mean and the covariance of the skin Gaussian, use the matlab function:
  `skin_MN = mean(skin_data_rgb); skin_CV = cov(skin_data_rgb);`
- Now you are going to compute for the entire image (and successive images in the video) the probabilities that a pixel contains a skin region.
- Write a matlab function **`P = gaussdensity(all_data_rgb,MN,CV);`**
  If the input (all_data_rgb) is a Nx3 matrix, the output is a Nx1 matrix of probability values.
  Use following formula for the gaussian:

$$p(x) = \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} e^{-1/2(x-\mu)^T \Sigma^{-1} (x-\mu)}$$

If you do it in a smart way in matlab, you can fully vectorize it (no for-loops, just matrix inputs and matrix outputs).

- You can reshape the result into a 2D layer with:
  `[rows,cols,d] = size(im);  L1 = reshape(P,rows,cols);.`

You can do the same for a "background layer" `L2` (using a different gaussian trained on background pixles). The last layer should be an "outlier layer" that has constant probability for each pixel. For example,

```
L3 = ones(rows,cols)/256;.
```

The final output of your algorithm should be normalized layers (posteriories). You get them in normalizing the values of each layer such that they add up to **1** for each pixel location (L1(x,y)+L2(x,y)+L3(x,y)=1). Do that with

```
S = L1+L2+L3; L1 = L1./S; L2 = L2./S; L3 = L3./S;
```

(Assuming uniform prior probabilities, those values could be interpreted as "posteriori probabilities": P(skin | RGB). And the un-normalized L1 values as "conditional density": p(RGB | skin).)

- You can visualize your results in generating a video of the skin layer (L1). Use the matlab function: **`avifile`**
- `Write out the resulting movie and display it on your web page.`

Extra Credit: ***This is required work for graduate students.***

- Try different color spaces (RGB vs nRGB vs HSV vs HS (with H on a circle)). Compute and plot those layers for your face and 1 other face. Converting a RGB image into a HSV image can be done with: **`rgb2hsv`**.
  The "HSV on a circle" representation can be done with: S*[cos(2*π*H),sin(2* π *H)].
  Discuss the difference in results.

Graduate Students Extra Credit:

- If you are totally excited about this, you could already implement the next step. Using the L1 matrix you can find the biggest "blob" of skin. Use for example some smoothing of the L1 layer and then connected components, and then estimate the center of mass of that blob. That could be your estimate of the "head location". Keep repeating that computation for each succesive frame.

`Scoring:`

Full credit – implement and turn in a skin detection algorithm. Create a web page with your resulting movies.
Provide a link to your webpage.
Students in the grad version of the class are required to do the extra credit.

Problem 2: *Morphing*

A morph is a simultaneous warp of the image shape and a cross-dissolve of the image colors. The warp is controlled by defining a correspondence between the two pictures. The correspondence should map eyes to eyes, mouth to mouth, chin to chin, ears to ears, etc., to get the smoothest transformations possible.

Generate a 2 seconds animation of your face morphing into another person's face. Morph still picture A into still picture B and produce 61 frames, where frame 0 must be identical to picture A and frame 60 must be identical to picture B. In the video, each frame will be displayed for 1/30 of a second.

Defining Correspondences:
Define pairs of corresponding points on the two images by hand (the more points, the better the morph, generally). The simplest way is probably to use the `cpselect` tool or write your own little tool using `ginput` and `plot` commands (with `hold on` and `hold off`). Now, you need to provide a triangulation of these points that will be used for morphing. You can compute a triangulation any way you like, or even define it by hand. A Delaunay triangulation (see `delaunay` and related functions) is a good choice since it does not produce overly skinny triangles. You can compute the Delaunay triangulation on either of the point sets (but not both -- the triangulation has to be the same throughout the morph!). But the best approach would probably be to compute the triangulation at midway shape (i.e. mean of the two point sets) to lessen the potential triangle deformations.

The Morph:
Write a function that produces a warp between `im1` and `im2` using point correspondences defined in `im1_pts` and `im2_pts` (which are both n-by-2 matrices of (x,y) locations) and the triangulation structure `tri` .

```
morphed_im = morph(im1, im2, im1_pts, im2_pts, tri, warp_frac, dissolve_frac);
```

The parameters `warp_frac` and `dissolve_frac` control shape warping and cross-dissolve, respectively. In particular, images `im1` and `im2` are first warped into an intermediate shape configuration controlled by `warp_frac`, and then cross-dissolved according to `dissolve_frac`. For interpolation, both parameters lie in the range [0,1]. They are the only parameters that will vary from frame to frame in the animation. For your starting frame, they will both equal 0, and for your ending frame, they will both equal 1. Given a new intermediate shape, the main task is implementing an affine warp for each triangle in the triangulation from the original images into this new shape. This will involve computing an affine transformation matrix A between two triangles:

```
A = computeAffine(tri1_pts,tri2_pts)
```

A set of these transformation matrices will then need to be used to implement an inverse warp (as discussed in class) of all pixels. Functions `tsearch` and `interp2` can come very handy here. But note that you are not allowed to use Matlab's build-in offerings for computing transformations, (e.g. `imtransform`, `cp2tform`, `maketform`, etc). Note, however, that `tsearch` assumes that your triangulation is always Delaunay. In our case, this might not always be true -- you may start with a Delaunay triangulation, but through the course of the morph it might produce skinny triangles and stop being Delaunay. David Martin from Boston College has kindly given access to his versions of tsearch that work on any triangulation: `mytsearch.m` and `mytsearch.c` (compile by typing "mex mytsearch.c" in Matlab; if you get compiler errors try replacing `mytsearch.c` with this one).

Face Labeling:
I ask everyone to label their own face (and maybe a few others of their own choosing), but do it in a consistent manner as shown in the following two images: points, point_order. For each face image, you should put on your website a text file with coordinates of x,y positions, one per line (43 lines total). You can read/write such files with the save -ascii and load -ascii commands.

Extra Credit: ***This is required work for graduate students.***

- Non-uniform face morphing
- Take three picture of a scene at three different times during the day (at sunrise, at noon and at sunset). Perform a global warp and cross-dissolve.

Graduate Student Extra Credit:

- Video morph

Scoring
To get full credit, you will need to implement a warping algorithm and turn in a video morph of your faces.
Generate a web page that displays your results. See the example webpage.

Students in the grad version of the class are required to do the extra credit.