ENGN 6528 Computer Vision, Semester 1, 2003

Lab 2: Detecting Skin in Colour Images

In this lab you will learn how to use chrominance[1] to segment coloured images. Here you be detecting skin, however, you could use this method to detect other coloured regions in images.

Firstly you will build a model describing the likelihood that any pair of chrominance values belongs to a piece of skin. Secondly you will test images and identify likely skin regions.

Getting Started

1. Go to the course website http://www.syseng.anu.edu.au/~luke/cvcourse.htm
   Download
   - the image 'face.jpg'.
   - the incomplete file 'Lab2.m', and
   - the file 'RGB2Lab.m'

   Save these in the directory you are using for this lab.
2. Open Matlab from the Start>Programs menu.
3. Change to your directory (use `cd`).
4. Type `edit Lab2`, and you are ready to start work.

Deliverable:

To gain the marks for this lab you will need to show me your completed `Lab2` function running during the lab. Complete the function Lab2 by writing the two sub-functions

`make_chroma_model`, and

`find_chroma`

The specifications of these functions can be found in the downloadable file Lab2.m

What to do

Firstly have a look at the Lab2 file. Run it and see what it does. First it defines some constants `num_bins` and `gsize`. Then it loads an image and converts it to double in the range [0,1], but leaves it in RGB format. It then crops a region defined by the user.

While you write and debugging this program it is convenient not to have to select a region each time you run the program. Comment out the lines that display the prompt to the user and interactively crop the image, we'll enable them again later once the code's working. For the meantime crop the image automatically like this:

[1] Colour has 3 channels, eg RGB, HSV. Chrominance is a two channel derivative of a colour that is independent of intensity. See lecture on Colour theory.

```
sample_colour = im_RGB(130:150,90:138,:);
```

This takes rows 130 to 150, columns 90 to 138, and all 3 colour channels. Notice that the ':' is used to mean 'everything inbetween' when it's between two indexes, or 'everything' when it's on it's own.

Now let's start on the function `make_chroma_model`. Firstly we need to convert from RGB colour space to CIE Lab colour space using RGB2Lab.m,

```
[L,a_chroma,b_chroma] = RGB2Lab(sample_colour);
```

RGB2Lab returns real *a,b* chrominance values in the range [-120,120] . We want to use these as indexes to an accumulator array, so we need to convert them to integers in the range [0, `num_bins`], where `num_bins` is a constant defining the size of each dimension of our accumulator array. To do this we will write a very short sub-function.

Write a sub-function `ab2ind` that is passed 2 parameters `ab_chroma` and `num_bins` and returns an index `ind`. This will only take two lines. Firstly convert `ab_chroma` from the range [-120,120] to [0,1].

```
ab_01 = ((ab_chroma)+120)/240;
```

Secondly discretise this into an integer index `ind` in the range [1,`num_bins`].

```
ind = round(ab_01*(num_bins-1)) + 1;
```

`round` rounds to the nearest integer. Note the minus 1 and plus 1 that are necessary to firstly scale to the range [0, `num_bins`-1] then increase this to [1, `num_bins`].

Getting back to `make_chroma_model`, we can now call our new sub-function to convert our a,b chrominance values to indexes for the accumulator array, e.g. for `a_chroma`

```
a_ind = ab2ind(a_chroma, num_bins);
```

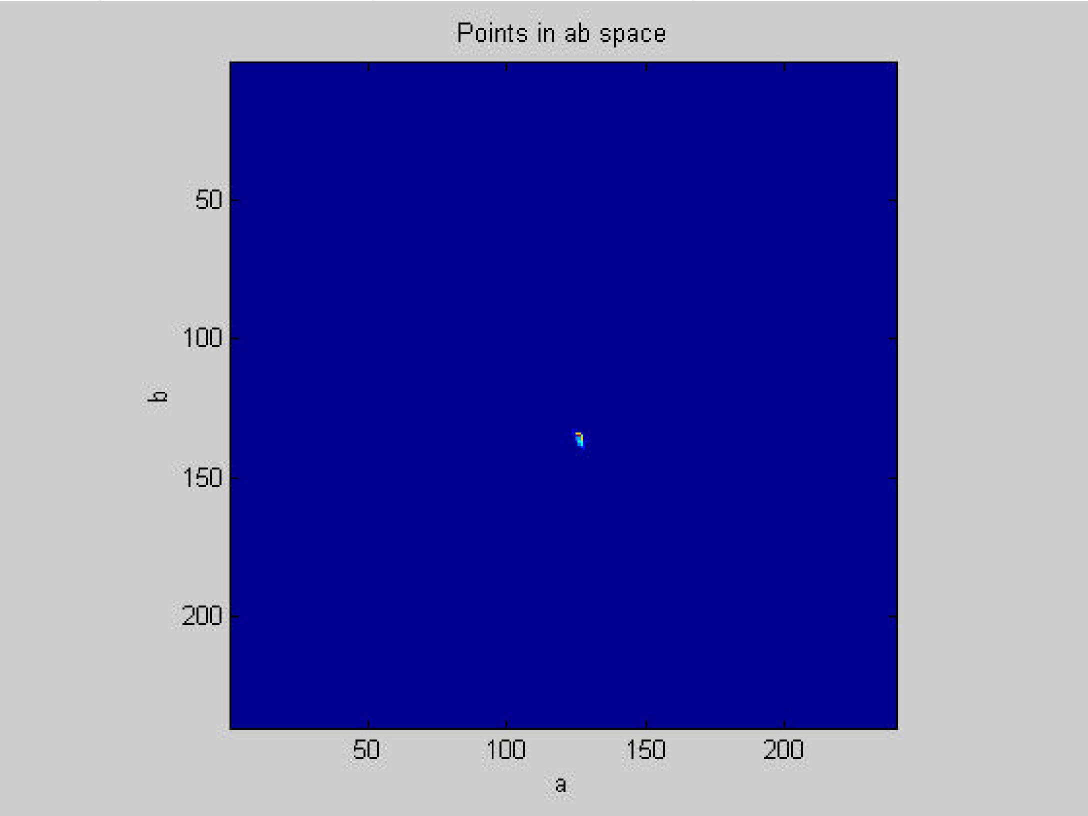
Now we are ready to start building the skin chrominance model. Create an accumulator array, this will be a `num_bins` × `num_bins` matrix of zeroes, use the `zeros` function to do this, call this matrix `accum_array`. We will use this for counting the occurrences of different chroma pairs and it will form the basis of our skin chrominance model.

Now we need to count the number of occurrences of each chroma pair. To do this you will need to use `for` loops to consider every pixel in the image. Look at the `a_ind` and `b_ind` values of each pixel and increment the appropriate cell in the accumulator array.

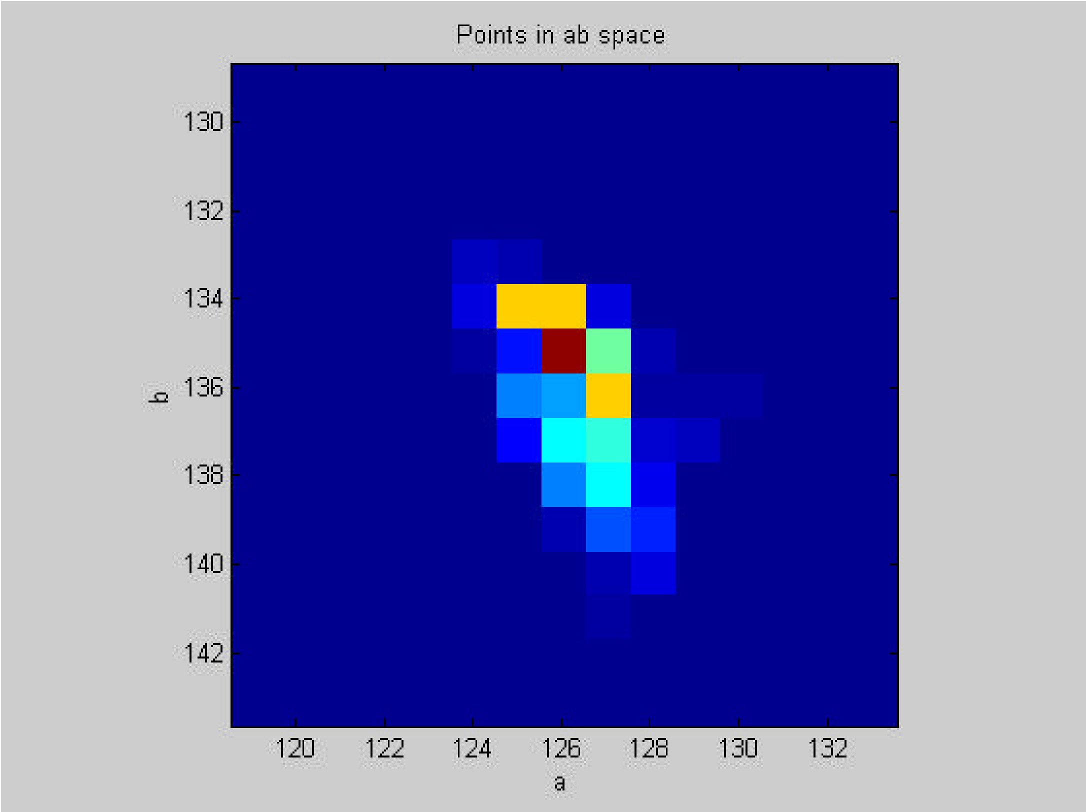
Done that? Have a look at your result using `imagesc` as follows,

```
figure;
imagesc(accum_array); axis image;
xlabel('a'); ylabel('b'); title('Points in ab space');
```

May be your axes will be the other way around – that's fine it depends on how you created your accumulator array. Your accumulator array should look like this



Zoom in by using the magnifying glass on the figure toolbar, close up it should look like this



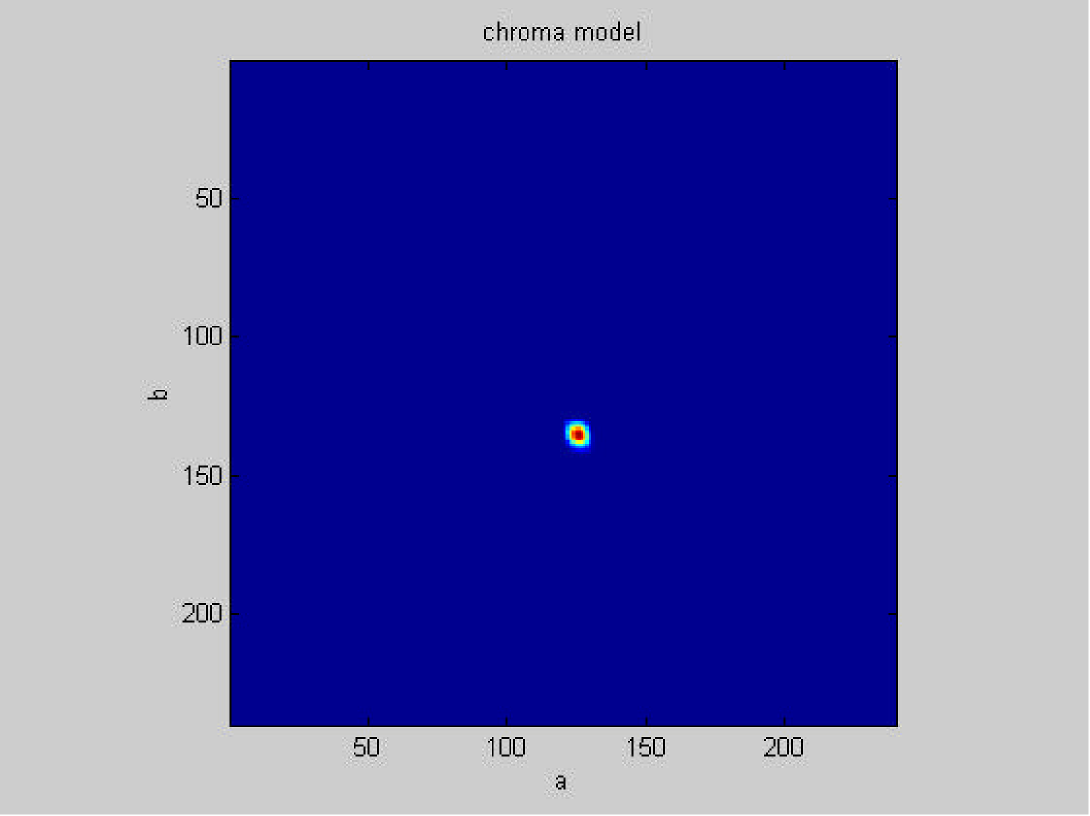
Now we'll convolve with a Gaussian to blur these points. We'll use `make_Gauss_vec` which I have written for you and included as a sub-function. The size of the Gaussian is specified by the parameter `gsize`. `make_Gauss_vec` will return a 1D Gaussian of length `gsize`. We can make a 2D Gaussian **G** by the convolution of two 1D gaussians **g**

$\mathbf{G} = \mathbf{g} * \mathbf{g}^T$

so this convolution is separable. Use `conv2` to perform a separable convolution of the Gaussian with the image. Your separable components are `gauss_vec` and its transpose `gauss_vec'`. Type `help conv2` to see how to do this. Make sure you use the `'same'` setting so you don't change the size of the accumulator array.

Call the output of this convolution `chroma_model`. Normalise this by dividing it by its maximum. Get the maximum value of the matrix using `max(max(chroma_model))`, have a look at `help max` to see why you need 2 calls to `max`.

Have a look at your chrominance model, it should look like



Now you've finished building the chrominance model. Lets try it out!

Firstly we'll see how well it detects other skin regions in our training image 'face.jpg'. To do this we need to complete the function `find_chroma`. This uses much the same process as `make_chroma_model` except here we are only accessing the chroma model not changing it. Go ahead and write this function, again you will need for loops to look at all the pixels and execution can take several seconds, so I recommend including a `waitbar` to indicate how far you are through. The syntax for including the `waitbar` is

```
h = waitbar(0,'Please wait...');
for i=1:num_bins,
      %
      % contents of loop %
      %
      waitbar(i/ num_bins,h)
end
close(h)
```

Type `help waitbar` for more information if you need it. Note that we do not need to pass `num_bins` to this function, since it can be obtained from the size of `chroma_model`, use `size` to do this: `size(chroma_model, 1)` will give you the number of rows, and `size(chroma_model, 2)` the number of columns, either will do.

Once you've got this working have a look at your skin probability image. Does it look right? If so re-enable the manual cropping code in the `Lab2` function and try detecting different colours in the image. Try a different image. Try training (building the chroma model) on one image and testing (looking for skin) in another. Now try training with more that one image. Notice that, in general, the bigger you training set, the better your result - this is hardly surprising!

Now we can try different chrominance spaces.
First use normalised R and G:

```
normR = im_RGB(:,:,1)./(im_RGB(:,:,1)+im_RGB(:,:,2)+ ...
     im_RGB(:,:,3));
normG = im_RGB(:,:,2)./(im_RGB(:,:,1)+im_RGB(:,:,2)+ ...
     im_RGB(:,:,3));
```

Make a new function `normRG2ind` similar to `ab2ind` to rescale normR and normG into the 240 bins (remember normR and normG will be between [0,1]). Now modify your program to use your new function.
How does the normalised RG colourspace compare?

Next use the HSV colour space. Use the matlab function rgb2hsv to perform the colour space transformation and then make another new function: `hs2ind`.
Why are we using H and S?
How does the HS space compare?
Is there a way to use the CIE Lab space without converting the test image to the CIE Lab space?

Additional Exercises (not directly assessable during the lab)

Have you got Lab1 working to your satisfaction using `atan2`? You will need this for Assignment 1.

- Blur an image using conv2 and kernels of different sizes.
- Sharpening an image, look at the intermediate image here **I*K**, see how adding this to the original image tend to 'sharpen' it.
- See what happens when you convolve a kernel with an image that is zero everywhere except one pixel that is equal to 1. Define this image like this

```
im1 = zeros(30,30);       % make a 30 x 30 mx of zeros
im1(10,10) = 1;           % set the (10,10) pixel to 1;
```

Consider the 2D convolution example from lectures, implement this in Matlab and verify that my answer is correct – let me know if it's not.

Make a start on the assignment.