LAB 4: COLOR IMAGE ANALYSIS

ECE180: Introduction to Signal Processing

OVERVIEW

Images are two-dimension (2-D) signals with the sample values called “pixels,” a shortened name for “picture elements”. Images are most commonly one of two types: grayscale and true-color. In this project you will learn how to load, display, process, and save each of these image types. You will also become acquainted with the notion of a *color space transformation* as a signal processing technique to make it easier to detect image objects by color.

OUTLINE

1. Discover image file information with `imfinfo`
2. Study the properties of grayscale and true-color images
3. Load, display, process, and save images
4. Apply color space transformations and thresholding to process a color image
5. Detect features of interest in a binary image

PREPARATION – TO BE COMPLETED BEFORE LAB

Study these tutorial videos:

1. Functions -- http://youtu.be/0zTmMIh6I8A (4:16) (this is new)
2. Scripting efficiency -- http://youtu.be/td09_g89X9w (2:19) (study now if you skipped it earlier)
3. “for” loop -- http://youtu.be/q2xfz8mOuSI (5:00) (review, if necessary)

Ensure that you have installed the ECE180 MATLAB folders (see Lab 1).

LAB ACTIVITIES

1. *Discover image information:*

You can learn many details about an image file before you load it into MATLAB:

1.1. Enter `imfinfo('camera.png')` at the MATLAB command line. Record the following information: `ColorType`, `BitDepth` (number of bits per pixel), `Width`, and `Height`.

1.2. Repeat for the “SunCountry” PNG image.

1.3. Repeat for the “PortlandHeadLight” PNG image.

2. *Study the “camera” grayscale image:*

2.1. Enter `imtool('camera.png')` at the MATLAB command line.
2.2. Hover the cursor over the tool buttons and make note of the purpose of each tool.
2.3. Click the “Display image information” tool button; compare the results to those you obtained in Step 1.1.

2.4. Observe the "Pixel info" display in the lower-left corner as you move the mouse and then respond to each of these questions:

(a) Explain the meaning of the three values (hint: move the cursor off of the image),

(b) In which corner is the origin (X,Y) = (0,0) located?

(c) What is the direction of increasing Y?

(d) What is the minimum pixel intensity you can find, and in which feature of the scene did you find it?

(e) What is the maximum pixel intensity you can find, and in which feature of the scene did you find it?

2.5. Experiment with the "Navigate image using overview" and "Inspect pixel values" tools. Try zooming in on the pixels until you can see the numerical values for the intensities. Describe how these two tools are similar and yet different.

2.6. Click the "Adjust contrast" tool button:

(a) Record the "Data range" values listed at the top left.

(b) Compare these to the values you found earlier in Step 2.4. How close did you get?

2.7. The *histogram display* shows the number of pixel values found in the image at a given intensity level for each of the 256 possible intensity levels.

(a) Which two intensities levels are most common in the image? (hint: move the left red slider to a desired position on the histogram and read its value from the "Window | Minimum" field in the top of the display)

(b) Which image features correspond most closely to these two peak values in the histogram?

2.8. Experiment with the left and right red sliders on the histogram. Try narrow versus wide windows; try narrow windows in various ranges:

(a) What happens to the displayed pixels when they have an intensity that exceeds the right-most slider value? Hint: Remember to look at the "Window" numerical fields that tell you where the cursor is located.

(b) Similarly, what happens to pixels with intensities below the left-most slider value?

(c) Adjust the window sliders to reveal as much detail as possible in the cameraman's coat; this is a basic type of *contrast enhancement*. Record the minimum/maximum values of your window and then screenclip the enhanced image.

2.9. Set the "Window | Width" to 1 and move the window by dragging the top pointer. The window center now serves as an *intensity threshold*, that is, intensities below the threshold display as black and intensities above the threshold display as white. This two-level image is called a *binary image*.

2.10. Adjust the contrast to return the display range to its original state (minimum 0, maximum 255); observe how these values appear in the lower-right corner of the main image display.

2.11. Experiment with the "Measure distance" tool. Right-click on a measurement line to see more options and to delete the line.

3. *Study the "PortlandHeadLight" true-color image:*

A *true-color* image allocates 24 bits to each pixel; 8 bits for red, 8 bits for green, and 8 bits for blue. You can think of a true-color image as a "stack" of three grayscale images called *color planes*.

3.1. Open the "PortlandHeadLight" image in `imtool`.

3.2. Click the "Display image information" tool button; compare the results to those you obtained in Step 1.3.

3.3. Use the "Inspect pixel value" tool to study various regions of the image. Tip: drag the pixel window by its white handles. Report the RGB triples for these features (note that the RGB values are now scaled to the range 0 to 255): sky, roof of the connected building, side of the light tower, and ball-shaped feature at the top of the light.

4. *Load and display images:*

4.1. Create an array variable `a` that contains the "camera" image: `a=imread('camera.png');`

4.2. Display the image in a figure window: `imshow(a)`
Type `colorbar` to add a color bar to the figure; the color bar shows how a pixel's numerical value appears as a color. NOTE: If the image seems to shrink a bit, type `truesize` to restore the image to 100% scaling to make each image pixel correspond to one physical display pixel.

4.3. Select the figure window that shows "camera" and then find the "Edit" menu item on the figure window itself (not on the MATLAB command window). Select "Edit | Colormap" to start the colormap editor. Select "Tools | Standard Colormaps" and try some of the available colormaps. Study the colorbar and try to relate this to what you see in the main image. Remember, the image pixel values are not changing, but instead the RGB lookup table values are changing.

4.4. Screenclip the image display window for your favorite colormap; record the colormap name, too.

5. *Save a processed image:*

5.1. Load and display the "camera" image into the variable `a`.

5.2. Display the *matrix transpose* of the image `a'`, i.e., type `imshow(a')`. What is the visual effect of the transpose operation?

5.3. Display `flip(a)`, `fliplr(a)`, and `rot90(a)`.

5.4. Choose one of these four image manipulation functions and save the processed image to a file: `imwrite(`*image*`,'processed.png')`, replacing "*image*" with one of the image manipulation functions operating on the image variable `a`.

5.5. Type `pwd` to print your working directory; you can also find this information just below the menu tabs. Open a Windows Explorer window (hold the Windows button and press E), navigate to this folder, and double-click the "processed" image file to confirm that you have created a valid image file.

6. *Process a color image:*

Now that you have established confidence working with the two standard types of images (grayscale and true color), let's develop some techniques to extract useful information from color images. The overall objective for this part is to create a *binary* image that detects an object in the image based on the object's color and saturation.

6.1. Load and display the image "marbles.png".

6.2. Note the dimension of the true-color image variable by either looking at the "Workspace" variable list window or by typing `whos` at the command line. The first two dimensions are the image height and width, while the third dimension is the color plane index (1 = red, 2 = green, and 3 = blue). For example, the green plane of the true-color image `img` is `img(:,:,2)`.

6.3. Use `subplot` to create a 2x2 image panel with the original color image at the top left and the red, green, and blue planes in the remaining quadrants. NOTE: Each color plane *displayed by itself* appears as a grayscale image.

6.4. Once you have worked out the details with `subplot`, create a function `showplanes` that accepts the image variable (*not* the image filename!) and creates the figure described in the previous step. Use `title` to label each image panel. For example, display the true-color image variable `img` with the command sequence `img=imread('marbles.png'); figure, showplanes(img).`
NOTE: This function simply creates a figure, it does not return any values.

6.5. Finish adding comments to document your function, and then:

(a) Copy and paste your finished and documented `showplanes.m` function text.

(b) Screenclip the figure window created by your `showplanes` function for the "marbles" image.

6.6. Study each of the color planes. Describe the relationship of each color plane to the color features that you see in the original color image. For example, which colors have a strong red component? Which color in the original image has almost no green component? Do any of the colors have a significant blue component?

6.7. Consider each of the color planes. Is it possible to define a *single* intensity threshold within any of the color planes that would separate the yellow marbles from all of the others? Explain your answer. Hint: Refer back to your work for Step 2.9 regarding the "thresholding" concept and technique. Optional, but likely helpful: use `imtool` to look at each of the color planes.

6.8. The RGB (red-green-blue) *color space* is only one of many different ways to represent color in the image. The hue-saturation-value (HSV) color space often provides a more convenient way to work with color images when you want to identify features by color. Study `helpwin` or `doc` for `rgb2hsv` and then create a modified version of `showplanes` called `showHSVplanes` to show the H, S, and V planes along with the original color image. Add a feature to return the three planes as distinct images, i.e., call the function with `[h,s,v]=showHSVplanes(img)` to obtain the H, S, and V planes as distinct grayscale images.

6.9. Type `whos img h s v` and look at the "Bytes" and "Class" (a.k.a. data type) columns:

(a) What is the data type of each of these image?

(b) What is the required memory for a *single* plane of the original image in bytes per pixel?

(c) What is the required memory for a *single* plane of the converted HSV image in bytes per pixel? By what factor has the required memory increased?

(d) Many MATLAB functions such as `rgb2hsv` default to the `double` data type. However, in this case the increased memory does not add additional information.

6.10. Use the data type conversion function `uint8` to return the H, S, and V images with this more efficient data type; simply enclose the `rgb2hsv` function call within a call to the `uint8` function.

(a) Does this work properly, i.e., does `showHSVplanes` still work properly?

(b) Remove the `uint8` function temporarily and create a fresh set of H, S, and V images. Images stored as `double`s scale the pixel intensities to the range 0 to 1. Type `max(h(:))` to see the maximum intensity in the hue image, and compare this value to 1.

(c) The intensity range of the unsigned 8-bit integer data type is 0 to 255, consequently the double data type image must be scaled (multiplied) by 255 *before* converting to integer format. Make this change to your `showHSVplanes` function and confirm that the generated image looks correct.

(d) Copy and paste your finished and documented `showHSVplanes` function text.

(e) Screenclip the figure window created by your `showHSVplanes` function for the "marbles" image.

6.11. Study each of the color planes. “Hue” assigns an intensity value to each color, “saturation” indicates the degree to which the color is *pure*, i.e., undiluted by white, and “value” is the remaining grayscale image with all color information removed. Is it possible now to define a single intensity threshold within any of the color planes that would separate the yellow marbles from the others? Explain your answer. Hint: You may still need to use information from the saturation plane.

6.12. Open the H plane with `imtool`. Watch the “Pixel info” display in the lower left corner as you move the mouse within the yellow marbles region, or use the “Adjust contrast” tool to make it easier to see the boundaries of the yellow marbles. Record the minimum and maximum hue values for this feature.

6.13. Repeat Step 6.12 for the S plane. Record the minimum and maximum saturation values for this feature.

6.14. Threshold the H plane and the S plane to extract the feature of interest from the H plane:

```
featH = h>=hmin & h<=hmax; imshow(featH)
```

This statement creates a binary feature image `featH` in which all pixel intensities of the image `h` greater than or equal to `hmin` AND less than or equal to `hmax` have value 1; pixel intensities outside this range have value zero. Use the minimum and maximum values you recorded in Step 6.12.

6.15. Repeat for the S plane using the values you recorded in Step 6.13:

```
featS = s>=smin & s<=smax; figure, imshow(featS)
```

6.16. Now combine the hue and saturation feature images into a single result :

```
feat = featH & featS; imshow(feat)
```

You should see the yellow marbles appear as prominent white objects with possibly some small extraneous features scattered throughout the image, too. Refine your threshold values as needed to minimize these extraneous features (some will remain, but try to eliminate them as much as possible while still retaining the desired features).

(a) Copy and paste the line two lines of code that you used to threshold the H and S images.

(b) Screenclip the figure window created for your finished feature image.

(c) Comment on your results; how well are you able to isolate the yellow marbles?

6.17. Type `whos h feat` and look at the Bytes and Class columns.

(a) What is the data type of the thresholded feature image `feat`?

(b) How does the memory requirement of this new data type compare to the “uint8” data type?

7. *Analyze the processed image:*

As a final touch, you can easily analyze the binary feature image to determine the centroid (center of mass) of each feature and then update the original image to label and box the detected features. Create a new script for this purpose.

7.1. Start the script with `clear` to ensure that your script will run properly with a clean workspace; add `close all hidden` to close any open figure windows. Set up the initial script to read the “marbles” image and then step through the process to create the feature image `feat`.

7.2. Remove any small extraneous connected features containing less than *P* pixels and display the result:

```
feat=bwareaopen(feat,P); imshow(feat)
```

Start with P = 100 and increase the value by steps of 50 or 100 if you have larger features to remove. Try to retain as much of the desired features as possible.

7.3. Wrap a *convex hull* around the remaining features to fill in the gaps and display the result:

```
feat=bwconvhull(feat,'objects'); imshow(feat)
```

7.4. Create the object `cc` to detect all of the *connected components* (features) in the image:

```
cc=bwconncomp(feat);
```

Confirm that the `NumObjects` property of the `cc` object (`cc.NumObjects`) matches the number of features that you can see in the result of Step 7.3.

7.5. Create the object `rp` to calculate the *region properties* of each feature:

```
rp=regionprops(cc);
```

Observe that `rp` is an array of structures that contain the centroid and bounding box coordinates of each detected feature. For example, enter `rp(2).Centroid` to see the x-y coordinates of the second detected feature.

7.6. Draw a bounding box around each detected feature and add a label to show the feature number:

```
for k=1:length(rp)
   img=insertShape(img,'Rectangle',rp(k).BoundingBox);
   img=insertText(img,rp(k).Centroid,num2str(k),'AnchorPoint','Center');
end
imshow(img)
```

7.7. Copy and paste your finished and documented script.

7.8. Screenclip the figure window for your finished result of Step 7.6.

7.9. Optional: Adjust your H and S threshold values to detect another marble color. Screenclip the figure window and include with your result for 7.8.

8. *Wrap Up:*

8.1. ⇈ Ensure that you have named the completed worksheet for your lab team “Lab 4 _ Firstname Lastname.docx” (first and last name of the scribe) and then submit it to the Gradescope by the due date.