Edge Detection using the Haar Transform

In this lab we will apply the 2D Haar coefficient formulas to explore how the discrete Haar transform can be used to detect edges in images.

Instructions: Submit the requested figures to tleise@amherst.edu by Wed 11/4.

Exercise 1 (Working with images in Matlab) First we need to learn how to load and plot images in Matlab. Download the MagicSquare.tif image from the Math 320 website (a small section from Albrecht Dürer's woodcut Melencolia I; see `http://en.wikipedia.org/wiki/Melencolia_I`). To load and plot the image into Matlab, type

```
A=double(imread('MagicSquare.tif'));
figure;imagesc(A);
```

The default colormap is `jet` may not be the best choice for viewing this image. Type `colormap('gray')` to view the image as grayscale. You can also experiment with other coloring choices, which are listed in the Matlab help section for `colormap`. Choose whichever colormap you like best.

To see the image with the correct aspect ratio, add the following code:

```
[M,N]=size(A);
axis equal;
axis([0 N 0 M])
```

Download an image of your choice from the web. Matlab's `imread` should work on tif, gif, png, and jpg files. If you want to work with a color image, then you should convert it to grayscale for purposes of applying the Haar transform. Color images in rgb format are $M \times N \times 3$ with red values in the 1st matrix, green in the 2nd, and blue in the 3rd. Convert to grayscale using the following formula, so you have a single matrix instead of a stack of 3 matrices:

```
A=0.2989*A(:,:,1)+0.5870*A(:,:,2)+0.1140*A(:,:,3);
```

Load your image into Matlab and plot it with the correct aspect ratio.

Exercise 2 (2D Haar transform) The basic idea behind the 2D discrete Haar transform is similar to the 1D version: decompose an image, represented as a matrix of pixel grayscale values, into an approximation plus details or residuals by taking averages and differences of adjacent coefficients. However, since the data is now in the form of a matrix rather than an array, there are 3 sets of detail coefficients at each level j, representing differences in the horizontal, vertical, and diagonal directions.

Let's create a function to carry out the 2D Haar transform, to make our code a bit more elegant than copying and pasting each time we want to apply another iteration of the transform. Go to the Matlab editor window and hit New Function (in the upper left corner of the editor window). A new file is generated with a template for defining a function. The output arguments will be the approximation and the three details, and the input argument will be the image matrix A:

```
function [A1,BH,BV,BD] = HT2D(A)
```

Fill in the comment section with a brief description of what the function does and what the input and output arguments are, then fill in the Haar 2D coefficient formulas:

```
A1=1/2*(A(1:2:end,1:2:end)+A(2:2:end,1:2:end)+A(1:2:end,2:2:end)+A(2:2:end,2:2:end));
BH=1/2*(A(1:2:end,1:2:end)-A(2:2:end,1:2:end)+A(1:2:end,2:2:end)-A(2:2:end,2:2:end));
BV=1/2*(A(1:2:end,1:2:end)+A(2:2:end,1:2:end)-A(1:2:end,2:2:end)-A(2:2:end,2:2:end));
BD=1/2*(A(1:2:end,1:2:end)-A(2:2:end,1:2:end)-A(1:2:end,2:2:end)+A(2:2:end,2:2:end));
```

Load the Dürer magic square image into Matlab again and apply the `HT2D` function to it:

```
[A1,BH,BV,BD] = HT2D(A);
```

Submit a 2x2 figure showing the four matrices generated by the 2D transform. Be sure to label which graph is which and to correct the aspect ratio, noting that these new matrices will be $\frac{M}{2} \times \frac{N}{2}$. The magic square image is particularly good for illustrating how the Haar transform detects vertical, horizontal, and diagonal edges in the different detail matrices.

Exercise 3 (Iterating the 2D transform)

Apply the 2D Haar transform to the new approximation matrix, called `A1` above. Give the 4 new matrices names to distinguish them from the first iterate, for instance, `A2, BH2`, etc., to indicate they come from the 2nd iteration. Submit a 2x2 figure of the 4 matrices in this 2nd iteration as well as a 2x2 figure of the 4 matrices from the 3rd iteration (`HT2D` applied to `A2`). As before, make sure the aspect ratios are correct, and note that the matrices keep shrinking with each iteration.

Exercise 4 (Inverting the 2D transform) Make a function `IHT2D` that inverts the transform:

```
function A = IHT2D(A1,BH,BV,BD)
% Inverse 2D Haar transform
[M,N]=size(A1);
A=zeros(2*M,2*N);
A(1:2:end,1:2:end)=(A1+BH+BV+BD)/2;
A(1:2:end,2:2:end)=(A1+BH-BV-BD)/2;
A(2:2:end,1:2:end)=(A1-BH+BV-BD)/2;
A(2:2:end,2:2:end)=(A1-BH-BV+BD)/2;
end
```

Test it on your magic square coefficient matrices to make sure it works correctly.

Exercise 5 (Extracting edges from an image) Download the car.png image file from the Math 320 website. Apply 3 iterations of the HT2D. Replace the `A3` matrix (approximation matrix from the 3rd iterate) with a matrix of zeros, and then apply the IHT2D 3 times to build back up to a single matrix `A0` that is the same size as the original. This process keeps only edges of the image:

```
figure;imagesc(-abs(A0));
colormap('gray');axis equal;axis([0 N 0 M])
```

Try applying this process to an image of your choice and submit a figure showing the resulting edges-only image. You'll need each dimension (M and N) to be a multiple of 8, so crop if needed:
`A=A(1:M-mod(M,8),1:N-mod(N,8));[M,N]=size(A); % crops then updates M and N`