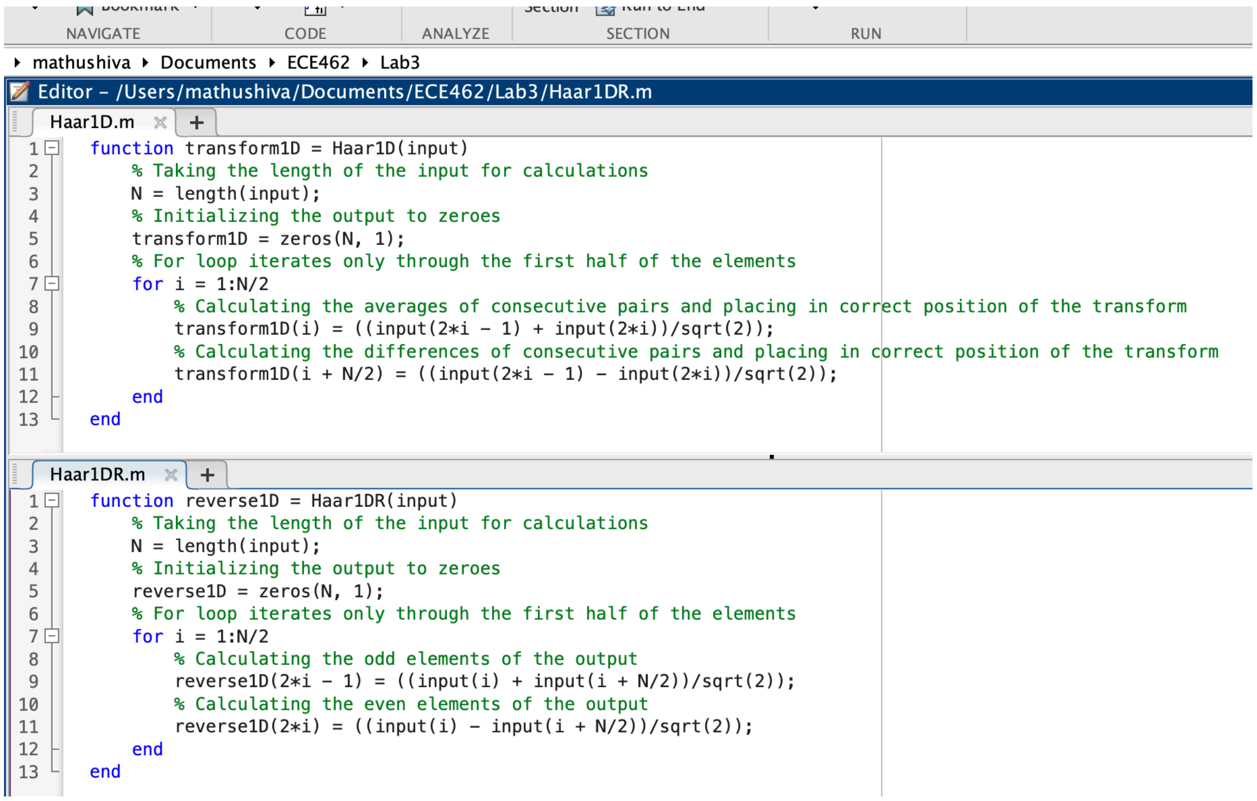
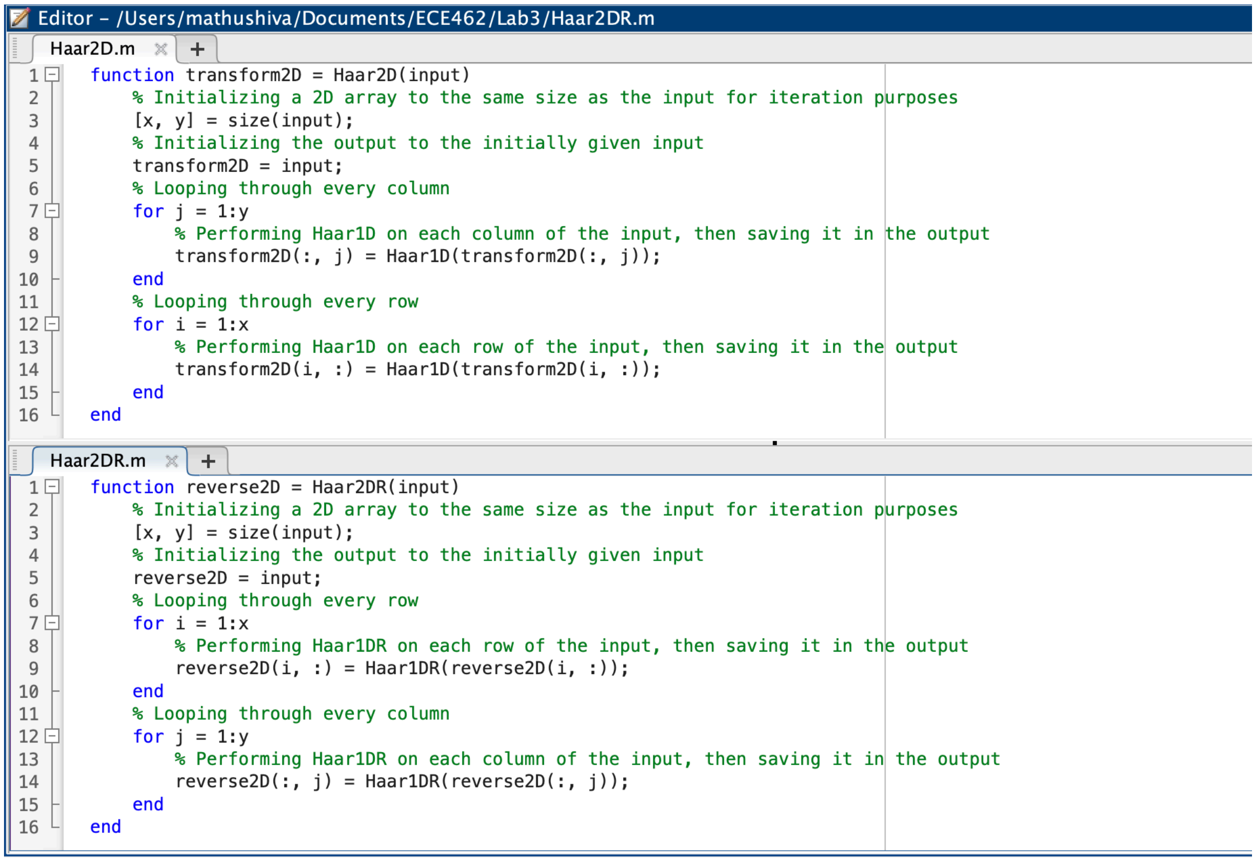

NAVIGATE
CODE
ANALYZE
SECTION
RUN
mathushiva ▸ Documents ▸ ECE462 ▸ Lab3
Editor – /Users/mathushiva/Documents/ECE462/Lab3/Haar1DR.m
Haar1D.m

```
function transform1D = Haar1D(input)
    % Taking the length of the input for calculations
    N = length(input);
    % Initializing the output to zeroes
    transform1D = zeros(N, 1);
    % For loop iterates only through the first half of the elements
    for i = 1:N/2
        % Calculating the averages of consecutive pairs and placing in correct position of the transform
        transform1D(i) = ((input(2*i - 1) + input(2*i))/sqrt(2));
        % Calculating the differences of consecutive pairs and placing in correct position of the transform
        transform1D(i + N/2) = ((input(2*i - 1) - input(2*i))/sqrt(2));
    end
end
```

```
function reverse1D = Haar1DR(input)
    % Taking the length of the input for calculations
    N = length(input);
    % Initializing the output to zeroes
    reverse1D = zeros(N, 1);
    % For loop iterates only through the first half of the elements
    for i = 1:N/2
        % Calculating the odd elements of the output
        reverse1D(2*i - 1) = ((input(i) + input(i + N/2))/sqrt(2));
        % Calculating the even elements of the output
        reverse1D(2*i) = ((input(i) - input(i + N/2))/sqrt(2));
    end
end
```

The first function in the above screenshot is the 1D Haar Transform. We first get the length of the input, and then loop half the amount of the length. For each iteration, we compute the normalized average and the normalized difference of consecutive pairs and save the calculated value in the appropriate element in the output transform. We save the normalized averages in the first half of the transform and the normalized differences in the second half of the transform.

The second function in the above screenshot is the 1D Reverse Haar Transform. We first get the length of the input, and then loop half the amount of the length. We then basically do the opposite of the 1D Haar Transform. We follow the formula for this transform. We calculate the normalized averages and normalized differences between element 1 and 4, element 2 and 5, and element 3 and 6. The normalized averages are saved in the odd elements of the output and the normalized differences are saved in the even elements of the output.

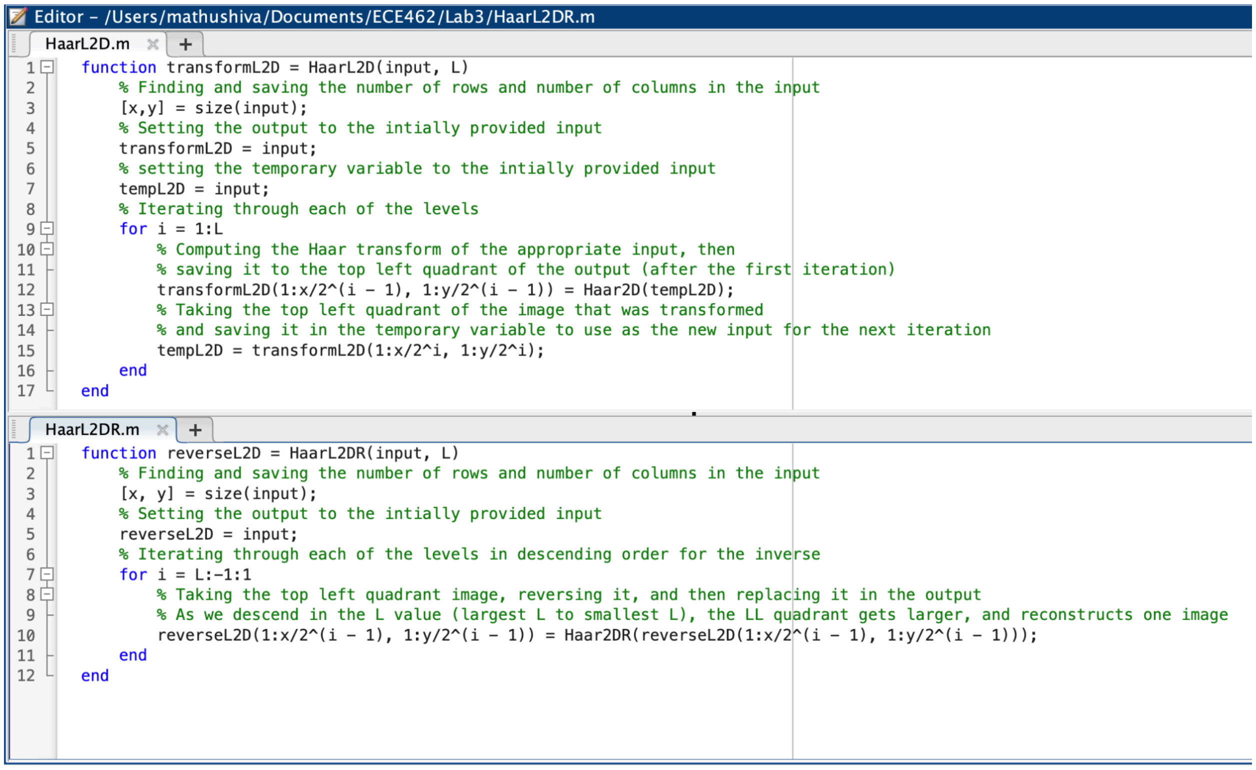


```
function transform2D = Haar2D(input)
    % Initializing a 2D array to the same size as the input for iteration purposes
    [x, y] = size(input);
    % Initializing the output to the initially given input
    transform2D = input;
    % Looping through every column
    for j = 1:y
        % Performing Haar1D on each column of the input, then saving it in the output
        transform2D(:, j) = Haar1D(transform2D(:, j));
    end
    % Looping through every row
    for i = 1:x
        % Performing Haar1D on each row of the input, then saving it in the output
        transform2D(i, :) = Haar1D(transform2D(i, :));
    end
end
```

```
function reverse2D = Haar2DR(input)
    % Initializing a 2D array to the same size as the input for iteration purposes
    [x, y] = size(input);
    % Initializing the output to the initially given input
    reverse2D = input;
    % Looping through every row
    for i = 1:x
        % Performing Haar1DR on each row of the input, then saving it in the output
        reverse2D(i, :) = Haar1DR(reverse2D(i, :));
    end
    % Looping through every column
    for j = 1:y
        % Performing Haar1DR on each column of the input, then saving it in the output
        reverse2D(:, j) = Haar1DR(reverse2D(:, j));
    end
end
```

The first function in the above screenshot is the 1-Level 2D Haar Transform. We first initialize the x and y variables to the number of rows and columns of the input. We also initialize the output to be the initial input. We then loop through every column and perform the 1D Haar Transform with the previously written function. We then place this transform in the output. We then do the same process and loop through every row. After both loops are complete, we will have successfully completed a 1-Level 2D Haar Transform.

The second function in the above screenshot is the 1-Level 2D Reverse Haar Transform. We first initialize the x and y variables to the number of rows and columns of the input. We also initialize the output to be the initial input. We then loop through every row and perform the 1D Reverse Haar Transform with the previously written function. We then place this reversed transform in the output. We then do the same process and loop through every column. After both loops are complete, we will have successfully completed a 1-Level 2D Reverse Haar Transform.



Editor – /Users/mathushiva/Documents/ECE462/Lab3/HaarL2DR.m

HaarL2D.m

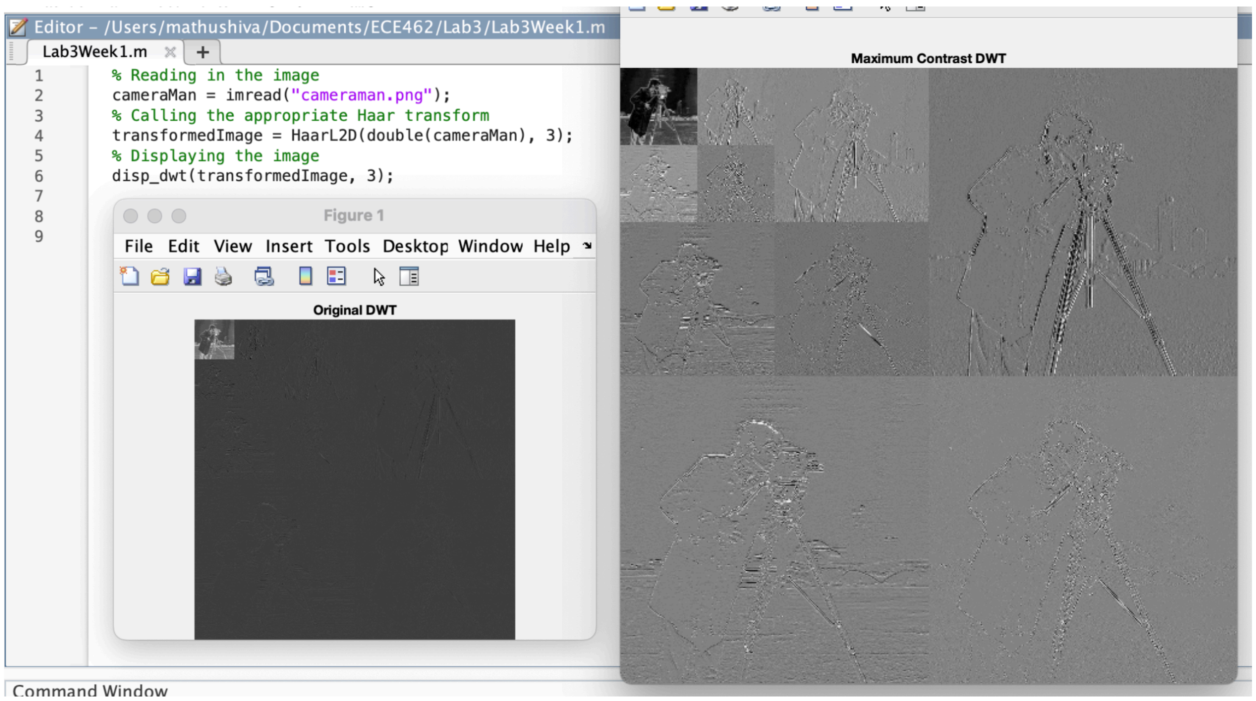
```matlab
function transformL2D = HaarL2D(input, L)
    % Finding and saving the number of rows and number of columns in the input
    [x,y] = size(input);
    % Setting the output to the intially provided input
    transformL2D = input;
    % setting the temporary variable to the intially provided input
    tempL2D = input;
    % Iterating through each of the levels
    for i = 1:L
        % Computing the Haar transform of the appropriate input, then
        % saving it to the top left quadrant of the output (after the first iteration)
        transformL2D(1:x/2^(i - 1), 1:y/2^(i - 1)) = Haar2D(tempL2D);
        % Taking the top left quadrant of the image that was transformed
        % and saving it in the temporary variable to use as the new input for the next iteration
        tempL2D = transformL2D(1:x/2^i, 1:y/2^i);
    end
end
```

HaarL2DR.m

```matlab
function reverseL2D = HaarL2DR(input, L)
    % Finding and saving the number of rows and number of columns in the input
    [x, y] = size(input);
    % Setting the output to the intially provided input
    reverseL2D = input;
    % Iterating through each of the levels in descending order for the inverse
    for i = L:-1:1
        % Taking the top left quadrant image, reversing it, and then replacing it in the output
        % As we descend in the L value (largest L to smallest L), the LL quadrant gets larger, and reconstructs one image
        reverseL2D(1:x/2^(i - 1), 1:y/2^(i - 1)) = Haar2DR(reverseL2D(1:x/2^(i - 1), 1:y/2^(i - 1)));
    end
end
```

The first function in the above screenshot is the Multi-Level 2D Haar Transform. We first initialize the x and y variables to the number of rows and columns of the input. We also initialize the output and a temporary variable to be the initial input. We then loop through however many levels there are, as extracted from the passed parameters. We calculate the 2D Haar Transform on the input (located in the temporary variable), and then set this equivalent to the LL subband of the output. For iteration one, the transform will be the whole output. For the following iterations, the transform will be in the LL of the area from the previous iteration. After this, we will take the LL of the area where the transform was placed and reassign the temporary variable. This is because after every level, the new input should be the LL subband of the previous transform. After iterating each level, we will have successfully completed a L-Level 2D Haar Transform.

The second function in the above screenshot is the Multi-Level 2D Reverse Haar Transform. Ideally, we just do the opposite of the first function. We first initialize the x and y variables to the number of rows and columns of the input. We also initialize the output to be the initial input. We then loop through however many levels there are, as extracted from the passed parameters. Importantly, for the reverse we start with the largest L-value and decrease by 1 to the smallest L-value. We then calculate the 2D Reverse Haar Transform on the smallest LL subband. As the L value decreases, the area of the image calculated in the for loop increases, meaning the image essentially “overtakes” the image of the previous image. Once all levels are iterated, the output should have the reconstructed image that is identical to the original, and we will have successfully completed a L-Level 2D Reverse Haar Transform.


Editor – /Users/mathushiva/Documents/ECE462/Lab3/Lab3Week1.m
Lab3Week1.m
Figure 1
File Edit View Insert Tools Desktop Window Help
Original DWT
Maximum Contrast DWT
Command Window

```
% Reading in the image
cameraMan = imread("cameraman.png");
% Calling the appropriate Haar transform
transformedImage = HaarL2D(double(cameraMan), 3);
% Displaying the image
disp_dwt(transformedImage, 3);
```

Here we are reading in a monochrome image of a man using a camera. We then use our Multi-Level 2D Haar Transform function that was previously written to complete a 3-Level 2D Haar Transform on the image, and save the output in "transformedImage". We then use the disp_dwt function provided in the lab details to display the transform.