COMPENG 3SK3: Computer-Aided Engineering

Project #1

Lauren Stephens – stephl4 – 400270259

Due: April 8th, 2023

Overview

The demosaicing algorithm uses a linear regression model based on machine learning to estimate the values of the missing colours in an image that has been passed through a colour filter array. First, it loads in several training images, passes them through a Bayer filter, and takes random 5 pixel x 5 pixel samples from each, totally 300,000 training samples. Using the Bayer filtered training images and their known ground truths, linear regression is performed to estimate the demosaicing coefficients. These coefficients are applied to new test images, and the accuracy is compared to MATLAB's demosaicing function. The full code can be viewed in Appendix C and D.

Results

Overall, my linear regression model outperformed MATLAB's demosaicing function. Visually, the two outputted images are very similar for each test case, and the remaining artifacts are in similar locations. In general, the linear regression model was seen to mitigate the intensity of the artifacts as compared to MATLAB's function. The demosaiced images can be viewed below, and the original images can be found in Appendix A.

Figure 1: Staircase test image comparing MATLAB's demosaic function (left) with regression model (right)

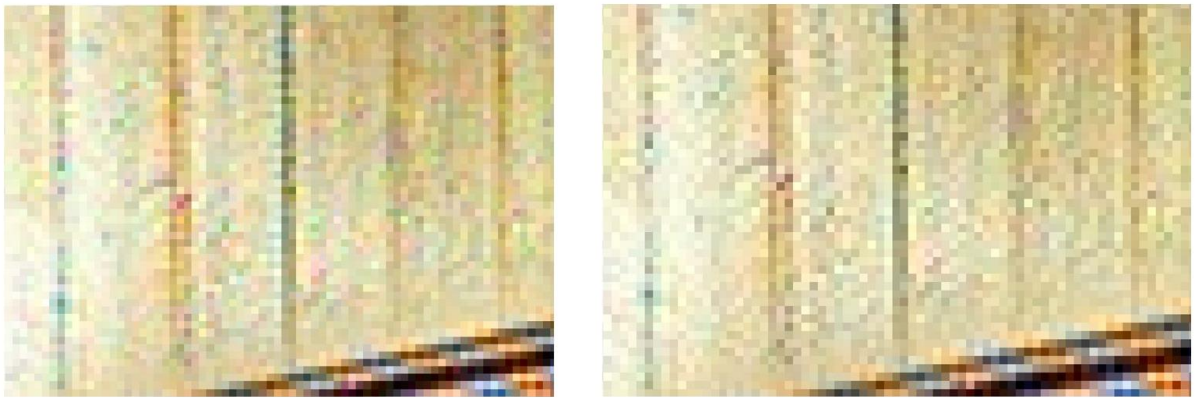

Figure 2: Zoomed in staircase test image comparing MATLAB's demosaic function (left) with regression model (right)

In the staircase test image (Figure 1), there were many areas of sharp transitions between colours that both algorithms struggled with. This image is also not very colourful, which makes the prediction errors very obvious. In the zoomed in image (Figure 2), striping artifacts can be seen in the bottom right corner

along the banister. The right image does a better job at replicating the true colour of the vertical lines in the wall, as the left image shows the stripes as having a stronger blue or red colour.

The RMSE values for the linear regression model and the MATLAB demosaic function were 6.0748 and 7.7486 respectively (Figure 3). This shows that the regression algorithm was more accurate in its estimates of the unknown colour values.

```
my_algorithm_rmse =

    6.0748


MATLAB_algorithm_rmse =

    7.7486
```

Figure 3: Staircase RMSE values for both algorithms

Figure 4: Seaside houses test image comparing MATLAB's demosaic function (left) with regression model (right)

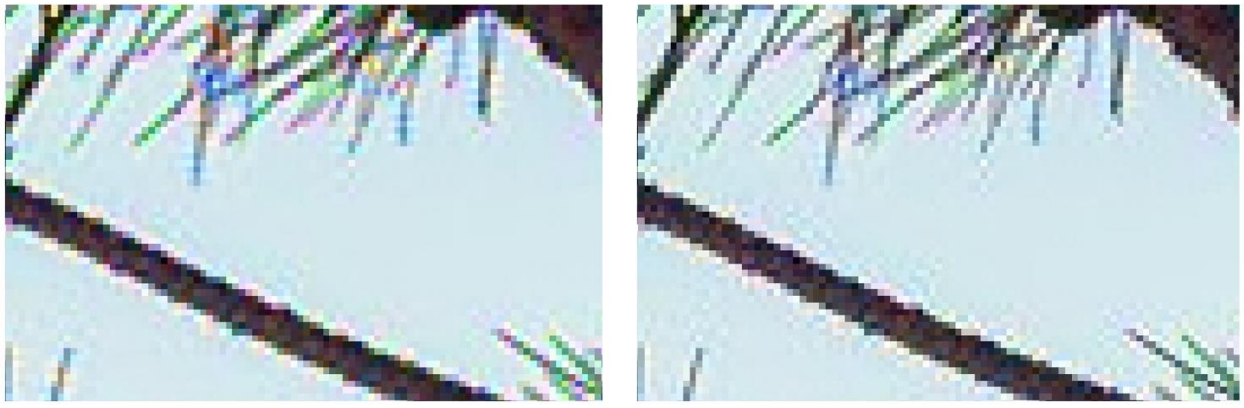

Figure 5: Zoomed in seaside houses test image comparing MATLAB's demosaic function (left) with regression model (right)

In the seaside houses test image (Figure 4), there are again many areas of sharp transition, bright colour, and high contrast leading to artifacts. The zoomed in image (Figure 5) shows these artifacts in the small branches in the upper right corner of the image. The right side does a better job at having these branches be overall dark green, whereas the left image has more prominent striping and zipper artifacts.

The RMSE values for the linear regression model and the MATLAB demosaic function were 7.7794 and 9.1568 respectively (Figure 6). This shows that the regression algorithm was more accurate in its estimates of the unknown colour values.

```
my_algorithm_rmse =

    7.7794


MATLAB_algorithm_rmse =

    9.1568
```

Figure 6: Seaside houses RMSE values for both algorithms

Figure 7: Swing set test image comparing MATLAB's demosaic function (left) with regression model (right)

Figure 8: Zoomed in swing set test image comparing MATLAB's demosaic function (left) with regression model (right)

For the final test image (Figure 7), the blurry background leads to less areas of sharp transitions, so it was assumed that both algorithms would perform well. Focusing on the swing in the foreground (Figure 8), the zoomed in image shows some zipper artifacts present on the right side of the chain, which are less pronounced in the regression model image.

The RMSE values for the linear regression model and the MATLAB demosaic function were 1.1062 and 1.2467 respectively. As assumed, the error is much less in this image due to the blurry background allowing for smooth transitions. Still, his shows that the regression algorithm was more accurate in its estimates of the unknown colour values.

```
my_algorithm_rmse =

    1.1062


MATLAB_algorithm_rmse =

    1.2467
```

Figure 9: Swing set RMSE values for both algorithms

Figure 10: Provided statue test image comparing MATLAB's demosaic function (left) with regression model (right)

Figure 11: Zoomed in statue test image comparing MATLAB's demosaic function (left) with regression model (right)

In the first provided test image (Figure 10), both MATLAB's function and the linear regression model were able to demosaic the image, but there were several areas with artifacts and noise present. In the zoomed in image (Figure 11), the column has a striping effect, but this was less vivid and less noticeable

using linear regression. In both images, areas of consistent colour that were piecewise smooth (like the statue or bushes) looked quite consistent and accurate.

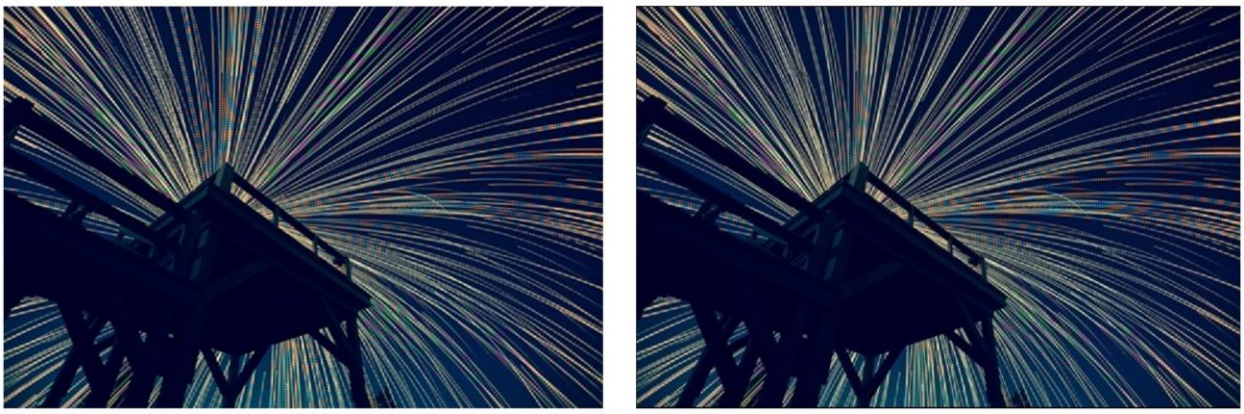

Figure 12: Provided night lights test image comparing MATLAB's demosaic function (left) with regression model (right)

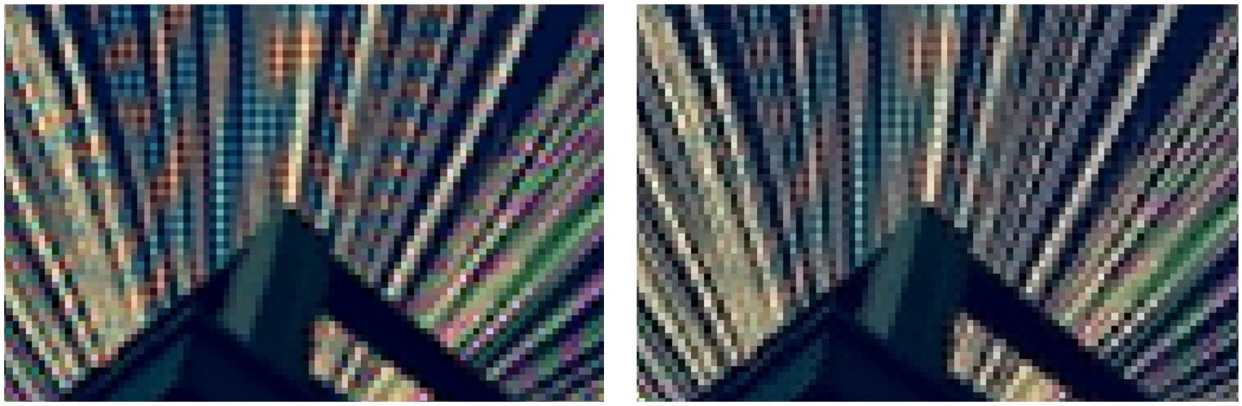

Figure 13: Zoomed in night lights test image comparing MATLAB's demosaic function (left) with regression model (right)

In the second provided test image (Figure 12), there are many areas of colour in the bright streaks, which are assumed to be artifacts and noise. In the zoomed in image (Figure 13), it is seen that striping is significantly mitigated in the regression model as compared to the demosaic function, especially if you compare the top left corners.

Limitations and Possible Improvements

Overall, the linear regression algorithm outperforms MATLAB's demosaic function, but it still has some limitations and possible improvements. Since the regression model uses a 5x5 grid to estimate the centre pixel's missing values, pixels along the edge of the image do not have all the necessary values. This algorithm simply starts estimating at the (3, 3) pixel index, which leaves a black border around the demosaiced image. One improvement could be handling these edge cases by performing regression using only the information available. Another improvement could be investigating whether linear regression is the best regression model to use, or if another model would be more accurate. Including interaction terms or non-linear components to the model may increase its accuracy. Finally, a possible improvement could be increasing the number of training samples, or choosing a more diverse set of

training images. Using only 5 training images and 300,000 training samples may limit the model's accuracy.

Demo Video

See below a link to a demo video that get the results of the first provided test image. The first figure represents MATLAB's demosaic function, while the second figure represents the linear regression image. Also note that while the program is running, the video is sped up.

Link: https://drive.google.com/file/d/1MK4SsAt3T1-yU2Ce-KIJiX_CLZd1ZU4U/view?usp=share_link

Appendices

Appendix A: Test Images

Figure A1: Staircase test image

Figure A2: Seaside houses test image

Figure A3: Swing set test image

Appendix B: Training Images

Figure B1: Mortar and pestle training image

Figure B2: Birthday cake training image

Figure B3: Statue training image

Figure B4: Peacock training image

Figure B5: Beach training image

Appendix C: Code used for provided test images

```
clear variables;

sample_img = imread('test1.png');
sample_size = size(sample_img);

% demosaic using MATLAB's demosaic function
sample_demosaic = demosaic(uint8(sample_img),'rggb');
figure(1);
imshow(sample_demosaic);

% load all training images
img1 = imread("test_1.jpg");
img2 = imread("test_2.jpg");
img3 = imread("test_3.jpg");
img4 = imread("test_4.jpg");
img5 = imread("test_5.jpg");

%produce Bayer mosaics
bayer_matrix_1 = apply_filtering(img1);
bayer_matrix_2 = apply_filtering(img2);
bayer_matrix_3 = apply_filtering(img3);
bayer_matrix_4 = apply_filtering(img4);
bayer_matrix_5 = apply_filtering(img5);

% gets random samples of matrix (X) and ground truth red values
[raw_1_red, samples_1_red] = sample_matrix(img1, bayer_matrix_1, "red");
[raw_2_red, samples_2_red] = sample_matrix(img2, bayer_matrix_2, "red");
[raw_3_red, samples_3_red] = sample_matrix(img3, bayer_matrix_3, "red");
[raw_4_red, samples_4_red] = sample_matrix(img4, bayer_matrix_4, "red");
[raw_5_red, samples_5_red] = sample_matrix(img5, bayer_matrix_5, "red");

raw_red = horzcat(raw_1_red, raw_2_red, raw_3_red, raw_4_red, raw_5_red);

red_1 = reshape(permute(samples_1_red, [2 1 3]), [25, 30000])';
red_2 = reshape(permute(samples_2_red, [2 1 3]), [25, 30000])';
red_3 = reshape(permute(samples_3_red, [2 1 3]), [25, 30000])';
red_4 = reshape(permute(samples_4_red, [2 1 3]), [25, 30000])';
red_5 = reshape(permute(samples_5_red, [2 1 3]), [25, 30000])';

X_matrix_red = vertcat(red_1, red_2, red_3, red_4, red_5);
X_matrix_red_ones = [ones(size(X_matrix_red,1),1), X_matrix_red];

% gets random samples of matrix (X) and ground truth blue values
[raw_1_blue, samples_1_blue] = sample_matrix(img1, bayer_matrix_1, "blue");
[raw_2_blue, samples_2_blue] = sample_matrix(img2, bayer_matrix_2, "blue");
[raw_3_blue, samples_3_blue] = sample_matrix(img3, bayer_matrix_3, "blue");
[raw_4_blue, samples_4_blue] = sample_matrix(img4, bayer_matrix_4, "blue");
[raw_5_blue, samples_5_blue] = sample_matrix(img5, bayer_matrix_5, "blue");

raw_blue = horzcat(raw_1_blue, raw_2_blue, raw_3_blue, raw_4_blue,
raw_5_blue);

blue_1 = reshape(permute(samples_1_blue, [2 1 3]), [25, 30000])';
blue_2 = reshape(permute(samples_2_blue, [2 1 3]), [25, 30000])';
blue_3 = reshape(permute(samples_3_blue, [2 1 3]), [25, 30000])';
```

```
blue_4 = reshape(permute(samples_4_blue, [2 1 3]), [25, 30000])';
blue_5 = reshape(permute(samples_5_blue, [2 1 3]), [25, 30000])';

X_matrix_blue = vertcat(blue_1, blue_2, blue_3, blue_4, blue_5);
X_matrix_blue_ones = [ones(size(X_matrix_blue,1),1), X_matrix_blue];

% gets random samples of matrix (X) and ground truth green 1 values
[raw_1_green1, samples_1_green1] = sample_matrix(img1, bayer_matrix_1,
"green1");
[raw_2_green1, samples_2_green1] = sample_matrix(img2, bayer_matrix_2,
"green1");
[raw_3_green1, samples_3_green1] = sample_matrix(img3, bayer_matrix_3,
"green1");
[raw_4_green1, samples_4_green1] = sample_matrix(img4, bayer_matrix_4,
"green1");
[raw_5_green1, samples_5_green1] = sample_matrix(img5, bayer_matrix_5,
"green1");

raw_green1 = horzcat(raw_1_green1, raw_2_green1, raw_3_green1, raw_4_green1,
raw_5_green1);

green_1 = reshape(permute(samples_1_green1, [2 1 3]), [25, 30000])';
green_2 = reshape(permute(samples_2_green1, [2 1 3]), [25, 30000])';
green_3 = reshape(permute(samples_3_green1, [2 1 3]), [25, 30000])';
green_4 = reshape(permute(samples_4_green1, [2 1 3]), [25, 30000])';
green_5 = reshape(permute(samples_5_green1, [2 1 3]), [25, 30000])';

X_matrix_green1 = vertcat(green_1, green_2, green_3, green_4, green_5);
X_matrix_green1_ones = [ones(size(X_matrix_green1,1),1), X_matrix_green1];

% gets random samples of matrix (X) and ground truth green 2 values
[raw_1_green2, samples_1_green2] = sample_matrix(img1, bayer_matrix_1,
"green2");
[raw_2_green2, samples_2_green2] = sample_matrix(img2, bayer_matrix_2,
"green2");
[raw_3_green2, samples_3_green2] = sample_matrix(img3, bayer_matrix_3,
"green2");
[raw_4_green2, samples_4_green2] = sample_matrix(img4, bayer_matrix_4,
"green2");
[raw_5_green2, samples_5_green2] = sample_matrix(img5, bayer_matrix_5,
"green2");

raw_green2 = horzcat(raw_1_green2, raw_2_green2, raw_3_green2, raw_4_green2,
raw_5_green2);

green2_1 = reshape(permute(samples_1_green2, [2 1 3]), [25, 30000])';
green2_2 = reshape(permute(samples_2_green2, [2 1 3]), [25, 30000])';
green2_3 = reshape(permute(samples_3_green2, [2 1 3]), [25, 30000])';
green2_4 = reshape(permute(samples_4_green2, [2 1 3]), [25, 30000])';
green2_5 = reshape(permute(samples_5_green2, [2 1 3]), [25, 30000])';

X_matrix_green2 = vertcat(green2_1, green2_2, green2_3, green2_4, green2_5);
X_matrix_green2_ones = [ones(size(X_matrix_green2,1),1), X_matrix_green2];

% compute coefficients
green_from_red = compute_coefficients(raw_red(2, :)', X_matrix_red_ones);
blue_from_red = compute_coefficients(raw_red(3, :)', X_matrix_red_ones);
```

```
green_from_blue = compute_coefficients(raw_blue(2, :)', X_matrix_blue_ones);
red_from_blue = compute_coefficients(raw_blue(1, :)', X_matrix_blue_ones);

red_from_green1 = compute_coefficients(raw_green1(1, :)',
X_matrix_green1_ones);
blue_from_green1 = compute_coefficients(raw_green1(3, :)',
X_matrix_green1_ones);

red_from_green2 = compute_coefficients(raw_green2(1, :)',
X_matrix_green2_ones);
blue_from_green2 = compute_coefficients(raw_green2(3, :)',
X_matrix_green2_ones);

demosaiced_image = zeros(sample_size(1), sample_size(2), 3);

for j = 3:sample_size(2)-2
    for i = 3:sample_size(1)-2

        grey = double(im2gray(sample_img));

        pred_values = grey(i-2:i+2,j-2:j+2);

        X_reshaped = reshape(pred_values, 1, []);
        X = [ones(size(X_reshaped,1),1), X_reshaped];

        if mod(i, 2) == 0
            if mod(j, 2) == 0
                %blue;

                blue = grey(i, j);
                red = predict_colour(X, red_from_blue);
                green = predict_colour(X, green_from_blue);

            else
                %green2

                green = grey(i, j);
                red = predict_colour(X, red_from_green2);
                blue = predict_colour(X, blue_from_green2);

            end
        else
            if mod(j, 2) == 0
                %green1

                green = grey(i, j);
                red = predict_colour(X, red_from_green1);
                blue = predict_colour(X, blue_from_green1);

            else
                %red

                red = grey(i, j);
                green = predict_colour(X, green_from_red);
                blue = predict_colour(X, blue_from_red);
```

```
            end
        end

    demosaiced_image(i, j, 1) = red;
    demosaiced_image(i, j, 2) = green;
    demosaiced_image(i, j, 3) = blue;

    end

end

figure(2)
actual_demosaic = uint8(demosaiced_image);
imshow(actual_demosaic)



% this function takes an image and an input and outputs the Bayer matrix
function bayer_filter = apply_filtering(img)

    [height, width, ~] = size(img);   %% not right :(

    % Extract the ground truth R, G, and B channels
    R = img(:,:,1);
    G = img(:,:,2);
    B = img(:,:,3);

    % Define the Bayer filter pattern
    bayer = [1 2; 3 4];

    % Initialize the output image
    out = zeros(size(img));

    % Loop through each pixel and extract the RGB values
    for x = 1:height
        for y = 1:width

            % Determine the filter position
            filter = bayer(mod(x, 2) + 1, mod(y, 2) + 1);

            % Apply the filter
            if filter == 4
                out(x,y,:) = [R(x,y),0,0];
            elseif filter == 2
                out(x,y,:) = [0,G(x,y),0];
            elseif filter == 3
                out(x,y,:) = [0,G(x,y),0];
            elseif filter == 1
                out(x,y,:) = [0,0,B(x,y)];
            end
        end
    end

    %creates 2D bayer_matrix
    bayer_filter = sum(out, 3);

end
```

```
% This function makes 10 random training samples of the bayer mosaic at a
specific configuration and
% then takes raw data and holds the corresponding ground truth for each
sample
function [output_truth, output_sample] = sample_matrix(raw_data,
bayer_matrix, colour)

    % Get the size of the matrix
    matrixSize = size(bayer_matrix);

    % Initialize output array for random samples
    output_sample = zeros(5, 5, 30000);

    % Initialize output array for ground truths
    output_truth = zeros(3, 30000);

    for i = 1:30000
        % Generate a random even row index
        random_row = randi([5, matrixSize(1)-2], 1);

        if colour =="blue" || colour == "green1"
            if mod(random_row, 2) ~= 0
                random_row = random_row - 1;
            end
        else
            if mod(random_row, 2) == 0
                random_row = random_row - 1;
            end
        end

        % Generate a random even column index
        random_col = randi([5, matrixSize(2)-2], 1);
        if colour =="blue" || colour == "green2"
            if mod(random_col, 2) ~= 0
                random_col = random_col - 1;
            end
        else
            if mod(random_col, 2) == 0
                random_col = random_col - 1;
            end
        end

        output_sample(:, :, i) = bayer_matrix(random_row-2:random_row+2,
random_col-2:random_col+2);
        output_truth(:, i) = raw_data(random_row, random_col, :);
    end

end

function b = compute_coefficients(ground_truth, X_matrix)

    b = inv(X_matrix' * X_matrix)*X_matrix'*ground_truth;

end

function pred = predict_colour(X, a)
```

```
    pred = X*a;

end

function rmse = calc_rmse(actual, predicted)

    dimensions = size(actual);
    pixels = (dimensions(1)-4)*(dimensions(2)-4)*dimensions(3);

    ssd = 0;

    for i = 3:dimensions(1)-2
        for j = 3:dimensions(2)-2
            for k = 3:dimensions(3)
                diff = actual(i,j,k) - predicted(i,j,k);
                ssd = ssd + diff^2;
            end
        end
    end

    mse = ssd / pixels;

    rmse = sqrt(mse);


end
```

Appendix D: Code used for test images from RGB images

```
clear variables;

sample_img = imread('attempt3.jpg');
bayer_matrix_actual = apply_filtering(sample_img);
sample_size = size(sample_img);

% demosaic using MATLAB's demosaic function
sample_demosaic = demosaic(uint8(bayer_matrix_actual),'rggb');
figure(1);
imshow(sample_demosaic);

% load all training images
img1 = imread("test_1.jpg");
img2 = imread("test_2.jpg");
img3 = imread("test_3.jpg");
img4 = imread("test_4.jpg");
img5 = imread("test_5.jpg");

%produce Bayer mosaics
bayer_matrix_1 = apply_filtering(img1);
bayer_matrix_2 = apply_filtering(img2);
bayer_matrix_3 = apply_filtering(img3);
bayer_matrix_4 = apply_filtering(img4);
bayer_matrix_5 = apply_filtering(img5);

% gets random samples of matrix (X) and ground truth red values
[raw_1_red, samples_1_red] = sample_matrix(img1, bayer_matrix_1, "red");
[raw_2_red, samples_2_red] = sample_matrix(img2, bayer_matrix_2, "red");
[raw_3_red, samples_3_red] = sample_matrix(img3, bayer_matrix_3, "red");
[raw_4_red, samples_4_red] = sample_matrix(img4, bayer_matrix_4, "red");
[raw_5_red, samples_5_red] = sample_matrix(img5, bayer_matrix_5, "red");

raw_red = horzcat(raw_1_red, raw_2_red, raw_3_red, raw_4_red, raw_5_red);

red_1 = reshape(permute(samples_1_red, [2 1 3]), [25, 30000])';
red_2 = reshape(permute(samples_2_red, [2 1 3]), [25, 30000])';
red_3 = reshape(permute(samples_3_red, [2 1 3]), [25, 30000])';
red_4 = reshape(permute(samples_4_red, [2 1 3]), [25, 30000])';
red_5 = reshape(permute(samples_5_red, [2 1 3]), [25, 30000])';

X_matrix_red = vertcat(red_1, red_2, red_3, red_4, red_5);
X_matrix_red_ones = [ones(size(X_matrix_red,1),1), X_matrix_red];

% gets random samples of matrix (X) and ground truth blue values
[raw_1_blue, samples_1_blue] = sample_matrix(img1, bayer_matrix_1, "blue");
[raw_2_blue, samples_2_blue] = sample_matrix(img2, bayer_matrix_2, "blue");
[raw_3_blue, samples_3_blue] = sample_matrix(img3, bayer_matrix_3, "blue");
[raw_4_blue, samples_4_blue] = sample_matrix(img4, bayer_matrix_4, "blue");
[raw_5_blue, samples_5_blue] = sample_matrix(img5, bayer_matrix_5, "blue");

raw_blue = horzcat(raw_1_blue, raw_2_blue, raw_3_blue, raw_4_blue,
raw_5_blue);

blue_1 = reshape(permute(samples_1_blue, [2 1 3]), [25, 30000])';
blue_2 = reshape(permute(samples_2_blue, [2 1 3]), [25, 30000])';
```

```
blue_3 = reshape(permute(samples_3_blue, [2 1 3]), [25, 30000])';
blue_4 = reshape(permute(samples_4_blue, [2 1 3]), [25, 30000])';
blue_5 = reshape(permute(samples_5_blue, [2 1 3]), [25, 30000])';

X_matrix_blue = vertcat(blue_1, blue_2, blue_3, blue_4, blue_5);
X_matrix_blue_ones = [ones(size(X_matrix_blue,1),1), X_matrix_blue];

% gets random samples of matrix (X) and ground truth green 1 values
[raw_1_green1, samples_1_green1] = sample_matrix(img1, bayer_matrix_1,
"green1");
[raw_2_green1, samples_2_green1] = sample_matrix(img2, bayer_matrix_2,
"green1");
[raw_3_green1, samples_3_green1] = sample_matrix(img3, bayer_matrix_3,
"green1");
[raw_4_green1, samples_4_green1] = sample_matrix(img4, bayer_matrix_4,
"green1");
[raw_5_green1, samples_5_green1] = sample_matrix(img5, bayer_matrix_5,
"green1");

raw_green1 = horzcat(raw_1_green1, raw_2_green1, raw_3_green1, raw_4_green1,
raw_5_green1);

green_1 = reshape(permute(samples_1_green1, [2 1 3]), [25, 30000])';
green_2 = reshape(permute(samples_2_green1, [2 1 3]), [25, 30000])';
green_3 = reshape(permute(samples_3_green1, [2 1 3]), [25, 30000])';
green_4 = reshape(permute(samples_4_green1, [2 1 3]), [25, 30000])';
green_5 = reshape(permute(samples_5_green1, [2 1 3]), [25, 30000])';

X_matrix_green1 = vertcat(green_1, green_2, green_3, green_4, green_5);
X_matrix_green1_ones = [ones(size(X_matrix_green1,1),1), X_matrix_green1];

% gets random samples of matrix (X) and ground truth green 2 values
[raw_1_green2, samples_1_green2] = sample_matrix(img1, bayer_matrix_1,
"green2");
[raw_2_green2, samples_2_green2] = sample_matrix(img2, bayer_matrix_2,
"green2");
[raw_3_green2, samples_3_green2] = sample_matrix(img3, bayer_matrix_3,
"green2");
[raw_4_green2, samples_4_green2] = sample_matrix(img4, bayer_matrix_4,
"green2");
[raw_5_green2, samples_5_green2] = sample_matrix(img5, bayer_matrix_5,
"green2");

raw_green2 = horzcat(raw_1_green2, raw_2_green2, raw_3_green2, raw_4_green2,
raw_5_green2);

green2_1 = reshape(permute(samples_1_green2, [2 1 3]), [25, 30000])';
green2_2 = reshape(permute(samples_2_green2, [2 1 3]), [25, 30000])';
green2_3 = reshape(permute(samples_3_green2, [2 1 3]), [25, 30000])';
green2_4 = reshape(permute(samples_4_green2, [2 1 3]), [25, 30000])';
green2_5 = reshape(permute(samples_5_green2, [2 1 3]), [25, 30000])';

X_matrix_green2 = vertcat(green2_1, green2_2, green2_3, green2_4, green2_5);
X_matrix_green2_ones = [ones(size(X_matrix_green2,1),1), X_matrix_green2];

% compute coefficients
green_from_red = compute_coefficients(raw_red(2, :)', X_matrix_red_ones);
```

```
blue_from_red = compute_coefficients(raw_red(3, :)', X_matrix_red_ones);

green_from_blue = compute_coefficients(raw_blue(2, :)', X_matrix_blue_ones);
red_from_blue = compute_coefficients(raw_blue(1, :)', X_matrix_blue_ones);

red_from_green1 = compute_coefficients(raw_green1(1, :)',
X_matrix_green1_ones);
blue_from_green1 = compute_coefficients(raw_green1(3, :)',
X_matrix_green1_ones);

red_from_green2 = compute_coefficients(raw_green2(1, :)',
X_matrix_green2_ones);
blue_from_green2 = compute_coefficients(raw_green2(3, :)',
X_matrix_green2_ones);

% intialize matrix for demosaiced image
demosaiced_image = zeros(sample_size(1), sample_size(2), 3);

for j = 3:sample_size(2)-2
    for i = 3:sample_size(1)-2

        pred_values = bayer_matrix_actual(i-2:i+2,j-2:j+2);

        X_reshaped = reshape(pred_values, 1, []);
        X = [ones(size(X_reshaped,1),1), X_reshaped];

        if mod(i, 2) == 0
            if mod(j, 2) == 0
                %blue

                blue = bayer_matrix_actual(i, j);
                red = predict_colour(X, red_from_blue);
                green = predict_colour(X, green_from_blue);

            else
                %green2

                green = bayer_matrix_actual(i, j);
                red = predict_colour(X, red_from_green2);
                blue = predict_colour(X, blue_from_green2);

            end
        else
            if mod(j, 2) == 0
                %green1

                green = bayer_matrix_actual(i, j);
                red = predict_colour(X, red_from_green1);
                blue = predict_colour(X, blue_from_green1);

            else
                %red

                red = bayer_matrix_actual(i, j);
                green = predict_colour(X, green_from_red);
                blue = predict_colour(X, blue_from_red);
```

```
            end
        end

    demosaiced_image(i, j, 1) = red;
    demosaiced_image(i, j, 2) = green;
    demosaiced_image(i, j, 3) = blue;

    end

end

figure(2)
actual_demosaic = uint8(demosaiced_image);
imshow(actual_demosaic)


my_algorithm_rmse = calc_rmse(double(sample_img), double(actual_demosaic))
MATLAB_algorithm_rmse = calc_rmse(double(sample_img),
double(sample_demosaic))


% this function takes an image and an input and outputs the Bayer matrix
function bayer_filter = apply_filtering(img)

    [height, width, ~] = size(img);   %% not right :(

    % Extract the ground truth R, G, and B channels
    R = img(:,:,1);
    G = img(:,:,2);
    B = img(:,:,3);

    % Define the Bayer filter pattern
    bayer = [1 2; 3 4];

    % Initialize the output image
    out = zeros(size(img));

    % Loop through each pixel and extract the RGB values
    for x = 1:height
        for y = 1:width

            % Determine the filter position
            filter = bayer(mod(x, 2) + 1, mod(y, 2) + 1);

            % Apply the filter
            if filter == 4
                out(x,y,:) = [R(x,y),0,0];
            elseif filter == 2
                out(x,y,:) = [0,G(x,y),0];
            elseif filter == 3
                out(x,y,:) = [0,G(x,y),0];
            elseif filter == 1
                out(x,y,:) = [0,0,B(x,y)];
            end
        end
    end
```

```
    %creates 2D bayer_matrix
    bayer_filter = sum(out, 3);

end

% This function makes 10 random training samples of the bayer mosaic at a
specific configuration and
% then takes raw data and holds the corresponding ground truth for each
sample
function [output_truth, output_sample] = sample_matrix(raw_data,
bayer_matrix, colour)

    % Get the size of the matrix
    matrixSize = size(bayer_matrix);

    % Initialize output array for random samples
    output_sample = zeros(5, 5, 30000);

    % Initialize output array for ground truths
    output_truth = zeros(3, 30000);

    for i = 1:30000
        % Generate a random even row index
        random_row = randi([5, matrixSize(1)-2], 1);

        if colour =="blue" || colour == "green1"
            if mod(random_row, 2) ~= 0
                random_row = random_row - 1;
            end
        else
            if mod(random_row, 2) == 0
                random_row = random_row - 1;
            end
        end

        % Generate a random even column index
        random_col = randi([5, matrixSize(2)-2], 1);
        if colour =="blue" || colour == "green2"
            if mod(random_col, 2) ~= 0
                random_col = random_col - 1;
            end
        else
            if mod(random_col, 2) == 0
                random_col = random_col - 1;
            end
        end

        output_sample(:, :, i) = bayer_matrix(random_row-2:random_row+2,
random_col-2:random_col+2);
        output_truth(:, i) = raw_data(random_row, random_col, :);
    end

end

function b = compute_coefficients(ground_truth, X_matrix)

    b = inv(X_matrix' * X_matrix)*X_matrix'*ground_truth;
```

```
end

function pred = predict_colour(X, a)
    pred = X*a;

end

function rmse = calc_rmse(actual, predicted)

    dimensions = size(actual);
    pixels = (dimensions(1)-4)*(dimensions(2)-4)*dimensions(3);

    ssd = 0;

    for i = 3:dimensions(1)-2
        for j = 3:dimensions(2)-2
            for k = 3:dimensions(3)
                diff = actual(i,j,k) - predicted(i,j,k);
                ssd = ssd + diff^2;
            end
        end
    end

    mse = ssd / pixels;

    rmse = sqrt(mse);


end
```