COMPENG 4TN4: Image Processing
Project Phase 1

Author: Farhan Rouf

Contents

1 INTRODUCTION

The intended focus for Phase 1 is to implement a demosaicing algorithm based on linear regression. I have implemented the demosaicing algorithm in two ways, one way where the missing color component is interpolated with all the information in the Bayer patch, which the other way is where the missing color component is interpolated with only the information of the same color within the Bayer patch. In either case, I have used the normal equations to find the 8 coefficient matrices. Additional features to develop were white balance and histogram equalization which will be discussed as well.

2 METHODS

The training data set consists of 5 images shown in the Figure 1 below.

Figure 1

The coefficient matrices were generated using two different methods. The first method uses all the information in the bayer patch to interpolate the missing color. While the second method only takes the information from the colors that match the missing color in the patch i.e. if the missing color is green only the green information in the patch is used for interpolation.

Before discussing the training method, let us take a look at the mathematical equation we are trying to solve.

$$\tilde{Y} = A\theta$$

Here, A represents the Bayer patches and theta represents the coefficient matrix multiplied to it to find the missing colors. The estimated Y value shown above is the estimated missing colors. For the same coefficient matrix this equation can give all the missing colors given all the patches if A is a matrix consisting of all the patches in an image and Y is all the missing components of that color (green, red, or blue). The function to be minimized then is just:

$$min_{\theta}(Y - A\theta)^2$$

Further simplifying this equation by taking the derivative results in the coefficient matrix for that individual color as the normal equations:

$$\theta = (A^T A)^{-1} A^T Y$$

This is the method that I have used to compute my coefficient matrices. In the first method, my A matrix remains unchanged for both missing color components. While in the second method, I create 2 A matrices each representing only the values of the missing colors.

2.1 Full Information Linear Regression

To train the coefficient matrices, each image in the training data set is converted to the 4 pattern types RGGB, GBRG, GRBG, and BGGR.

For each pattern, the A matrix is generated by randomly collecting a set number of patches from each image. Concurrently, an array of the ground truth is generated from the original images corresponding to the center of each patch in A. This results in A containing an array of patches and the ground truth middle pixels of those patches.

The ground truth middle pixels are then split into the corresponding missing colors as y1 and y2 i.e. if the middle pixel Bayer color is green, the missing colors blue and red are returned as y1 and y2 respectively.

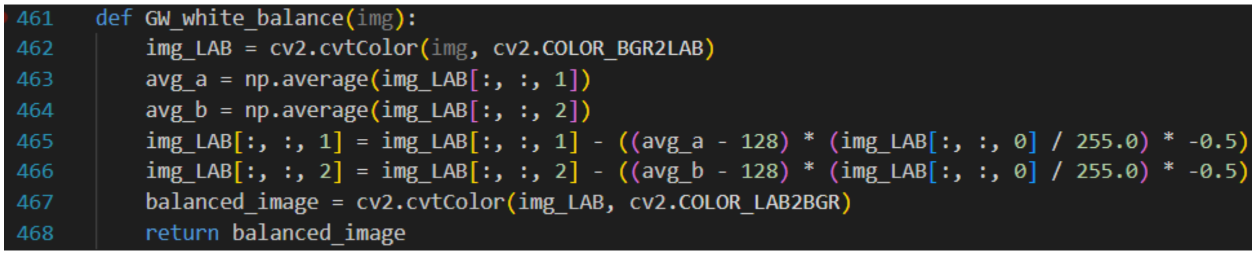
Finally, we apply the normal equations using A and y1 to find the first coefficient matrix and using A and y2 to find the second coefficient matrix.

2.2 Color-Matched Information Linear Regression

This method does everything done in the first method except the A matrix is also split into two matrices A1 and A2, where all the entries are 0 except the entries related to their corresponding missing components. For example, if green is the first missing component A1 contains only the green values with the other values set to 0.

2.3 White Balancing and Histogram Equalization

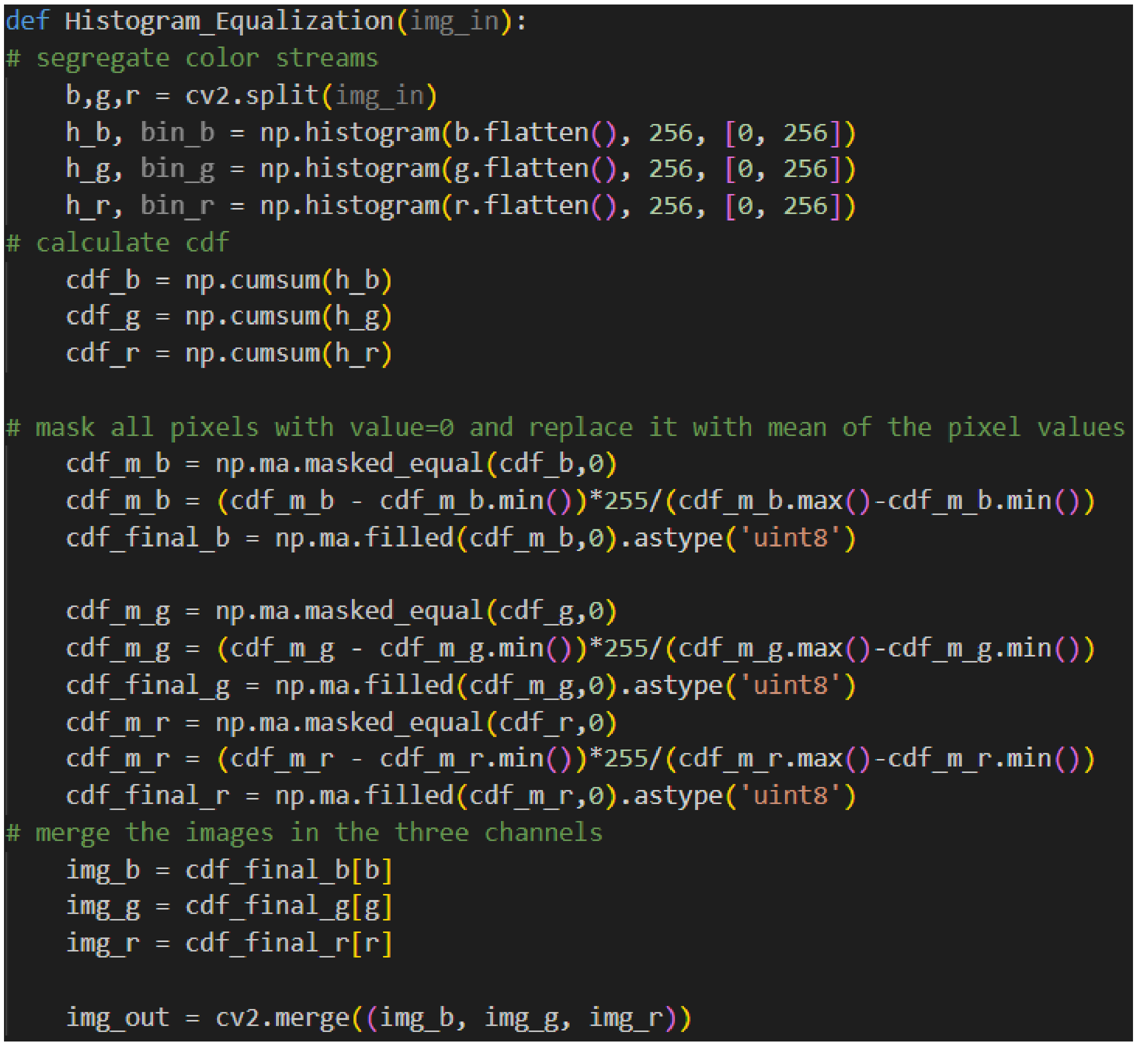
To implement white balancing a simple algorithm was used where the BGR image was converted to L*a*b. Then the code in Figure 2 was implemented. **NOTE**: the -0.5 scale factor applied at the end of line 465 and 466 can be tuned to increase whiteness in image.



```python
461  def GW_white_balance(img):
462      img_LAB = cv2.cvtColor(img, cv2.COLOR_BGR2LAB)
463      avg_a = np.average(img_LAB[:, :, 1])
464      avg_b = np.average(img_LAB[:, :, 2])
465      img_LAB[:, :, 1] = img_LAB[:, :, 1] - ((avg_a - 128) * (img_LAB[:, :, 0] / 255.0) * -0.5)
466      img_LAB[:, :, 2] = img_LAB[:, :, 2] - ((avg_b - 128) * (img_LAB[:, :, 0] / 255.0) * -0.5)
467      balanced_image = cv2.cvtColor(img_LAB, cv2.COLOR_LAB2BGR)
468      return balanced_image
```

Figure 2

To implement histogram equalization, on each color channel, first the image is split into its individual color channels and then equalized using the histogram equalization algorithm shown in class.



```python
def Histogram_Equalization(img_in):
# segregate color streams
    b,g,r = cv2.split(img_in)
    h_b, bin_b = np.histogram(b.flatten(), 256, [0, 256])
    h_g, bin_g = np.histogram(g.flatten(), 256, [0, 256])
    h_r, bin_r = np.histogram(r.flatten(), 256, [0, 256])
# calculate cdf
    cdf_b = np.cumsum(h_b)
    cdf_g = np.cumsum(h_g)
    cdf_r = np.cumsum(h_r)

# mask all pixels with value=0 and replace it with mean of the pixel values
    cdf_m_b = np.ma.masked_equal(cdf_b,0)
    cdf_m_b = (cdf_m_b - cdf_m_b.min())*255/(cdf_m_b.max()-cdf_m_b.min())
    cdf_final_b = np.ma.filled(cdf_m_b,0).astype('uint8')

    cdf_m_g = np.ma.masked_equal(cdf_g,0)
    cdf_m_g = (cdf_m_g - cdf_m_g.min())*255/(cdf_m_g.max()-cdf_m_g.min())
    cdf_final_g = np.ma.filled(cdf_m_g,0).astype('uint8')
    cdf_m_r = np.ma.masked_equal(cdf_r,0)
    cdf_m_r = (cdf_m_r - cdf_m_r.min())*255/(cdf_m_r.max()-cdf_m_r.min())
    cdf_final_r = np.ma.filled(cdf_m_r,0).astype('uint8')
# merge the images in the three channels
    img_b = cdf_final_b[b]
    img_g = cdf_final_g[g]
    img_r = cdf_final_r[r]

    img_out = cv2.merge((img_b, img_g, img_r))
```

Figure 3

3 RESULTS

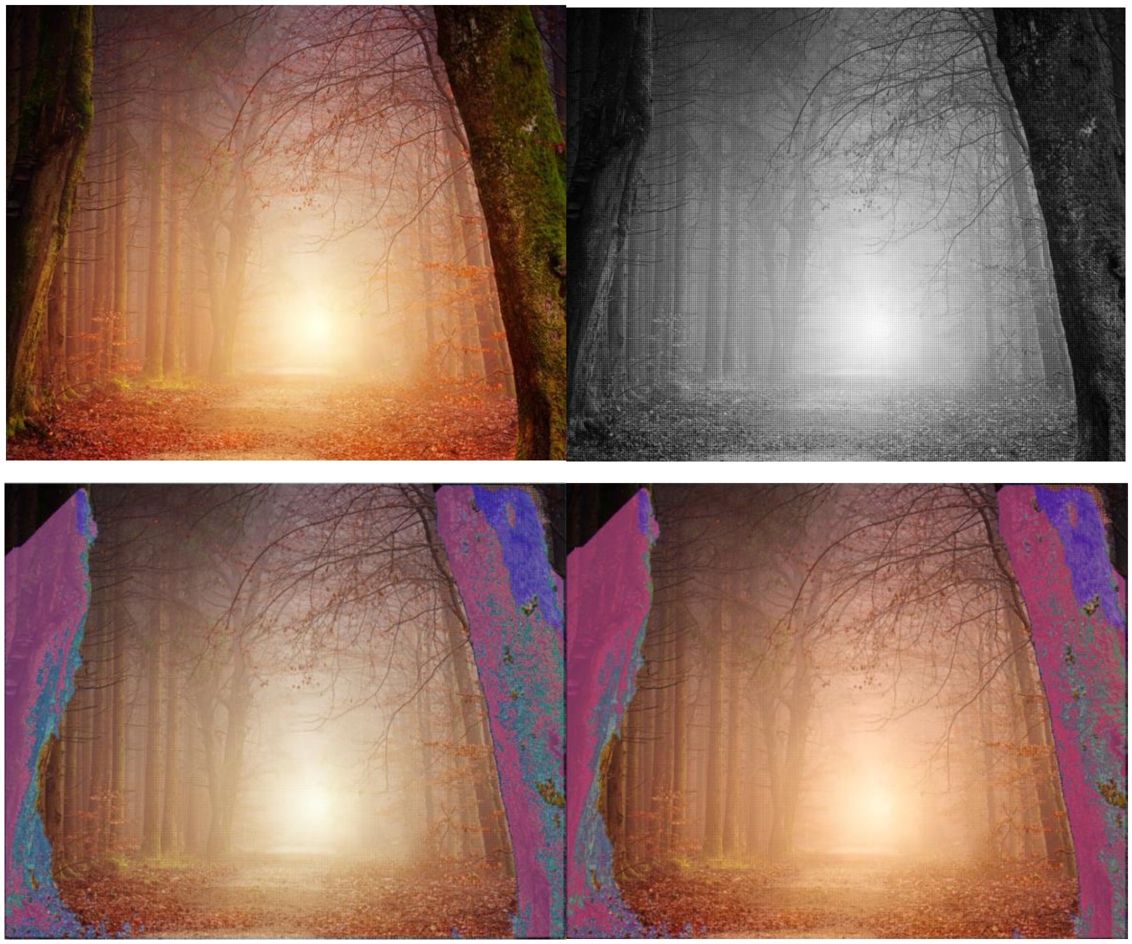
Using the computed coefficient matrices from both methods, I have demosaiced the 5 training images. Additionally, I have also altered computed the coefficient matrices with different patch sizes as well as different number of patches for training purposes to produce lowest possible error.

To reiterate, all the training images were converted to the 4 Bayer patterns and the corresponding coefficient matrices were trained on each Bayer Pattern.

3.1 Full Information

Here are the results of the algorithms and the root mean square error (RMSE) between the outputs and the MATLAB demosaic() outputs using the full information patches.

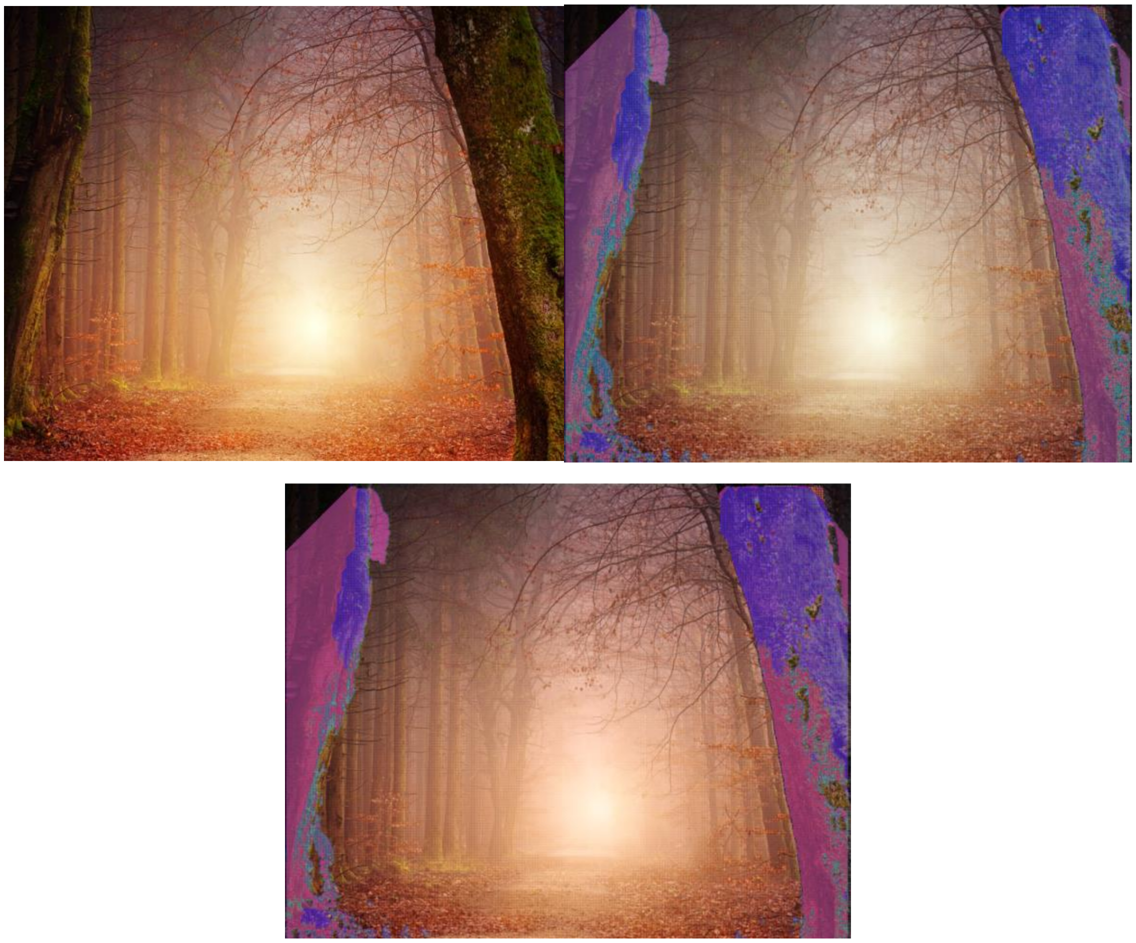
1. **GBRG** Training Configuration: Patch Size = 5 Training Samples per Pattern = 1000



Top Left: Original Image. Top Right: Mosaic Image (GBRG)
Bottom Left: Demosaiced Bottom Right: Demosaiced + White Balance + Histogram Equalization

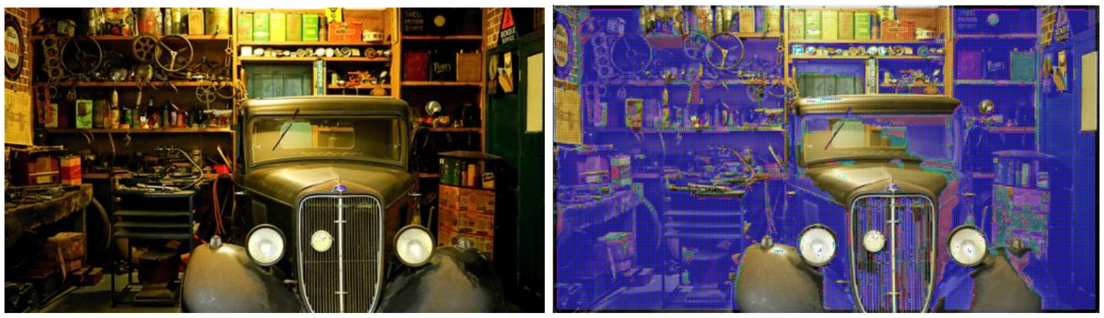
MATLAB RMSE: 3.603680334099297
DEMOSAIC RMSE: 8.129742044532332
DEMOSAIC + WB + HE RMSE: 8.57294390911002

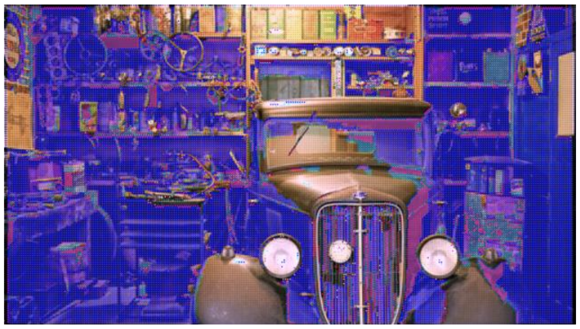
2. **GBRG** Training Configuration: Patch_Size = 7 Training Samples per pattern = 5000


Top Left: Original Image. Top Right: Demosaiced Bottom: Demosaiced + White Balance + Histogram Equalization

MATLAB RMSE: 3.603680334099297
DEMOSAIC RMSE: 8.172384111687698
DEMOSAIC + WB + HE RMSE: 8.555832536265259

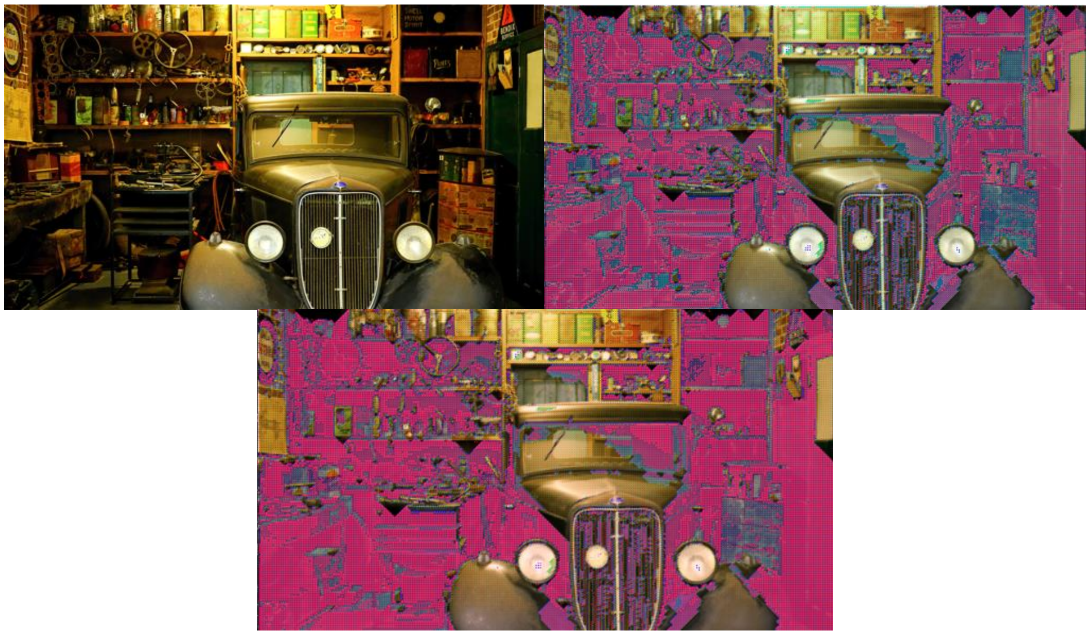
3. **RGGB** Training Configuration: Patch Size = 7 Training Samples per pattern = 5000


Left: Original Image. Right: Demosaic (RGGB)


Demosaic + HE + WB

MATLAB RMSE: 4.919107117452335
DEMOSAIC RMSE: 8.308448274313479
DEMOSAIC + WB + HE RMSE: 9.547869400782991

4. **RGGB** Training Configuration: Patch Size = 3 Training Samples per pattern = 5000

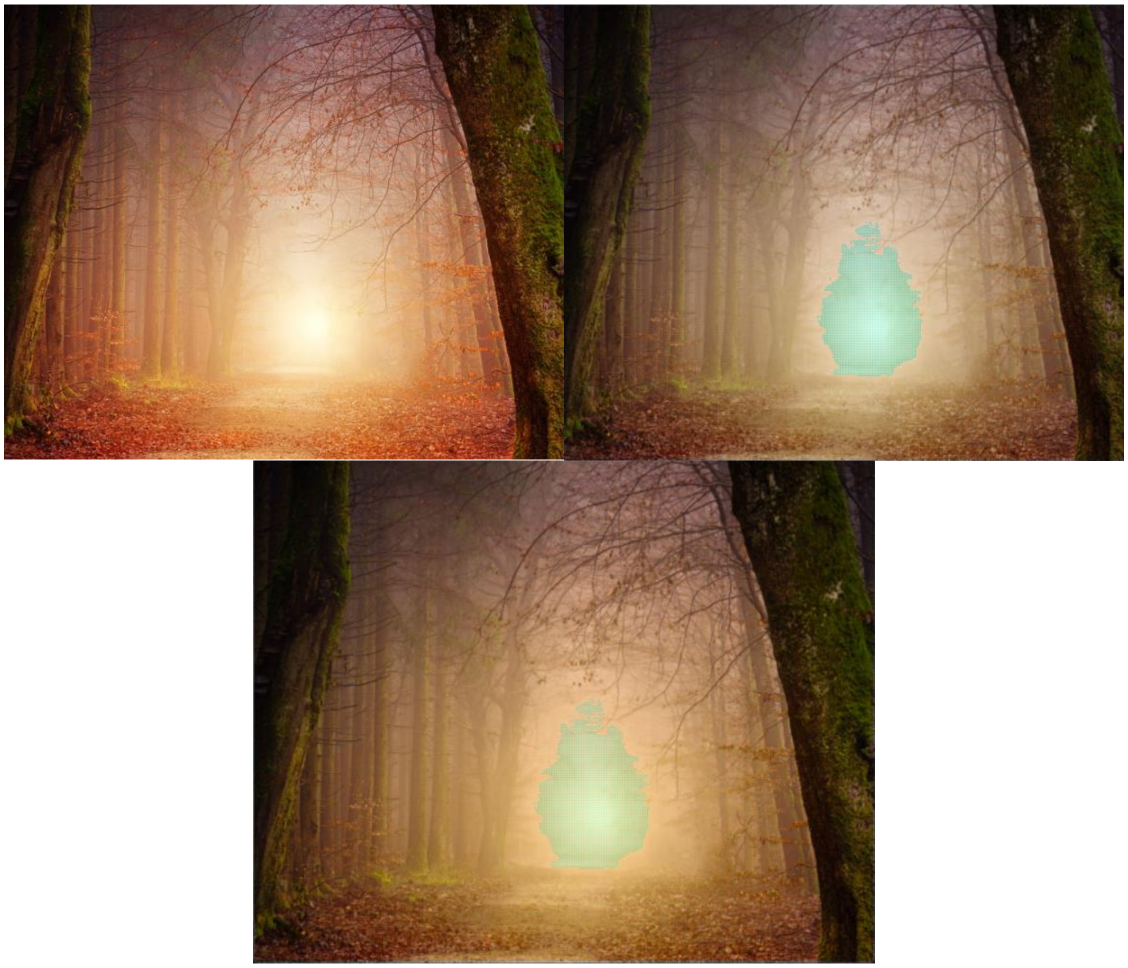

Top Left: Original Image. Top Right: Demosaic RGGB
Bottom: Demosaic + HE + WB

MATLAB RMSE: 4.919107117452335
DEMOSAIC RMSE: 8.134204188350436
DEMOSAIC + WB + HE RMSE: 9.211740539214128

3.2 Color-Matched Information

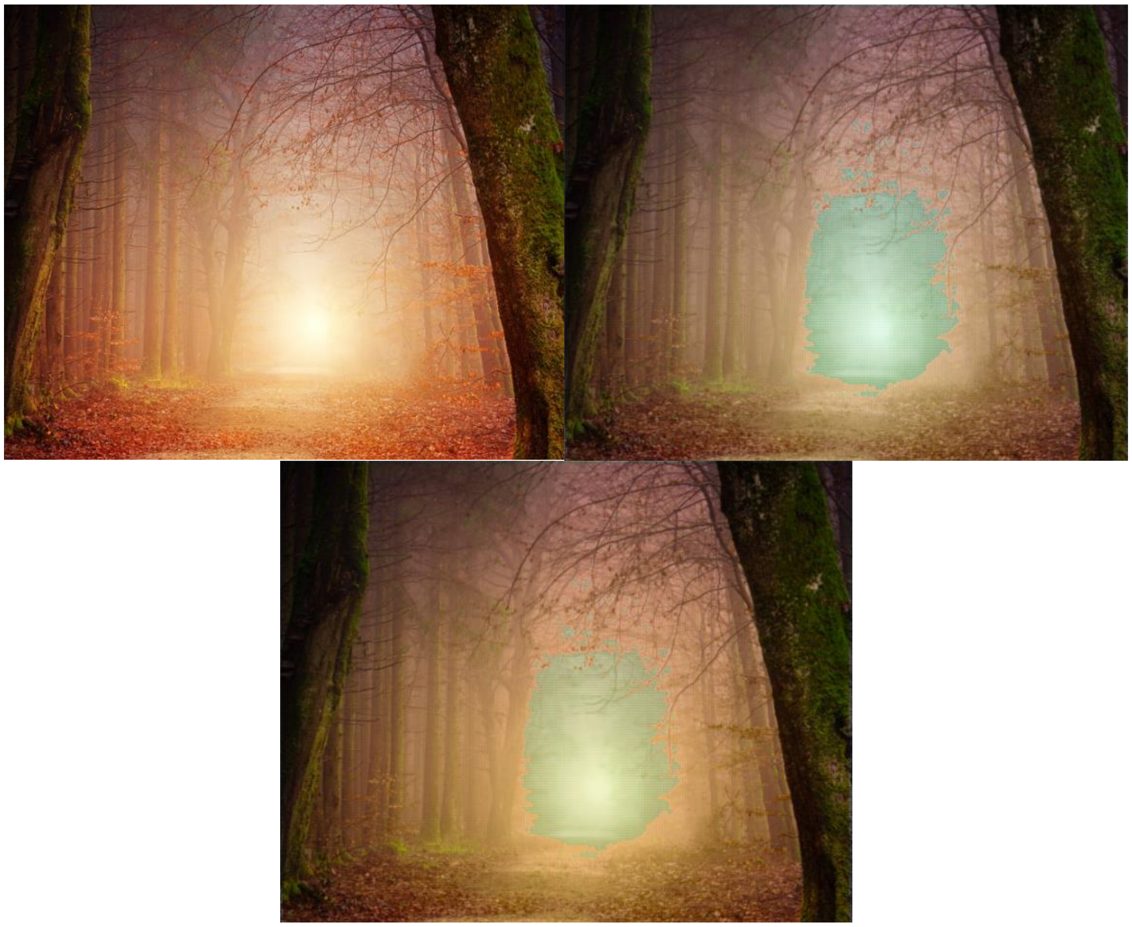
Here are the results of the algorithms and the root mean square error (RMSE) between the outputs and the MATLAB demosaic() outputs using the color-matched information patches.

1. **GBRG** Training Configuration: Patch Size = 5 Training Samples per Pattern = 5000


Top Left: Original Image. Top Right: Demosaic Image
Bottom: Demosaic + HE + WB

MATLAB RMSE: 3.603680334099297
DEMOSAIC RMSE: 7.211625858835398
DEMOSAIC + WB + HE RMSE: 8.544742133931061

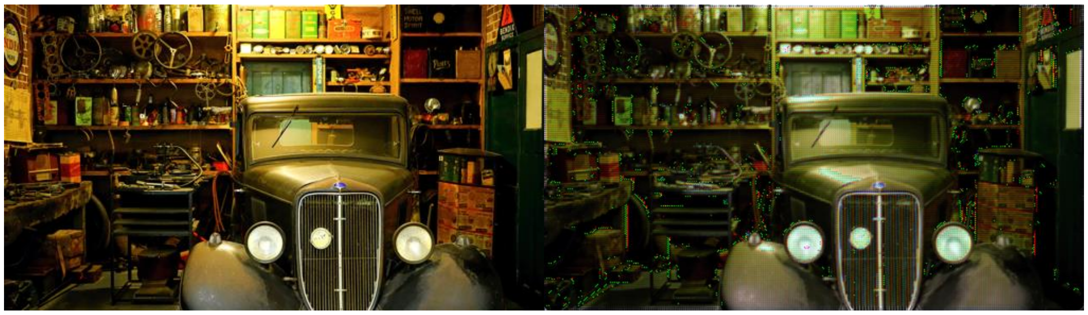
2. **GBRG** Training Configuration: Patch_Size = 7 Training Samples per pattern = 5000



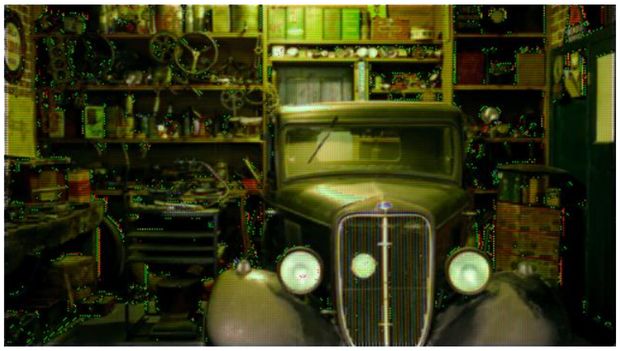
Top Left: Original Image. Top Right: Demosaic Image
Bottom: Demosaic + HE + WB

MATLAB RMSE: 3.603680334099297
DEMOSAIC RMSE: 7.191973064144019
DEMOSAIC + WB + HE RMSE: 8.540342120325283

3. **RGGB** Training Configuration: Patch Size = 7 Training Samples per pattern = 5000

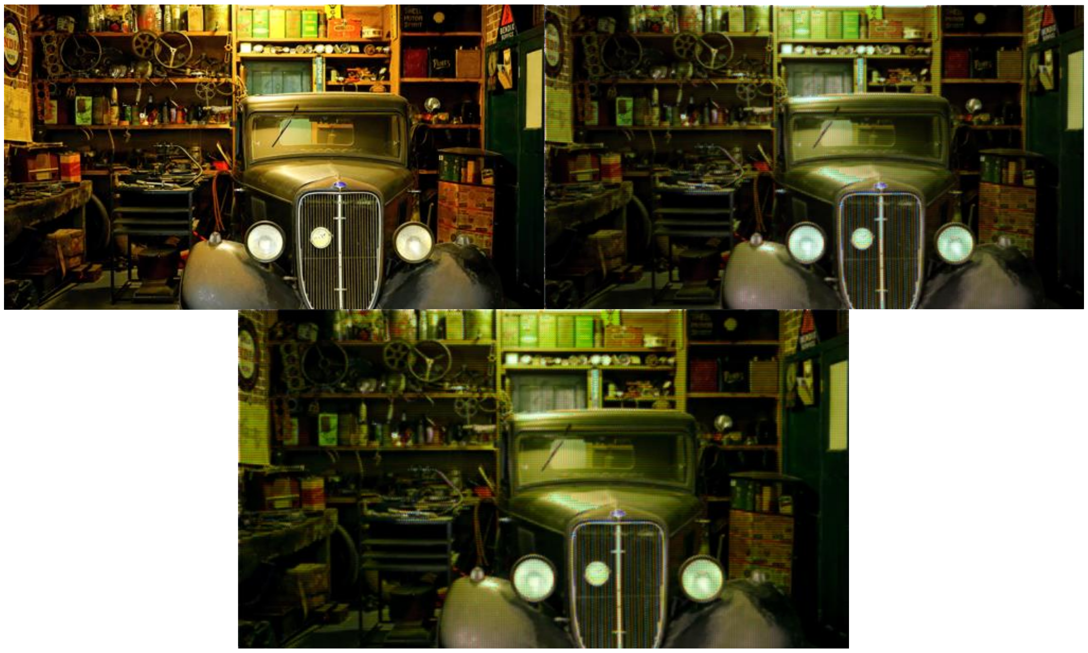


Left: Original Image. Right: Demosaic Image


Demosaic + HE + WB

MATLAB RMSE: 4.919107117452335
DEMOSAIC RMSE: 6.340722455395316
DEMOSAIC + WB + HE RMSE: 7.131068648455893

4. **RGGB** Training Configuration: Patch Size = 3 Training Samples per pattern = 5000


Top Left: Original Image. Top Right: Demosaic Image
Bottom: Demosaic + HE + WB

MATLAB RMSE: 4.919107117452335
DEMOSAIC RMSE: 6.308087007333066
DEMOSAIC + WB + HE RMSE: 7.100321004690151

4 DISCUSSION

According to the results, the color-matched information linear regression performs better overall than the full information linear regression. Additionally, the patch size shown clearly matters and depends on the image being processed.

For the bigger resolution image in tests 1 and 2 for both methods the error was smaller when using a bigger patch size. However, for the smaller resolution image in tests 3 and 4 both methods had a lower error for a smaller patch size. It can also be noted that changing the training sample size does alter the error slightly but has no big impact as there is only 5 training images.

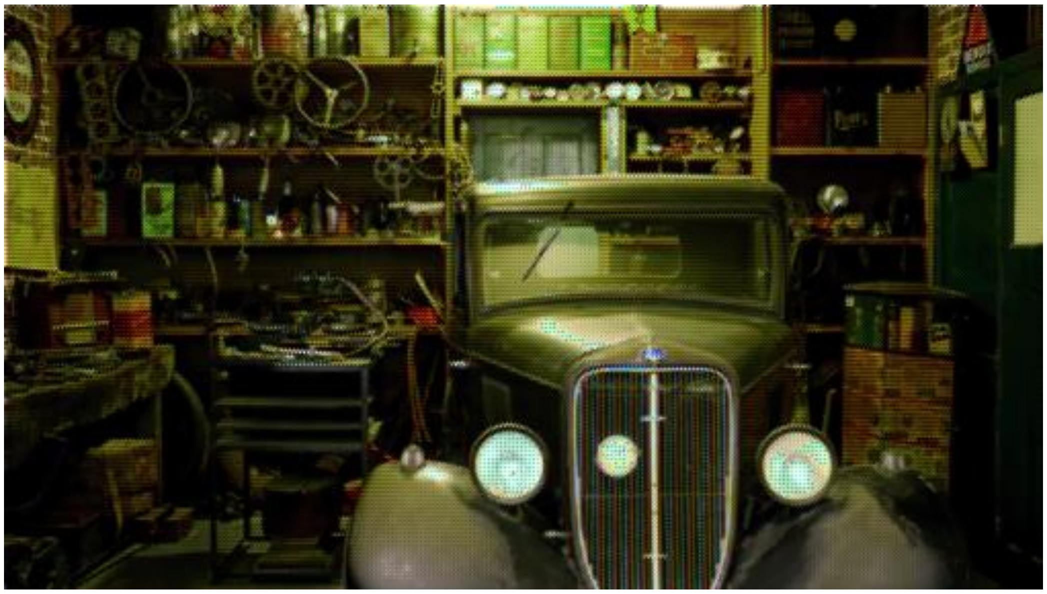
The pros of this current method is that it is easy to implement and can give a decent demosaiced result with some tuning. The cons of this method is that using the normal equations with 1000s of entries is computationally heavy. Another issue with this method is the limited amount of training images used, this limits the linear regression model from finding the appropriate weights that covers many pictures.

The color-matched information method was much more accurate in interpolating the colors than the full information method. These methods were also unable to produce a lower error than the demosaic function from MATLAB. The lowest error produced is ~1.28236 times the MATLAB error. This may due to the few training images used. However, on further research and adding about 15 training images, the results have not been improved to an extent that is better than the MATLAB demosaicing algorithm.

Moving to the white balancing and histogram equalization algorithms, both algorithms are very simplistic approaches and similar to the applications thought in class. In order to perform the white balancing, the BGR image was converted to L*a*b color space and the a and b values were scaled accordingly. Finally, here is the outputs from the demosaicing algorithm created.

Original Image:

DEMOSAIC:



Histogram Equalization + White Balancing on DEMOSAIC: