**Link for Solution Code:**
**https://github.com/uran001/Computer_Vision/blob/main/demosaic_try.m**

**Name: Uran Sabyr**
**StudentID: 20172008**

TASK 1-2:

**Mainly this task is about DEMOSAICING the given mosaic image.**

**I read the mosaic file with im2double format not with uint8 format because it may not fit into uint8 format. Then I divided into three channels**

---

```
I = im2double(imread('/home/uran001/Desktop/Academic 2021 Fall/Computer Vision/data/mosaic/crayons_mosaic.bmp'));
figure(1); clf; imshow(I(1:480,1:600));
%As it may not fit into uint8 format, so we use double format
input = im2double(I((1:480), (1:600)));
```

As it's rggb pattern, so for each channel I created new matrix with use of .*repmat()

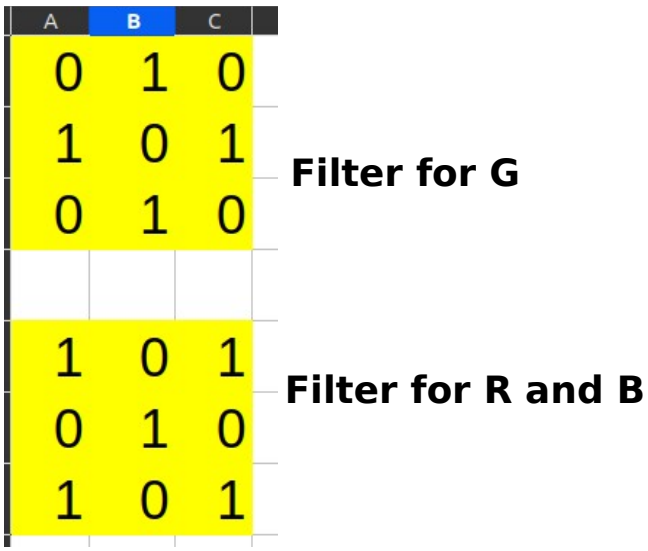
For Red: [1 0; 0 0] which means in 1,1 position is red

For Green: [0 1; 1 0] which means in 1,2 and 2 1 positions are Green

For Blue: [0 0; 0 1] which means in 2,2 position is blue

```
red = input.*repmat([1 0; 0 0], 240, 300);
green = input.*repmat([0 1; 1 0], 240, 300);
blue = input.*repmat([0 0; 0 1], 240, 300);
```

## TASK 2: Filtering each channels




```
g_Filter = [0 1 0; 1 0 1; 0 1 0]/4;
r_bFilter = [1 0 1; 0 0 0; 1 0 1]/4;

%Applying filter Red-to-Blue and Blue-to-Red values
red = red + imfilter(red,r_bFilter);
blue = blue + imfilter(blue,r_bFilter);
red = red + imfilter(red,g_Filter);
green = green + imfilter(green,g_Filter);
blue = blue + imfilter(blue,g_Filter);

J(:,:,1)= red;
J(:,:,2)= green;
J(:,:,3)= blue;
```

So, after applying, provided above filter, for each channel using builtin **imfilter** function we get filtered channels by averaging four neighbors or two.
As the green channel is the main channels, we don't apply R-B filter for G channels as provided in Szeliski book. However we apply the green filter for red and blue channel as well.

After applying the above filter I've got following result.

**TASK 3: Comparing the Original with Received Image**

**I have got about to similarly to the original image. But we need to compare it pixel by pixel.**

**Original:**

**Received:**

**So then I tried to see the difference between the Original and Received image after squaring their channels.**

```
Original= im2double(imread('/home/uran001/Desktop/Academic 2021 Fall/Computer Vision/data/mosaic/crayons.jpg' ));

R = Original(:,:, 1);
G = Original(:,:, 2);
B = Original(:,:, 3);

R = R.^2;
G = G.^2;
B = B.^2;

red = red.^2;
green = green.^2;
blue = blue.^2;

Diff(:,:,1) = R - red;
Diff(:,:,2) = G - green;
Diff(:,:,3) = B - blue;

figure(1); clf; imshow(Diff);
```

**Here I got the difference of Original and Received Images.**

**IMAGE ALIGNMENT**

**The main goal of this task 4 and 5 is convert 3 grays image into 1 colored(RGB) image. So my thought was brute-forcefully cut the images into 3 equal sizes. And combine them bottom to top. Then I realized that even though I divided them three equal sizes and combining them into one there is slice matching pixel error occurred.**

**Task 4:**

**Brute-force Method:**

```
imname = '/home/uran001/Desktop/Academic 2021 Fall/Computer Vision/Asignment 1/data/data/00125v.jpg';
img = (imread(imname));

[n,m]= size(img);
part = fix(n/3);
img_1=img(1:part,:);
img_2=img(part+1:2*part,:);
img_3=img(2*part+1:n-1,:);
% figure(i); clf; imshow(img_1);
img_3 = img_3 + img_2;
img_3 = img_3 + img_1;
figure(1); clf; imshow(img_3);
```

**Here what I got a result after combining:**

**It's kind of noisy and not properly filtered image.**

****** Algorithm ******

**That means we need to find match point of three images by shifting their matrix elements. For the first time, I went over the slides and didn't get much idea about the project. I've read the Szeliski_Book then I've got the idea that we need find such a shift of the elements of matrix such that their difference of their Square Sum is minimum. Having said that we will have much better matching three images. But we need to some constant matrix to refer to check are we need to shift or not. So as I've mentioned in the task 2, the Green matrix will be main matrix, and the Red and Blue matrix will be shifted according to the Green matrix. Also key point is we need to take some 50x50 shade from channels otherwise it becomes expensive.**

**Even though, it becomes expensive if the given image is large, as I'm searching for shifting point for a row and column in order to align three pictures more accurately. For a given 1024x400 it works fine.**

```
lcear all;
img = (imread('/home/uran001/Desktop/Academic 2021 Fall/Computer Vision/Asignment 1/data/data/00125v.jpg'));
```

## Task 4:

As all images have 1024xwidth sizes. I divided the heigh into three parts. Actually if to divide into three parts we have got 341, 341, 342. So which means we will have an issue while combining. I just reduced the last pixel to 1023.

```
[Height, Width] = size(img);
b = double(img(1:341,1:Width));
g = double(img(342:682,1:Width));
r = double(img(683:1023,1:Width));
```

## Task 5:

Here, I'm trying for 50x50 shade in about center of channels make some shifts in order to finder better alignment with less error rate which calculated Square sum of Green-to-Red and Green-to-Blue.

```
[shiftr_row, shiftr_col] = get(double(g(147:197,175:225)), double(r(147:197,175:225)));
[shiftb_row, shiftb_col] = get(double(g(147:197,175:225)), double(b(147:197,175:225)));
shiftr= uint8(circshift(r,[shiftr_row,shiftr_col]));
shiftb = uint8(circshift(b,[shiftb_row,shiftb_col]));
final = cat(3,shiftr,uint8(g),shiftb);
figure(4); clf; imshow(final);
```

This function finds some shift for row and column. I'm trying to find better match every time looping through -20:20, -20:20 points for row and column.

```
function [shift_row, shift_col] = get(main, temp)
        error = inf;
        for i = -20:20
            for j = -20:20
                shift_temp = circshift(temp,[i,j]);
                temp_error = sum(sum((double(main) - double(shift_temp)) .^ 2));
                if temp_error < error
                    error = temp_error;
                    shift_row = i;
                    shift_col = j;
                end
            end
        end
end
```

## Sample 1 (341x400):

**As shown in picture it has much better alignment comparing brute-force method shown above. However it has edge issues.**
**First idea was to crop the edges of the image.**
**Second find some filtering method to remove noises in the corner.**

```
targetsize = [300 350];
r = centerCropWindow2d(size(final), targetsize);
cropped = imcrop(final, r);
figure(4); clf;  imshow(cropped);
```

**Cropped image:**

**It looks pretty much better comparing to image with noisy image.**

**But, still doesn't look perfectly. So we can other filtering algorithms to make it much accurate.**

**Summarize the algorithm, I didn't use the "normxcorr2" because using over -20:20 was easier for me personally. However, for big scale images would be better to normxcorr2.**

**********

**For data_hires files which are big enough I've tried the same way which was done for above pictures. The only difference is the changing the bounds of the channels and shifting square**

**Here what I've got.**

```
clear all;
img = (imread('/home/uran001/Desktop/Academic 2021 Fall/Computer Vision/Asignment
1/data/data_hires/01047u.tif ));
[Height, width] = size(img);
b = double(img(1:3218,1:width));
g = double(img(3219:6436,1:width));
r = double(img(6437:9654,1:width));
part = 3218;
[shiftr_row, shiftr_col] = get(double(g(part / 2 - 225:part / 2 + 225,width/2-225: width / 2 +
225)), double(r(part / 2 - 225:part / 2 + 225,width/2-225: width / 2 + 225)));
[shiftb_row, shiftb_col] = get(double(g(part / 2 - 225:part / 2 + 225,width/2-225: width / 2 +
225)), double(b(part / 2 - 225:part / 2 + 225,width/2-225: width / 2 + 225)));
shiftr= uint16(circshift(r,[shiftr_row,shiftr_col]));
shiftb = uint16(circshift(b,[shiftb_row,shiftb_col]));
final = cat(3,shiftr,uint16(g),shiftb);
figure(3); clf; imshow(final);
function [shift_row, shift_col] = get(main, temp)
        error = inf;
        for i = -100:100
            for j = -100:100
                shift_temp = circshift(temp,[i,j]);
                temp_error = sum(sum((double(main) - double(shift_temp)) .^ 2));
                it temp_error < error
                    error = temp_error;
                    shift_row = i;
                    shift_col = j;
                end
            end
        end
end
```

**Received Image from 01047u.tif**

**[3218,3741]**

**The same problem with the noisy borders**

**Here the cropped image with size [2900,3300]**

```
targetsize = [2900 3300];

r = centerCropWindow2d(size(final), targetsize);

cropped = imcrop(final, r);
%figure(3); clf; imshow(final);
figure(4); clf; imshow(cropped);
```

Overall, I've tried with Gaussian or Laplacian method of rendering the Image pyramid and receive the original image. However I've got stuck there so then I tried my own method which take a little bit more time to execute the big file but works fine.