HOCHSCHULE OSNABRÜCK

UNIVERSITY OF APPLIED SCIENCES

Fakultät Ingenieurwissenschaften und Informatik

Studiengang Medieninformatik

Modul Audio- und Videotechnik

Praktikumsbericht

Meilenstein 02
Bildsignale – Digitalisierung

Wintersemester 2020/2021

Dozent:

Prof. Dr. Julius Schöning

Verfasser:

Patrick Felschen, Julian Voß

Matrikelnummer:

932056, 934505

Datum der Abgabe:

12.11.2020

I. Inhaltsverzeichnis

II. Abbildungsverzeichnis

1 Aufgabe: Von *.dng zu *.png

In diesem Abschnitt wird ein RAW-Image einer Canon Kamera ausgelesen und mittels MATLAB in ein farbiges PNG-Image konvertiert. [Anhang 1]

1.1 Implementierung

Zunächst wird die gegebene Datei „canon.dng“ aus einem Verzeichnis in MATLAB eingelesen. Unter anderem werden Meta-Daten, Offsets und das RAW-Image in Variablen gespeichert, um in den folgenden Prozessschritten verwendet zu werden.

1.2 Prozessschritte

Abbildung 1: RAW

Rohe Sensordaten mit Zahlenwerten (Integer) im Bereich zwischen Schwarz und Weiß des Sensors (Matrixgröße: m x n). [1]

1.2.1 Linearisierung

Abbildung 2: Linearizing

Da vom Kamerasensor nicht lineare Werte gespeichert werden, müssen diese in einen Wertebereich von 0 und 1 gebracht werden. In diesem Fall, geschieht dies über die Schwarz - und Weißstufen, welche sich in den Meta-Infos befinden (Matrixgröße: m x n). [1]

1.2.2 Weißabgleich

Abbildung 3: White Balancing

Anhand eines Referenzpunktes (ein Punkt für den die korrekte Farbe bekannt ist, meistens ein weißer oder grauer Pixel) und Kompensierung der Beleuchtung der Szene, werden die RGB Werte aller Pixel entsprechend angepasst (Matrixgröße: m x n). [1]

1.2.3 Demosaicing

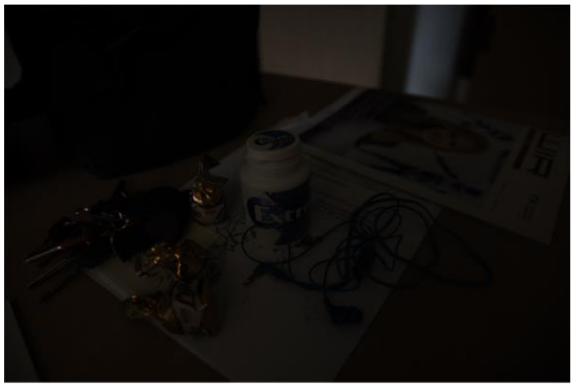

Abbildung 4: Demosaicing

Der Wert für einen Farbkanal ist bekannt, aus Farbwerten der Nachbarpixel lassen sich Werte der anderen 2 Farbkanäle berechnen. Ausgabe ist ein m x n x 3 Array mit RGB Werten für jeden Pixel. [1]

1.2.4 Farbraumanpassung

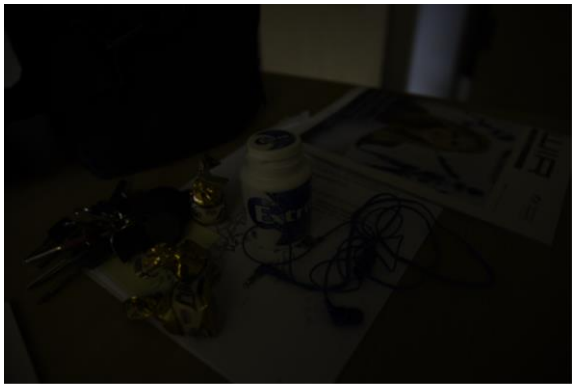

Abbildung 5: Color Space Conversion

Die Pixel des aktuellen RGB Bilds besitzen keine korrekten Koordinaten im RGB Farbraum, welcher der Monitor erwartet. Die Farbraumanpassung geschieht mit einer linearen Transformation. Auf jeden Pixel wird eine 3 x 3 Matrix Transformation angewendet. [1]

1.2.5 Helligkeitskorrektur

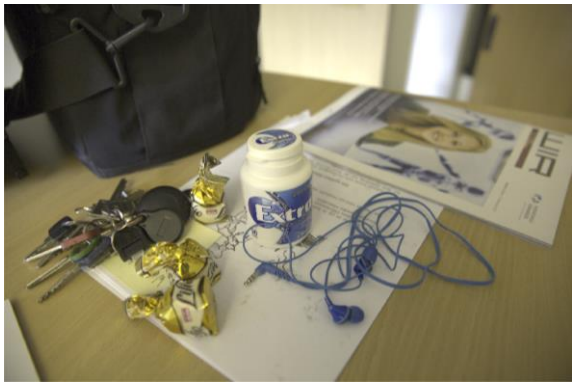

Abbildung 6: Brightness and Gamma Correction

Die Helligkeit wird korrigiert, indem die mittlere Luminanz[1] auf 1/4 des Maximums gesetzt wird. Anschließend wird das Bild mit 1/2.2 potenziert, welches einer Gammakorrnktur etspricht. Generell lässt sich für diesen Schritt keine eindeutige Lösung finden, das Bild wird hier nur "hübscher" gemacht (Matrixgröße: m x n x 3). [1]

2 Aufgabe: RAW ist nicht gleich RAW

Der folgende Abschnitt beschreibt das Verhalten des Nikon RAW-Images, welches zunächst wie im Abschnitt zuvor in MATLAB eingelesen wird. Danach werden Änderungen beschrieben, welche vorgenommen werden müssen, um das resultierende Bild richtig darstellen zu können. Abschließend werden auf die Unterschiede der Canon und Nikon Kamera eingegangen.

2.1 Beobachtung

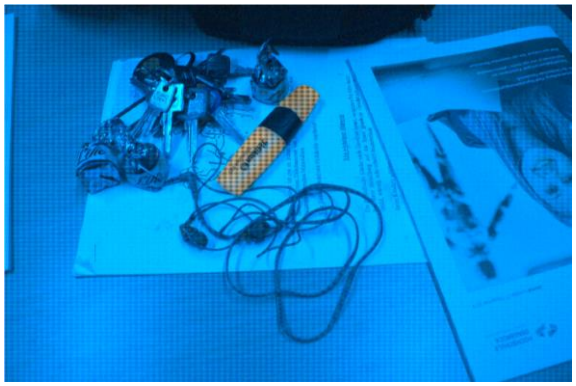

Abbildung 7: nikon.dng

Das gesamte Bild ist in Blau gefärbt, nur der Orangeton des Stifts ist erkennbar. Außerdem ist über das gesamte Bild ein Raster aus hellen und dunklen Kästchen zu sehen.

[1] Helligkeit der Bildpunkte

2.2 Anpassung des Skripts

Für die richtige Darstellung des Nikon Bildes, werden Änderungen im „Weißabgleich“, „Demosaicing“ und in der „Farbraumanpassung“ vorgenommen.

Da die Nikon Kamera eine andere Anordnung des Bayer CFA Layouts (s. 2.3), als die Canon Kamera besitzt, müssen die zwei String-Parameter „rggb“ im Weißabgleich und Demosaicing zu „gbrg“ geändert werden. Dadurch tritt das Raster (s. Abbildung 7) nicht mehr auf. [2]

Zudem wird die Farbraumanpassung nicht durchgeführt, da sonst das Bild einen Grünstich erhält. (s. Abbildung 8)

Erwähnenswert ist ebenso, dass im Schritt der Linearisierung eine „look-up table“ aus den Meta-Infos verwendet wird, um die Linearisierung durchzuführen. [1] [Anhang 2]

Abbildung 8: Nikon mit Farbraumanpassung

2.3 Unterschiede der Sensorchips

Der Sensorchip der Canon Kamera verwendet ein RGGB Layout, der Sensorchip der Nikon Kamera hingegen ein GBRG Layout. [2]

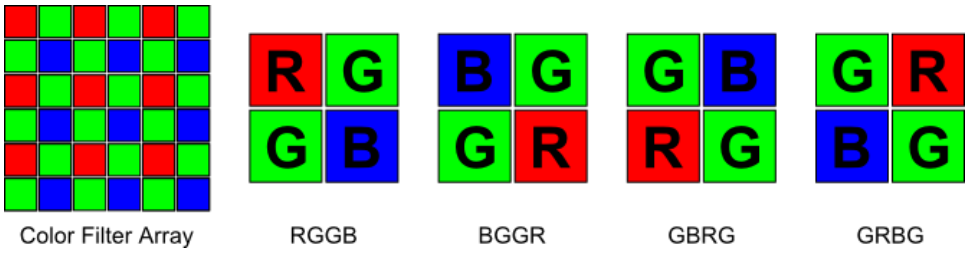

Abbildung 9: Bayer CFA Layouts [3]

3 Aufgabe: *.jpg aus der Kamera und *.png aus RAW

Auffallend ist, dass die Bilder, welche die Kamera erzeugt (s. Abbildung 11), weitaus sattere Farben besitzen. Die Bilder, die das Skript erzeugt (s. Abbildung 10), wirken dagegen sehr blass. Dies liegt daran, dass der Schritt der Farbraumanpassung anders verläuft und die Helligkeits- und Gammakorrektur mit anderen Werten berechnet wird. Weiteres zur Helligkeitsanpassung und Gammakorrektur in Aufgabe 4.

Abbildung 11: canon.jpg

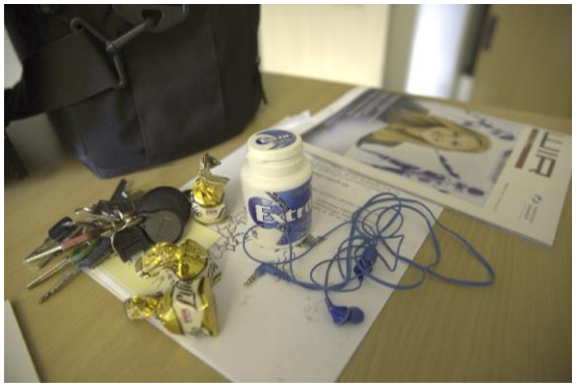

Abbildung 10: canon.png

4 Aufgabe: alternative Gamma Korrektur und Helligkeitsanpassung

Die Helligkeitsanpassung wurde hier mit 1/5 der mittleren Luminanz berechnet. Das Ergebnis wird nun mit 1/1.5 potenziert. Rechnet man in der Farbraumanpassung mit der Color-Matrix1, ist das vom Skript produzierte Bild sehr ähnlich zu dem Kamerabild. [Anhang 3]

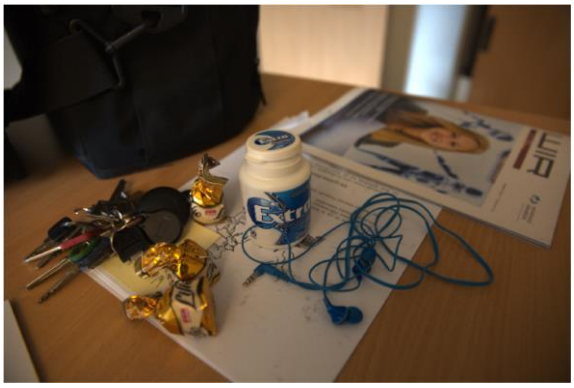

Abbildung 12: alternative Gamma und Helligkeits Korrektur

5 Aufgabe: Farbraumanpassung

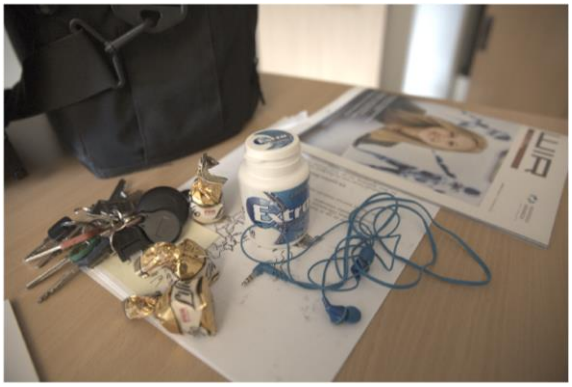

Abbildung 13: canon ohne Farbraumanpassung

Das Bild (s. Abbildung 13) sieht im Gegensatz zum Bild mit Farbraumanpassung (s. Abbildung 10) weniger gelblich aus.

Ohne die Farbraumanpassung, fehlen den Pixeln die Informationen für die richtige Farbdarstellung auf dem Monitor. [1]

IV. Literaturverzeichnis

[1] R. Summer, „Processing RAW Images in MATLAB,“ 19 05 2014. [Online]. Available: http://www.rcsumner.net/raw_guide/RAWguide.pdf. [Zugriff am 09 11 2020].

[2] K. Kindermann. [Online]. Available: http://www.ciando.com/img/books/extract/3772337937_lp.pdf. [Zugriff am 09 11 2020].

[3] „Pixinsight,“ [Online]. Available: https://pixinsight.com/doc/tools/Debayer/images/CFA.png. [Zugriff am 09 11 2020].

VI. Anhang

Anhang 1: Quellcode Aufgabe 1

```
%% Reading the CFA Image into MATLAB
filename = "./raw_dng/canon.dng"; % Put file name here
warning off MATLAB:tifflib:TIFFReadDirectory:libraryWarning
t = Tiff(filename,"r");
offsets = getTag(t,"SubIFD");
setSubDirectory(t,offsets(1));
raw = read(t); % Create variable 'raw', the Bayer CFA data
close(t);
meta_info = imfinfo(filename);
% Crop to only valid pixels
x_origin = meta_info.SubIFDs{1}.ActiveArea(2)+1; % +1 due to MATLAB indexing
width = meta_info.SubIFDs{1}.DefaultCropSize(1);
y_origin = meta_info.SubIFDs{1}.ActiveArea(1)+1;
height = meta_info.SubIFDs{1}.DefaultCropSize(2);
raw = double(raw(y_origin:y_origin+height-1,x_origin:x_origin+width-1));

%% Linearizing
if isfield(meta_info.SubIFDs{1},"LinearizationTable")
    ltab = meta_info.SubIFDs{1}.LinearizationTable;
    raw = ltab(raw+1);
end

black = meta_info.SubIFDs{1}.BlackLevel(1);
saturation = meta_info.SubIFDs{1}.WhiteLevel;
lin_bayer = (raw-black)/(saturation-black);
lin_bayer = max(0,min(lin_bayer,1));

%% White Balancing
wb_multipliers = (meta_info.AsShotNeutral).^-1;
wb_multipliers = wb_multipliers/wb_multipliers(2);
mask = wbmask(size(lin_bayer,1),size(lin_bayer,2),wb_multipliers,"rggb");
balanced_bayer = lin_bayer .* mask;

%% Demosaicing
temp = uint16(balanced_bayer/max(balanced_bayer(:))*2^16);
lin_rgb = double(demosaic(temp,"rggb"))/2^16;

%% Color Space Conversion
xyz2cam = reshape(meta_info.ColorMatrix2, 3, 3);
rgb2xyz = [0.4124564, 0.3575761,0.1804375; 0.2126729, 0.7151522, 0.0721750; 0.0193339, 0.1191920, 0.9503041];

rgb2cam = xyz2cam * rgb2xyz; % Assuming previously defined matrices
rgb2cam = rgb2cam ./ repmat(sum(rgb2cam,2),1,3); % Normalize rows to 1
cam2rgb = rgb2cam^-1;
lin_srgb = apply_cmatrix(lin_rgb, cam2rgb);
lin_srgb = max(0,min(lin_srgb,1)); % Always keep image clipped b/w 0-1

%% Brightness and Gamma Correction
grayim = rgb2gray(lin_srgb);
grayscale = 0.25/mean(grayim(:));
bright_srgb = min(1,lin_srgb*grayscale);

nl_srgb = bright_srgb.^(1/2.2);

%% Show Image
figure;
imshow(nl_srgb);


%% Functions
function colormask = wbmask(m,n,wbmults,align)
% COLORMASK = wbmask(M,N,WBMULTS,ALIGN)
%
% Makes a white-balance multiplicative mask for an image of size m-by-n
% with RGB while balance multipliers WBMULTS = [R_scale G_scale B_scale].
% ALIGN is string indicating Bayer arrangement: 'rggb','gbrg','grbg','bggr'
colormask = wbmults(2)*ones(m,n); %Initialize to all green values
switch align
case "rggb"
colormask(1:2:end,1:2:end) = wbmults(1); %r
colormask(2:2:end,2:2:end) = wbmults(3); %b
case "bggr"
colormask(2:2:end,2:2:end) = wbmults(1); %r
colormask(1:2:end,1:2:end) = wbmults(3); %b
case "grbg"
colormask(1:2:end,2:2:end) = wbmults(1); %r
colormask(1:2:end,2:2:end) = wbmults(3); %b
case "gbrg"
colormask(2:2:end,1:2:end) = wbmults(1); %r
colormask(1:2:end,2:2:end) = wbmults(3); %b
end
end

function corrected = apply_cmatrix(im,cmatrix)
% CORRECTED = apply_cmatrix(IM,CMATRIX)
%
% Applies CMATRIX to RGB input IM. Finds the appropriate weighting of the
% old color planes to form the new color planes, equivalent to but much
% more efficient than applying a matrix transformation to each pixel.
if size(im,3)~=3
error("Apply cmatrix to RGB image only.")
end
r = cmatrix(1,1)*im(:,:,1)+cmatrix(1,2)*im(:,:,2)+cmatrix(1,3)*im(:,:,3);
g = cmatrix(2,1)*im(:,:,1)+cmatrix(2,2)*im(:,:,2)+cmatrix(2,3)*im(:,:,3);
b = cmatrix(3,1)*im(:,:,1)+cmatrix(3,2)*im(:,:,2)+cmatrix(3,3)*im(:,:,3);
corrected = cat(3,r,g,b);
end
```

Anhang 2: Quellcode Aufgabe 2

```
%% Reading the CFA Image into MATLAB
filename = "./raw_dng/nikon.dng"; % Put file name here
warning off MATLAB:tifflib:TIFFReadDirectory:libraryWarning
t = Tiff(filename,"r");
offsets = getTag(t,"SubIFD");
setSubDirectory(t,offsets(1));
raw = read(t); % Create variable 'raw', the Bayer CFA data
close(t);
meta_info = imfinfo(filename);
% Crop to only valid pixels
x_origin = meta_info.SubIFDs{1}.ActiveArea(2)+1; % +1 due to MATLAB indexing
width = meta_info.SubIFDs{1}.DefaultCropSize(1);
y_origin = meta_info.SubIFDs{1}.ActiveArea(1)+1;
height = meta_info.SubIFDs{1}.DefaultCropSize(2);
raw = double(raw(y_origin:y_origin+height-1,x_origin:x_origin+width-1));

%% Linearizing
if isfield(meta_info.SubIFDs{1},"LinearizationTable")
    ltab = meta_info.SubIFDs{1}.LinearizationTable;
    raw = ltab(raw+1);
end

black = meta_info.SubIFDs{1}.BlackLevel(1);
saturation = meta_info.SubIFDs{1}.WhiteLevel;
lin_bayer = (raw-black)/(saturation-black);
lin_bayer = max(0,min(lin_bayer,1));

%% White Balancing
wb_multipliers = (meta_info.AsShotNeutral).^-1;
wb_multipliers = wb_multipliers/wb_multipliers(2);
mask = wbmask(size(lin_bayer,1),size(lin_bayer,2),wb_multipliers,"gbrg");
balanced_bayer = lin_bayer .* mask;

%% Demosaicing
temp = uint16(balanced_bayer/max(balanced_bayer(:))*2^16);
lin_rgb = double(demosaic(temp,"gbrg"))/2^16;

%% Color Space Conversion
xyz2cam = reshape(meta_info.ColorMatrix2, 3, 3);
rgb2xyz = [0.4124564, 0.3575761,0.1804375; 0.2126729, 0.7151522, 0.0721750; 0.0193339, 0.1191920, 0.9503041];

rgb2cam = xyz2cam * rgb2xyz; % Assuming previously defined matrices
rgb2cam = rgb2cam ./ repmat(sum(rgb2cam,2),1,3); % Normalize rows to 1
cam2rgb = rgb2cam^-1;
lin_srgb = apply_cmatrix(lin_rgb, cam2rgb);
lin_srgb = max(0,min(lin_srgb,1)); % Always keep image clipped b/w 0-1

%% Brightness and Gamma Correction
grayim = rgb2gray(lin_rgb);
grayscale = 0.25/mean(grayim(:));
bright_srgb = min(1,lin_rgb*grayscale);

nl_srgb = bright_srgb.^(1/2.2);

%% Show Image
figure;
imshow(nl_srgb);

%% Functions
function colormask = wbmask(m,n,wbmults,align)
% COLORMASK = wbmask(M,N,WBMULTS,ALIGN)
%
% Makes a white-balance multiplicative mask for an image of size m-by-n
% with RGB while balance multipliers WBMULTS = [R_scale G_scale B_scale].
% ALIGN is string indicating Bayer arrangement: 'rggb','gbrg','grbg','bggr'
colormask = wbmults(2)*ones(m,n); %Initialize to all green values
switch align
case "rggb"
colormask(1:2:end,1:2:end) = wbmults(1); %r
colormask(2:2:end,2:2:end) = wbmults(3); %b
case "bggr"
colormask(2:2:end,2:2:end) = wbmults(1); %r
colormask(1:2:end,1:2:end) = wbmults(3); %b
case "grbg"
colormask(1:2:end,2:2:end) = wbmults(1); %r
colormask(1:2:end,2:2:end) = wbmults(3); %b
case "gbrg"
colormask(2:2:end,1:2:end) = wbmults(1); %r
colormask(1:2:end,2:2:end) = wbmults(3); %b
end
end

function corrected = apply_cmatrix(im,cmatrix)
% CORRECTED = apply_cmatrix(IM,CMATRIX)
%
% Applies CMATRIX to RGB input IM. Finds the appropriate weighting of the
% old color planes to form the new color planes, equivalent to but much
% more efficient than applying a matrix transformation to each pixel.
if size(im,3)~=3
error("Apply cmatrix to RGB image only.")
end
r = cmatrix(1,1)*im(:,:,1)+cmatrix(1,2)*im(:,:,2)+cmatrix(1,3)*im(:,:,3);
g = cmatrix(2,1)*im(:,:,1)+cmatrix(2,2)*im(:,:,2)+cmatrix(2,3)*im(:,:,3);
b = cmatrix(3,1)*im(:,:,1)+cmatrix(3,2)*im(:,:,2)+cmatrix(3,3)*im(:,:,3);
corrected = cat(3,r,g,b);
end
```

Anhang 3: Quellcode Aufgabe 4

```
%% Brightness and Gamma Correction
grayim = rgb2gray(lin_rgb);
grayscale = 0.2/mean(grayim(:));
bright_srgb = min(1,lin_rgb*grayscale);

nl_srgb = bright_srgb.^(1/1.5);
```