Hochschule Konstanz
Technik, Wirtschaft und Gestaltung

Signale, Systeme und Sensoren

Kalibrierung von Digitalkameras

D. Wollmann, V. Bratulescu

Konstanz, 29. November 2020

Zusammenfassung (Abstract)

Thema:	Kalibrierung von Digitalkameras	
Autoren:	D. Wollmann	da161wol@htwg-konstanz.de
	V. Bratulescu	vl161bra@htwg-konstanz.de
Betreuer:	Prof. Dr. Matthias O. Franz	mfranz@htwg-konstanz.de
	Jürgen Keppler	juergen.keppler@htwg-konstanz.de
	Mert Zeybek	me431zey@htwg-konstanz.de

In diesem Versuch werden die Eigenschaften von digitalen Kameras untersucht. Dabei wird die Python Bibliothek OpenCV eingesetzt, um die Kamerasensoren zu kalibrieren. Ein Grauwertkeil wird mit einer digitalen Kamera aufgenommen und für jede Grauwertstufe den Mittelwert und die Standardabweichung berechnet. Als nächstes wird versucht die Auswirkungen des Dunkelstroms zu eliminieren. Dafür wird ein sogenanntes Dunkelbild erstellt, mit dem dann das Originalbild korrigiert wird. Außerdem gibt es noch das Problem, dass die Optik der Kamera die Helligkeit nicht gleichmäßig auf den Sensor überträgt und somit eine Vignettierung entsteht. Dafür wird ein Weißbild erstellt. Im letzten Teil der Aufgabe werden die Bilder auf funktionsuntüchtige Pixel, wie zum Beispiel hot, stuck und dead pixel untersucht.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

Kapitel 1

Einleitung

In dem zweiten Versuch der Versuchsreihe geht es um die Kalibrierung von Kameras. Dabei wird die Python Bibliothek OpenCV eingesetzt, um die Kamerasensoren zu kalibrieren.

Ein Grauwertkeil wird mit einer digitalen Kamera aufgenommen und für jede Grauwertstufe der Mittelwert und die Standardabweichung berechnet.

Als nächstes wird versucht die Auswirkungen des Dunkelstroms zu eliminieren. Dafür werden zehn Dunkelbilder aufgenommen und daraus der pixelweise Mittelwert berechnet, welcher dann vom Eingangsbild subtrahiert wird.

Außerdem gibt es noch das Problem, dass die Optik der Kamera die Helligkeit nicht gleichmäßig auf den Sensor überträgt und somit eine Vignettierung entsteht. Dafür werden zehn Weißbilder aufgenommen und ebenfalls der pixelweise Mittelwerte berechnet. Nach der Subtraktion des Dunkelbildes vom Weißbild wird dieses normalisiert, welches dann vom Eingangsbild dividiert werden kann.

Im letzten Versuch wird der Sensor auf funktionsuntüchtige Pixel, wie zum Beispiel hot, stuck und dead pixel untersucht. Dabei wird das Dunkel- und das Weißbild analysiert. Anschließend wird das korrigierte Eingangsbild mit dem originalen Eingangsbild verglichen.

Kapitel 2

Versuch 1: Aufnahme und Analyse eines Grauwertkeiles

2.1 Fragestellung, Messprinzip, Aufbau, Messmittel

Im ersten Versuch wird einen Grauwertkeil mithilfe einer Webcam aufgenommen. In Absprache mit Herrn Franz haben wir alle Bilder der Versuche mit einer digitalen Kamera aufgenommen. Dabei verwenden wir die Sony A6300. Der Grauwertkeil wurde an der Wand befestigt und senkrecht dazu die Kamera aufgestellt. Die Kamera wurde ebenfalls so ausgerichtet, dass die Grauwertstufen parallel zum Bildrand verlaufen. Der Lichteinfluss im Raum wurde so angepasst, dass kein Schatten auf das Bild einfällt.

Das aufgenommene Bild wird mit der OpenCV Bibliothek in Python eingelesen und in ein Grauwertbild umgewandelt. Dieses wird in die einzelnen Stufen eingeteilt und jeweils die Standardabweichung und der Mittelwert berechnet. Dabei sollen die Unterbilder möglichst viele Pixel der jeweiligen Stufe umfassen ohne die Stufenränder zu berühren.

Da unser aufgenommenes Bild eine Auflösung von 6000x4000 Pixel hat und mit der Methode cv2.imshow nicht angezeigt werden kann, haben wir uns dazu entschieden das Bild auf die Auflösung 600x400 Pixel zu skalieren, was 10% der originalen Größe entspricht. Die Skalierung wurde mithilfe von Python durchgeführt.

Abbildung 2.1: Versuchsaufbau

2.2 Messwerte

Die Aufnahme mit den Einstellungen aus der Tabelle 2.1 ergab folgendes Bild 2.2.

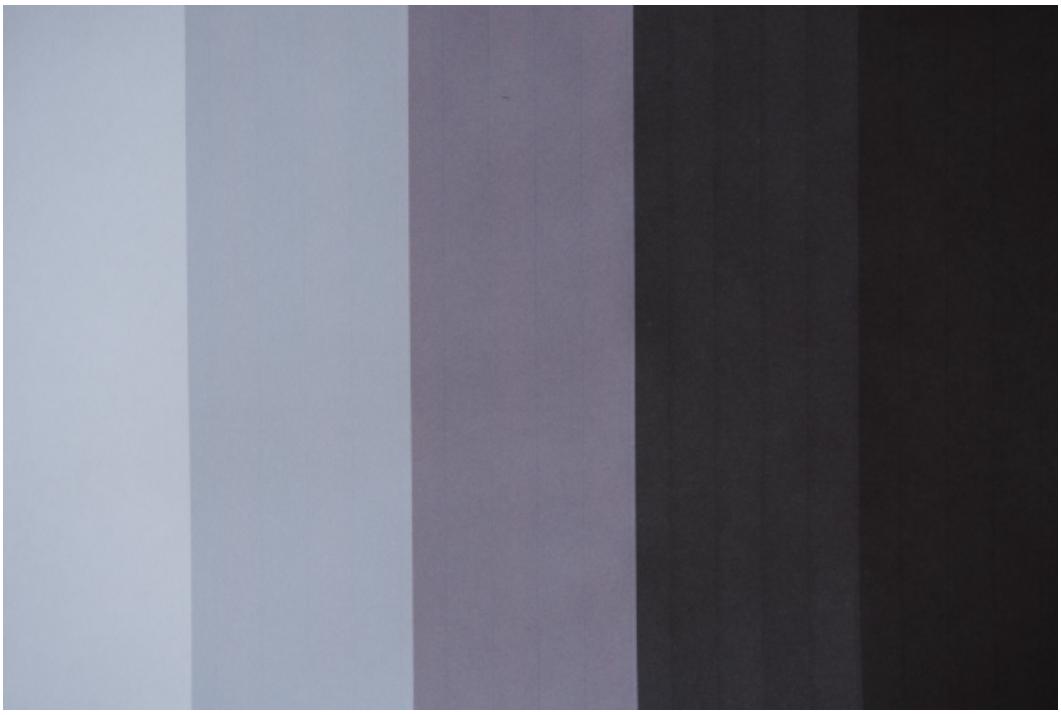


Abbildung 2.2: Der Grauwertkeil

Beschreibung	**Wert**
Frame Width	600
Frame Height	400
ISO	100
Blende	f3.2
Verschlusszeit	1/13
Manueller Fokus	Ja
Abstand von Objektiv zu Grauwertkeil	32.5cm

Tabelle 2.1: Einstellungen der Sony A6300

2.3 Auswertung

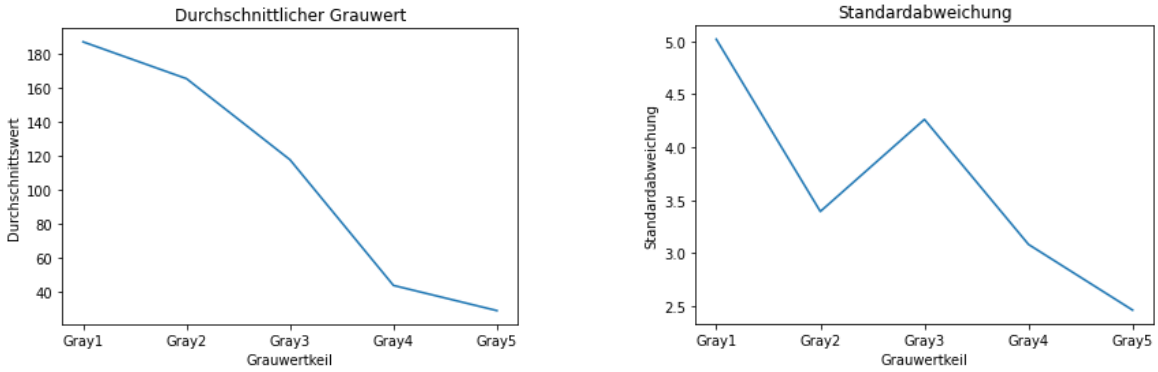
Das Originalbild wird mit der Funktion cv2.cvtcolor in ein Grauwertbild umgewandelt. Mithilfe von Indizierung und Index Slicing wurde das Grauwertbild in fünf Unterbilder unterteilt.

(a) Grauwert 1 (b) Grauwert 2 (c) Grauwert 3 (d) Grauwert 4 (e) Grauwert 5

(f) Grauwertkeil

Abbildung 2.3: Die einzelnen Grauwertsstufen und das zusätzliche Gesamtbild

Durch die Funktion cv2.meanStdDev wird die empirische Standardabweichung und der Mittelwert der einzelnen Unterbilder des Grauwertkeils berechnet. Die Ergebnisse der Standardabweichungen und Mittelwerte sehen wir in Abbildung 2.4.



(a) Mittelwert der Grauwerte (b) Standardabweichung der Grauwerte

Abbildung 2.4: Standardabweichung und Durchschnitt der jeweiligen Grauwerte

2.4 Interpretation

In Abbildung 2.3 ist zu sehen, dass die einzelnen Grauwertkeile unterschiedlich hoch sind, was darauf zurückzuführen ist, dass wir die Array Breite unterschiedlich gewählt haben, um die Stufenränder nicht zu berühren. Die große Standardabweichung bei Gray 1 in 2.4, könnte daran liegen, dass wir in diesem Bereich am wenigsten Messwerte genommen haben, was das Ergebnis verfälschen könnte.

Zusammenfassend kann man sagen, dass die einzelnen Grauwertstufen eine geringe Standardabweichung aufweisen, was bedeutet, dass fast alle Pixel den gleichen Wert haben.

Kapitel 3

Versuch 2: Aufnahme eines Dunkelbildes

3.1 Fragestellung, Messprinzip, Aufbau, Messmittel

Aufgrund von Fertigungstoleranzen und von spontan entstehenden Ladungsträgerpaaren durch Wärmezufuhr, entsteht ein sogennanter Dunkelstrom. Dieser und das thermische Rauschen der Ausleseelektronik führt dazu, dass jeder Pixel ein leicht unterschiedlicher Nullpunkt hat. Um diesen pixelweisen Offset zu eliminieren, wird in Versuch 2 ein Dunkelbild erstellt. Das Dunkelbild wird von einem Eingabebild subtrahiert und somit bekommt man ein korrigiertes Endbild. Das Dunkelbild haben wir aufgenommen, indem wir das Objektiv komplett abgedeckt haben. In diesem Versuch werden 10 Bilder aufgenommen und dabei pro Pixel ein Mittelwert berechnet. Wenn man alle berechneten Mittelwerte zu einem Bild zusammenfügt, hat man das sogenannte Dunkelbild, das vom Eingabebild subtrahiert werden kann.

3.2 Messwerte

In der nachfolgenden Abbildung 3.1 ist eines der aufgenommenen Bilder, wo das Objektiv verdeckt war, dargestellt.

Abbildung 3.1: Eins der zehn aufgenommenen Bilder

3.3 Auswertung

Um das Objektiv abzudecken, haben wir die standard mitgelieferte Schutzkappe verwendet. Dadurch konnten wir ein maximal dunkles Bild erzielen. Zu aller erst haben wir die 10 aufgenommenen Bilder eingelesen, in ein Grauwertbild umgewandelt und anschließend in ein float image konvertiert. Danach haben wir den pixelweisen Mittelwert berechnet und in einem Ausgabebild gespeichert, welches wir zuvor wieder zurück konvertiert haben als int image. Das Ausgabebild wurde dann mit der Funktion cv2.equalizeHist kontrastmaximiert und zum Schluss vom Grauwertkeilbild subtrahiert. Das konstrastmaximierte Bild ist in Abbildung 3.2 zu sehen.

Abbildung 3.2: Konstrastmaximiertes Dunkelbild

Der korrigierte Grauwertkeil ist in Abbildung 3.3 dargestellt.

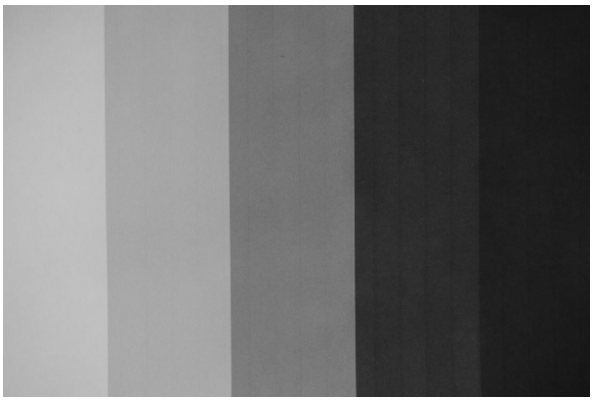


Abbildung 3.3: Korrigierter Grauwertkeil

3.4 Interpretation

Da unser korrigierter Grauwertkeil relativ ähnlich, wenn nicht sogar identisch zu unserem Ausgangsbild ist, können wir davon ausgehen, dass unsere verwendete Kamera, so gut wie keine Dunkelströme aufweist und die schwarzen Farben sehr gut aufnimmt. Des weiteren hatten auch unsere Pixel der aufgenommenen zehn Bilder, einen sehr nahen Wert an null, bzw. nur vereinzelte Pixel standen auf eins. Was dementsprechend ein Dunkelbild ergibt, welches kein starken Einfluss auf die Subtraktion des Eingangsbildes hat.

Kapitel 4

Versuch 3: Aufnahme eines Weißbildes

4.1 Fragestellung, Messprinzip, Aufbau, Messmittel

Im dritten Versuch handelt es sich um den Effekt der Vignettierung. Die ungleichmäßige Übertragung der Helligkeit auf den Sensor, führt zu einer Abdunkelung des Bildes zu den Rändern hin. Zur Kompensation wird deshalb ein Weißbild aufgenommen. Um dieses zu bekommen, wurden zehn Aufnahmen eines weißen Blatt Papiers gemacht. Der Abstand zur Kamera hat sich nicht geändert. Während der Aufnahme der Bilder wurde darauf geachtet, dass der Lichteinfluss auf das Blatt Papier gleichmäßig verteilt ist. Theoretisch bräuchte man für jede Fokuseinstellung ein eigenes Weißbild, dies soll in diesem Versuch aber nicht beachtet werden. Mithilfe eines Python Skripts wird der Mittelwert jedes einzelnen Pixels berechnet, um damit das thermische Rauschen eliminieren zu können. Von dem errechneten Mittelwertbild wird das Dunkelbild subtrahiert, welches kontrastmaximiert dargestellt werden soll. Das berechnete Bild soll normiert werden, sodass es den Mittelwert 1 erhält und wird anschließend vom korrigierten Eingangsbild dividiert.

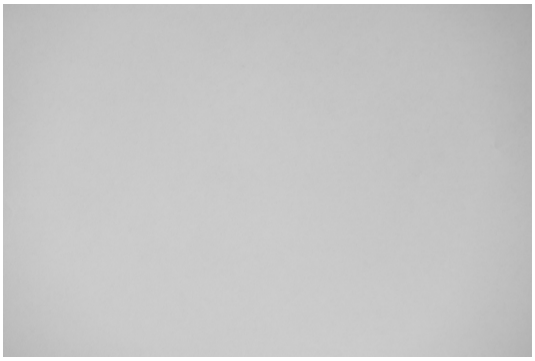
4.2 Messwerte

In der nachfolgenden Abbildung 4.1 ist eines der aufgenommenen Weißbilder dargestellt.

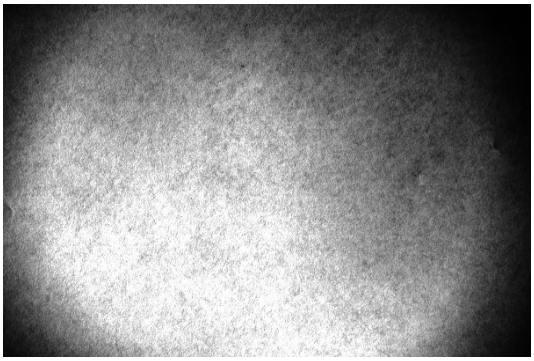
Abbildung 4.1: Eins der zehn aufgenommenen Bilder

4.3 Auswertung

Zuerst wurden die zehn Bilder eingelesen und als Grauwertbilder umgewandelt. Danach wurde der pixelweise Mittelwert berechnet und in einem neuen Bild gespeichert (a), wovon wir dann das kontrastmaximierte Dunkelbild subtrahiert haben. Anschließend wurde das Ergebnis kontrastmaximiert dargestellt. Dies ist in Abbildung 4.2 zu sehen.



(a) Pixelweiser Mittelwert der einzelnen Aufnahmen



(b) Konstrastmaximiertes Weißbild

Abbildung 4.2: Auswertung Weißbild

Als nächtes wird das Weißbild normalisiert. Dafür wurden alle Pixel mit dem Mittelwert des ganzen Bildes dividiert. Der Mittelwert des normalisierten Bildes beträgt in unserem Fall

1.0000002. Um das finale Bild aus Abbildung 4.3 zu erhalten, wird das korrigierte Eingangsbild mit dem normalisierten Weißbild dividiert.

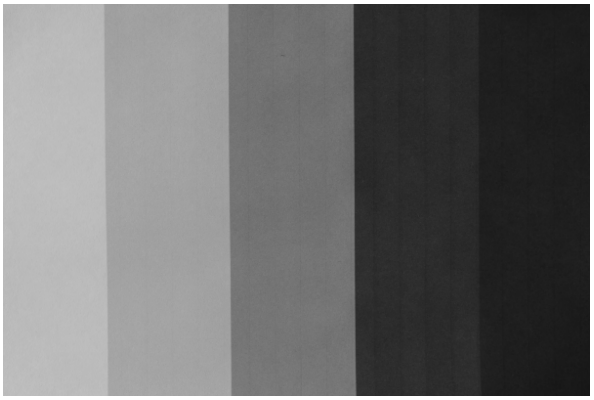


Abbildung 4.3: Das korrigierte Bild aus 3.3 dividiert mit dem normalisierten Weißbild

4.4 Interpretation

In dem kontrastmaximierten Weißbild ist die Vignettierung sehr deutlich zu sehen und man erkennt gut an welchen Stellen das Licht nicht gut auf den Sensor übertragen wird. Wie zu erwarten, befindet sich die Vignettierung an den Bildrändern. Bei unserem finalen Bild, erkennt man leider kein großen Unteschied zu dem korrigierten Eingangsbild. Deshalb haben wir die Bilder im Detail analysiert und festgestellt, dass ein Unterschied vorhanden ist, den man aber nicht mit bloßem Auge erkennen kann.

Betrachtet man die Arrays des korrigierten und des finalen Bildes im oberen linken Bereich, wo eine Vignettierung vorhanden ist, sieht man, dass diese im finalen Bild kompensiert wurde und hellere Pixel aufweist. Des Weiteren ist im Bereich der unteren Mitte, wo nur eine schwache Vignettierung vorhanden ist, zu erkennen, dass durch das normalisierte Weißbild die Pixel sich nur schwach verändern. Dieser Vergleich ist in Abbildung 4.4 dargestellt und Abbildung 4.5 verdeutlicht welche Bereiche wir verglichen haben.

	0	1	2	3	4	5	6	7
0	165	164	165	166	165	164	165	168
1	166	164	168	164	167	166	165	167
2	165	163	164	167	165	165	166	165
3	165	168	166	167	166	165	168	166
4	166	167	166	165	167	167	166	167
5	167	166	167	167	167	165	165	167
6	168	166	167	165	168	168	166	166
7	168	167	168	164	166	166	167	166
8	166	167	167	167	165	168	167	166
9	166	168	165	168	167	166	168	163
10	167	166	167	166	169	166	168	168
11	166	166	168	167	167	168	167	168
12	165	165	165	167	168	168	168	166
13	165	167	167	168	167	169	168	169
14	166	166	168	168	166	169	169	168
15	168	166	168	166	169	167	169	168
16	165	169	170	165	169	167	170	169
17	168	168	168	166	167	169	169	169
18	169	168	167	167	169	170	170	169

(a) Korrigiertes Eingangsbild links oben

	0	1	2	3	4	5	6	7
0	177.988	177.881	179.954	179.066	178.965	178.864	179.954	181.224
1	181.045	177.881	182.219	178.864	182.136	180.05	180.954	181.135
2	179.954	177.773	176.909	181.135	178.965	178.965	182.051	178.965
3	177.988	182.219	180.05	181.135	179.066	178.965	182.219	179.066
4	179.066	179.166	180.05	178.965	180.145	180.145	180.05	180.145
5	180.145	178.093	181.135	179.166	181.135	177.02	177.02	181.135
6	181.224	179.066	182.136	177.02	180.05	181.224	177.13	179.066
7	180.239	180.145	181.224	176.909	178.093	178.093	178.197	178.093
8	180.05	179.166	180.145	178.197	176.063	180.239	179.166	176.178
9	180.05	181.224	177.02	178.301	178.197	177.13	180.239	172.069
10	180.145	179.066	179.166	177.13	179.362	177.13	180.239	177.347
11	179.066	178.093	179.265	178.197	178.197	180.239	179.166	177.347
12	177.988	177.988	175.117	178.197	180.239	179.265	179.265	175.236
13	178.965	179.166	177.239	181.224	179.166	180.332	178.301	179.362
14	178.093	178.093	178.301	180.239	178.093	179.362	178.403	178.301
15	179.265	177.13	178.301	176.178	181.312	177.239	177.454	177.347
16	175.117	180.332	180.423	175.117	179.362	176.292	180.423	179.362
17	177.347	179.265	178.301	177.13	177.239	178.403	179.362	179.362
18	179.362	178.301	178.197	178.197	178.403	179.459	180.423	179.362

(b) Finales Bild aus 4.3 links oben

	300	301	302	303	304	305	306	307	308
387	117.088	116.293	118.637	119.61	118.637	114.185	123.499	120.959	118.056
388	118.056	117.665	119.991	121.926	121.554	112.25	122.294	121.926	118.056
389	120.959	118.637	121.554	120.959	119.61	113.628	119.991	121.926	117.48
390	119.023	117.665	121.554	122.294	119.991	119.023	121.179	119.023	118.056
391	122.527	118.443	122.527	124.829	117.48	120.202	120.582	120.369	120.369
392	119.61	121.926	124.472	122.527	120.959	116.293	122.527	124.22	121.926
393	117.48	119.023	118.637	119.991	121.554	116.692	122.894	119.61	122.527
394	117.088	123.499	118.637	119.61	118.637	116.12	122.156	124.111	120.202
395	119.61	120.959	117.088	120.582	117.088	115.153	123.499	123.133	123.499
396	119.991	117.665	118.056	120.959	120.959	119.61	122.894	123.499	120.582
397	121.554	119.023	119.023	120.959	119.023	116.12	127.042	123.133	120.959
398	120.582	119.023	119.61	118.056	119.023	118.056	122.156	121.554	121.926
399	118.247	121.926	122.527	121.926	119.991	116.692	124.472	122.527	121.179
400	121.179	121.926	122.527	122.527	118.056	118.247	122.527	124.472	121.554

(c) Korrigiertes Eingangsbild unten mitte

	300	301	302	303	304	305	306	307	308
387	121	119	122	123	122	118	127	125	122
388	122	121	124	126	125	116	127	126	122
389	125	122	125	125	123	118	124	126	122
390	123	121	125	127	124	123	124	123	122
391	126	123	126	129	122	123	124	125	125
392	123	126	128	126	125	119	126	129	126
393	122	123	122	124	125	120	127	123	126
394	121	127	122	123	122	120	125	127	123
395	123	125	121	124	121	119	127	126	127
396	124	121	122	125	125	123	127	127	124
397	125	123	123	125	123	120	130	126	125
398	124	123	123	122	123	122	125	125	126
399	121	126	126	126	124	120	128	126	124
400	124	126	126	126	122	121	126	128	125

(d) Finales Bild aus 4.3 unten mitte

Abbildung 4.4: Auszüge der Arrays aus dem Python Skript

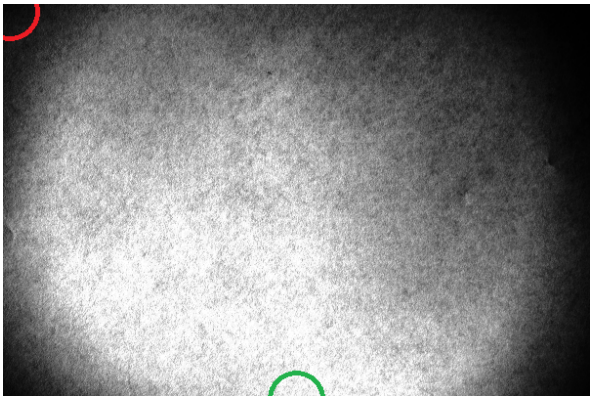


Abbildung 4.5: Analysierte Bereiche in einem roten und grünen Kreis dargestellt

Kapitel 5

Versuch 4: Pixelfehler

5.1 Fragestellung, Messprinzip, Aufbau, Messmittel

Im letzten Aufgabenteil wird das Dunkel- und Weißbild auf Funktionsuntüchtige Pixel untersucht. Es gibt verschiedene Arten von funktionsuntüchtigen Pixel, wie z.B. dead, hot und stuck pixel. Dead Pixel bleiben immer auf dem niedrigsten Wert stecken und lassen sich sehr gut auf einem Weißbild identifizieren. Stuck pixel hingegend bleiben immer auf dem Maximalwert und hot pixel gehen bei längerer Belichtungszeit in die Sättigung. Beide können sehr gut auf einem Dunkelbild erkannt werden. Am Schluss wird das finale Bild anhand des Mittelwertes und der Standardabweichung erneut, wie im ersten Versuch, ausgewertet.

5.2 Messwerte

In den nachfolgenden Abbildungen, ist die Untersuchung auf funktionsuntüchtige Pixel dargestellt.

Abbildung 5.1: Untersuchung Weißbild

Abbildung 5.2: Untersuchung Weißbild

5.3 Auswertung

In Abbildung 5.1 wird das Weißbild auf funktionsuntüchtige Pixel untersucht. Mithilfe unseres Python Skripts haben wir den höchsten und niedrigsten Werte mit verschiedenen Farben eingekreist. Der grüne Kreis stellt den höchsten Wert dar, während der rote den niedrigsten Wert umkreist.

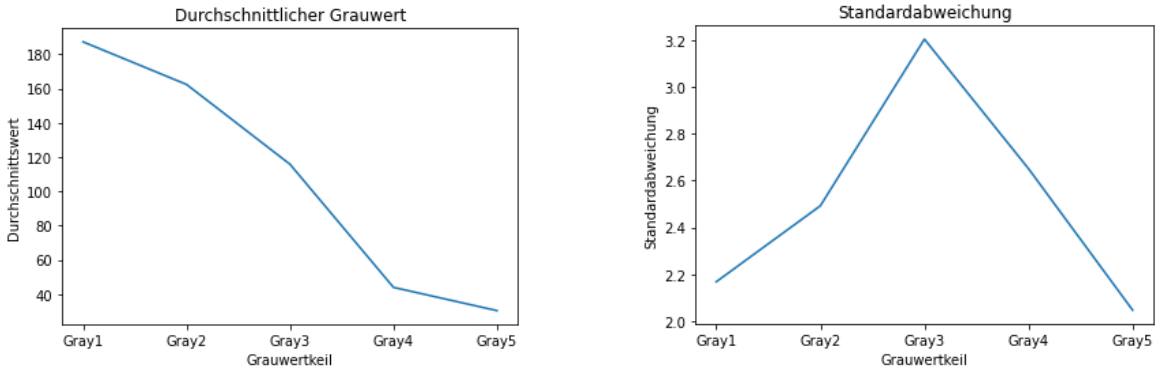
In Abbildung 5.2 wird das Dunkelbild auf funktionsuntüchtige Pixel untersucht, wobei hier der rote Kreis den grünen überdeckt. Das ist darauf zurückzuführen, dass das Dunkelbild ein Array aus Nullen ist und wir standardmäßig die Kreise auf die Koordinaten (0,0) gesetzt haben.

Die Werte dieser auffäligen Pixel werden in Tabelle 5.1 aufgelistet.

Beschreibung	**Wert**
Höchster Wert im Dunkelbild	0
Niedrigster Wert im Dunkelbild	0
Höchster Wert im Weißbild	206
Niedrigster Wert im Weißbild	166

Tabelle 5.1: Werte der eingekreisten Pixel

In Abbildung 5.3 wird das finale Bild 4.3 anhand des Mittelwerts und der Standardabweichung ausgewertet.



(a) Mittelwert der Grauwerte

(b) Standardabweichung der Grauwerte

Abbildung 5.3: Standardabweichung und Durchschnitt der jeweiligen Grauwerte

5.4 Interpretation

Da im Dunkelbild der höchste Pixel den Wert Null hat, ist daraus zu schließen, dass es sich hierbei nicht um hot oder stuck pixels handelt. Im Weißbild hat der niedrigste Pixel den Wert 166, was darauf hindeutet, dass es sich hier nicht um ein dead pixel handelt. Aus den zwei Erkenntnissen, schließen wir daraus, das unsere verwendete Kamera keine funktionsuntüchtigen Pixels besitzt.

Es ist zu sehen, dass die Standardabweichung generell etwas niedriger ausfällt, als im ersten Versuch. Dabei sticht deutlich der erste Grauwertkeil heraus, da die Standardabweichung von 5 auf 2.2 geschrumpft ist. Dass ist darauf zurückzuführen, dass es sich bei dem ersten Grauwertkeil um den weißen Teil handelt und die Vignettierung dabei aufgehoben wird.

Anhang

A.1 Quellcode

A.1.1 Quellcode Versuch 1

```
# −∗− coding: utf−8 −∗−

import cv2
import numpy as np
import matplotlib.pyplot as plt

path = r'C:\Users\danie\Desktop\Versuch2\Grauwertkeil\1.png'

img = cv2.imread(path)

#Bild von Originalgröße 6000x4000 runterskalieren damit es auf dem Monitor angezeigt werden kann
scale_percent = 10 # wird auf 10% der Originalgröße skaliert
width = int(img.shape[1] ∗ scale_percent / 100)
height = int(img.shape[0] ∗ scale_percent / 100)
dim = (width, height)
resized = cv2.resize(img, dim, interpolation = cv2.INTER_AREA)

#in Grauwertbild umwandeln
gray = cv2.cvtColor(resized, cv2.COLOR_BGR2GRAY)

#Das neue Bild ist jetzt 601x401
print('Resized Dimensions : ',resized.shape)

#Speichern des resized Grauwert und des resized Original Bildes
filename = 'Grauwertkeil.png'
cv2.imwrite(filename, gray)
filename = 'Original.png'
cv2.imwrite(filename, resized)
```

```
29
30 #Bereiche aus dem Gesamtbild extrahieren
31 cropGray1 = gray[0:400, 0:100]
32 cropGray2 = gray[0:400, 107:227]
33 cropGray3 = gray[0:400, 234:355]
34 cropGray4 = gray[0:400, 362:483]
35 cropGray5 = gray[0:400, 490:600]
36
37 #Einzelne Bereiche abspeichern
38 filename = 'cropGray1.png'
39 cv2.imwrite(filename, cropGray1)
40 filename = 'cropGray2.png'
41 cv2.imwrite(filename, cropGray2)
42 filename = 'cropGray3.png'
43 cv2.imwrite(filename, cropGray3)
44 filename = 'cropGray4.png'
45 cv2.imwrite(filename, cropGray4)
46 filename = 'cropGray5.png'
47 cv2.imwrite(filename, cropGray5)
48
49 arrayCropedImages = (cropGray1, cropGray2, cropGray3, cropGray4, cropGray5)
50 arrayMean = np.zeros(5)
51 arrayStd = np.zeros(5)
52
53 #Empirische Standardabweichung und Durchschnitt berechnen
54 for n in range(5):
55     mean, std = cv2.meanStdDev(arrayCropedImages[n])
56     arrayMean[n] = mean
57     arrayStd[n] = std
58     print(f"Gray {n + 1} mean: {mean}")
59     print(f"Gray {n + 1} std: {std}\n")
60
61
62 arrayPlot = ['Gray1', 'Gray2', 'Gray3', 'Gray4', 'Gray5']
63
64 plt.plot(arrayPlot, arrayMean)
65 plt.title('Durchschnittlicher Grauwert')
66 plt.xlabel('Grauwertkeil')
67 plt.ylabel('Durchschnittswert')
68 plt.show()
69
70 plt.plot(arrayPlot, arrayStd)
```

```
71 plt.title('Standardabweichung')
72 plt.xlabel('Grauwertkeil')
73 plt.ylabel('Standardabweichung')
74 plt.show()
```

Listing 6.1: Skript Versuch 1

A.1.2 Quellcode Versuch 2

```
# -*- coding: utf-8 -*-
"""
Created on Mon Nov 23 15:11:51 2020

@author: danie
"""
import cv2
import numpy as np

path = 'C:/Users/danie/Desktop/Versuch2/Schwarzbilder/'

arrayImages = [np.zeros((401,601)),np.zeros((401,601)),np.zeros((401,601)),np.zeros((401,601)),np.zeros((401,601)),np.zeros((

for n in range(10):
    pathToFile = path + str(n + 1) + '.png'
    img = cv2.imread(pathToFile)

    #Bild von Originalgröße 6000x4000 runterskalieren damit es auf dem Monitor angezeigt werden kann
    scale_percent = 10 # wird auf 10% der Originalgröße skaliert
    width = int(img.shape[1] * scale_percent / 100)
    height = int(img.shape[0] * scale_percent / 100)
    dim = (width, height)
    resized = cv2.resize(img, dim, interpolation = cv2.INTER_AREA)

    #eingelesenes Bild in Grauwertbild umwandeln
    gray = cv2.cvtColor(resized, cv2.COLOR_BGR2GRAY)

    #Von int Image in float Image convertieren
    info = np.iinfo(gray.dtype)
    arrayImages[n] = gray.astype(np.float64) / info.max

blackFloatImage = np.zeros((401,601))

for n in range(10):
    for i in range(401):
        for j in range(601):
            blackFloatImage[i,j] = blackFloatImage[i,j] + (arrayImages[n][i,j] / 10)

#Von float Image in int Image convertieren
blackFloatImage = blackFloatImage * 255
```

```
41 blackIntImage = blackFloatImage.astype(np.uint8)
42
43 maxContrastBlackIntImage = cv2.equalizeHist(blackIntImage)
44
45 #Speichern des Kontrastmaximiertes Dunkelbilds
46 cv2.imwrite('maxContrastBlackImage.png', maxContrastBlackIntImage)
47
48 #cv2.imshow('image',maxContrastBlackIntImage)
49 #cv2.waitKey(0)
50 #cv2.destroyAllWindows()
51
52 #Grauwertbild einlesen
53 grayIntImage = cv2.imread('Grauwertkeil.png')
54
55 #Convertieren in Graubild
56 grayIntImage = cv2.cvtColor(grayIntImage, cv2.COLOR_BGR2GRAY)
57
58 #Kontrastmaximiertes Dunkelbild von Grauwertbild subtrahieren
59 subGrayIntImage = grayIntImage - maxContrastBlackIntImage
60
61 #Speichern des Graubilds wo das Dunkelbild abgezogen wurde
62 cv2.imwrite('subGrayIntImage.png', subGrayIntImage)
```

Listing 6.2: Skript Versuch 2

A.1.3 Quellcode Versuch 3

```
# -*- coding: utf-8 -*-

import cv2
import numpy as np

path = 'C:/Users/danie/Desktop/Versuch2/Weissbilder/'

arrayImages = [np.zeros((401,601)),np.zeros((401,601)),np.zeros((401,601)),np.zeros((401,601)),np.zeros((401,601)),np.zeros((

#Einlesen der 10 Weissbilder
for n in range(10):
    pathToFile = path + str(n + 1) + '.png'
    img = cv2.imread(pathToFile)

    #Bild von Originalgröße 6000x4000 runterskalieren damit es auf dem Monitor angezeigt werden kann
    scale_percent = 10 # wird auf 10% der Originalgröße skaliert
    width = int(img.shape[1] * scale_percent / 100)
    height = int(img.shape[0] * scale_percent / 100)
    dim = (width, height)
    resized = cv2.resize(img, dim, interpolation = cv2.INTER_AREA)

    #eingelesenes Bild in Grauwertbild umwandeln
    gray = cv2.cvtColor(resized, cv2.COLOR_BGR2GRAY)

    arrayImages[n] = gray

whiteIntImage = np.zeros((401,601))

#Mittelwertbild berechnen
for n in range(10):
    for i in range(401):
        for j in range(601):
            whiteIntImage[i,j] = whiteIntImage[i,j] + (arrayImages[n][i,j] / 10)

#print('for loop done')

#Dunkelbild einlesen
blackIntImage = cv2.imread('C:/Users/danie/.spyder-py3/maxContrastBlackImage.png', cv2.IMREAD_GRAYSCALE)

#Mittelwertbild - Dunkelbild
```

```
whiteMinusBlackIntImage = np.zeros((401,601))
whiteMinusBlackIntImage = whiteIntImage - blackIntImage

#Weißbild konvertieren um es kontrastmaximiert dazustellen
convertedWhiteImage = whiteMinusBlackIntImage.astype(np.uint8)
maxContrastWhiteIntImage = cv2.equalizeHist(convertedWhiteImage)

#Speichern der Bilder
cv2.imwrite('maxContrastWhiteImage.png', maxContrastWhiteIntImage)
cv2.imwrite('whiteMinusBlackImage.png', convertedWhiteImage)
cv2.imwrite('meanWhiteImage.png', whiteIntImage)

#print('done')
```

Listing 6.3: Skript Versuch 3

```
# -*- coding: utf-8 -*-

import cv2
import numpy as np

#Einlesen des meanWhiteImages und des InputImages
#Das InputImage ist das Bild welche durch Grauwertkeil - Dunkelbild berechnet wurde
inputImage = cv2.imread('C:/Users/danie/.spyder-py3/subGrayIntImage.png', cv2.IMREAD_GRAYSCALE)
meanWhiteImage = np.float32(cv2.imread('C:/Users/danie/.spyder-py3/meanWhiteImage.png', cv2.IMREAD_GRAYSCALE)

#Bild normalisieren => Mittelwert 1
meanOfMeanWhiteImage = np.mean(meanWhiteImage)
normalizedImage = meanWhiteImage/meanOfMeanWhiteImage

print(np.mean(normalizedImage))

#Das durch Abzug des Dunkelbildes korrigierte Eingangsbild wird
#durch das normierte Weißbild dividiert
finalImage = inputImage / normalizedImage

#Resultat abspeichern
cv2.imwrite('finalImage.png', finalImage)
```

Listing 6.4: Skript Versuch 3b

A.1.4 Quellcode Versuch 4

```
# -*- coding: utf-8 -*-

import cv2

whiteImage = cv2.imread('C:/Users/danie/.spyder-py3/meanWhiteImage.png')
blackImage = cv2.imread('C:/Users/danie/.spyder-py3/maxContrastBlackImage.png')

whiteImageGray = cv2.imread('C:/Users/danie/.spyder-py3/meanWhiteImage.png', cv2.IMREAD_GRAYSCALE)
blackImageGray = cv2.imread('C:/Users/danie/.spyder-py3/maxContrastBlackImage.png', cv2.IMREAD_GRAYSCALE)

black_highestValue = 0
black_lowestValue = 255

black_circleX_1 = 0
black_circleY_1 = 0
black_circleX_2 = 0
black_circleY_2 = 0

#Dunkelbild auf Stuck und Hot Pixel untersuchen
for i in range(401):
    for j in range(601):
        currentPixel = blackImageGray[i,j]

        if(currentPixel > black_highestValue):
            black_highestValue = currentPixel
            black_circleX_1 = i
            black_circleY_1 = j

        if(currentPixel < black_lowestValue):
            black_lowestValue = currentPixel
            black_circleX_2 = i
            black_circleY_2 = j

print("Höchster Wert im Dunkelbild: " + str(black_highestValue))
print("Niedrigster Wert im Dunkelbild: " + str(black_lowestValue))

circleBlackImage = blackImage
cv2.circle(circleBlackImage, (black_circleX_1, black_circleY_1), 4, (0,255,0), 2)#grüner Kis
cv2.circle(circleBlackImage, (black_circleX_2, black_circleY_2), 4, (0,0,255), 2)#roter Kreis

```

```
cv2.imwrite('circleBlackImage.png', circleBlackImage)

#----------------------------------------------------------------------------------------

white_highestValue = 0
white_lowestValue = 255

white_circleX_1 = 0
white_circleY_1 = 0
white_circleX_2 = 0
white_circleY_2 = 0

#Weißbild auf Dead Pixel untersuchen
for i in range(401):
    for j in range(601):
        currentPixel = whiteImageGray[i,j]

        if(currentPixel > white_highestValue):
            white_highestValue = currentPixel
            white_circleX_1 = j
            white_circleY_1 = i

        if(currentPixel < white_lowestValue):
            white_lowestValue = currentPixel
            white_circleX_2 = j
            white_circleY_2 = i

print("Höchster Wert im Weißbild: " + str(white_highestValue))
print("Niedrigster Wert im Weißbild: " + str(white_lowestValue))
color = [0, 0, 255]
circleWhiteImage = whiteImage
cv2.circle(circleWhiteImage, (white_circleX_1, white_circleY_1), 4, (0,255,0), 2)#grüner Kreis
cv2.circle(circleWhiteImage, (white_circleX_2, white_circleY_2), 4, (0,0,255), 2)#roter Kreis

cv2.imwrite('circleWhiteImage.png', circleWhiteImage)
```

Listing 6.5: Skript Versuch 4

```
# -*- coding: utf-8 -*-

import cv2
import numpy as np
import matplotlib.pyplot as plt
```

```
6
7  img = cv2.imread('C:/Users/danie/.spyder-py3/finalImage.png', cv2.IMREAD_GRAYSCALE)
8
9  #Bereiche aus dem Gesamtbild extrahieren
10 cropGray1 = img[0:400, 0:100]
11 cropGray2 = img[0:400, 107:227]
12 cropGray3 = img[0:400, 234:355]
13 cropGray4 = img[0:400, 362:483]
14 cropGray5 = img[0:400, 490:600]
15
16 #Einzelne Bereiche abspeichern
17 filename = 'cropGray1_final.png'
18 cv2.imwrite(filename, cropGray1)
19 filename = 'cropGray2_final.png'
20 cv2.imwrite(filename, cropGray2)
21 filename = 'cropGray3_final.png'
22 cv2.imwrite(filename, cropGray3)
23 filename = 'cropGray4_final.png'
24 cv2.imwrite(filename, cropGray4)
25 filename = 'cropGray5_final.png'
26 cv2.imwrite(filename, cropGray5)
27
28 arrayCropedImages = (cropGray1, cropGray2, cropGray3, cropGray4, cropGray5)
29 arrayMean = np.zeros(5)
30 arrayStd = np.zeros(5)
31
32 #Empirische Standardabweichung und Durchschnitt berechnen
33 for n in range(5):
34     mean, std = cv2.meanStdDev(arrayCropedImages[n])
35     arrayMean[n] = mean
36     arrayStd[n] = std
37     print(f"Gray {n + 1} mean: {mean}")
38     print(f"Gray {n + 1} std: {std}\n")
39
40
41 arrayPlot = ['Gray1', 'Gray2', 'Gray3', 'Gray4', 'Gray5']
42
43 plt.plot(arrayPlot, arrayMean)
44 plt.title('Durchschnittlicher Grauwert')
45 plt.xlabel('Grauwertkeil')
46 plt.ylabel('Durchschnittswert')
47 plt.show()
```

```
48
49 plt.plot(arrayPlot, arrayStd)
50 plt.title('Standardabweichung')
51 plt.xlabel('Grauwertkeil')
52 plt.ylabel('Standardabweichung')
53 plt.show()
```

Listing 6.6: Skript Versuch 4b