

Prof. Dr. Matthias O. Franz

KALIBRIERUNG VON DIGITALKAMERAS

Versuch 2, Praktikum Technische Grundlagen der künstlichen Intelligenz

In diesem Versuch werden die Eigenschaften von digitalen Kameras untersucht. Wir führen dazu eine Kalibrierung des Kamerasensors durch, wie sie etwa bei industriellen Inspektionsanlagen, bei der Fernerkundung durch Satelliten oder in der Astronomie gemacht wird. Wie immer sind die Vorüberlegungen, Berechnungen und Ergebnisse zum nächsten Praktikumstermin in Form eines Jupyter-Notebooks zu präsentieren. Bitte alle Programme in das Notebook einbauen und kommentieren.

Lernziele:

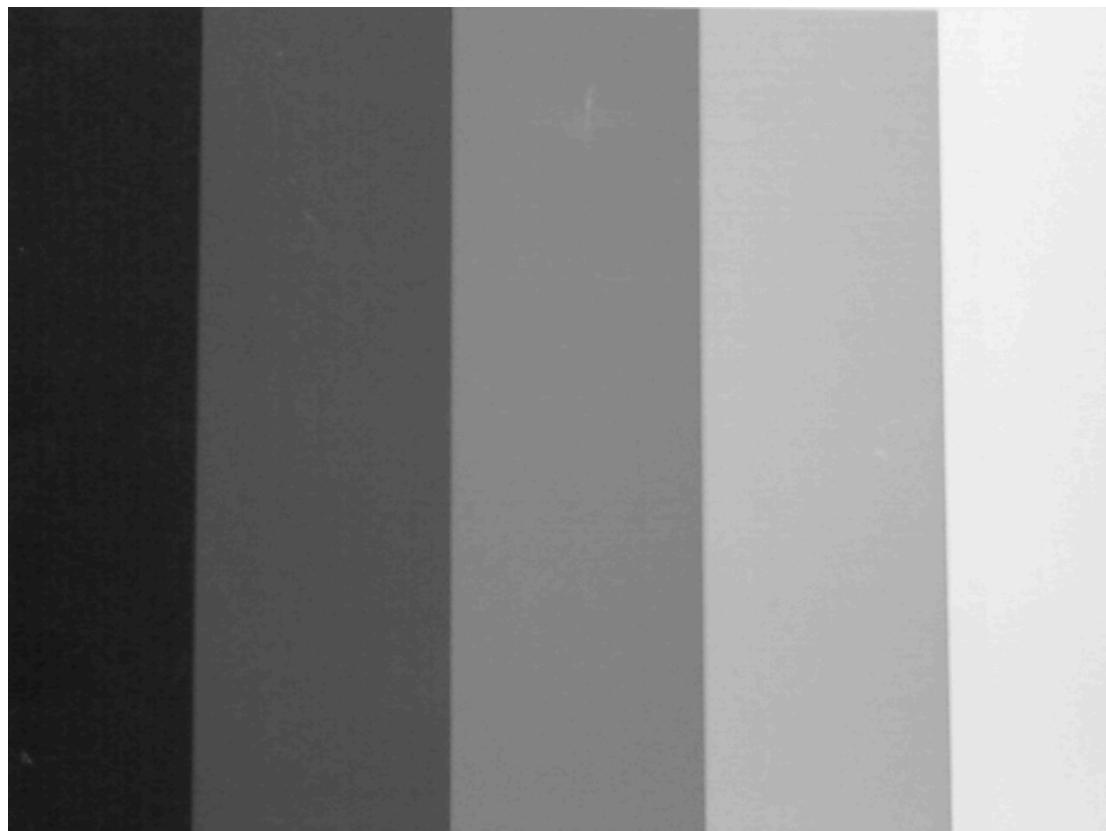
- Durchführung einfacher Bildverarbeitungsoperationen
- Praktische Erfahrung mit Kamerasensoren
- Verständnis der Fehlerquellen in Bildsensoren
- Kennenlernen einfacher Korrekturverfahren für Bildsensorfehler

1. Aufnahme und Analyse eines Grauwertkeiles

In diesem Versuch überprüfen wir, wie gut die Webcam am Arbeitsplatz einen stufenförmigen Grauwertverlauf aufnehmen kann. Da das aufgenommene Muster vorher bekannt ist, kann man anhand der Aufnahme die Wiedergabequalität messen. In unserem Fall sollte der Grauwert innerhalb jeder Stufe konstant sein. Die reale Aufnahme wird aber aufgrund von Bildfehlern und Sensorrauschen Abweichungen aufweisen.

- Nehmen Sie den Grauwertkeil (im Labor vorhanden) mit der Webcam an Ihrem Arbeitsplatz so auf, dass er das gesamte Bildfeld ausfüllt, den Helligkeitsbereich möglichst gut ausschöpft und die Grauwertstufen parallel zu den Bildrändern verlaufen. Stellen Sie sicher, dass kein Teil des Grauwertkeils in der Hell- oder Dunkelsättigung verschwindet. Nutzen Sie zu diesem Zweck das Python-Paket `OpenCV-Python`. Ein Kurzanleitung zur Installation, zum Einlesen von Bildern und Veränderung der Belichtungsparameter der Kamera finden Sie in Moodle. Notieren Sie, welche Belichtungsparameter Sie eingestellt haben, und die Entfernung zwischen Kamera und Testbild. Stellen Sie sicher, dass Sie die Belichtungsparameter im Verlauf der Versuche nicht mehr verändern. Speichern Sie das Bild für die Weiterverarbeitung und für das Notebook (Befehl: `cv2.imwrite()`) im verlustfreien Format `png` ab (das sonst übliche JPEG-Format ist verlustbehaftet).

Zu Finden In: `greyscale.png` (Ist bereits ein Graubild):



```
PS C:\Users\Nikita> pip install imageio
Collecting imageio
  Downloading imageio-2.37.2-py3-none-any.whl.metadata (9.7 kB)
Requirement already satisfied: numpy in c:\programmiersprachen\python\lib\site-packages (from imageio)
(2.2.6)
Requirement already satisfied: pillow>=8.3.2 in c:\programmiersprachen\python\lib\site-packages (from
imageio) (11.3.0)
  Downloading imageio-2.37.2-py3-none-any.whl (317 kB)
Installing collected packages: imageio
Successfully installed imageio-2.37.2

[notice] A new release of pip is available: 25.2 → 25.3
[notice] To update, run: python.exe -m pip install --upgrade pip
```

In [148...]

```
import sys
print(sys.executable)
```

c:\IDE\Anaconda\python.exe

In [148...]

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import imageio.v3 as iio
import cv2
```

- Schreiben Sie eine Routine, die das Bild einliest (Befehl: `pyopencv.imread()`) und die einzelnen Grauwertstufen als Unterbilder aus dem Originalbild ausliest. Schauen Sie sich dazu nochmals das Thema Indizierung und Index Slicing aus der Python-Einführung an.

Die Webcam liefert standardmäßig nur Farbbilder. Ein Farbbild können Sie mit der Funktion `cv2.cvtColor()` in ein Grauwertbild umwandeln. Anzeigen lässt sich das Bild bequem mit dem Befehl `cv2.imshow()`. Achten Sie darauf, dass die Unterbilder möglichst viele Pixel der jeweiligen Stufe umfassen, ohne die Stufenränder zu berühren.

In [148...]

```
def grayscaleZerstückeln(img):
    ausschnittBoundsArr=[
        (100, 400, 0, 100),
```

```

        (100, 400, 120, 250),
        (100, 400, 260, 390),
        (100, 400, 410, 530),
        (100, 400, 550, 620)
    ]
ausschnittBildArr = []
for bounds in ausschnittBoundsArr:
    ausschnitt= img[bounds[0]:bounds[1],bounds[2]:bounds[3]]
    #if(ausschnitt.max != 0):
        #ausschnittBildArr.append(ausschnitt/ausschnitt.max())
    #else:
        ausschnittBildArr.append(ausschnitt)
return ausschnittBildArr

```

In [149...]

```

def einBildEinlesen(dateiname, Farbe = 0, keinUrBild =0, asSubplot = 1, keinTeilBild = 0):
    img = iio.imread(dateiname)
    img = img.astype('float64')
    # Nur umwandeln, wenn Bild tatsächlich 3 Kanäle hat (Farbbild ist)
    if img.ndim == 3 and img.shape[2] == 3:
        img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    else:
        img = img
    if(Farbe== 1 and  keinUrBild ==0):
        plt.figure()
        #print("Eingelesenes Bild:\n ",img)
        plt.title(f"{dateiname} als Farbbild")
        plt.imshow(img/255) #255 = Maxwert in 8 Bit Bildern scaliert Farben
        #plt.imshow(img/img.max())#/Maximaler Grauwert scaliert Farben

    elif(keinUrBild ==0):
        plt.figure()
        #print("Eingelesenes Bild:\n ",img)
        plt.title(f"{dateiname} als Graubild")
        #plt.imshow(img/255) #255 = Maxwert in 8 Bit Bildern scaliert Farben
        #plt.imshow(img/img.max())#/Maximaler Grauwert scaliert Farben
        plt.imshow(img, cmap='gray')

    if("scale" in dateiname and garnichtTeilen==0):
        ausschnittBildArr = grayscaleZerstückeln(img)

    if(asSubplot==1 and keinTeilBild == 0):
        fig, axes = plt.subplots(nrows=1, ncols=len(ausschnittBildArr), figsize=(10, 5))
        #fig → die gesamte Figure (das komplette Bild)
        #axes → die einzelnen Zeichenbereiche (Plot-Felder), also Unterplots

        for colums,ausschnittBild in zip(axes,ausschnittBildArr):
            colums.imshow(ausschnittBild,cmap='gray')
            #colums.axis('off')
        fig.suptitle(f"{dateiname} als Graubild nach Graustufe aufgeteilt", fontweight='bold')
        plt.show()
        return ausschnittBildArr

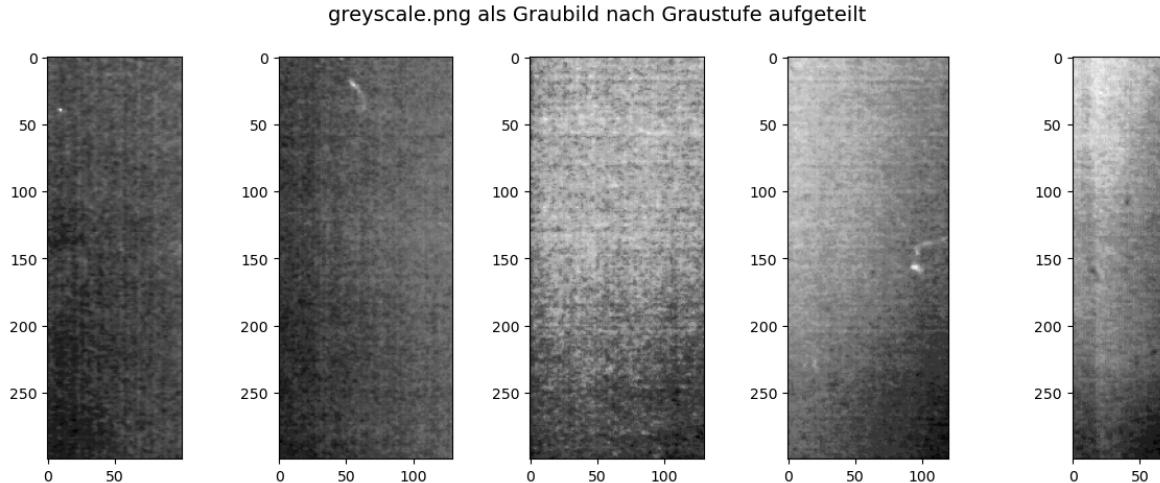
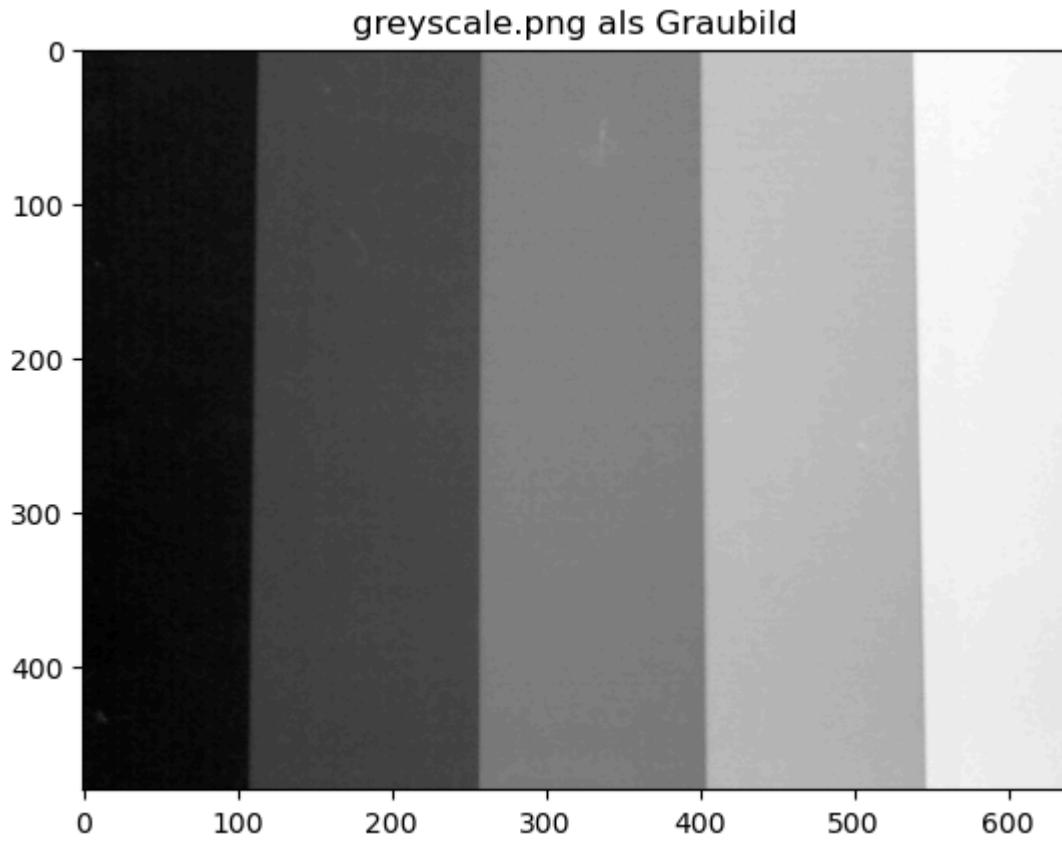
    elif(keinTeilBild == 0):
        i=0
        for ausschnittBild in ausschnittBildArr:
            i=i+1
            plt.figure()
            plt.imshow(ausschnittBild,cmap='gray')
            plt.title(f"{dateiname} als Graubild Teil{i}")
        return ausschnittBildArr

```

```
    img = np.array(img)
    return img
```

In [149...]

```
ausschnittBildArr = einBildEinlesen(r"greyscale.png", Farbe =0, keinUrBild=0, asSub
```



Plott

In [149...]

```
def plottstart():
    plt.figure(figsize=(8,4))           #neuer Plot

def plottend(ueberschrift, xlabel, ylabel):
    # doppelte Label vermeiden-----
    # plt.getCurrentAxes()
    # Gibt die aktuelle Achse (das aktuelle Diagramm) zurück, auf der man zei
    # Rückgabe Objekt vom Typ matplotlib.axes._axes.Axes
    # .get_legend_handles_labels() liefert zwei Listen zurück: handles, labels
    handles, labels = plt.gca().get_legend_handles_labels()
    # zip(labels, handles) verbindet beide Listen paarweise:
```

```

#   [("Messpunkte", obj1), ("Messpunkte", obj2), ("Kennlinie", obj3)]
# durch dict() wird das zu einem Dictionary umgewandelt:
# doppelte Schlüssel („Messpunkte“) werden automatisch überschrieben,
# es bleibt nur ein Eintrag pro Label übrig
by_label = dict(zip(labels, handles))
# plt.legend(by_label.values(), by_label.keys()) Zeigt jetzt die Legende oh
#.values = Plot-Objekte
#.keys = Textbeschriftung
plt.legend(by_label.values(), by_label.keys())
#-----
plt.title(f"{{ueberschrift}}")
#Achsenbeschriftung
plt.xlabel(f"{{xlabel}}")
plt.ylabel(f"{{ylabel}}")
plt.grid(True)
plt.legend(loc='upper right') #Legende oben rechts
#plt.tight_layout() passt automatisch die Abstände zwischen Plot und Rand s
#dass alles schön sichtbar bleibt
plt.tight_layout()
plt.show()

```

- Bestimmen Sie für jede Stufe den Mittelwert und die Standardabweichung der Grauwerte und tragen Sie diese in eine Tabelle ein. Die zugehörigen Befehle finden sich ebenfalls in der Python-Einführung.

Ein wichtiges Detail: wir bestimmen hier nicht die Standardabweichung des Mittelwertes, sondern nur die normale Standardabweichung, denn uns interessiert hier nicht das Vertrauensintervall des Mittelwertes, sondern die Streuung der Messwerte. Damit können wir in der letzten Aufgabe die Qualität unserer Kalibrierung überprüfen.

```

In [149...]: def plottMittStd(pixelMeanArr, pixelStdArr, erhaltenesPixelArr, sortet =0):
    farbArr=["orange", "pink", "purple", "blue", "cyan"]
    if(sortet==1):
        ausschnittPixelArr =erhaltenesPixelArr
        for ausschnitt in range(len(ausschnittPixelArr)):
            plottstart()
            plt.hist(ausschnittPixelArr[ausschnitt], bins=50, color=farbArr[auss
                plt.axvline(pixelMeanArr[ausschnitt], color="red", linestyle="--", l
                plt.axvline(pixelStdArr[ausschnitt], color="black", linestyle="-", l
                plt.axvline(pixelMeanArr[ausschnitt]+pixelStdArr[ausschnitt], color=
                plt.axvline(pixelMeanArr[ausschnitt]-pixelStdArr[ausschnitt], color=
            plottend(f"Helligkeitsverteilung im Ausschnitt {ausschnitt}", "Helli
    else:
        unsortedPixelArr= erhaltenesPixelArr
        mean = pixelMeanArr
        std = pixelStdArr
        plottstart()
        plt.hist(unsortedPixelArr, bins=100, color=farbArr[0], edgecolor="black"
            plt.axvline(mean, color="red", linestyle="--", label=f"Mean: {mean:.3f}")
            plt.axvline(std, color="black", linestyle="-", label=f"Std: {std:.3f}")
            plt.axvline(mean+std, color="green", linestyle="--", label=f"Mean + Std:
            plt.axvline(mean-std, color="green", linestyle="--", label=f"Mean - Std:
        plottend(f"Helligkeitsverteilung Gesamtstreuung", "Helligkeit (0-250)", "

```

```
In [149...]: def proStufeMittStd(ausschnittBildArr ,plotEachAusschnitt=0, plotGesamtStreuung
```

```

pixelMeanArray = []
pixelStdArray = []
ausschnittPixelArr = []      #helligkeiten der Pixel
unsortedPixelArr=[]          #helligkeiten der Pixel
for ausschnittNr in range(len(ausschnittBildArr)):
    # Pixel-Liste pro Ausschnitt anlegen
    ausschnittPixelArr.append([])
    # Pixelwerte sammeln
    for zeile in range(len(ausschnittBildArr[ausschnittNr])):
        for spalte in range(len(ausschnittBildArr[ausschnittNr][zeile])):
            ausschnittPixelArr[ausschnittNr].append(ausschnittBildArr[ausschnittNr][zeile][spalte])
    #MEAN pro pixel
    pixelMeanArray.append(np.mean(ausschnittPixelArr[ausschnittNr]))
    #STD pro pixel
    pixelStdArray.append(np.std(ausschnittPixelArr[ausschnittNr], ddof=1))

gesamtMean = np.mean(unsortedPixelArr)
gesamtStd = np.std(unsortedPixelArr, ddof=1)

if(plotEachAusschnitt == 1):
    plottMittStd(pixelMeanArray, pixelStdArray, ausschnittPixelArr, sortet=1)
if(plotGesamtStreuung == 1):
    plottMittStd(gesamtMean, gesamtStd, unsortedPixelArr, sortet=0)

return gesamtMean, gesamtStd, pixelMeanArray, pixelStdArray, ausschnittPixelArr

```

In [149...]

```

def tabelleGraustufen(pixelMeanArr, pixelStdArr, save_csv=False, csv_name="tabelle.csv"):
    from enum import Enum

    farb_text = ["schwarz", "dunkelgrau", "grau", "hellgrau", "weiss"]

    anzahl = len(pixelMeanArr)

    tabelle = pd.DataFrame({
        "Grauwert-Stufe": farb_text,
        "Mittelwert": np.round(pixelMeanArr, 3),
        "Standardabweichung": np.round(pixelStdArr, 3)
    })

    print("\n===== Tabelle Graustufen =====")
    print(tabelle.to_string(index=False))
    print("=====\\n")

    if save_csv:
        tabelle.to_csv(csv_name, index=False)
        print(f">>> Tabelle gespeichert als: {csv_name}")

    return tabelle

```

In [149...]

```

ausschnittBildArr = einBildEinlesen(r"greyscale.png", Farbe =0, keinUrBild=1, asSubImage=True)
gesamtMean, gesamtStd, pixelMeanArray, pixelStdArray, erhaltenesPixelArr = proStufentabellenGraustufen(ausschnittBildArr)
tabelle = tabelleGraustufen(pixelMeanArray, pixelStdArray, save_csv=True, csv_name="tabelle.csv")

```

```

-----
```

```

TypeError                                     Traceback (most recent call last)
Cell In[1496], line 2
    1 ausschnittBildArr = einBildEinlesen(r"greyscale.png", Farbe =0,keinUrBild=
1,asSubplot=1,keinTeilBild = 1)
----> 2 gesamtMean, gesamtStd,pixelMeanArray, pixelStdArray,erhaltenesPixelArr =
proStufeMittStd(ausschnittBildArr, plotEachAusschnitt =0, plotGesamtStreuung= 1)
    3 tabelle = tabelleGraustufen(pixelMeanArray, pixelStdArray,save_csv=True,
csv_name="graustufenTabelle_A1.csv")

Cell In[1494], line 12, in proStufeMittStd(ausschnittBildArr, plotEachAusschnitt,
plotGesamtStreuung)
    10 # Pixelwerte sammeln
    11 for zeile in range(len(ausschnittBildArr[ausschnittNr])):
---> 12     for spalte in range(len(ausschnittBildArr[ausschnittNr][zeile])):
    13         ausschnittPixelArr[ausschnittNr].append(ausschnittBildArr[ausschn
ittNr][zeile][spalte])
    14         unsortedPixelArr.append(ausschnittBildArr[ausschnittNr][zeile][sp
alte])

TypeError: object of type 'numpy.float64' has no len()

```

2. Aufnahme eines Dunkelbildes

Nicht jeder Pixel einer Kamera liefert den Grauwert 0, wenn der Sensor abgedeckt ist. Das liegt zum einen am thermischen Rauschen der Ausleseelektronik, zum anderen am sogenannten *Dunkelstrom*, das aufgrund von Fertigungstoleranzen und von spontan durch Wärmezufuhr entstehenden Ladungsträgerpaaren zu einem leicht unterschiedlichen Nullpunkt jedes Pixels führt. Diesen pixelweisen Offset kann man durch Erstellung eines *Dunkelbildes* eliminieren, den man von jeder Aufnahme subtrahiert. Dadurch wird ein Großteil des Rauschens aus der Aufnahme entfernt. Achtung: der Offset jedes Pixels hängt stark von der Belichtungszeit ab, d.h. im Prinzip braucht man für jede unterschiedliche Belichtungszeit ein eigenes Dunkelbild! Deshalb ist es wichtig, bei der Aufnahme des Dunkelbildes die Belichtungsparameter konstant auf den gleichen Wert der zu korrigierenden Aufnahme zu stellen. Vom Gesichtspunkt der Kalibrierung her bestimmen wir mit dem Dunkelbild alle Nullpunkte jeden einzelnen Pixels des gesamten Sensors.

Vorgehensweise:

- Stellen Sie die Belichtungsparameter genau gleich wie bei der Aufnahme des Grauwertkeiles ein. Decken Sie das Objektiv der Kamera so ab, dass das Bild komplett schwarz wird. Machen Sie 10 Aufnahmen.

black_Number.png Dateien Nummer 0<1<11

```

In [ ]: def schwarzweißDateiname( nummer, hell = 0):
    if hell == 0:
        farbe = "black"
    if hell == 1:
        farbe = "white"
    if(0 < nummer and nummer < 11):
        Nummer = nummer
        return f"{farbe}_{Nummer}.png"
    else:
        print("Nummer gibts net")
        return "Nummer gibts net"

```

- Schreiben Sie eine Routine, die die 10 Bilder einliest, in *double* umwandelt, die Farbbilder in Grauwertbilder umrechnet und ihren pixelweisen Mittelwert berechnet. Sie erhalten also nicht einen einzigen Mittelwert wie in Aufgabe 1, sondern ein ganzes Bild von Mittelwerten in derselben Größe wie die Einzelaufnahmen. Auf diese Weise wird das thermische Ausleserauschen eliminiert, es bleibt nur noch der Offset bzw. der Dunkelstrom jedes Pixels übrig. Das Mittelwertbild ist das Dunkelbild für die von Ihnen gewählte Belichtungszeit. Stellen Sie das Dunkelbild kontrastmaximiert in Ihrem Notebook dar.

```
In [ ]: def kontrast_maximieren(img):
    min_val = np.min(img)
    max_val = np.max(img)

    print("----kontrast_maximieren----")
    print("bild: min =", min_val)
    print("bild: max =", max_val)
    print("bild: mean =", np.mean(img))
    print("img = (img - min_val) / (max_val - min_val)")
    print("-----")

    if(max_val!= min_val):
        return (img - min_val) / (max_val - min_val)
    else:
        return img
```

```
In [ ]: def zehnKontrastBilder(showPicture=0, controllausgabe=0, hell = 0):
    zehnImg=[]
    #Werte holen
    for i in range (0, 10):
        dateiname = schwarzweißDateiname(i+1, hell=hell)
        img = einBildEinlesen(dateiname, keinUrBild =1)
        img = np.array(img, dtype=np.float64)
        zehnImg.append(img)
    #pixelweiser Mittelwert
    imgMean = []
    zehnPixelArr = np.empty(10)
    for zeile in range(len(zehnImg[0])):
        imgMean.append([])
        for spalte in range(len(zehnImg[0][0])):
            imgMean[zeile].append([])
            for i in range (0, 10):
                zehnPixelArr[i] = zehnImg[i][zeile][spalte]
            imgMean[zeile][spalte]= np.mean(zehnPixelArr)

    if(controllausgabe==1):
        for zeile in range(len(zehnImg[0])):
            print("[", ", ".join(f"{wert:.3f}" for wert in imgMean[zeile]), end="")
            print("]")

    if(showPicture==1):
        imag=kontrast_maximieren(imgMean)
        imag = np.array(imag)
        plt.imshow(imag, cmap='gray')

    if(hell == 0):
        contrast = "black"
```

```

else:
    contrast = "white"

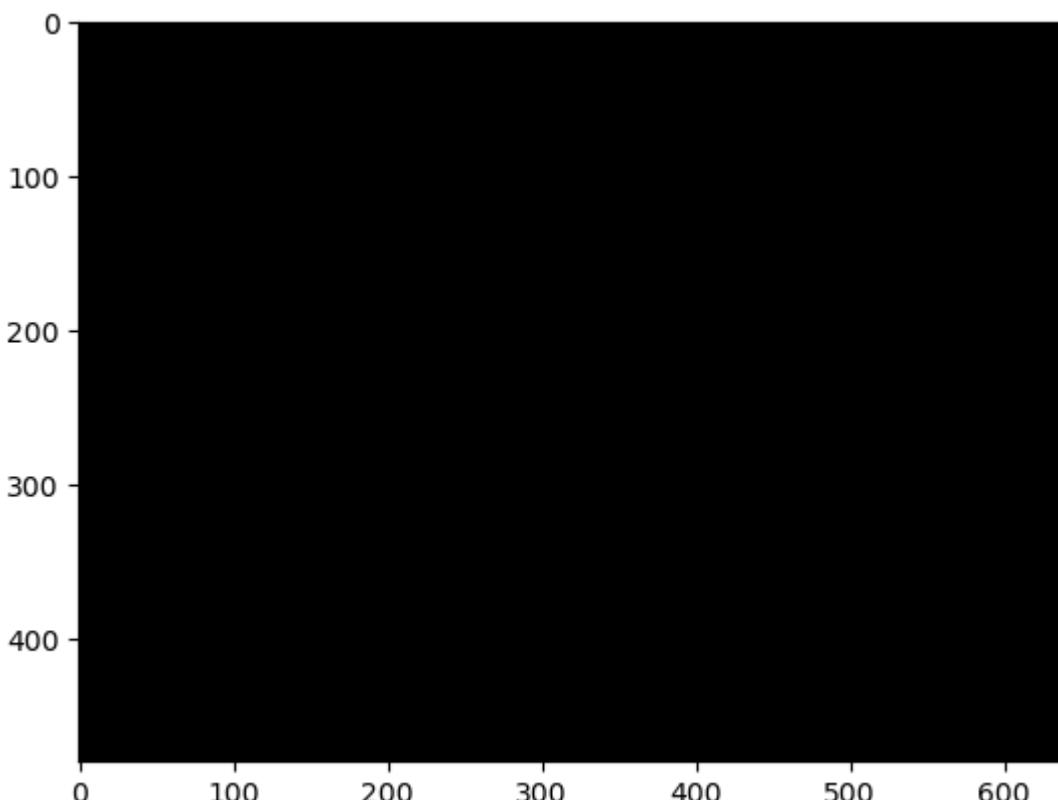
imgMean = np.array(imgMean)
dateiname=f"{contrast}_MittelwertBild.png"
cv2.imwrite(dateiname, imgMean.astype(np.uint8))
return imgMean, dateiname

```

In []: kontrastbild = zehnKontrastBilder(showPicture=1, hell=0)

```

----kontrast_maximieren----
bild: min = 0.0
bild: max = 0.0
bild: mean = 0.0
img = (img - min_val) / (max_val - min_val)
-----
```



- Schreiben Sie eine Routine, die das Dunkelbild einliest, von einem Eingabebild subtrahiert und das korrigierte Bild abspeichert. Wir werden das Programm dann in den folgenden Aufgaben erweitern.

In []: `def bildKorrigieren(dateiname, mitHell = 0, weißbild=None, zusatz = ""):`

```

eingabeImg=einBildEinlesen(dateiname=dateiname,keinUrBild=1, keinTeilBild=1,
eingabeImg = np.array(eingabeImg, dtype=np.float64)

if(mitHell == 0):
    kontrastbild, dateinamikon = zehnKontrastBilder(showPicture=0,hell=mitHe
    kontrastbild = np.array(kontrastbild, dtype=np.float64)
elif weißbild is not None:
    kontrastbild = np.array(weißbild, dtype=np.float64)
else:
    print("kein Bild übergeben")
# print("ein",eingabeImg.shape)
# print("dunkel",kontrastbild.shape)
ausgabeImg=eingabeImg-kontrastbild

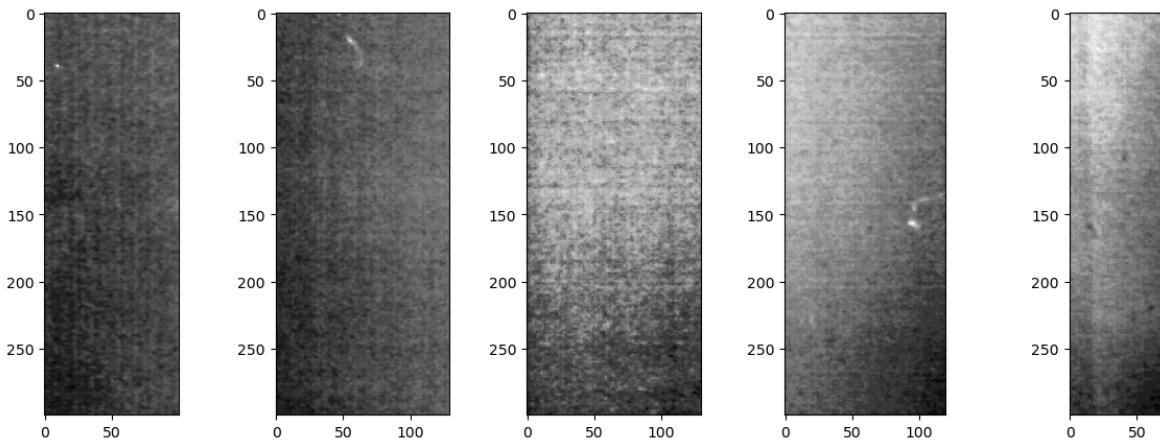
```

```
clean = dateiname.replace(".png", "")  
dateibezeichnung = f"KorrekturNorm_{clean}{zusatz}.png"  
cv2.imwrite(dateibezeichnung, ausgabeImg.astype(np.uint8))  
return ausgabeImg, dateibezeichnung
```

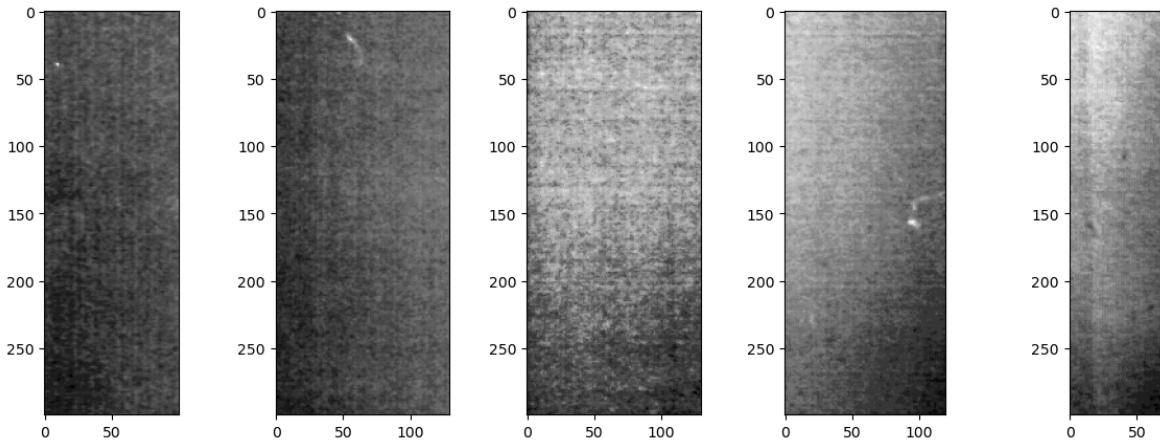
```
In [ ]: ausgabeImg, dateibezeichnung= bildKorrigieren("greyscale.png", mitHell=0)
```

```
In [ ]: eingabeImg=einBildEinlesen(dateiname="greyscale.png", Farbe=0, keinUrBild=1)  
eingabeImg=einBildEinlesen(dateiname="KorrekturNorm_greyscale.png", Farbe=0, kei
```

greyscale.png als Graubild nach Graustufe aufgeteilt



KorrekturNorm_greyscale.png als Graubild nach Graustufe aufgeteilt



3. Aufnahme eines Weißbildes

Obwohl die einzelnen Pixel einer Kamera eine hervorragende Linearität mit der Beleuchtungsstärke aufweisen, ist ihre Sensitivität aufgrund von Fertigungstoleranzen nicht völlig gleich. Zusätzlich kommt noch die sogenannte *Vignettierung* hinzu, d.h. die Optik der Kamera überträgt die Helligkeit nicht gleichmäßig auf den Sensor. Typischerweise findet man eine Abdunkelung des Bildes zu den Rändern hin. Zur Kompensation dieser Effekte nimmt man ein sogenanntes *Weißbild* auf. Dazu braucht man eine möglichst homogene Fläche, z.B. ein Blatt Papier oder den wolkenlosen Himmel. Durch Division durch das Weißbild kann man die unterschiedlichen Sensitivitäten der einzelnen Pixel herausrechnen. Leider hängt das Weißbild vom eingestellten Fokus des Kameraobjektives ab, d.h. im Prinzip braucht man für jede Fokuseinstellung ein eigenes Weißbild. In unserem Fall bedeutet das, dass das Weißbild in der gleichen Entfernung wie das zu korrigierende Bild aufgenommen werden muss.

Vom Gesichtspunkt der Kalibrierung her nehmen wir mit dem Weißbild einen zweiten Punkt der Kennlinie jeden einzelnen Pixels auf. Zusammen mit dem Dunkelbild haben wir damit die Kennlinie für alle Pixel gleichzeitig und eindeutig bestimmt: wir wissen an jeder Stelle des Bildes die Steigung und den Nullpunkt und können durch Subtraktion des Dunkelbildes und Division durch das Weißbild die tatsächliche Intensität des einfallenden Lichtes bestimmen.

Vorgehensweise:

- Nehmen Sie ein leeres Blatt Papier in der gleichen Entfernung wie das Testbild auf. Achten Sie darauf, die Belichtung auf 30-50% der Hellsättigung einzustellen. Wichtig ist auch, dass die Helligkeit des Blattes so gleichmäßig wie möglich ist, also keine Schatten oder unterschiedlich helle Beleuchtung. Machen Sie auch hier 10 Aufnahmen zur Elimination des thermischen Rauschens.

black_Number.png Dateien Nummer 0<1<11

- Schreiben Sie eine Routine, die die 10 Bilder einliest und ihren Mittelwert berechnet. Subtrahieren Sie von dem Mittelwertbild das Dunkelbild und speichern Sie das resultierende Weißbild ab. Stellen Sie das Weißbild kontrastmaximiert dar und nehmen Sie es in das Protokoll auf.

```
In [ ]: def weißSubSchwarz(showPicture=0):
    hellBild, dateiname = zehnKontrastBilder(showPicture=0,hell=1)
    hellBild = np.array(hellBild)

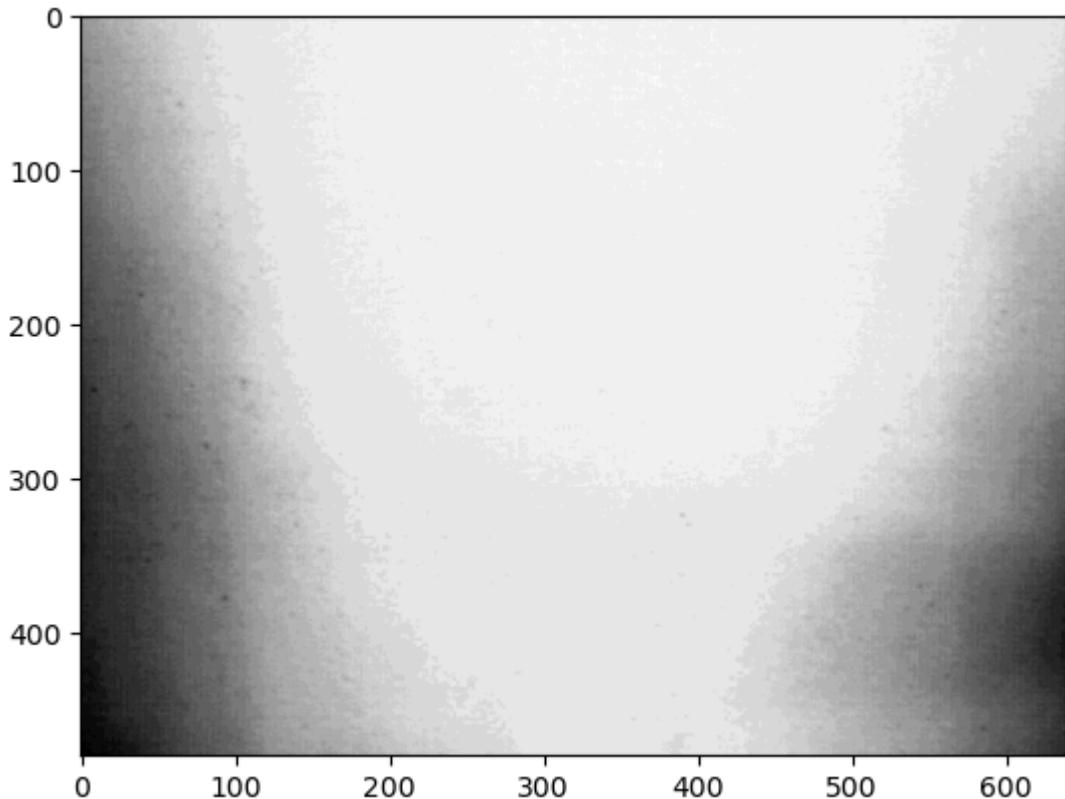
    resWeißbild, dateibezeichnung = bildKorrigieren(dateiname=dateiname, mitHell=1)

    if(showPicture==1):
        resWeißbildK = kontrast_maximieren(resWeißbild)
        plt.imshow(resWeißbildK, cmap='gray')

    return resWeißbild
```

```
In [ ]: result = weißSubSchwarz(1)

----kontrast_maximieren---
bild: min = 229.0
bild: max = 250.0
bild: mean = 245.594365234375
img = (img - min_val) / (max_val - min_val)
-----
```



- Erweitern Sie die Routine zur Subtraktion des Dunkelbildes aus Aufgabe 2 so, dass das Weißbild eingelesen, in *double* umgewandelt wird und so normiert wird, dass sein Mittelwert 1 ist. Das durch Abzug des Dunkelbildes korrigierte Eingangsbild wird anschliessend durch das normierte Weißbild dividiert.

```
In [ ]: def normierBild(bild):
    bild = np.array(bild, dtype=np.float64)
    mean_value = np.mean(bild)
    print(mean_value)
    if mean_value==0:
        print("WARNUNG: Mittelwert ≈ 0 → Normierung übersprungen!")
        return bild
    bild = bild / mean_value

    dateiname=f"NormiertesBild.png"
    cv2.imwrite(dateiname, bild.astype(np.uint8))
    return bild, dateiname
```



```
In [ ]: def weißSubSchwarzNormiert(showPicture=0):
    # Weißbild einlesen (hell=1)
    hellBild, dateiname = zehnKontrastBilder(showPicture=0, hell=1)
    hellBild = np.array(hellBild, dtype=np.float64)

    # Weißbild normieren (MEAN = 1)
    weiss_norm, dateinameN = normierBild(hellBild)

    # Eingang korrigieren (inkl. Dunkelbild-Abzug)
    korrbild, dateibezeichnung = bildKorrigieren(dateiname=dateinameN, mitHell=0)
    korrbild = np.array(korrbild, dtype=np.float64)

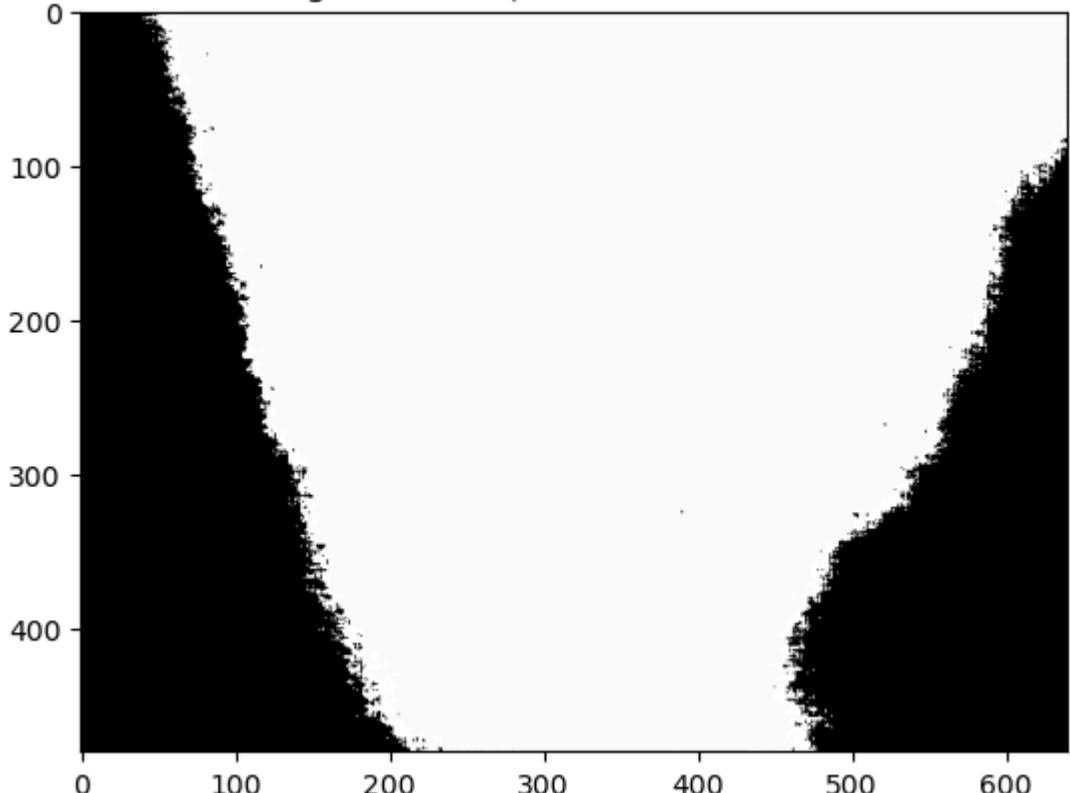
    result = korrbild / weiss_norm
```

```
if showPicture == 1:  
    plt.imshow(result, cmap='gray')  
    plt.title("Normiertes korrigiertes Bild (Division durch normiertes Weißbild)  
  
return result
```

```
In [ ]: result = weißSubSchwarzNormiert(1)
```

```
245.91652083333338
```

Normiertes korrigiertes Bild (Division durch normiertes Weißbild)

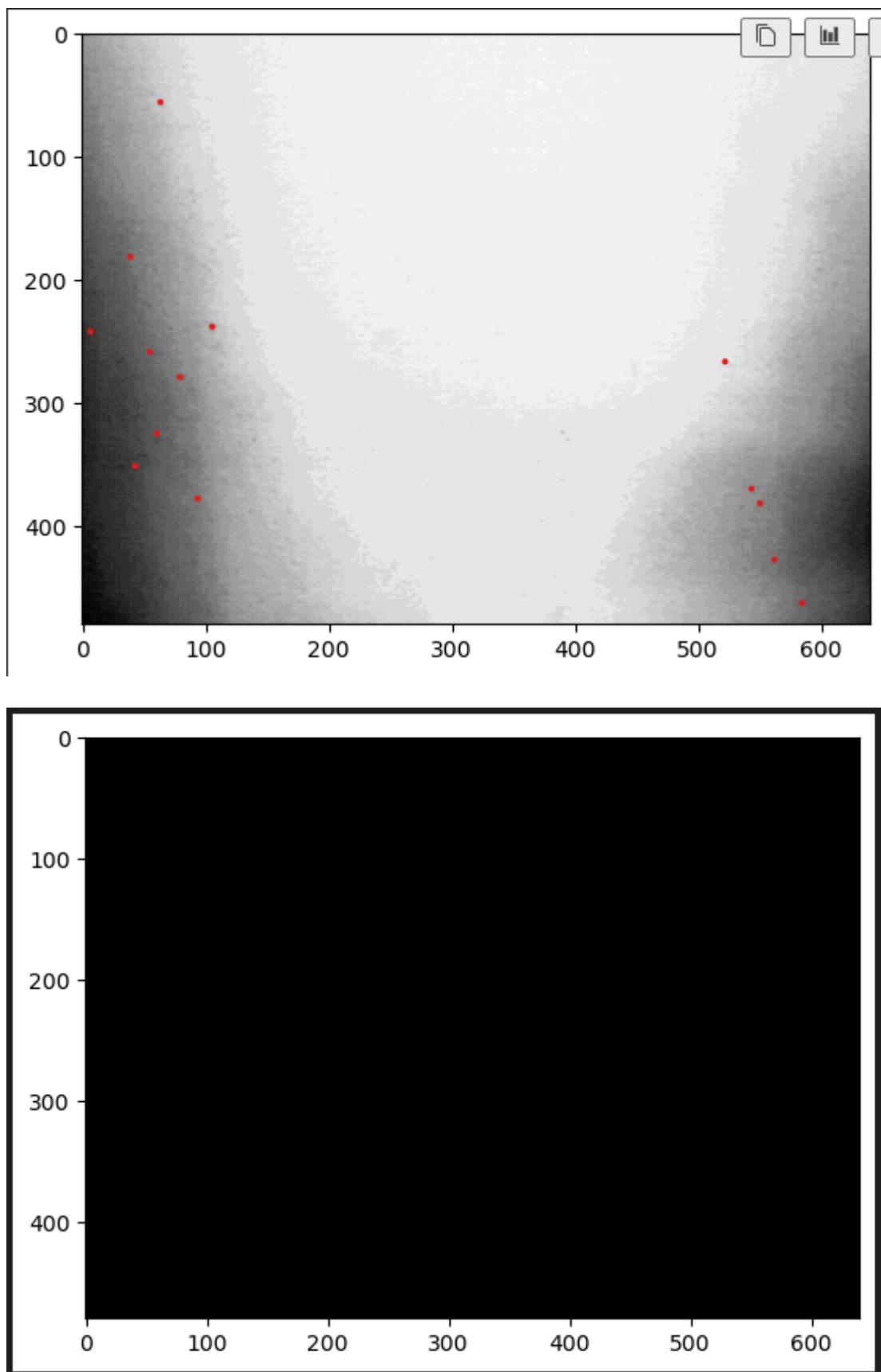


4. Pixelfehler

Je nach Qualität des Bildsensors entstehen beim Fertigungsprozess eine Anzahl von funktionsuntüchtigen Pixeln. Es gibt *dead pixels*, die immer auf ihrem niedrigsten Wert steckenbleiben, *stuck pixels*, die immer auf ihrem Maximalwert bleiben, und sogenannte *hot pixels*, die bei längeren Belichtungszeiten in die Sättigung gehen. Stuck und hot pixels entdeckt man am einfachsten im Dunkelbild, bei dem diese Pixel als helle Punkte auffallen. Dead Pixels findet man im Weißbild, wo sie als dunkle Punkte auffallen. Je nach Anwendung werden diese Pixelwerte im zu korrigierenden Bild durch Interpolation aus ihren Nachbarwerten ersetzt, so dass sie nicht mehr auffallen.

Vorgehensweise:

- Überprüfen Sie Ihr Dunkelbild auf dem Bildschirm auf stuck und hot pixels und Ihr Weißbild auf dead pixels. Markieren Sie diese im Bild und fügen es in Ihr Notebook ein.



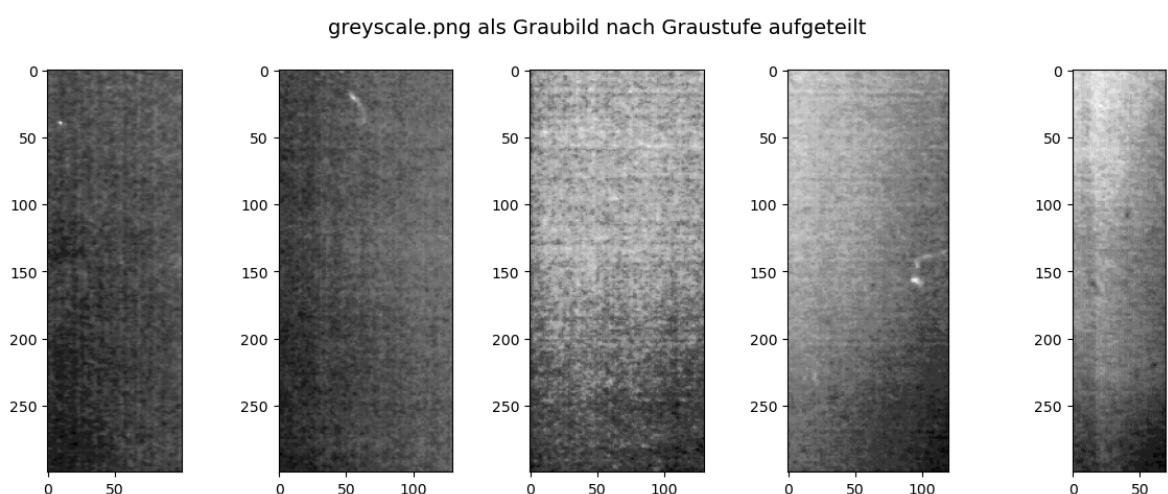
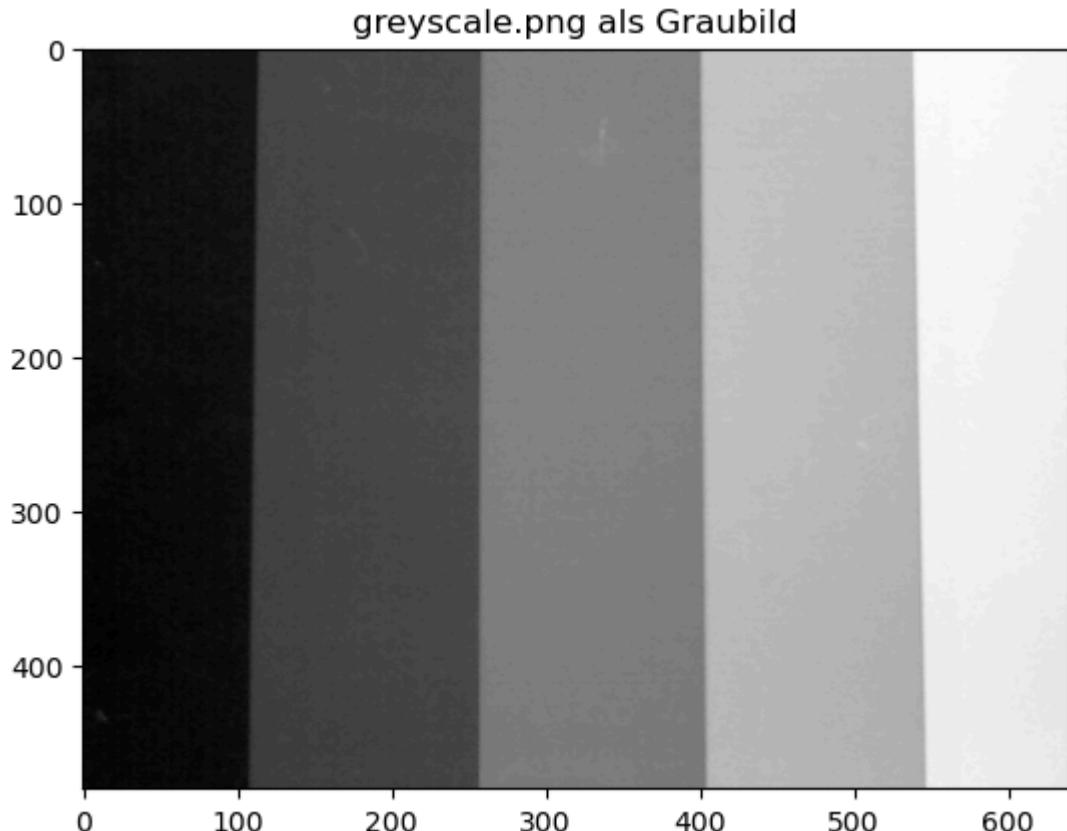
- Korrigieren Sie mithilfe Ihres Programms aus Aufgabe 3 das Bild des Grauwertkeils aus Aufgabe 1, speichern Sie es ab und bauen Sie es in Ihr Notebook ein.

```
In [ ]: def korrigierGrayscale(original = 0):
    dateiname = "greyscale.png"
    normWeißBild = weißSubSchwarzNormiert(0)
    bildNachSchwarz,dateibezeichnung = bildKorrigieren(dateiname=dateiname, mitH
```

```
result,dateibezeichnung = bildKorrigieren(dateiname=dateibezeichnung, mitHel  
  
if(original==1):  
    plottstart()  
    bild = einBildEinlesen(dateiname=dateiname)  
  
    plottstart()  
    result = einBildEinlesen(dateiname=dateibezeichnung)  
  
return dateibezeichnung
```

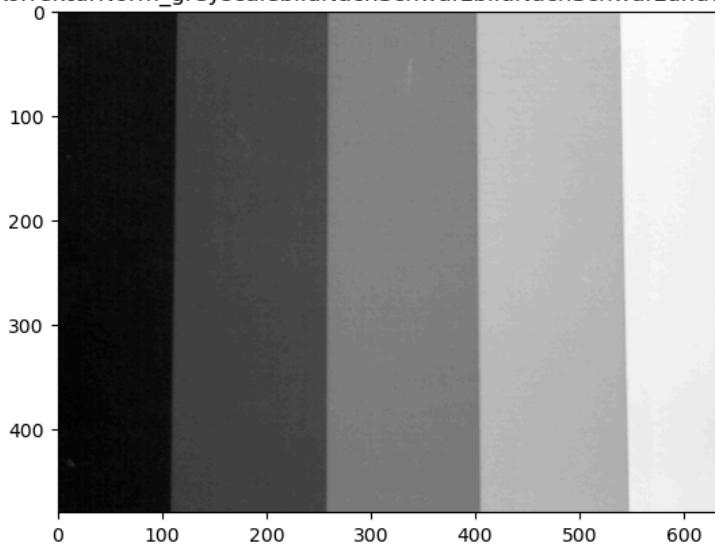
In []: korrigierGrayscale(1)

245.91652083333338
<Figure size 800x400 with 0 Axes>

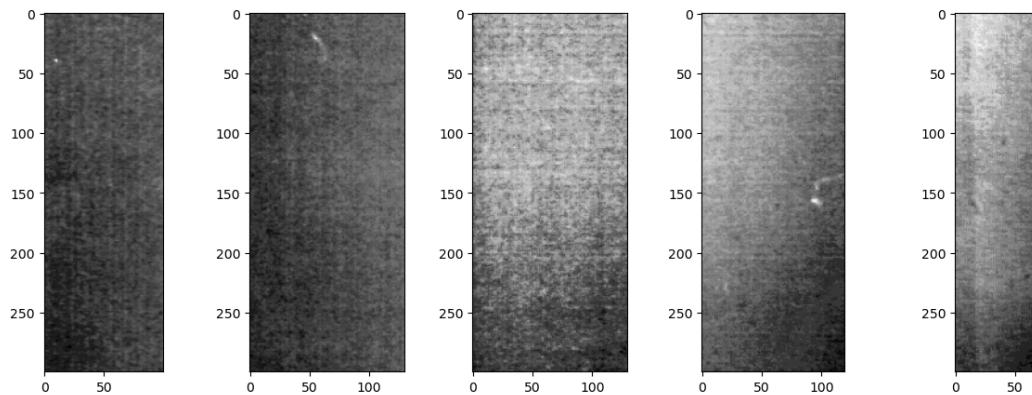


<Figure size 800x400 with 0 Axes>

KorrekturNorm_KorrekturNorm_greyscalebildNachSchwarzbildNachSchwarzundWeiss.png als Graubild



KorrekturNorm_KorrekturNorm_greyscalebildNachSchwarzbildNachSchwarzundWeiss.png als Graubild nach Graustufe aufgeteilt



Out[]: 'KorrekturNorm_KorrekturNorm_greyscalebildNachSchwarzbildNachSchwarzundWeiss.png'

- Werten Sie nun erneut das korrigierte Bild aus Aufgabe 4 mit diesem Programm aus und erstellen Sie die zugehörige Tabelle wie in Aufgabe 1. Ergibt sich eine Verbesserung?

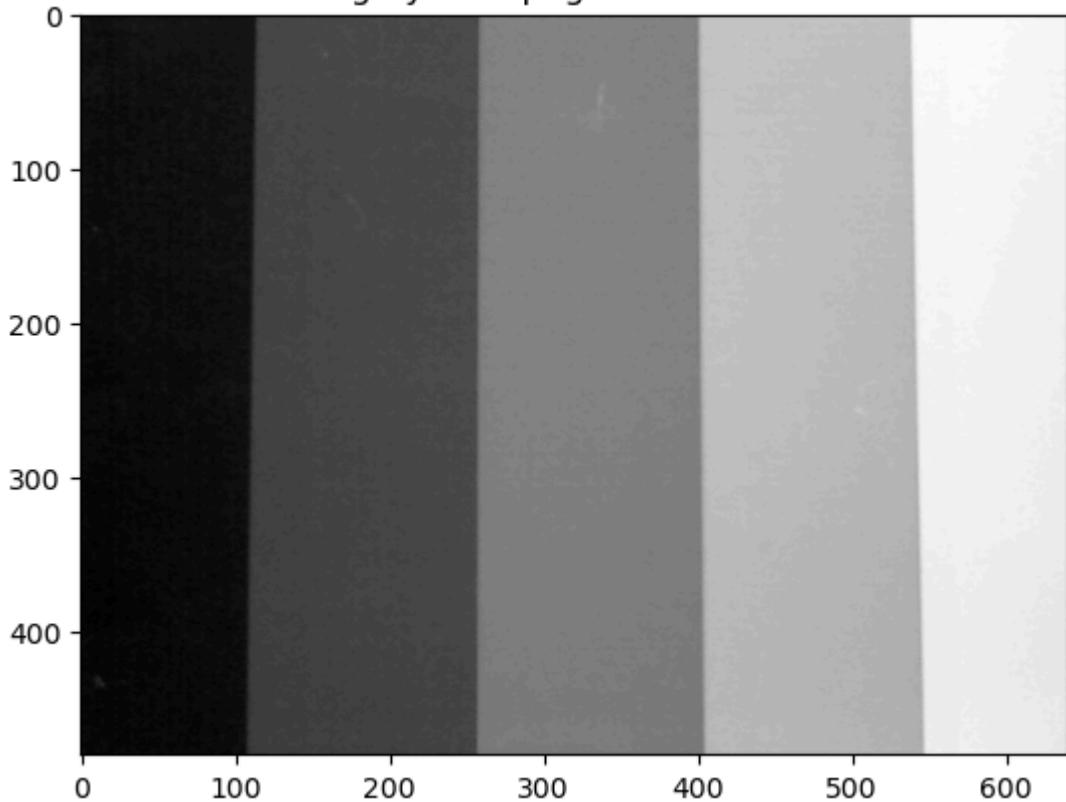
In []: `def tabelle2():`

```
dateiname = korrigierGrayscale(1)
ausschnittBildArr = einBildEinlesen(dateiname=dateiname, Farbe =0, keinUrBild=1,
gesamtMean, gesamtStd, pixelMeanArray, pixelStdArray, erhaltenesPixelArr = pro
print("Neue Tabelle: ")
tabelle = tabelleGraustufen(pixelMeanArray, pixelStdArray, save_csv=True, csv
#originaltabelle
print("\n\nOriginaltabelle: ")
ausschnittBildArr = einBildEinlesen(r"greyscale.png", Farbe =0, keinUrBild=1, a
gesamtMean, gesamtStd, pixelMeanArray, pixelStdArray, erhaltenesPixelArr = pro
tabelle = tabelleGraustufen(pixelMeanArray, pixelStdArray, save_csv=True, csv
```

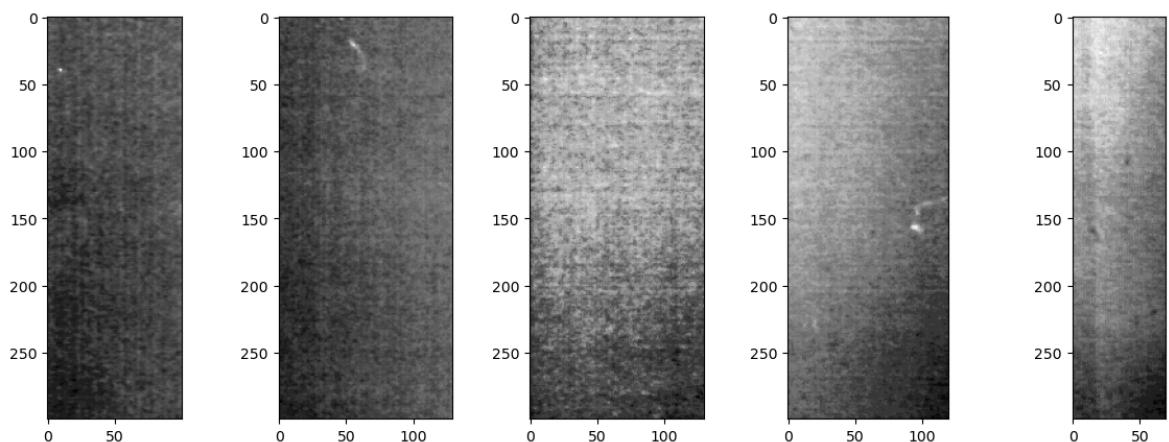
In []: `tabelle2()`

```
245.91652083333338
<Figure size 800x400 with 0 Axes>
```

greyscale.png als Graubild

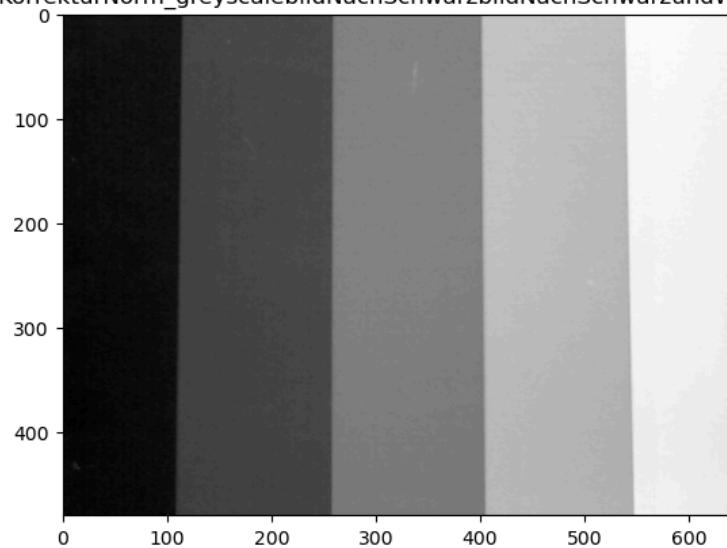


greyscale.png als Graubild nach Graustufe aufgeteilt

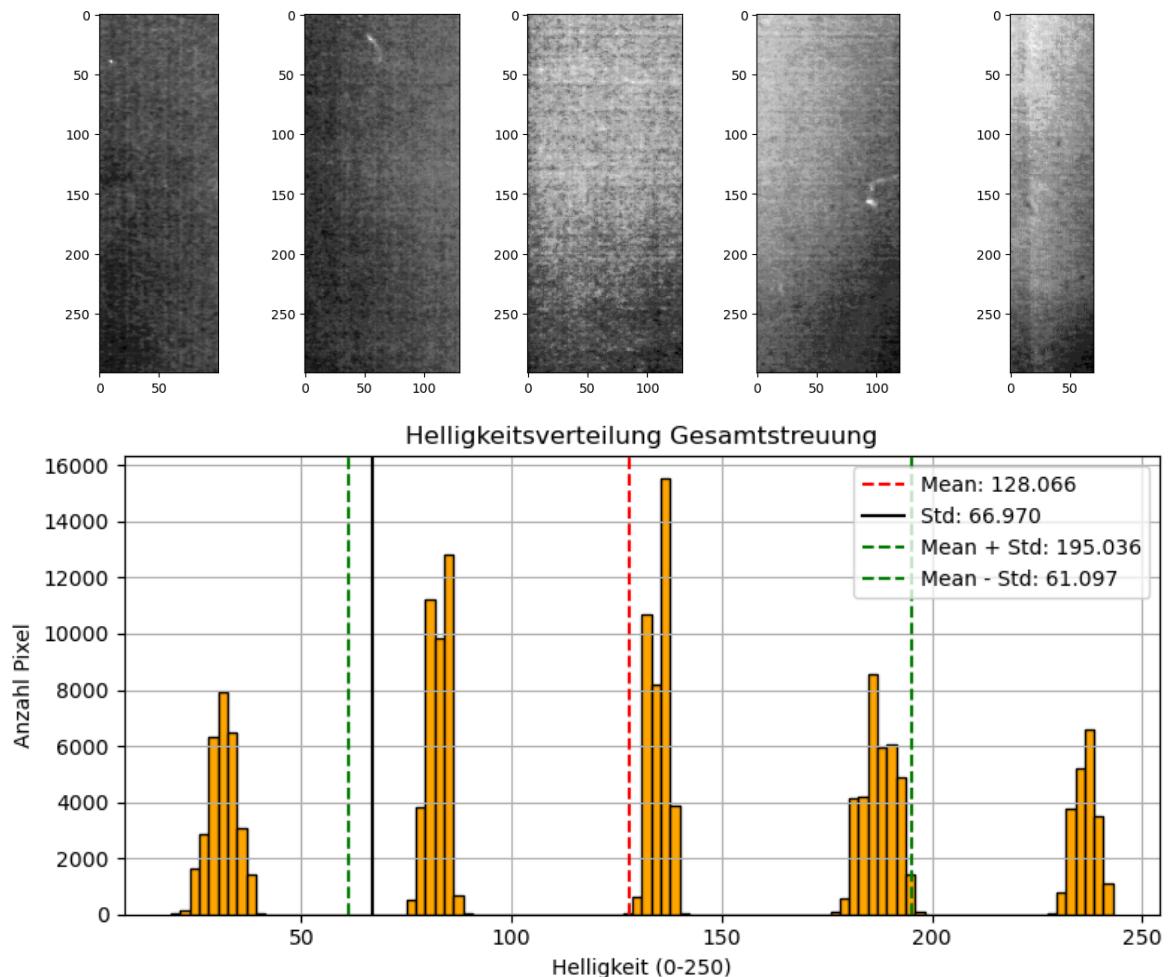


<Figure size 800x400 with 0 Axes>

KorrekturNorm_KorrekturNorm_greyscalebildNachSchwarzbildNachSchwarzundWeiss.png als Graubild



KorrekturNorm_KorrekturNorm_greyscalebildNachSchwarzbildNachSchwarzundWeiss.png als Graubild nach Graustufe aufgeteilt



Neue Tabelle:

===== Tabelle Graustufen =====		
Grauwert-Stufe	Mittelwert	Standardabweichung
schwarz	31.286	3.316
dunkelgrau	82.353	2.362
grau	134.957	2.298
hellgrau	187.500	3.821
weiss	236.536	2.585

>>> Tabelle gespeichert als: graustufenTabelle_A2.csv

Originaltabelle:

===== Tabelle Graustufen =====		
Grauwert-Stufe	Mittelwert	Standardabweichung
schwarz	31.316	3.353
dunkelgrau	83.263	2.493
grau	135.957	2.298
hellgrau	188.413	3.968
weiss	236.856	2.906

>>> Tabelle gespeichert als: graustufenTabelle_A1.csv

die Standardabweichungen der einzelnen Grautöne sind weniger geworden =>
Verbesserung