



**Hochschule Konstanz**  
Technik, Wirtschaft und Gestaltung

**Signale, Systeme und Sensoren**

# **Kalibrierung von digitalen Kameras**

**T. Schoch, L. Stratmann**

**Konstanz, 17. April 2019**

## Zusammenfassung (Abstract)

Thema:	Kalibrierung von digitalen Kameras	
Autoren:	T. Schoch	tobias.schoch@htwg-konstanz.de
	L. Stratmann	luca.stratmann@htwg-konstanz.de
Betreuer:	Prof. Dr. Matthias O. Franz Jürgen Keppler	mfranz@htwg-konstanz.de juergen.keppler@htwg-konstanz.de
	Mert Zeybek	me431zey@htwg-konstanz.de

In dem Versuch haben wir einen Entfernungsmeßer dazu verwendet, um die bereits in der Vorlesung behandelten Vorgehensweisen zum Thema Kalibrierung, Fehlerbehandlung und Fehlerrechnung anzuwenden. Der Distanzsensor der Marke "Sharp" benutzt für das Triangulationsprinzip Infrarot-LEDS mit einer Linse. Diese geben Lichtstrahlen von sich, um dann wiederum reflektiert zu werden und durch die zweite Linse zu gelangen. Je nachdem wo der Lichtstrahl auftrifft, wandelt der Signalprozessor die Leitfähigkeit in eine Spannung um. Da das Ausgangssignal anti-proportional ist, wird mit der zunehmenden Entfernung das Ausgangssignal kleiner. Die Entfernung liegen zwischen 10cm und 70cm. Durch ein Oszilloskop können wir den Spannungsverlauf des Abstandssensors überprüfen.

# **Inhaltsverzeichnis**

# **Abbildungsverzeichnis**

# **Tabellenverzeichnis**

# **Listingverzeichnis**

# 1

## Einleitung

In dem ersten Versuch geht es um die "Kalibrierung und Einsatz eines Infrarot-Entfernungsmessers". Mittels eines aktiven Abstandssensors der Marke 'Sharp' mit dem Namen 'GP2Y0A21YK0F' messen wir 21 unterschiedliche Distanzen die mit einem Meterstab überprüft werden.

Ein 5V Netzgerät versorgt den Abstandssensor mit Strom. Der Abstandssensor gibt eine Ausgangsspannung an das Oszilloskop, mit welchem wiederum man .csv Dateien ausgeben kann. Im ersten Kapitel dieses Versuches haben wir die Aufgabe diese Geräte miteinander zu verbinden. Nachdem wir unter Anderem das Oszilloskop kalibriert haben, starten wir mit den einzelnen Messungen. Wir erstellen dazu eine Pythonfunktion um die Daten aus den .csv Dateien zu verwerten.

Mit matplotlib stellen wir die Übertragungsfunktion und die Kennlinie dar. Im zweiten Kapitel arbeiten wir mit logarithmierten Werten und stellen diese in einem Graphen visuell dar unter anderem als Kennlinie.

Eines der Hauptthemen des Versuches ist die lineare Regression.

Deshalb haben wir selbstverständlich die Aufgabe eine Ausgleichsgerade in Python darzustellen mittels der linearen Regression. Im dritten und mathematisch anspruchvollsten Kapitel beschäftigen wir uns mit der Ermittlung des Messfehlers und der Flächenmessung. Bei der Ermittlung des Messfehlers behandeln wir auch die Gaußsche Fehlerfortpflanzungsge setzt mit 68% und 95%.

# 2

## **Versuch 1: Ermittlung der Kennlinie des Abstandssensors**

### **2.1 Fragestellung, Messprinzip, Aufbau, Messmittel**

Im ersten Versuch werden wir die Kennlinie des Abstandssensors ermitteln. Für den Aufbau des Projektes verbinden wir den Abstandssensor an 'Output 3' des Labornetzgerätes 'EA-PS 2342-06 B' durch Ground ( - ) und dem 5V Anschluss ( + ). Der Abstandssensor lautet 'GP2Y0A21YK0F' und wurde von der Firma 'Sharp' entworfen. Das Netzgerät wird auf 5V Gleichspannung eingestellt. Das Oszilloskop von 'Tektronix' mit dem Namen 'TDS 2022B' wird an den Abstandssensor mit Ground( - ), sowie an den Signalausgang angeschlossen. Dies wird im Oszilloskop mit einem Adapter an Channel1 angeschlossen. Nachdem das Oszilloskop richtig eingestellt wurde, haben wir es mit dem PC verbunden. Über ein Programm konnten wir so das Oszilloskop mit dem Computer verbinden. Ein Programm hat uns geholfen den aktuellen Bildschirm des Oszilloskopes auf dem Bildschirm zu empfangen. Zudem kann das Programm die empfangenen Daten über die Ausgangsspannung in eine '.csv' Datei ausgeben. So konnten wir für jede Messung einen Screenshot und eine '.csv' Datei erstellen. Ein hochkant stehendes Holzbrett definiert den Abstand. Die 21 zu messenden Werte liegen zwischen 10cm und 70cm in jeweils 3cm Abständen. Mit einem Meterstab haben wir einen Richtwert für den Abstand zwischen Abstandssensor und Holzbrett. Nachdem wir durch die erschwerten Lichtverhältnisse die richtige Lage des Abstandssensors gefunden haben, haben wir über das Programm sowohl Screenshots vom Bildschirm des Oszilloskopes gemacht, als auch die Daten in einer '.csv' Datei gespeichert. Zudem haben wir die gemessenen Längen mit deren dazugehörigen Ausgangsspannung handschriftlich in einer Tabelle aufgeschrieben.

ben, welche am Ende des Versuches vom Tutor unterschrieben wurde. Anschließend haben wir in Python programmiert, um die Dateien aus den '.csv' einzulesen mit genfromtxt(). Um den Einschwingvorgang nicht mit zu berechnen haben wir die ersten 1000 Zeilen übersprungen. Nachdem werden wir den Durchschnitt sowie die Standardabweichung berechnet haben, visualisieren wir in einer Kennlinie den Durchschnitt sowie die Standardabweichung mittels matplotlib.

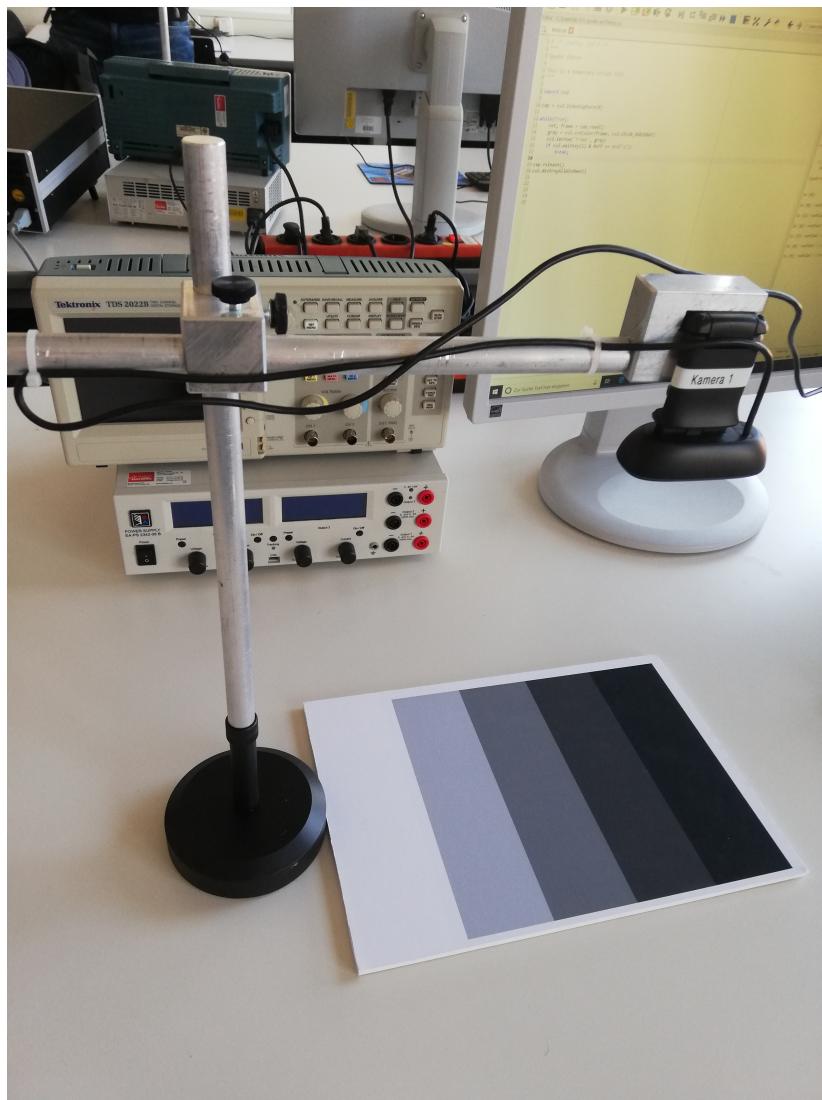


Abbildung 2.1: Aufbau im Labor

## 2.2 Messwerte

Tabelle [2.2] zeigt die von Hand notierten, sowie die in Python programmierten Werte.

Stufe	Blau, Rot und Grün	Hex-Value	Standartabweichung
Stufe 1	86.39761904761905	#161616	2.082341260388545
Stufe 2	112.00498456790123	#1d1d1d	2.6116529874626044
Stufe 3	161.2048205596107	#292929	2.2861194924984116
Stufe 4	199.60105744949496	#333333	2.968662423351764
Stufe 5	235.58106563421828	#3c3c3c	2.582844019720398

Tabelle 2.1: Messwerte Kalibrierung

## 2.3 Auswertung

In der folgenden Abbildung sind die Messergebnisse der durchschnittlichen Spannung nochmals visuell dargestellt. Die Messergebnisse wurden mit matplotlib in Python visualisiert.

Zuerst wurden die Werte von der .csv Datei mit `genfromtxt()` in das Programm eingelesen. Nachdem für die einzelnen Dateien mit der numpy Funktion `np.mean()` der Durchschnitt der von dem Oszilloskop gemessen Ausgangsspannung in Volt ausgerechnet wurde, haben wir diese mit der y-Achse initialisiert.

Die x-Achse wurde mit dem Abstand in cm definiert. Durch eine For-Schleife kann nun jedem Abstandswert eine durchschnittliche Spannung zugewiesen werden.

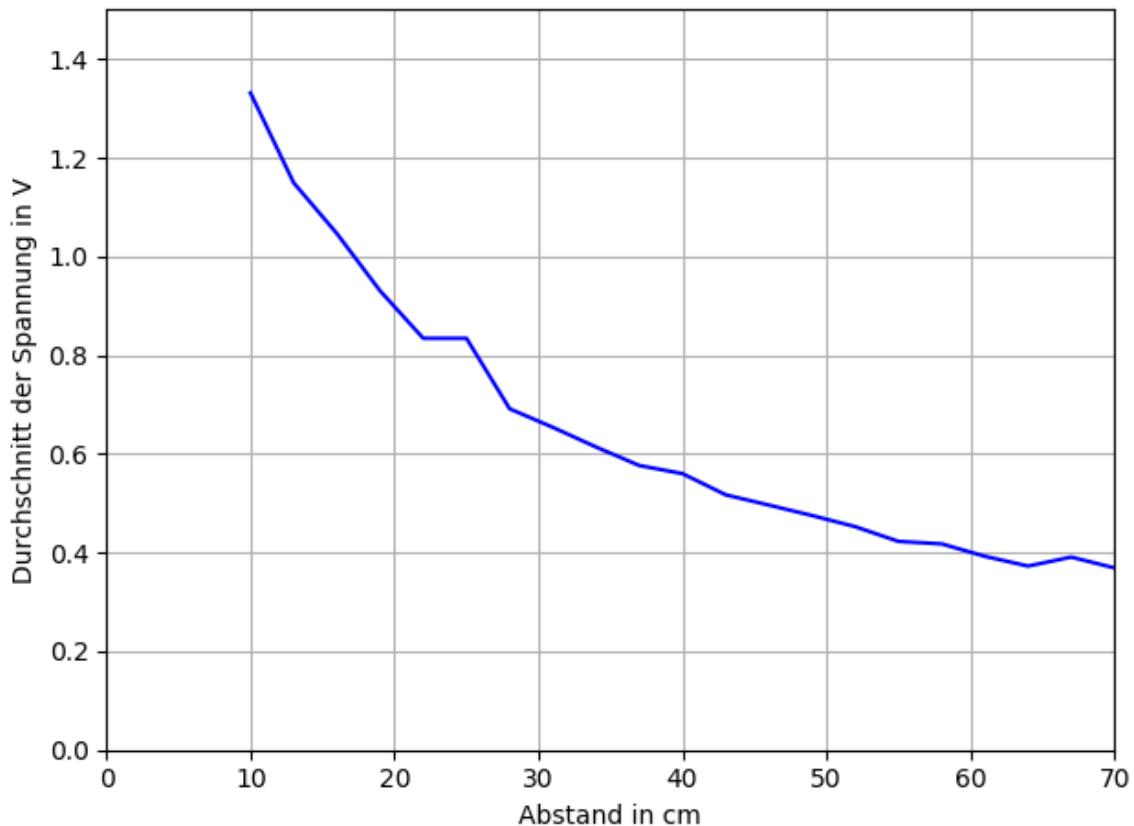


Abbildung 2.2: Durchschnittliche Spannung

In der folgenden Abbildung sind die Messergebnisse der Standardabweichung visuell dargestellt. Die Messergebnisse wurden mit matplotlib in Python visualisiert. Hierfür wurde mit der numpy Funktion `np.std()` die Standardabweichung in Volt berechnet und mit der y-Achse initialisiert.

Die x-Achse wurde wieder mit dem Abstand in cm deklariert. Wieder wurden durch die For-Schleife jedem Abstand ein Standardabweichung der Spannung zugewiesen werden.

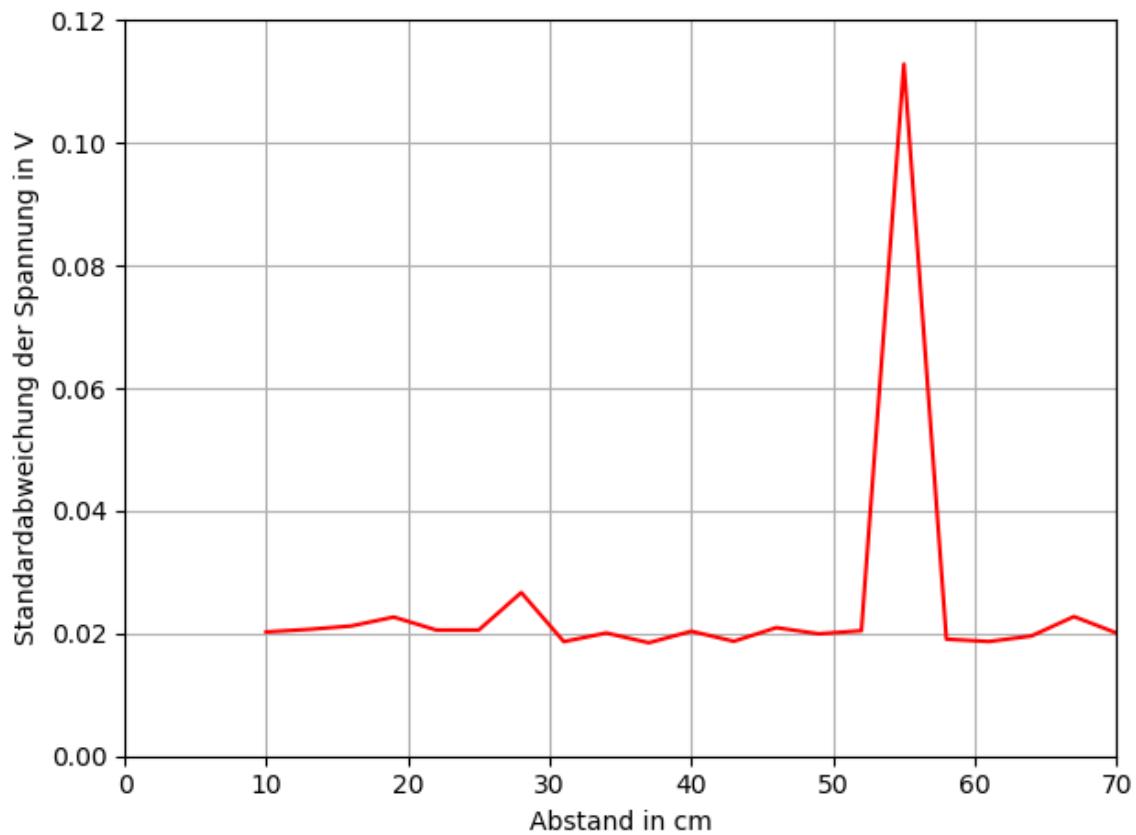


Abbildung 2.3: Standartabweichung der Spannung

## 2.4 Interpretation

Wie man gut in der Tabelle und der Abbildung 2.2 ablesen kann, wird die Spannung stets niedriger. Dies liegt an der Anti-proportionalität, dass mit der zunehmenden Entfernung zwischen Holzbrett und Abstandssensor die vom Signalprozessor übertragene Spannung geringer wird.

Leider haben wir bei der Generierung der Dateien einen Fehler gemacht und bei 22cm und 25cm zu spät die Single Sequenz aktualisiert, weshalb eine Gerade in dem Plot zwischen 22cm und 25cm genau gleich ist. Zudem geht bei Messung zwischen 64cm und 67cm Abstand die Spannung nach oben. An dem Tag der Messung war das Wetter sehr wechselhaft, was zu einer Erhöhung der Werte führen könnte.

Bei der Messung 55cm ist eine sehr hohe Standardabweichung im Vergleich zu den anderen Werten. Dies liegt daran, dass das Oszilloskop durch andere Störfaktoren gestört wurde und so eine ungleiche Single Sequenz ergeben hat. In dem rechten Bild der Abbildung 2.4 ist im Vergleich nochmals gut zu sehen, wie die hohe Standartabweichung zu stande kommt.

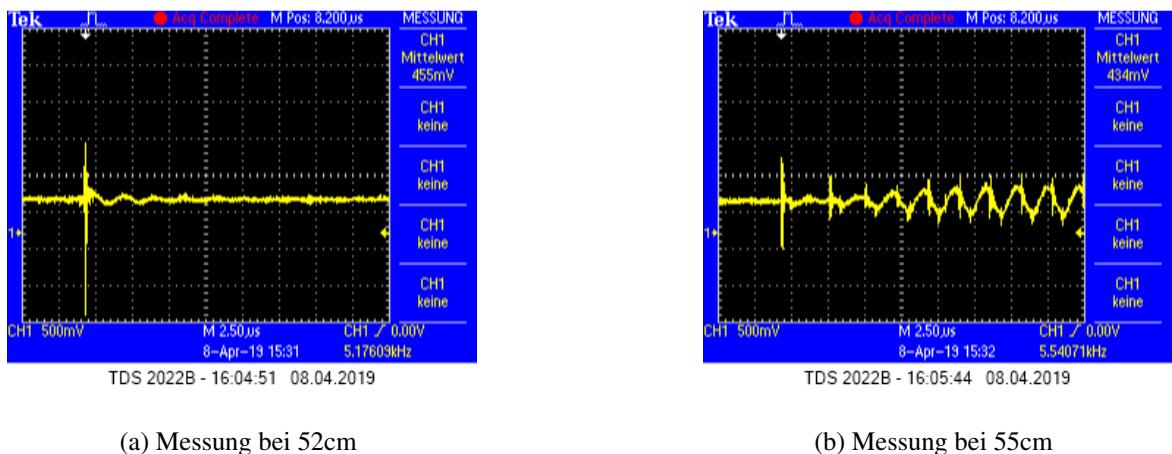


Abbildung 2.4: Unterschiede durch Störfaktoren

# 3

## **Versuch 2: Modellierung der Kennlinie durch lineare Regression**

### **3.1 Fragestellung, Messprinzip, Aufbau, Messmittel**

In Versuch Nummer 2 war es die Aufgabe die Ein- und Ausgangswerte, also die Distanz und die durchschnittliche Spannung in der Tabelle zu logarithmieren.

Zudem soll man so eine Kennlinie, logarithmierte Werte sowie die lineare Regression darstellen.

So soll man die bereits erlernten Formeln aus den Vorlesungen in System, Signale und Sensoren anwenden und für die Daten des Sharp-Sensors umwandeln um so die Prinzipien der linearen Regression praktisch zu verwenden.

Durch die Errechnung und Darstellung der Kennlinie, soll man auch prüfen ob ein systematischer Fehler geschehen ist. Falls dies nicht der Fall ist, sollte die Kennlinie einer typischen Kennliniensteigung entsprechen.

Für den Versuch benötigt man lediglich die bereits gemessenen Daten, eine Python IDE, sowie die nützlichen Formeln aus den Vorlesungs- und Aufgabenblättern.

## 3.2 Messwerte

Tabelle [3.2] zeigt die in Python logarithmierten Werte.

Distanz	log cm	log Durchschnitt
10cm	2.302585092994046	0.28658251236288684
13cm	2.5649493574615367	0.13950997769809692
16cm	2.772588722239781	0.04584587706574458
19cm	2.9444389791664403	-0.0717653956802384
22cm	3.091042453358316	-0.18091598142798918
25cm	3.2188758248682006	-0.18091598142798918
28cm	3.332204510175204	-0.3688220785875914
31cm	3.4339872044851463	-0.42460823943365683
34cm	3.5263605246161616	-0.48755538925392977
37cm	3.6109179126442243	-0.5504620343701235
40cm	3.6888794541139363	-0.579426134298413
43cm	3.7612001156935624	-0.659127119920277
46cm	3.828641396489095	-0.7003660010868976
49cm	3.8918202981106265	-0.7438412790564594
52cm	3.9512437185814275	-0.792597814482305
55cm	4.007333185232471	-0.8607681589957209
58cm	4.060443010546419	-0.8726037472131969
61cm	4.110873864173311	-0.9348943146021321
64cm	4.1588830833596715	-0.9865867500420398
67cm	4.204692619390966	-0.9389225676676274
70cm	4.248495242049359	-0.9965499402651674

Tabelle 3.1: Logarithmus von der Distanz und der Durchschnittsspannung

### 3.3 Auswertung

In der folgenden Abbildung wurden die Messergebnisse logarithmiert und daraufhin mit matplotlib in Python visualisiert. Die Werte wurden mit der Funktion `genfromtxt()` aus der .csv Datei in das Programm eingelesen. Die Distanz in cm wurde mit der numpy Funktion `np.log()` logarithmiert. Die durchschnittlichen Spannung in Volt wurde ebenfalls mit `np.log(np.mean())` logarithmiert. Während die x-Achse mit dem logarithmierten Wert von dem Abstand belegt wird, initialisiert man die y-Achse mit dem logarithmierten Durchschnitt der Spannung in Volt.

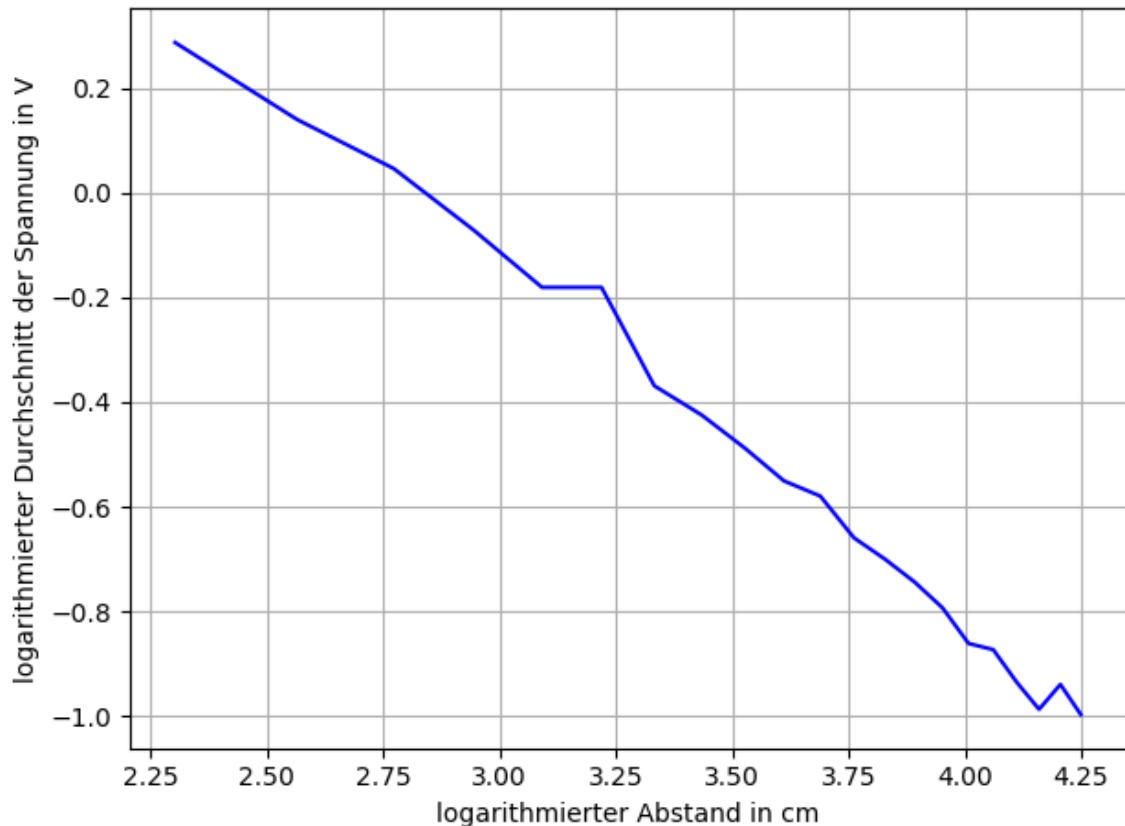


Abbildung 3.1: Logarithmus der Ein- und Ausgangswerte

In der folgenden Abbildung wurde die Kennlinie visualisiert mit matplotlib in Python. Da die Werte bereits eingelesen wurden in der vorherigen Darstellung müssen wir dies nicht erneut machen. Hier werden erneut die logarithmierten Abstände in cm verwendet, sowie der logarithmierte Durchschnitt der Spannung in Volt. Ebenfalls brauchen wir die Steigung. Wir benötigen die folgende Formel, welche auf dem Aufgabenblatt auf Seite 3 vorzufinden ist:  $y' = \ln y = \ln(x^a) = a * \ln(x) = a * \ln(e^{x'}) = a * x'$

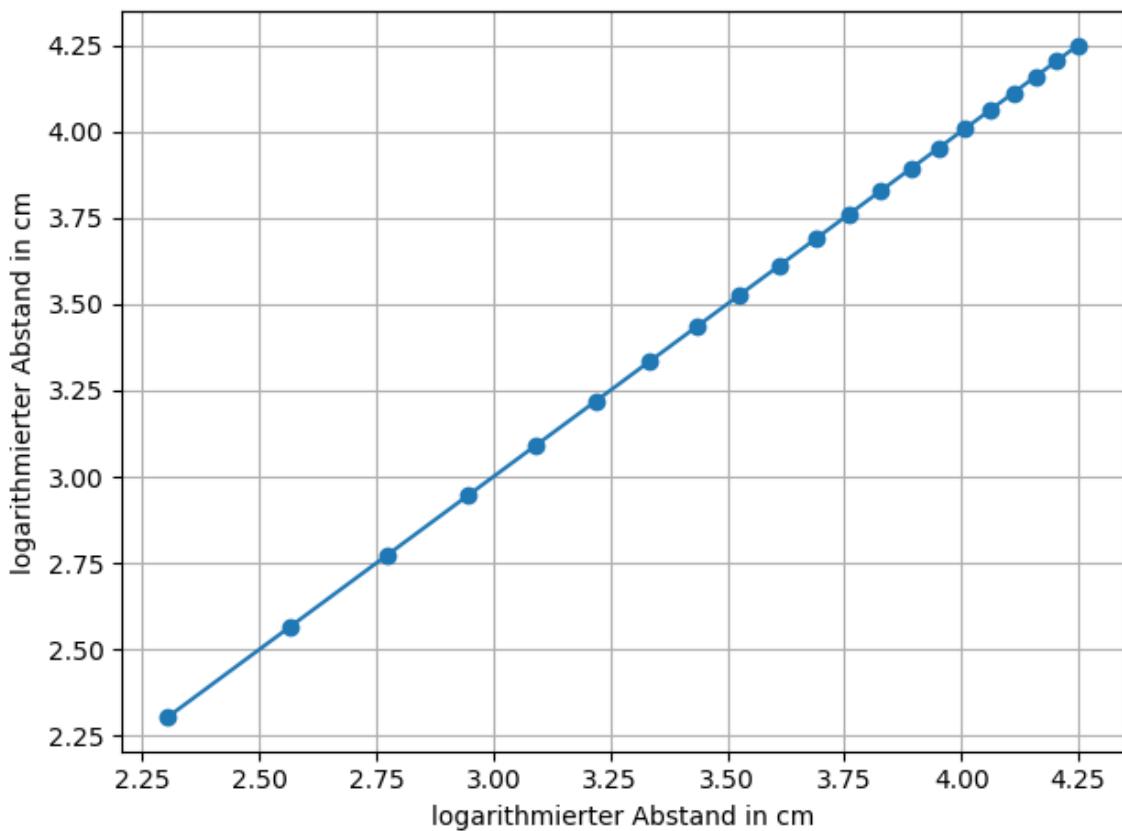


Abbildung 3.2: Kennlinie

In der folgenden Abbildung wurde die Lineare Regression realisiert mit matplotlib in Python. Auch hier wurden die Werte die Werte bereits eingelesen, weshalb dieser Schritt entfällt.

Hier werden erneut die logarithmierten Abstände in cm verwendet, sowie der logarithmierte Durchschnitt der Spannung in Volt. Sämtliche logarithmierten Abstände in cm wurden in ein Array gemacht, was später für die x-Achse verwendet wird. Die Werte für die logarithmierten Durchschnitte der Spannung in Volt wurden ebenfalls in ein separates Array gepackt, welche später wiederum für die y-Achse verwendet werden.

Danach wurde stats.linregress() in folgender Darstellung verwendet:

```
gradient, intercept, r_value, p_value, std_err = stats.linregress(x, y)
```

Danach wird noch die folgende Formel angewendet:  $y' = a * x' + b$

In Python sieht diese Formel folgendermaßen aus:  $y1 = gradient * x1 + intercept \& x1 = np.linspace(mn, mx, 500)$   $y1$  wird mit  $x1$  als Ausgleichsgerade der linearen Regression angezeigt.  $x$  und  $y$  wurden bereits vorher in ein Array gespeichert und nun als Punkte angezeigt. Die Ausgleichsgerade macht den Abstand zu den Punkten im Gesamten minimal.

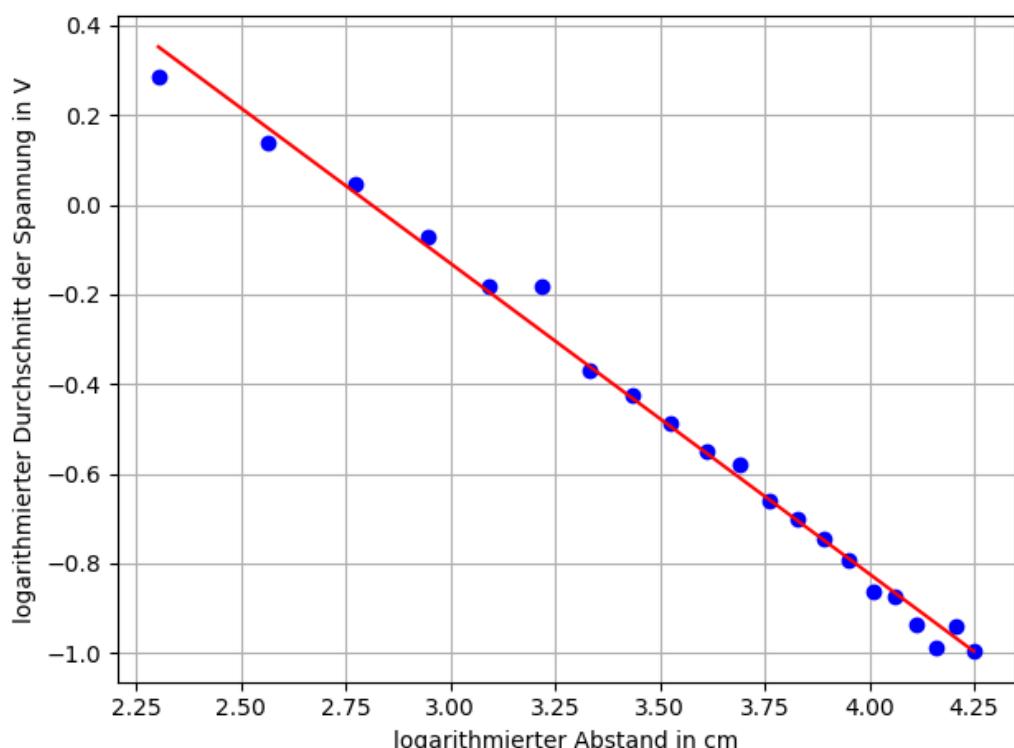


Abbildung 3.3: Lineare Regression

## 3.4 Interpretation

In Abbildung 3.1 sieht man nur wenige Unstimmigkeiten. Bei der x-Achse an der Stelle 3.25cm liegt der Fehler an der Messung, für die unsere Messung verantwortlich ist. Leider haben wir denselben Wert zweimal eingelesen, ohne die Single Sequenz zu aktualisieren, was zu diesem Ausschlag führt.

Kurz vor dem Ende der Visualisierung geht der Wert nach oben, was an einer Erhöhung der Spannung liegt. Dies ist auch sichtbar in der Tabelle 2.1. Vermutlich ist dies passiert durch veränderte Wetterverhältnisse, die die Lichtverhältnisse im Raum veränderten.

Die Kennlinie in Abbildung 3.2 weißt auch keine systematischen Fehler auf, da sie einer Kennliniensteigung beziehungsweise der Sensitivität eines Sensors entspricht.

Das heißt, dass bei einem doppelten Eingangswert auch der doppelte Ausgangswert ausgegeben wird.

In Abbildung 3.3 zieht sich die Ausgleichsgerade durch die einzelnen Punkte um den Abstand minimal werden zu lassen. Abgesehen von einzelnen Ausreißern wie zum Beispiel dem Wert bei 3.25cm, der durch einen Messfehler entstanden ist wie schon weiter oben berichtet, liegen die Punkte sehr nah an der Ausgleichsgeraden.

Dies lässt darauf deuten, dass die meisten Punkte durch den Abstandssensor richtig gemessen wurden.

# 4

## Versuch 3: Flächenmessung mit Fehlerrechnung

### 4.1 Fragestellung, Messprinzip, Aufbau, Messmittel

Im dritten Versuch ist eine Aufgabe die Ermittlung des Messfehlers des Abstandsmessers. Zuerst müssen wir die lange Seite eines DinA4 Blattes vom Sensor müssen und danach das Resultat als .csv Datei speichern.

So müssen wir zur Ermittlung des Messfehlers die Fehlerfortpflanzung durch die folgende Kennlinie berechnen:

$e^b * x^a$  Zudem muss man berechnen wie groß der Vertrauensbereich für eine Sicherheit von 68% und für eine Sicherheit von 95% berechnen. Danach sollten wir die Fehlerfortpflanzung dazu benutzen um das Ergebnis der Abstandsmessung in cm angeben zu können.

Im zweiten Teil des dritten Versuchs müssen wir nun die breite Seite eines DinA4 Blattes messen. Daraufhin sollen wir mit der selben Methode die Breite des Blattes berechnen und die Fläche des Blattes berechnen. Dies geschieht mithilfe dem Gaußschen Fehlerfortpflanzungsgesetz.

## 4.2 Messwerte

Tabelle [4.2] zeigt die durchschnittliche Spannung in Volt für die breite und die lange Seite eines DinA4 Blattes.

Distanz	durchschnittliche Spannung in Volt
29,7cm	0.6583416425026114V
21cm	0.8572027683256641V

Tabelle 4.1: Messwerte DinA4

## 4.3 Auswertung

Tabelle [4.2] zeigt die berechneten Messfehler im Vertrauensbereich bei einer Distanz von 29,7cm. Für die Berechnung des Messfehlers bei den entsprechenden Vertrauensbereichen der Gaußverteilung benötigt man die folgende Formel:

$x = \text{Durchschnitt der Spannung in Volt} + \text{Korrekturfaktor} * \text{Standardabweichung} + (2sx \text{ oder } 4sx, \text{ je nach Vertrauensbereich})$

In Python wird die Formel folgendermaßen ausgeführt:

`result1 = np.mean(data2) + 1,95 * (np.std(data2) / np.sqrt(1000)) * 4` Der Vertrauensbereich bei 68% hat einen Korrekturfaktor von 1 und ist  $+/-1sx$  groß , während der Vertrauensbereich bei 95% einen Korrekturfaktor von 1,96 hat und  $+/-2sx$  groß ist.

`np.mean` ist hier eine numpy Funktion welche den Durchschnitt berechnet, wie hier die durchschnittliche Spannung in Volt für die lange Seite der DinA4 Seite.

`np.std` ist ebenfalls eine numpy Funktion und berechnet die Standardabweichung in cm für die lange Seite der DinA4 Seite.

Diese wird durch die numpy Funktion `np.sqrt(1000)` ist ebenfalls eine numpy Funktion und berechnet die Wurzel. In diesem Fall soll die Funktion die Anzahl der verwendeten Spannungen pro Messung die Wurzel ziehen.

Formel	Ergebnis
$68\% = 0.6583416425026114V + 1 * 0.0006520905314570869V * 2sx$	0.6596458235655256cm
$95\% = 0.6583416425026114V + 1,96 * 0.0006520905314570869V * 4sx$	0.6634540322692349cm

Tabelle 4.2: Messfehler in Gaußverteilung

Tabelle [4.2] zeigt die Ergebnisse der Abstandsmessung mittels der Fehlerfortpflanzung an. Verglichen werden die Gaußverteilungen von 68% und 95% mit den Distanzen eines DIN A4 Blattes. Um diese Formel zu berechnen benötigen wir noch  $a$  und  $b$ .

$a$  erhält man durch den Logarithmus aus der Länge bzw. der Breite des Blattes geteilt durch die Logarithmierung der durchschnittlichen Spannung:

$$a1 = np.log(29.7) / np.log(np.mean(data2))$$

$b$  wiederrum erhält man aus dem Logarithmus der Distanz subtrahiert von der Multiplikation von dem Logarithmus der durchschnittlichen Spannung in Volt und  $a$ :

$$b1 = np.log(29.7) - (a1 * np.log(np.mean(data2)))$$

Um die Abstandsmessung mit der Fehlerfortpflanzung zu berechnen benötigt man zuerst die folgende Formel:

$$e^b * x^a.$$

Bzw. deren Ableitung:  $e^b * a * x^{a-1}$ . Nun nur noch die errechneten Werte einfügen.

Wenn man ein Vertrauensintervall von 68% verwendet, muss man nur noch die abgeleitete Funktion mit der Standardabweichung geteilt durch die Wurzel mit der Anzahl der Messungen multiplizieren und schon hat man den Vertrauensintervall. Für den Vertrauensintervall von 95% hingegen muss man noch zusätzlich den Vertrauensfaktor von 1,96 multiplizieren.

Sicherheit	Distanz	Abstandsmessung
68%	21cm	0.31835908731736245cm
68%	29,7cm	0.23864419265666054cm
95%	21cm	0.6239838111420121cm
95%	29,7cm	0.4677426176070423cm

Tabelle 4.3: Abstandsmessung mit Fehlerfortpflanzung

Wenn man die Länge und die Breite eines DinA4 Blattes multipliziert, erhält man den Flächeninhalt.

$$29,7\text{cm Länge} * 21\text{cm Breite} = 623,7\text{cm}^2.$$

Damit hat ein DinA4 Blatt einen Flächeninhalt von  $623,7 \text{ cm}^2$ .

Um den Messfehler in einem Vertrauensbereich von 68% mit  $623,7\text{cm}^2$  Flächeninhalt zu berechnen muss man nur die Werte von 68% addieren:

$$0.557003279974023\text{cm}.$$

Um den Messfehler in einem Vertrauensbereich von 95% hat mit  $623,7\text{cm}^2$  zu berechnen muss man nur die Werte von 95% addieren:

$$1.0917264287490545\text{cm}.$$

## 4.4 Interpretation

Die Messfehler auf den Flächeninhalt eines DinA4 Blattes bezogen

Vertrauensintervall	Messfehler
68%	0.557003279974023cm
95%	1.0917264287490545cm

Tabelle 4.4: Messfehler im Gaußsche Intervall

Die Höhe der Messunsicherheit hängt mit der Anzahl der Messungen zusammen: Zum Beispiel bei einem Vertrauensintervall von 95%, aber nur 2 Messungen gibt es einen Korrekturfaktor von 12,71.

Zudem gilt: Desto höher der Vertrauensintervall, desto höher sind auch die Werte. Dies trifft hier, als auch in Tabelle 4.3 zu, da die Messunsicherheit hier bei 95% doppelt so hoch ist, wie die bei 68%.

Diese erhöhten Werte kommen durch einen erhöhten Korrekturfaktor.

# Anhang

## A.1 Quellcode

### A.1.1 Quellcode Versuch 1: Messung Kalibrierung

```
1 import cv2
2
3 # ----- Aufgabe1.1 -----
4
5 cap = cv2.VideoCapture(0)
6
7 cap.set(3, 640)
8 cap.set(4, 480)
9 cap.set(10, 200)
10 cap.set(11, 32)
11 cap.set(12, 32)
12 cap.set(14, 20)
13 cap.set(15, -4)
14 cap.set(17, 10000)
15
16 while (True):
17     ret, frame = cap.read()
18     gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
19     cv2.imshow("frame", gray)
20     for x in range(1, 11):
21         cv2.imwrite("data/bild" + str(x) + ".png", frame)
22     if cv2.waitKey(1) & 0xFF == ord("q"):
23         break;
24
25 # cv2.imwrite("data/bild.png", frame)
26
27 print("frame width: " + str(cap.get(3)))
28 print("frame height: " + str(cap.get(4)))
```

```

29 print("-----")
30 print("brightness: " + str(cap.get(10)))
31 print("contrast: " + str(cap.get(11)))
32 print("saturation: " + str(cap.get(12)))
33 print("-----")
34 print("gain: " + str(cap.get(14)))
35 print("exposure: " + str(cap.get(15)))
36 print("-----")
37 print("white balance: " + str(cap.get(17)))
38
39 cap.release()
40 cv2.destroyAllWindows()

```

Listing 5.1: Ermittlung der Kennlinie des Abstandssensors

```

1 import numpy as np
2 import cv2
3
4 # ----- Aufgabe1.2 -----
5
6 vec = np.zeros((5, 4))
7 crop = ["crop1", "crop2", "crop3", "crop4", "crop5"]
8 ten = -1
9
10 image = cv2.imread('data/Versuch1a.png')
11 image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
12
13 vec[0, 0] = 0
14 vec[0, 1] = 480
15 vec[0, 2] = 105
16 vec[1, 0] = 111
17 vec[1, 1] = 480
18 vec[1, 2] = 135
19 vec[2, 0] = 249
20 vec[2, 1] = 480
21 vec[2, 2] = 137
22 vec[3, 0] = 389
23 vec[3, 1] = 480
24 vec[3, 2] = 132
25 vec[4, 0] = 529
26 vec[4, 1] = 480
27 vec[4, 2] = 111
28

```

```

29 for z in range(1, 6):
30     ten += 1
31
32     y = 0
33     x = int(vec[ten, 0])
34     h = int(vec[ten, 1])
35     w = int(vec[ten, 2])
36
37     crop[ten] = image[y:y + h, x:x + w]
38     cv2.imshow("Crop" + str(z), crop[ten])
39     cv2.imwrite("bild" + str(z) + ".png", crop[ten])
40
41 cv2.waitKey(0)

```

Listing 5.2: Ermittlung der Kennlinie des Abstandssensors

```

1 import numpy as np
2 import cv2
3 import matplotlib
4
5 # ----- Aufgabe 1.3 -----
6 vec1 = np.zeros((5, 3))
7 vec2 = np.zeros((5, 4))
8 ten = -1
9
10 print("\hline")
11 print("Stufe & Mittelwert & Hex-Value & Standartabweichung \\\\")

12
13 for x in range(1, 6):
14     image = cv2.imread("data/bild" + str(x) + ".png")
15     ten += 1
16
17     b, g, r, a = cv2.mean(image)
18     b1 = b / 1000
19     g1 = g / 1000
20     r1 = r / 1000
21     hex = matplotlib.colors.to_hex([b1, g1, r1])
22
23     vec2[ten, 0] = x
24     vec2[ten, 1] = (b + g + r)/3
25     vec2[ten, 2] = np.std(image)
26
27     print("\hline")

```

```
28 print("Stufe " + str(x) + " & " + str(vec2[ten, 1]) + " & " + hex + " & " + str(vec2[ten, 2]) + " \\\\")  
29  
30 print("\hline")
```

Listing 5.3: Ermittlung der Kennlinie des Abstandssensors

## A.1.2 Quellcode Versuch 2

```
1 import numpy as np
2 import cv2
3
4 # ----- Aufgabe2.1 -----
5
6 anzahl = 0
7 mean = 0
8
9 vec1 = np.zeros((480, 640))
10 vec2 = np.zeros((480, 640))
11 vec3 = np.zeros((480, 640))
12 vec4 = np.zeros((480, 640))
13 vec5 = np.zeros((480, 640))
14 vec6 = np.zeros((480, 640))
15 vec7 = np.zeros((480, 640))
16 vec8 = np.zeros((480, 640))
17 vec9 = np.zeros((480, 640))
18 vec10 = np.zeros((480, 640))
19
20 average = np.zeros((480, 640))
21
22 blackpic = ["black1", "black2", "black3", "black4", "black5", "black6", "black7", "black8", "black9", "black10"]
23 vector = [vec1, vec2, vec3, vec4, vec5, vec6, vec7, vec8, vec9, vec10]
24
25 for x in range(0, 10):
26     blackpic[x] = cv2.imread('data/' + blackpic[x] + '.png', cv2.IMREAD_GRAYSCALE)
27
28     print("Datei: " + str(x + 1) + ".png erfolgreich")
29     for y in range(0, 480):
30         for z in range(0, 640):
31             b = blackpic[x][y, z]
32             vector[x][y, z] = b
33             anzahl += 1
34     print("-----")
35
36 for y in range(0, 480):
37     for z in range(0, 640):
38         mean = 0
39         for file in range(0, 10):
40             mean += vector[file][y, z]
```

```

41     mean = float(mean / 10)
42     average[y, z] = mean
43
44 image = cv2.imread("data/blackaverage.png")
45 for x in range(0, 480):
46     for y in range(0, 640):
47         image[x, y] = average[x, y]
48
49 cv2.imwrite("data/blackaverage.png", image)
50 cv2.imshow('image', image)
51
52 # Convert to YUV
53 image_contrast = cv2.cvtColor(image, cv2.COLOR_BGR2YUV)
54 # Apply histogram equalization
55 image_contrast[:, :, 0] = cv2.equalizeHist(image_contrast[:, :, 0])
56 # Convert to RGB
57 image_contrast = cv2.cvtColor(image_contrast, cv2.COLOR_YUV2RGB)
58
59 cv2.imwrite("data/contrastblackaverage.png", image_contrast)
60 cv2.imshow('image_contrast', image_contrast)
61 cv2.waitKey(0)
62 cv2.destroyAllWindows()

```

Listing 5.4: Lineare Regression

```

1 import numpy as np
2 import cv2
3
4 # ----- Aufgabe2.2 -----
5
6 korrigiertes_bild = np.zeros((480, 640))
7
8 grey = cv2.imread('data/Versuch1a.png')
9 image = cv2.cvtColor(grey, cv2.COLOR_BGR2GRAY)
10 blackavg = cv2.imread('data/blackaverage.png')
11
12
13 for y in range(0, 480):
14     for z in range(0, 640):
15         r1, g1, b1 = grey[y, z]
16         r2, g2, b2 = blackavg[y, z]
17         korrigiertes_bild[y, z] = int(r1) - int(r2)
18

```

```
19 image = cv2.imread("data/korrigiertes_bild.png")
20 for x in range(0, 480):
21     for y in range(0, 640):
22         image[x, y] = korrigiertes_bild[x, y]
23
24 cv2.imwrite("data/korrigiertes_bild.png", image)
25 cv2.imshow('image', image)
26 cv2.waitKey(0)
27 cv2.destroyAllWindows()
```

Listing 5.5: Lineare Regression

### A.1.3 Quellcode Versuch 3

```
1 import numpy as np
2 import cv2
3
4 # ----- Aufgabe3.1 -----
5
6 anzahl = 0
7 mean = 0
8
9 vec1 = np.zeros((480, 640))
10 vec2 = np.zeros((480, 640))
11 vec3 = np.zeros((480, 640))
12 vec4 = np.zeros((480, 640))
13 vec5 = np.zeros((480, 640))
14 vec6 = np.zeros((480, 640))
15 vec7 = np.zeros((480, 640))
16 vec8 = np.zeros((480, 640))
17 vec9 = np.zeros((480, 640))
18 vec10 = np.zeros((480, 640))
19 average = np.zeros((480, 640))
20
21 whitepic = ["white1", "white2", "white3", "white4", "white5", "white6", "white7", "white8", "white9", "white10"]
22 vector = [vec1, vec2, vec3, vec4, vec5, vec6, vec7, vec8, vec9, vec10]
23
24 for x in range(0, 10):
25     whitepic[x] = cv2.imread('data/' + whitepic[x] + '.png')
26     whitepic[x] = cv2.cvtColor(whitepic[x], cv2.COLOR_BGR2GRAY)
27
28 print("Datei: " + str(x + 1) + ".png erfolgreich")
29 for y in range(0, 480):
30     for z in range(0, 640):
31         b = whitepic[x][y, z]
32         vector[x][y, z] = b
33         anzahl += 1
34 print("-----")
35
36 for y in range(0, 480):
37     for z in range(0, 640):
38         mean = 0
39         for file in range(0, 10):
40             mean += vector[file][y, z]
```

```

41     mean = float(mean / 10)
42     average[y, z] = mean
43
44 image = cv2.imread("data/whiteaverage.png")
45 for x in range(0, 480):
46     for y in range(0, 640):
47         image[x, y] = average[x, y]
48
49 cv2.imwrite("data/whiteaverage.png", image)
50 cv2.imshow('image', image)
51
52 # Convert to YUV
53 image_contrast = cv2.cvtColor(image, cv2.COLOR_BGR2YUV)
54 # Apply histogram equalization
55 image_contrast[:, :, 0] = cv2.equalizeHist(image_contrast[:, :, 0])
56 # Convert to RGB
57 image_contrast = cv2.cvtColor(image_contrast, cv2.COLOR_YUV2RGB)
58
59 cv2.imwrite("data/contrastwhiteaverage1.png", image_contrast)
60 cv2.imshow('image_contrast', image_contrast)
61 cv2.waitKey(0)
62 cv2.destroyAllWindows()

```

Listing 5.6: Flächenmessung mit Fehlerrechnung

```

1 import numpy as np
2 import cv2
3
4 # ----- Aufgabe3.2 -----
5
6 white_minus_black = np.zeros((480, 640))
7
8 whiteavg = cv2.imread('data/whiteaverage.png')
9 blackavg = cv2.imread('data/blackaverage.png')
10
11
12 for y in range(0, 480):
13     for z in range(0, 640):
14         r1, g1, b1 = whiteavg[y, z]
15         r2, g2, b2 = blackavg[y, z]
16         white_minus_black[y, z] = int(r1) - int(r2)
17
18 image = cv2.imread("data/whiteminusblack.png")

```

```

19 for x in range(0, 480):
20     for y in range(0, 640):
21         image[x, y] = white_minus_black[x, y]
22
23 cv2.imwrite("data/whiteminusblack.png", image)
24 cv2.imshow('image', image)
25
26 # Convert to YUV
27 image_contrast = cv2.cvtColor(image, cv2.COLOR_BGR2YUV)
28 # Apply histogram equalization
29 image_contrast[:, :, 0] = cv2.equalizeHist(image_contrast[:, :, 0])
30 # Convert to RGB
31 image_contrast = cv2.cvtColor(image_contrast, cv2.COLOR_YUV2RGB)
32
33 cv2.imwrite("data/contrastwhiteaverage2.png", image_contrast)
34 cv2.imshow('image_contrast', image_contrast)
35 cv2.waitKey(0)
36 cv2.destroyAllWindows()

```

Listing 5.7: Flächenmessung mit Fehlerrechnung

```

1 import numpy as np
2 import cv2
3 import math
4
5 # ----- Aufgabe2.2 -----
6
7 korrigiertes_bild = np.zeros((480, 640))
8 norm = np.zeros((480, 640))
9
10 grey = cv2.imread('data/Versuch1a.png')
11 image = cv2.cvtColor(grey, cv2.COLOR_BGR2GRAY)
12 blackavg = cv2.imread('data/blackaverage.png')
13 whiteminusblack = cv2.imread('data/whiteminusblack.png')
14
15 for y in range(0, 480):
16     for z in range(0, 640):
17         r, g, b = whiteminusblack[y, z] - np.mean(whiteminusblack)
18         r1, g1, b1 = grey[y, z]
19         r2, g2, b2 = blackavg[y, z]
20         korrigiertes_bild[y, z] = int(r1) - int(r2)
21         korrigiertes_bild[y, z] /= r
22         print("pixelvalue:", korrigiertes_bild[y, z])

```

```

23     print("-----")
24
25 image = cv2.imread("data/korrigiertes_bild.png")
26 for x in range(0, 480):
27     for y in range(0, 640):
28         image[x, y] = korrigiertes_bild[x, y]
29     print("finish2")
30 cv2.imwrite("data/korrigiertes_bild2.png", image)
31 cv2.imshow('image', image)
32 cv2.waitKey(0)
33 cv2.destroyAllWindows()

```

Listing 5.8: Flächenmessung mit Fehlerrechnung

```

1 import numpy as np
2 import cv2
3
4 # ----- Aufgabe1.2 -----
5
6 vec = np.zeros((5, 4))
7 crop = ["crop1", "crop2", "crop3", "crop4", "crop5"]
8 ten = -1
9
10 image = cv2.imread('data/korrigiertes_bild2.png')
11 image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
12
13 vec[0, 0] = 0
14 vec[0, 1] = 480
15 vec[0, 2] = 105
16 vec[1, 0] = 111
17 vec[1, 1] = 480
18 vec[1, 2] = 135
19 vec[2, 0] = 249
20 vec[2, 1] = 480
21 vec[2, 2] = 137
22 vec[3, 0] = 389
23 vec[3, 1] = 480
24 vec[3, 2] = 132
25 vec[4, 0] = 529
26 vec[4, 1] = 480
27 vec[4, 2] = 111
28
29 for z in range(1, 6):

```

```

30 ten += 1
31
32 y = 0
33 x = int(vec[ten, 0])
34 h = int(vec[ten, 1])
35 w = int(vec[ten, 2])
36
37 crop[ten] = image[y:y + h, x:x + w]
38 cv2.imshow("Crop" + str(z), crop[ten])
39 cv2.imwrite("korrigiert" + str(z) + ".png", crop[ten])
40
41 cv2.waitKey(0)

```

Listing 5.9: Flächenmessung mit Fehlerrechnung

```

1 import numpy as np
2 import cv2
3 import matplotlib
4
5 # ----- Aufgabe 1.3 -----
6 vec1 = np.zeros((5, 3))
7 vec2 = np.zeros((5, 4))
8 ten = -1
9
10 print("\hline")
11 print("Stufe & Mittelwert & Hex-Value & Standartabweichung \\\\")

12
13 for x in range(1, 6):
14     image = cv2.imread("data/korrigiert" + str(x) + ".png")
15     ten += 1
16
17     b, g, r = cv2.mean(image)
18     b1 = b / 1000
19     g1 = g / 1000
20     r1 = r / 1000
21     hex = matplotlib.colors.to_hex([b1, g1, r1])
22
23     vec2[ten, 0] = x
24     vec2[ten, 1] = (b + g + r) / 3
25     vec2[ten, 2] = np.std(image)
26
27 print("\hline")
28 print("Stufe " + str(x) + " & " + str(vec2[ten, 1]) + " & " + hex + " & " + str(vec2[ten, 2]) + " \\\"")

```

```
29  
30 print("\hline")
```

Listing 5.10: Flächenmessung mit Fehlerrechnung

```
1 import numpy as np  
2 import cv2  
3  
4 # ----- Aufgabe3.1 -----  
5  
6 anzahl = 0  
7 cir1 = 1000  
8 cir2 = 1000  
9 cir3 = 0  
10 cir4 = 0  
11 value = 0  
12 high1 = 1000  
13 high4 = 0  
14  
15 vec1 = np.zeros((480, 640))  
16  
17 white1 = cv2.imread('data/Versuch1b.png')  
18 white1 = cv2.cvtColor(white1, cv2.COLOR_BGR2GRAY)  
19 value = np.std(white1)  
20  
21 image = cv2.imread("data/zzz2.png")  
22 for x in range(0, 480):  
23     for y in range(0, 640):  
24         value = white1[x, y]  
25         if (value < high1):  
26             cir1 = x  
27             cir2 = y  
28             high1 = value  
29  
30         if (value > high4):  
31             cir3 = x  
32             cir4 = y  
33             high4 = value  
34  
35  
36 print("Der dunkelste Punkt liegt bei: " + str(cir1) + " " + str(cir2))  
37 print("Der hellste Punkt liegt bei: " + str(cir3) + " " + str(cir4))  
38
```

```
39 cv2.circle(image, (cir2, cir1), 5, (142, 123, 228), 2)
40 cv2.circle(image, (cir4, cir3), 5, (46, 179, 66), 2)
41
42 cv2.imshow('image',image)
43 cv2.imwrite("zzz2.png", image)
44 cv2.waitKey(0)
45 cv2.destroyAllWindows()
```

Listing 5.11: Flächenmessung mit Fehlerrechnung

## A.2 Messergebnisse

<u>Values</u>	
Brightness	133
Contrast	32
Saturation	32
gain	20
exposure	-4
whitebalance	10000
Abstand von Kamera zu Grauwertteil:	
- 31.5 cm	

Abbildung 5.1: Messungen aus dem Labor