Technische Universität Berlin
Fakultät IV (Elektrotechnik und Informatik)
Institut für Energie- und Automatisierungstechnik
Fachgebiet Lichttechnik

Entwicklung einer Messeinrichtung auf Basis eines Einplatinencomputers und des Sensors AS7264N von ams

Bachelorarbeit

Vorgelegt von: Matthias Schaale-Segeroth
Matrikelnr.: 347330
Studiengang: B.Sc. Elektrotechnik

Eingereicht am: 28. August 2019

Betreuung: Nils Weber
Dr. Martine Knoop

Prüfer/in: Prof. Dr.-Ing. Stephan Völker
Prof. Dr.-Ing. Sibylle Dieckerhoff

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

Name	Datum	Unterschrift
Matthias Schaale-Segeroth		

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1 Einleitung

Als Grundlage zur Erforschung des Tageslichtes ist eine spezifische Kenntnis über die Zusammensetzung des Lichtes obligatorisch. Dieses Tageslichtspektrum besteht aus allen Wellenlängen der elektromagnetischen Strahlung des sichtbaren Bereiches (Licht) ($380 - 780\,\mathrm{nm}$), welche vom Himmel und somit direkt oder indirekt von der Sonne emittiert wird.
Ein Tageslichtspektrum zu messen ist nicht trivial. Ein für diese Anwendung entwickelter Sky-Scanner, wie es ihn an der TU-Berlin gibt, misst die spektrale Leuchtdichte des Himmels in 145 Himmelsbereichen mit einem Wellenlängenintervall von maximal einem Nanometer. Diese komplexe Verfahrensweise führt zu einem sehr hohen Anschaffungspreis im 6-Stelligen Euro Bereich und benötigt zusätzlich eine umfangreiche Kalibrierung. Diese Eigenschaften stehen einer flächendeckenden Messung des Tageslichtspektrums mittels des Sky-Scanners im Wege.

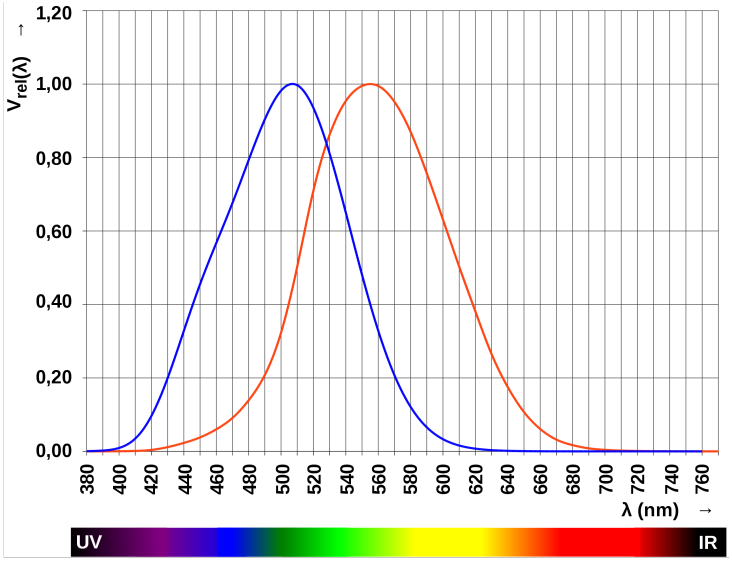
Eine andere Möglichkeit das Tageslichtspektrum zu bestimmen ist in Abschnitt 2 dargelegt. Hierbei wird lediglich der Farbort des Tageslichtes unabhängig seiner Zusammensetzung gemessen. Da das Tageslicht nur einen auf den 'CIE Daylight Locus' beschränkten Farbort annehmen kann, ist es möglich, anhand von Referenzmessungen das Tageslichtspektrum aus diesem zu rekonstruieren. Zur Bestimmung des Farbortes ist es nun ausreichend mit Hilfe von Farbsensoren das Tageslicht zu messen.

Das Thema dieser Arbeit ist die Entwicklung einer Messeinrichtung, welche die Grundlage für einen variablen Messaufbau zur Ermittelung des Farbortes des Tageslichtes bildet. Die Variabilität soll gewährleistet werden, indem die Anzahl der eingesetzten AS7264N Sensoren variabel zu gestalten ist und die Sensoren mit Platine eine platzsparende Bauform besitzen. Ebenfalls ist gefordert, dass das Benutzererlebnis der Messeinrichtung auf der Hardware-Ebene (Aufbau der Messinstrumente) sowie auf der Software-Ebene (Interaktion) intuitiv gestaltet sein soll.
Die Ergebnisse der Messungen sollen auf einer SD-Karte als `csv`-Datei abgespeichert werden können. Dies kann entweder im 'logging'-Betrieb, in welchem lediglich eine Stromversorgung benötigt wird, erfolgen, oder aber über einen zusätzlichen externen Computer, welcher mittels einer USB-Schnittstelle parallel auslesen kann, realisiert werden.

2 Lichttechnische Grundlagen

In diesem Abschnitt sollen kurz die lichttechnischen Grundlagen als theoretisches Fundament der vorliegenden Bachelorarbeit erläutert werden. Hierbei wird vor allem auf die Möglichkeit der Rekonstruktion des Tageslichtspektrums auf Grundlage von Normfarbwertanteilen eingegangen, um den Einsatzbereich der Messeinrichtung zu verdeutlichen. Das menschliche Auge nimmt die Intensität einer Lichtquelle und somit ihre Helligkeit in Abhängigkeit ihrer Farbe wahr. Die Helligkeitsempfindung aufgetragen über den Wellenlängen monochromatischer elektromagnetischer Strahlung formt die Hellempfindlichkeitskurve $v(\lambda)$, welche in Abbildung 1 dargestellt ist. Der rote Verlauf $v(\lambda)$ stellt das Tagessehen dar und die blaue $v'(\lambda)$ Kurve beschreibt das Nachtsehen[1].

Abbildung 1: Die $v(\lambda)$ Kurve

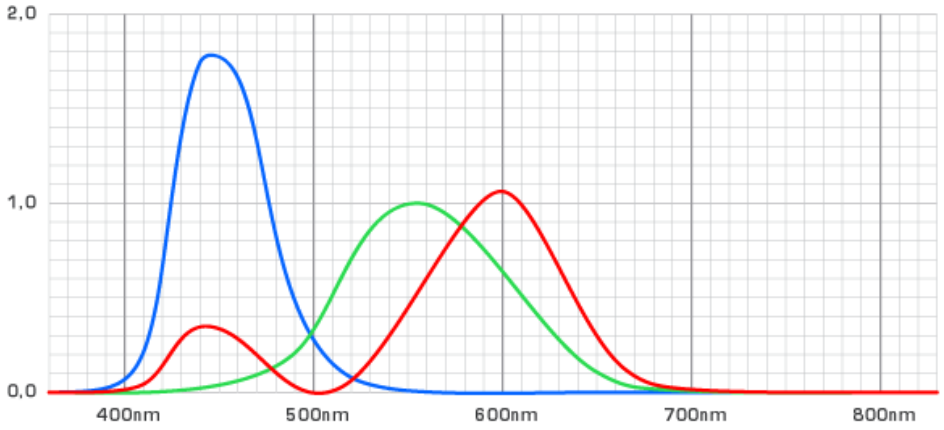


Quelle: `https://commons.wikimedia.org/wiki/File:V-lambda-phot-scot.svg`

Zur Darstellung des Farbeindrucks werden die CIE-xyz Normspektralwertfunktionen (Abbildung 2) genutzt. Diese bestehen aus drei Parametern (Farbvalenzen), welche eine dreidimensionale Beschreibung $(\overline{x}, \overline{y}, \overline{z})$ der Farbwahrnehmung vornehmen. Hervorzuheben ist, dass $\overline{y}(\lambda)$ der $v(\lambda)$ Kurve gleicht, die Skalierung der übrigen Farbvalenzen vorgibt, und zusätzlich Aufschluss über die Helligkeitsempfindung bietet.

[1] Bear/Barfuss/Seifert, *Beleuchtungstechnik - Grundlagen*, S. 21.

Abbildung 2: Die Normspektralwertfunktionen



Quelle: `https://commons.wikimedia.org/wiki/File:CIE_Tristimul.png`

Über die Normspektralwertfunktionen werden die Farbwerte (X,Y,Z) des Lichts mit der spektralen Strahldichte $\varphi(\lambda)$ nach den Formeln 2.0.1 bis 2.0.3 bestimmt, wobei k dem photometrischen Strahlungsäquivalent $k = 683\,\frac{lm}{W}$ entspricht. Durch diese Umrechnung sind die Farbwerte rein lichttechnische Größen[2].

$$X = k \cdot \int_{380nm}^{780nm} \varphi(\lambda) \cdot \overline{x}(\lambda) \qquad (2.0.1)$$

$$Y = k \cdot \int_{380nm}^{780nm} \varphi(\lambda) \cdot \overline{y}(\lambda) \qquad (2.0.2)$$

$$Z = k \cdot \int_{380nm}^{780nm} \varphi(\lambda) \cdot \overline{z}(\lambda) \qquad (2.0.3)$$

$$x = \frac{X}{X+Y+Z} \qquad (2.0.4)$$

$$y = \frac{Y}{X+Y+Z} \qquad (2.0.5)$$

Aus den Normfarbwerten werden nun die Normfarbwertanteile bestimmt, indem die einzelnen Normfarbwerte ins Verhältnis zur Summe der Normfarbwerte gestellt werden (Formeln 2.0.4 & 2.0.5). Der Vorteil der anteiligen Darstellung ist durch die Notwendigkeit begründet, nur die relativen x,y Werte angeben zu müssen, da sich die Summe aller Anteile zu eins addieren muss. Ein weiterer Vorteil dieser Konvention ist das Kürzen von $\varphi(\lambda)$ als Faktor im Zähler wie im Nenner. Als Indikation der Helligkeit ist es ausreichend, lediglich den Y-Normfarbwert anzugeben. Die Normfarbwertanteile $x\,\&\,y$ spannen den Farbraum auf, welcher durch die Purpurlinie und den Spektralfarbenzug (monochromatisch) begrenzt wird. Die resultierende Farbe jedes multichromatischen Lichtes liegt nun in diesem Farbraum und wird durch den Farbort, bestehend aus den Normfarbwertanteilen $x\,\&\,y$, hinreichend charakterisiert.
Aus dem Farbort des Tageslichtes, ist es möglich, dessen Spektrale Verteilung zu rekonstruieren. Das Tageslicht ist auf diskrete Farbörter beschränkt, welche durch den CIE-Daylight Locus beschrieben werden[3]. Allein durch die Annahme der Beschränktheit der Farbörter und eine daraus folgende eindeutige Zuordnung ist die Rekonstruktion der spektralen Verteilung möglich.

[2] Bear/Barfuss/Seifert, *Beleuchtungstechnik - Grundlagen*, S. 55.
[3] Judd/MacAdam/Wyszecki, *Spectral Distribution of Typical Daylight as a Function of Correlated Color Temperature*.

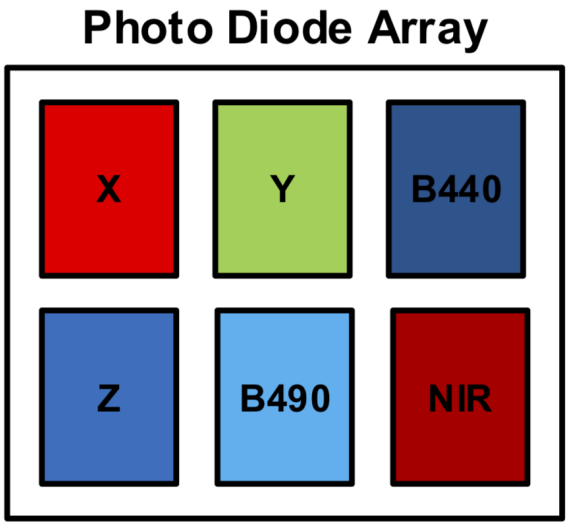
3 Hardware Komponenten

Die Auswahl der Hardware-Komponenten richtet sich nach vielen Kriterien. Das Wichtigste ist die Anwendbarkeit, also die Übereinstimmung der vom Bauteil theoretisch möglichen Leistung mit den Voraussetzungen im Anwendungsbereich. Hierzu zählen sowohl interne Fähigkeiten der Bauteile als auch eine platzsparende Bauform. Weitere Kriterien sind die Benutzerfreundlichkeit und die damit einhergehende einfache Reproduktion des Messaufbaus sowie der Preis.

3.1 Der AS7264N Sensor

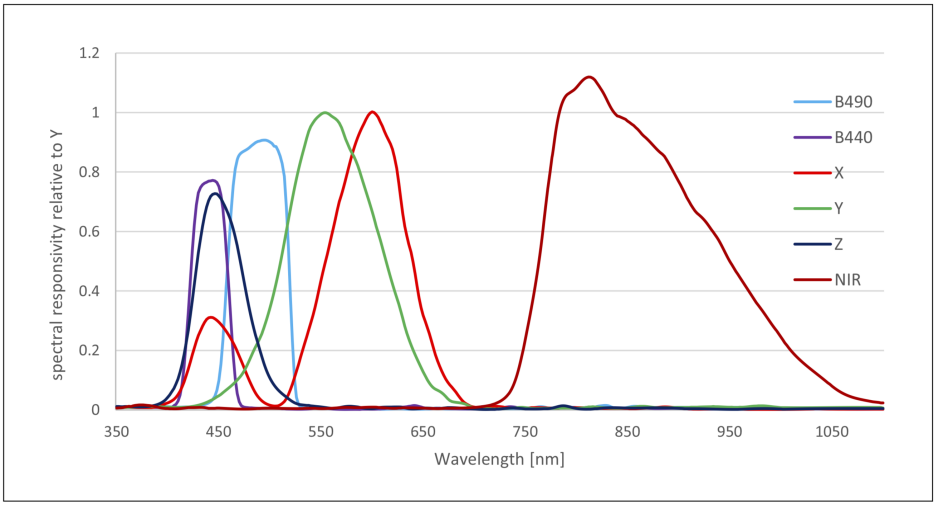
Der AS7264N ist ein Sensor, der elektromagnetische Strahlung unterschiedlicher Energie und somit Wellenlänge detektieren kann. Der Sensor misst mit sechs unterschiedlichen Fotodioden Farbspekren, welche im und um den menschlich wahrnehmbaren Spektralraum $\lambda = 380 - 780nm$ (Licht) liegen und in Abbildung 4 dargestellt sind. Für eine Rekonstruktion des Tageslichtspektrums werden nur die X , Y , Z Normfarbwerte bzw. deren prozentuale Zusammensetzung (Normfarbwertanteile) benötigt. Im Vergleich mit dem vom CIE veröffentlichten Verlauf der Normspektralwerte in Abbildung 2 fällt auf, dass die X und Y Verläufe mit den auf dem Datenblatt (Abbildung 4) angegebenen Verläufen übereinstimmen und nur der Z Wert in der Amplitude angepasst werden muss. Dies lässt die Vermutung zu, dass ein rechnerisch einfacher Rückschluss auf die tatsächlichen Normfarbwerte bzw., da die Kenntnis über das Verhältnis zueinander ausreicht, auf die Normfarbwertanteile möglich ist.

Abbildung 3: Fotodioden Anordnung



Quelle: https://ams.com/documents/20143/36005/AS7264N_DS000569_1-00.pdf/2888f82d-8c1f-2cfe-0ba1-ee8ed094d758#page=12

Abbildung 4: Spektralverläufe welche vom AS7264N gemessen werden



Quelle: `https://ams.com/documents/20143/36005/AS7264N_DS000569_1-00.pdf/2888f82d-8c1f-2cfe-0ba1-ee8ed094d758#page=9`

Zwei weitere positive Merkmale des AS7264N sind die Bauform, welche mit 6 mm · 6 mm · 2.5 mm sehr platzsparend ist und mit einem Preis von ca. 5€ günstig ausfällt. Dies lässt eine einfache Integration in einen gegebenen Messaufbau zu. Trotz der kleinen Abmessungen ist die Ansteuerung des Sensors über einen $I^2C - Bus$ möglich.
Der I^2C-Standard ist ein Bus System, welches von Philips Semiconductor 1982 entwickelt wurde. Es lässt die Kommunikation zwischen Master-Objekten (Mikrocontroller) und Slave-Objekten (AS7264N oder Real-Time-Clock) im Nahbereich zu. Dazu werden nur zwei Leitungen benötigt, die SCL-Line, welche die Taktfrequenz vorgibt, und die SDA-Line, welche für die Datenübertragung genutzt wird. Der AS7264N hat eine nicht veränderbare Bus-Adresse (0x49), somit kommt es zu Problemen der Doppelbelegung bei mehreren Sensoren (siehe Abschnitt 3.3).

3.2 Mikrocontroller

Die Auswahl der Mikrocontrollerunit (MCU) hängt vor allem von seinen Spezifikationen im Bereich Leistungsfähigkeit, dem internem Speicher sowie der physische Größe ab. Da die Dauer einer Abfrage des Sensors vor allem durch die Taktfrequenz bedingt wird, ist es, insbesondere im Hinblick auf die Kaskadierung der Sensoren und eine damit einhergehende Vervielfachung der Abfragezeit, von Vorteil den μC mit einer hohen maximalen Taktfrequenz zu wählen. Des Weiteren muss der Mikrocontroller in der Lage sein auf eine SD-Karte zu schreiben und einen großen internen Speicher besitzen.
Da der Compile- und Upload-Prozess auf Grund der Simplizität mit der Arduino IDE vorgenommen wird, sollte der μC diese unterstützen. Die Arduino IDE ist die Software-Anwendung des Arduino Projektes, welche lizenzfreie Soft- und Hardware mit Augenmerk auf Mikrocontrollern anbietet, und als Studentenprogramm an der 'Interaction Design Institute Ivrea' 2003 begonnen hat.

Tabelle 1: Vergleich von Arduino Nano mit Teensy 3.5

	Arduino Nano	**Teensy 3.5**
Abmessungen	43.18 mm × 18.54 mm	60.96 mm × 17.78 mm
Taktfequenz	16 MHz	120 MHz
Flash	32 KB	512 KB
RAM	2 KB	192KB
Preis	3€	25€

Da der von Arduino entwickelte 'Arduino-Nano', welcher als einzige Eigenproduktion den Größenverhältnissen entspricht, aber nicht den Ansprüchen an den internen Speicher gerecht wird, muss auf einen Teensy 3.5 zurückgegriffen werden. Dieser ist ein passender Kompromiss aus Platzsparendem Design und hoher interner Leistungsfähigkeit. Ein Vergleich der Spezifikationen ist in Tabelle 1 dargestellt.

Es ist gut zu erkennen, dass der Teensy 3.5 den Anforderungen im Hinblick auf eine Skalierung der Sensoren gerecht wird. Ein weiteres Feature ist der onboard SD-Slot, über welchen eine SD-Karte direkt eingeführt werden kann. Dies macht eine aufwändige Nachrüstung überflüssig.

Abbildung 5: Der Teensy 3.5

Quelle: `https://www.tinytronics.nl/shop/en/teensy/teensy-3.5-with-soldered-headers`

3.3 I2C Multiplexer

Wie in Abschnitt 3.1 erwähnt, kommt es zu Problemen der Adressierung, wenn man mehrere AS724N an einen Bus anschließt, da die Sensoren immer die Bus-Adresse 0x49 (hexadezimal) besitzen. Diese Mehrfachbelegung der Adressen kann nur verhindert werden, wenn jeder AS7264N an einen eigenen I^2C Bus angeschlossen wird. Hierzu werden Multiplexer verwendet, welche jeweils acht einzelne Bussysteme eröffnen und einzeln auf den Haupt-Bus zu und abgeschaltet werden können. Da an jedem einzelnen Bussystem des Multiplexers nur ein Sensor angeschlossen ist, kann somit jeder Sensor individuell auf den Haupt-Bus durchgeschleift und somit abgefragt werden.

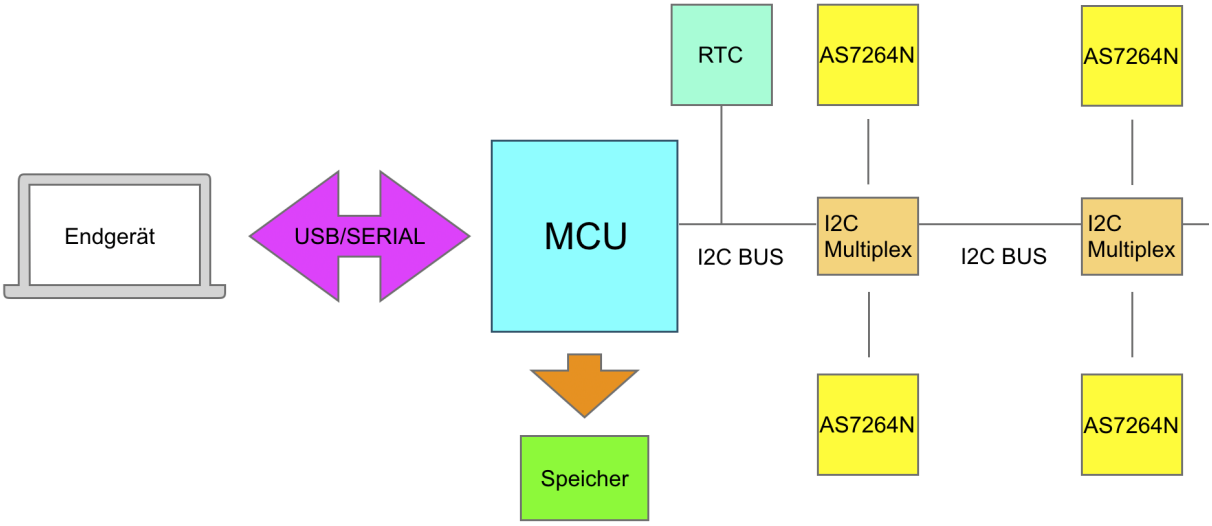


Abbildung 6: Schematischer Aufbau und Anschluss der I^2C Bauteile

Die Multiplexer werden auch über den I^2C Haupt-Bus gesteuert. Für die Kommunikation zu mehreren Multiplexern auf einem Bus System kann die Multiplexer Adresse in 3 bit beginnend bei 0x70 (hexadezimal) verstellt werden. Das ergibt eine maximale Anzahl von parallelen Multiplexern auf einem Bus von $(2^3 = 8)$ und somit eine maximale Anzahl der Sensoren:

$$8 \quad \text{Multiplexer} \cdot 8 \quad \text{Anschlüsse} = 64 \quad \text{Sensoren} \tag{3.3.1}$$

3.4 Real Time Clock

Die Real-Time-Clock (RTC) ist ein Zählwerk, welches seine Stromversorgung über eine externe Batterie gewährleistet. Sinn dieser RTC ist das Weiterzählen der Zeit nach dem diese einmal festgelegt wurde, auch wenn die Stromversorgung über den Mikrocontoller abbricht. Die Uhrzeit kann zu jedem Zeitpunkt über den I^2C Bus eingestellt und ausgelesen werden. Die Information der Uhrzeit ist im Logging-Modus unverzichtbar, dennoch sind RTC's nicht unproblematisch, da die taktgebenden Quartze im Bauteil Temperaturfühlig sind und es somit zu langfristigen Abweichungen von der richtigen Zeit kommen kann.

4 Software

Zum Auslesen der AS7364N Sensoren wird eine Software auf den Teensy 3.5 geladen, welche aus zwei Komponenten besteht. Die Hauptkomponente ist der Sketch für die Arduino IDE 'BachelorTeensy.ino', welche den Ablauf der Befehle steuert. Die zweite Komponente ist die eigens für diese Anwendung verfasste 'BaTeeLib.h' Library, welche die im vorherigen Teil aufgerufenen Funktionen definiert. In diesem Abschnitt wird zuerst der grundlegende Programmablauf erläutert, gefolgt von einer genauen Analyse des Programm-Code.

4.1 Programmablauf

Der Programmablauf des Teensy ist in Abbildung 7 dargestellt und beschreibt einen kontinuierlichen Kreislauf. Dabei steht die Funktion `SerialEvent` (siehe Abs.4.2.4), hier als `'GET NEW INPUT'`, im Mittelpunkt, da diese neu eingegebene Befehle verwaltet und in den `INPUT` übergibt. Falls kein neuer Befehl eingegeben wurde, wird dieser Abschnitt übersprungen. Im Folgenden wird der Wert von `MODE` ggf. verändert sowie der korrekte gelb hinterlegte Modus gefunden. Hierbei ist zu beachten, dass der `RECORDING MODE` nur eintritt falls die `acquisition time` erreicht ist, also das voreingestellte Intervall abgelaufen ist. Im `RECORDING MODE` sowie im `IDEPENDENT MODE` wird nun die Funktion `'GET all Sensor DATA'` aufgerufen, welche in Abbildung 8 dargestellt ist.

Der `PROGRAMMING MODE` führt entweder einen direkten Befehl aus (`'EXECUTE COMMAND OR SET CATEGORY'`) oder setzt Variablen in eine Kategorie (`'EXECUTE COMMAND FOR CATEGORY'`). Diese Kategorie muss zuvor durch einen Befehl ausgewählt werden. Eine genaue Beschreibung des Programmiervorgangs ist in Abschnitt 6.2 zu finden.

Im Abbildung 8 ist der Programmablauf der Datenakquise abgebildet, welche zuvor als `'GET all Sensor DATA'` aufgerufen wird. Der Ablauf der Sensor-Auslese wiederholt sich für jeden gefundenen Sensor und unterscheidet zwischen `AUTOMODE` an oder aus. Ohne diesen werden die manuell voreingestellten Werte als Parameter(`gain,intTime`) der Messung übernommen. Da nach einem Reset des Sensors in allen Registern eine Null steht, wird gewartet, bis ein neuer Wert im Register steht. So kann sicher gestellt werden, dass es sich um die zur Messung passenden Messwerte handelt, bevor diese auf die SD-Karte geschrieben werden.

Als Alternative zur Messung mit voreingestellten Parametern dient der `AUTOMODE`, welcher automatisiert den richtigen Verstärkungsfaktor und eine passende Integrationszeit ermittelt. Da sich die Register wie Zählwerke ohne Überlauf-Indikation verhalten, muss ein Überlauf des Registers verhindert werden. Um dies zu gewährleisten, wird der Sensor mit minimalen Parametern initialisiert, welche anschließend langsam erhöht werden, bis ein Registerwert von über 200 erreicht ist. Diese Vorgehensweise wird für jeden der X/Y/Z-Farbwerte wiederholt und die gefundenen Parameter abgespeichert. Final werden alle drei Messungen der unterschiedlichen Spektren mit den optimalen Parametern für die jeweilige spektrale Empfindlichkeit hintereinander durchgeführt. Somit ist eine Gleichzeitigkeit der einzelnen Farbwertmessungen gewährleistet, bevor diese auf der SD-Karte abgespeichert werden.

Nachdem für jeden Sensor eine Messung durchgeführt wurde, ist der Vorgang abgeschlossen(`DONE`) und der Programmablauf in Abbildung 7 beginnt von Anfang.

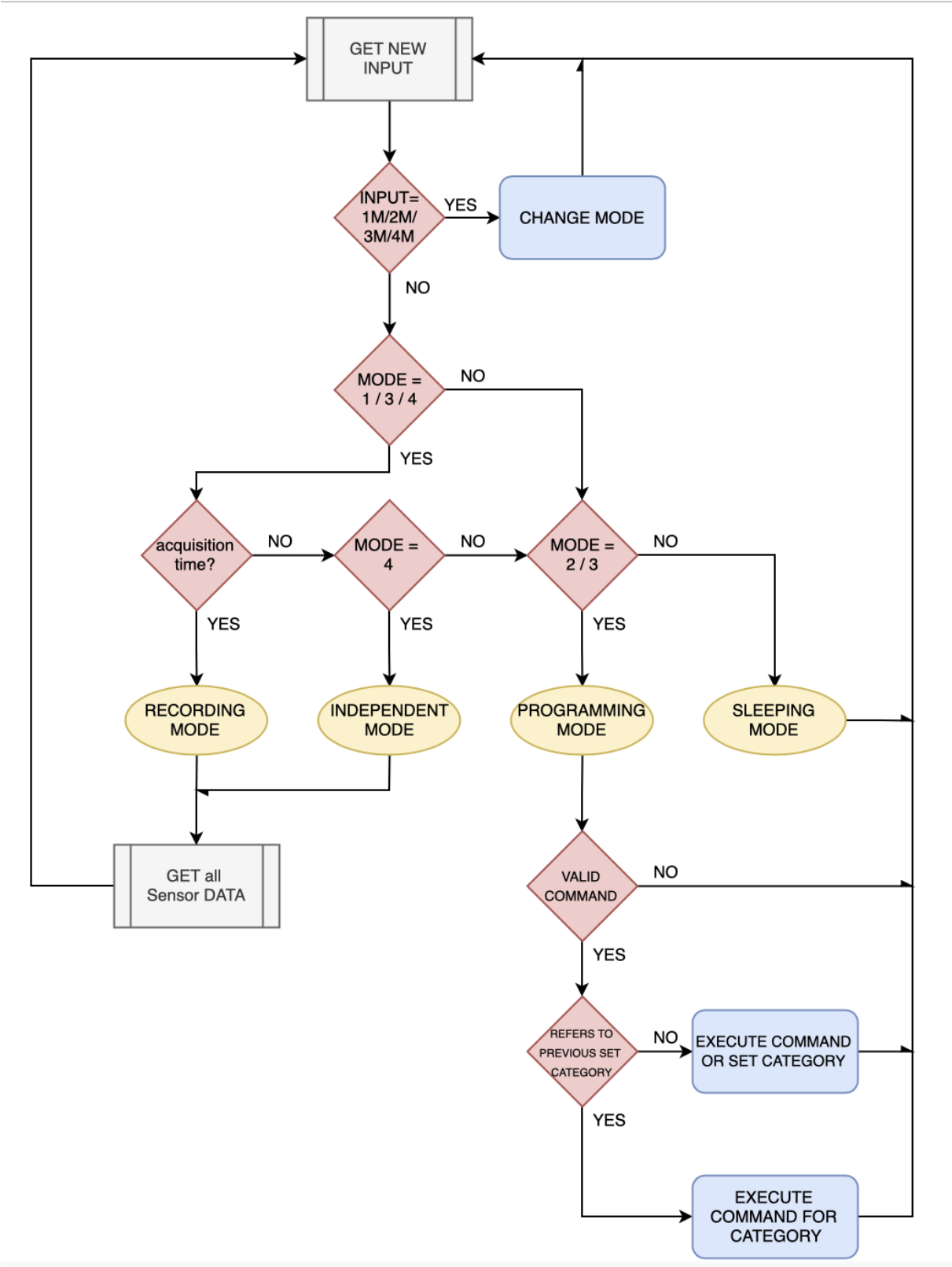


Abbildung 7: PAP

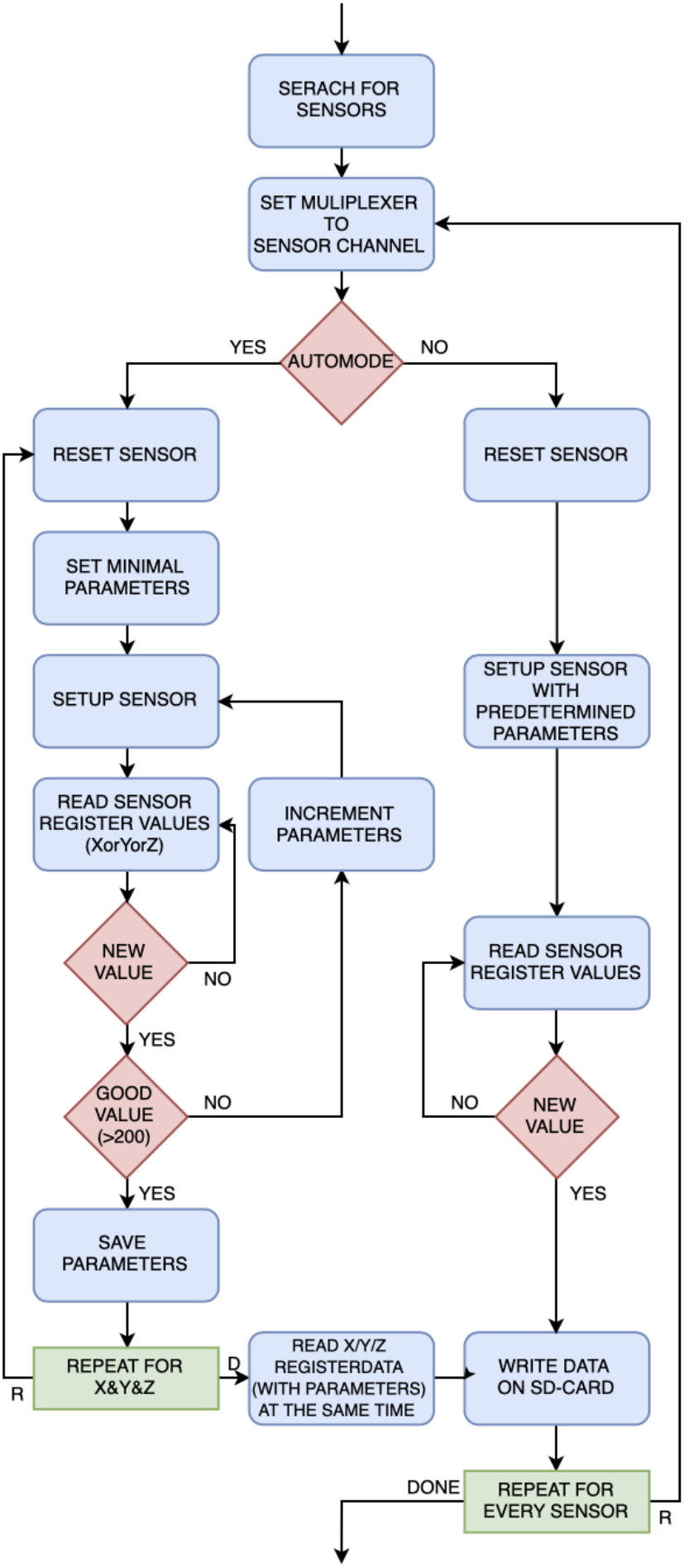


Abbildung 8: PAP DATA

4.2 Arduino Skript

Der Haupt-Code 'BachelorTeensy.ino' ist im Arduino-IDE (integrated development environment) Format mit dem Suffix '.ino' geschrieben, welches sich in der Programmiersprache nicht von C++ unterscheidet, sich aber an die Vorgaben des Arduino-Compilers halten muss. Ein '.ino' Format wird immer bei der Programmierung mit der Arduino-IDE erstellt und muss aus einem 'void setup', welches nur einmalig zu Beginn ausgeführt wird, und dem 'void loop', welcher als Schleife konstant wiederholt wird, bestehen. Die Bezeichnung void gibt an, dass aus dieser Funktion keine Werte zurückgegeben werden.

Listing 1: Grundgerüst eines Arduino-Codes

```
1  void setup() {
2    // put your setup code here, to
         run once:
3
4  }
5
6  void loop() {
7    // put your main code here, to
         run repeatedly:
8
9  }
```

4.2.1 Prefix, Library & globale Variablen

Das Programm fängt mit einem Präfix an, welches auskommentiert den Anlass, Anwendungsbereich, technische Voraussetzungen und den Autor angibt. Das Auskommentieren wird entweder durch /* begonnen und mit */ beendet, oder zeilenweise mit dem doppelten Schrägstrich: // vollzogen, was im Listing 1 als Lückenfüller der Funktionen angewandt wurde. Hervorzuheben sind die für den korrekten Ablauf des Programms notwendigen Libraries, welche auf der beigelegten CD zur Verfügung stehen. Diese Libraries sind, mit Ausnahme Einer, von Externen verfasst worden und stehen kostenfrei zum Download bereit (Liste in Tabelle 5). Da diese für den vorliegenden spezifischen Anwendungsfall teilweise angepasst wurden, ist die Entnahme über die CD vorzuziehen.
Die für diesen Anlass verfasste Library 'BaLibTEE.h' wird als erster Compiler Befehl eingefügt, welche die anderen Libraries bei Bedarf selber aufruft (siehe Abschnitt 4.3). Im Folgenden werden Objekte (engl: instances) aus den 3 Klassen der 'BaLibTEE.h' Library definiert: sensor, flash und interface. In Tabelle 2 ist ein Überblick auf die Objekte, Klassen und ihre Funktion gegeben, welcher in Abschnitt 3 vertieft wird.

Tabelle 2: Funktionsübersicht der Klassen

Objekt	**Klasse**	**Funktion**
sensor	AS7264N	I^2C Interaktion (mit Sensor & RTC)
flash	SDCARD	SD schreiben & Uhrzeit
interface	Teensy	Bedienoberfläche und Programmierung

Bevor das eigentliche Programm beginnt, werden globale Variablen initiiert. Der standardmäßig eingestellte Modus, in dem sich der Mikrocontroller nach einem Reboot befindet, kann in Zeile 30 verändert werden. Hier kann zwischen fünf Modi, welche in Tabelle 3 dargestellt sind, ausgewählt werden. Des Weiteren werden Strings zur Übergabe zwischen Funktionen als leer definiert und die booleschen Variablen 'erroroverwrite' und 'newcommand' auf false gesetzt.

Tabelle 3: Modi

Index	Modus
1	Recording
2	Programming
3	Rec. & Prog.
4	Independent
sonst	Sleeping

Listing 2: Initialisierung

```
1  /* This is the Teensy scatch using the Arduino IDE for the Bachelor thesis
2   * "Entwicklung einer Messeinrichtung auf Basis eines Einplatinencomputers und des
       Sensors AS7264N von ams"
3   * by Matthias Schaale-Segeroth
4   *
5   * For using this scribt the following libraries are needed
6   * and can be downloaded from this github page:
7   * - BaLibTEE.h     <-this Library is written by me for this occasion
8   * - EEPROM.h
9   * - Arduino.h
10  * - Wire.h
11  * - SD.h
12  * - SPI.h
13  * - TimeLib.h
14  * - DS1307RTC.h
15  */
16
17
18 //Beginning of the Sketch:
19
20 //inclusion of the libraries
21 #include <BaLibTEE.h> //my lib
22 #include <EEPROM.h>     //library for writing to internal memory
23
24 // Create instance of the classes provided by the 'BaLibTEE.h' library
25 AS7264N sensor;
26 SDCARD flash;
27 Teensy interface;
28
29 // introduction of the variables
30 int modus = 1;               //defauld mode of the Teensy
31 String inputString = "";     //inputString is empty
32 String commandString = "";   //commandString is empty
33 boolean newcommand = false; //no new command
34 bool erroverwrite = false;//no data while error
```

4.2.2 Void Setup

Wie in Abschnitt 4.2 beschrieben enthält ein Arduino Programm die einmalig zu Beginn aufgerufene Funktion 'void setup()', welche die Initialisierung von Funktionen übernimmt. Hier werden nach einer Verzögerung die LED-Pins als Outputs definiert und eine serielle Verbindung mit der Baud-Rate von 115200 bps aufgebaut. Ebenfalls muss das Objekt interface initialisiert werden und die Compiler-Zeit(__TIME__) sowie das Compiler-Datum(__DATE__) als aktuelle Zeit übernommen werden. Falls es sich um einen Reboot ohne vorherige Compilierung handelt, bleibt die Uhrzeit gleich. Im Anschluss wird nun das 'flash' Objekt der Klasse SDCARD initialisiert. Ein genauerer Einblick in die Initialisierung ist in der Tabelle 7 unter der Funktion initializing gegeben.

Listing 3: Setup & Loop

```
36 // Begin setup  (only run once at the beginning)
37 void setup() {
38   delay(1000);
39     pinMode(datapin, OUTPUT);    //LED pin init.
40     pinMode(ok_pin, OUTPUT);
41     pinMode(sd_error_pin, OUTPUT);
42     pinMode(i2c_error_pin, OUTPUT);
43
44     Serial.begin(115200);        //begin serial connection
45
46     Serial.println("RESTART");
47     Serial.println("--------------------");
48   interface.comptime(__TIME__,__DATE__);  //if newly uploaded sets to compiler time and
         calculates time difference
49   flash.initializing();                   //initializing of the SD-Card
50 }
51
52
53 // continues loop (repeats itself)
54 void loop() {
55
56   if (interface.problem() && !erroverwrite){
57       goto skiped;  //if there is a problem -> start again
58   }
59   //this part collects the data (RECORDING MODE)
60   if (modus == 1 || modus == 3 || modus == 4) {
61     if (interface.acquisition()|| modus == 4){  //if its the correct interval
62       Serial.println();
63       allSensorData();           //record the data
64     }
65   }
66
67   //this part enables the commands (PROGRAMMING MODE)
68   if (modus == 2 || modus == 3) {
69     if (newcommand == true) {   //  if a new command is entered
70       interface.command(commandString.toInt(), commandString);  //run the command ->(
           value,command)
71       commandString = ""; //erase command
72       newcommand = false; //no new comamnd
73     }
74   }
75
76   skiped:
77   delay(100);
78 }
```

4.2.3 Void Loop

Die Schleife `void loop()` wird, nachdem das Setup abgeschlossen ist, wiederkehrend ausgeführt und ist für den Funktionsablauf verantwortlich. Von dieser Ebene werden alle weiteren Unterfunktionen zur Aufnahme der Daten aufgerufen. Der Programmablaufplan ist in Abbildung 7 dargestellt.

In Zeile 56 wird zu Beginn abgefragt, ob das Interface ein Problem festgestellt hat. Dieses Problem kann sowohl das Fehlen einer SD Speicherkarte als auch das Fehlen der Real-Time-Cock bedeuten. In einem solchen Fall gibt die Funktion `interface.problem()` die Fehlermeldung auf dem seriellen Monitor aus und übergibt eine Eins (1 = `true`) an die Abfrage zurück, womit die void loop mit dem `return` Befehl sofort beendet wird. Dies geschieht so lange bis keine Fehlermeldung mehr vorliegt und der `return` Befehl übersprungen wird.

Der Nächste Block (Zeile 59-65) ist für die Datenakquise verantwortlich. Die äußere If-Abfrage ist erfüllt, falls der Modus eins oder drei ist und sich somit der Teensy nach der Tabelle 3 im Aufnahmemodus befindet. Darauf folgt die Abfrage nach der `interface.aquision()`

Funktion, welche berechnet, wann es Zeit ist, eine neue Messung zu beginnen. Dies tut sie, indem sie den Modulo aus dem einstellbaren Intervall und der aktuellen Zeit berechnet. Falls dieser Null ist, wird einmalig ein `true` an die Abfrage übergeben. Falls beide If-Bedingungen erfüllt sind, wird die Funktion `allSensorData()` aufgerufen, welche auch im Arduino Skript definiert ist und in Abschnitt 4.2.5 erläutert wird. Für `modus`=4 gilt dies nicht und eine kontinuierliche Aufnahme von Messdaten findet statt.

Der letzte Abschnitt entscheidet, ob man sich im Programmiermodus befindet, indem die Variable `modus` abgefragt wird, welche laut Tabelle 3 den Wert zwei oder drei betragen muss. Ebenso muss die Variable `newcommand` wahr sein. Dies ist der Fall, wenn ein neuer Befehl in der Funktion `serialEvent()` eingegangen ist (siehe folgenden Abschnitt 4.2.4). In diesem Fall wird die Funktion `interface.command()` mit den Übergaben des eingegebenen Zahlenwertes (`commandString.toInt()`) und des Befehls (`commandString`) aufgerufen. Dies passiert nur einmal pro Eingabe, da im Anschluss in den Zeilen 71 und 72 `commandString` und `newcommand` gelöscht werden.

4.2.4 SerialEvent

In jeder Iteration, nachdem die Funktion `loop()` ausgeführt wurde, wird auch die Funktion `serialEvent()` aufgerufen, welche die serielle Eingabe von Befehlen überprüft und diese gegebenenfalls der Funktion void loop bereit stellt.
Die `serialEvent()` Funktion besteht aus einer While-Schleife, welche, solange eine serielle Eingabe getätigt wird, durchlaufen und durch die Funktion `Serial.available()` überprüft wird (Zeile 82). Eine serielle Eingabe wird durch '\n' beendet (siehe If-Abfrage in Zeile 85). Solange werden in der While-Schleife die einzelnen Buchstaben/Zahlen der Eingabe hintereinander in den `inputString` geschrieben, wodurch eine vollständige Rekonstruktion der eingegebenen Zeile entsteht.
Der Teensy wird bedient, indem die eingebende Person den Modus verändert, wobei hierzu der Index nach Tabelle 3 gefolgt von dem Buchstaben 'M' im seriellen Monitor einzugeben ist. Nach der Vervollständigung des `inputString` wird in diesem nach der Stelle gesucht, an welcher der Buchstabe 'M' vorkommt (Zeile 86), und falls diese größer Null ist, also ein M eingegeben wurde, fortgefahren. Falls kein 'M' im `inputString` enthalten ist, wird der folgende Abschnitt übersprungen.
Um den Index des gewünschten Modus zu ermitteln, wird das Ergebnis der Funktion `inputString.toint()`, welche die Zahl des `inputSting` bis zum ersten Buchstaben liefert, in die Variable `modus` geschrieben. Nun werden die vier Modi abgefragt und der gewählte Modus ausgegeben. Im Falle der Modi mit Programmieranteil werden die weiteren Eingabemöglichkeiten durch den Befehl `flash.writeprogrammingcommands()` angezeigt.
Separat wird in Zeile 120 der `reset`-Befehl überprüft, welcher zufolge hat, dass die Einstellungen aus Tabelle 4 getroffen werden. Die Eingabe dieses Befehls ist aus jeder Ebene und zu jeder Zeit möglich. Der in Tabelle 4 angegebene AS7264N Modus bezieht sich auf die in Abschnitt 3.1 erwähnten sechs Fotodioden von denen im Modus 1 $\rightarrow$ (Bank = 0x80) nur die X,Y,Z Diodenregister ausgelesen werden.
Zum Ende der Funktion wird nun der `inputString` an den `commandString` zur Weiterverarbeitung im `void loop()` 4.2.3 übergeben und die Variable `newcommand` `true` gesetzt, womit der Eingang eines neuen Befehls angekündigt wird.

Tabelle 4: Reset Einstellungen

Variable	Registername	EEPROM-Register	RESET-Wert
AS7264N Modus	BANK	10	0x80
Gain	GAIN	20	0x03
Integration-Wartezeit	INT_WT	30	0x80
Interationszeit	INT_T	40	0x00
Intervall	inter	50	5 s
	inter+1	51	0 m
	inter+2	52	0 h

Listing 4: SerialEvent

```
80  // After every void loop run the seialEvent function is called
81  void serialEvent() {
82    while (Serial.available()) {        //if a new command is entered
83      char inChar = (char)Serial.read();
84      inputString += inChar;            //add all letters to inputString
85      if (inChar == '\n') {                 //if inputString is complete
86        if (inputString.indexOf("M") > 0) {   //searches for the 'M'
87          modus = inputString.toInt();        //the value in front of the M is set to mode
88          if (modus != 1 && modus != 2 && modus != 3 && modus != 4) {  //if modus is not
                1,2,3 go to sleep mode
89            Serial.println();
90            Serial.println("SLEEPING");
91            Serial.println();
92            erroverwrite=false;
93          }
94          if (modus == 1) {      //if modus is 1 go to recording mode (default)
95            Serial.println();
96            Serial.println("RECORDING MODE");
97            Serial.println();
98            erroverwrite=false;
99          }
100         if (modus == 2) {      //if modus is 2 go to programming mode
101           Serial.println();
102           Serial.println("PROGRAMMING MODE");
103           flash.writeprogcom();    //prints the programming commands
104           erroverwrite=false;
105         }
106         if (modus == 3) {      //if modus is 3 go to recording and programming mode
107           Serial.println();
108           Serial.println("RECORDING + PROGRAMMING MODE");
109           flash.writeprogcom();    //prints the programming commands
110           erroverwrite=false;
111         }
112         if (modus == 4) {      //if modus is 4 go to independent mode
113           Serial.println();
114           Serial.println("INDEPEDET MODE");
115           erroverwrite=true;
116         }
117         inputString = "";
118       }
119       // reset
120       if (inputString.indexOf("reset") >= 0) {  //if "reset" is written reset all
            parameters
121         interface.reset();  //reset function
122         erroverwrite=false;
123       }
124       newcommand = true;           //there is a new command
125       commandString = inputString;//define commandString
126       inputString = "";            //erase inputString
127     }
128   }
129 }
```

4.2.5 Hilfsfunktionen

Im Gegensatz zu den Funktionen `setup()`, `loop()` und `serialEvent()` werden die Hilfsfunktionen `allSensorData()` und `collectSensorData()` (Listing 5) ausschließlich von anderen Funktionen aufgerufen.
Die Funktion `allSensorData()` wird vom `void loop` aufgerufen, wenn Sensordaten ermittelt und abgespeichert werden sollen. Hierzu gibt diese den Zeitpunkt der Messung aus (Zeile 138) und schreibt das Ergebnis der Abfrage nach allen gefundenen Sensoren (`sensor.getSerachSrting()`) in den String sstr. Dieser beinhaltet nun abwechselnd die Nummer der Multiplexer (plex) und die Nummer des Kanals (ch), an welchem ein AS7264N Sensor gefunden wurde. Die For-Schleife führt nun für die Länge des sstr-Strings und somit der Anzahl der Sensoren die Funktion `collectSensorData()` für jeden Sensor mit den passenden Werten (`plex` & `ch`) aus. Nun wird die Funktion `flash.endnewheadder()` aufgerufen, welche jeden Aufnahmezyklus und das Schreiben der Überschriften auf der SD-Karte nach erstmaliger Ausführung beendet. Es ist zu erkennen, dass diese Funktion nur einmal nach dem Einfügen einer leeren SD-Karte bzw. dem Erkennen eines neuen Sensors benötigt wird.

Listing 5: Hilfsfunktionen

```
132 //------------------ the following functios are only run if triggert in other functions
        ------------------
133
134 // function saves and prints all data from every sensor
135 void allSensorData() {
136     Serial.println(flash.gettimeString());  //print timestamp
137
138     String sstr = sensor.getSearchString(); //gets adresses(plex'+'ch) of all conected
            sensors
139     for (int i=0; i< sstr.length();i=i+2) { //every adress has two digits -> repeats for
            each found sensor
140         byte plex = sstr[i] - 48;      //first digit of every adress is plex no. (-48
                because of ASCII)
141         byte ch = sstr[i + 1] - 48;    //second digit is channel number (-48 because of
                ASCII)
142         collectSensorData(plex, ch);  //run function below for every sensor
143     }
144     flash.endnewheadder(); // if all csv-headders are written the first time, stop
            writing them for next data aquision loop
145 }
146
147 // function writes sensor data of one sensor (at channel->ch and Miltiplexer->plex) on
        the SD-Card
148 void collectSensorData(byte plex, byte ch) {
149   if(flash.available()||erroverwrite){       // if SD-available
150       digitalWrite(datapin, HIGH);  // set indicator LED on
151       if(!erroverwrite){
152         flash.writeData(sensor.getMeasurement(plex, ch), plex, ch); //(Data, Multiplexer,
                Sensor) -> write Data to the SD-Card with the filepath: year/month/date/plex
               +ch
153       }
154       if(erroverwrite){
155         Serial.print("Sensor ");//prints sensor and channel
156         Serial.print(plex);
157         Serial.print(ch);
158         Serial.print(": ");
159         Serial.println(sensor.getMeasurement(plex, ch));//prints Data ->no SD-Card
                writing!
160       }
161       digitalWrite(datapin, LOW);    // set indicator LED off
162   }
163 }
```

Die Funktion `collectSensorData(plex,ch)` wird nur von `allSensorData()` aufgerufen und benötigt die Übergabe der Multiplexeradresse `plex` und des Kanals `ch`, an welchem der auszulesende Sensor angeschlossen ist. Falls die SD-Karte zu diesem Zeitpunkt erreichbar ist, wird der `Datapin HIGH` gesetzt. Dies initiiert die Stromversorgung einer LED, welche am `Datapin` (D13,OnboardLED) angeschlossen ist. Folgend wird die Funktion `flash.writedata(Datenstring,plex,ch)` aufgerufen, welche den Datenstring auf die SD-Karte unter dem Namen Channel:'`plex+ch`' in den Ordner des Aufnahmedatums abspeichert. In Zeile 152 wird der Daten-String durch die Funktion `sensor.getMeasurement(plex,ch)` ersetzt, welche die aktuellen Werte aus dem Sensor am Multiplexer (`plex` und dem Kanal (`ch`) ausliest und als String zurückgibt. Somit werden in Zeile 152 die Werte eines Sensors auf der SD-Karte in der korrekten Datei abgespeichert. Es ist zu erwähnen, dass bei jedem Speichervorgang die Werte auch automatisch auf dem seriellen Monitor ausgegeben werden. Im `INDEPENDENT MODE` ist overwrite auf `true` gesetzt, wodurch die Auslese der Daten in Zeile 159 aktiviert wird und lediglich auf dem seriellen Monitor ausgegeben wird. Im Folgenden wird die Onboard-LED wieder ausgeschaltet, sodass diese nur während der Datenakquise leuchtet.

4.3 Die BaLibTEE.h Bibliothek

Eine Bibliothek in der Programmiersprache C++ ist, wie auch im Falle von 'BaLibTEE.h', an dem Suffix '.h' zu erkennen und besteht aus zwei separaten Files: 'BaLibTEE.h' & 'BaLibTEE.cpp'. Hierbei bestimmt die '.h' Datei den Aufbau, bestehend aus der Initialisierung der Klassen und ihrer Funktionen sowie der Definition globaler Größen. Die Datei mit dem Suffix '.cpp' (für C++) definiert hingegen alle in der '.h' initialisierten Klassen und Funktionen separat.

Um die Library, welche durch den Befehl `#include <BaLibTEE.h>` eingefügt wird, benutzen zu können, muss der Ordner mit dem Namen 'BaLibTEE.h', bestehend aus den Files 'BaLibTEE.h' und 'BaLibTEE.cpp', in den Library Ordner im Arduino Verzeichnis hinzugefügt werden bzw. über den Library-Manager eingefügt werden. Dies gilt auch für alle Bibliotheken, welche von BaLibTEE.h aufgerufen werden und in der Tabelle 5 aufgelistet sind.

Tabelle 5: Bibliothekenliste

Bibliothek	**Funktionsbereich**
BaLibTEE.h	Für diese Anwendung
EEPROM.h	Nichtflüchtiger Speicher
Arduino.h	Arduino Umgebung
Wire.h	I^2C BUS
SD.h	SD-Karte
SPI.h	SD-Karte
TimeLib.h	Zeitverwaltung
DS1307RTC.h	Real-Time-Clock

Listing 6: Teensy Klasse in BaLibTEE.h

```
10 #include <Arduino.h>
11
12 class Teensy{
13 public:
14     Teensy();
15     void initializing();
16     void command (int value, String string);
17     bool problem ();
18     void reset();
19     void comptime(const char *t , const char *d);
20     void writeStringEEPROM(char add , String data);
21     String readStringEEPROM(char add);
22     bool acquisition();
23     void display();
24 };
```

Die Teensy Klasse wird im Listing 6 definiert und besteht nur aus Funktionen der Kategorie `public`, was einen Zugriff auf diese von außerhalb der Bibliothek zulässt. Fünf Funktionen sind `void` Funktionen, geben also kein Wert zurück. Die booleschen Funktionen `problem()` und `acquisition()` geben `true` oder `false` zurück.
Die beiden Funktionen `void writeStringEEPROM()` und `String readStringEEPROM()` ermöglichen es einen String auf den EEPROM-Speicher zu schreiben. Der EEPROM (electrically erasable programmable read-only memory)-Speicher ist ein nicht flüchtiger Speicher, behält also den gespeicherten Wert auch nach dem Reeboot bei. Der Teensy 3.5 besitzt einen Speicher von `4096 Bytes`, welche einzeln in Registern adressiert werden.
Im Anschluss werden die Werte der LED-Pins sowie die in Tabelle 4 dargestellten EEPROM-Register als Compiler-Befehl `#define` gesetzt, womit die Variablen-Namen nur als Lückenfüller fungieren und beim Compilier-Vorgang automatisch durch die zugeordneten Werte ersetzt werden.

Listing 7: SDCARD Klasse in BaLibTEE.h

```
26 class SDCARD{
27 public:
28     SDCARD();
29     void writeprogcom();
30     void writeData(String dataString,byte plex,byte channel);
31     void initializing();
32     void endnewheadder();
33     void newheadder();
34     bool i2cavailable();
35     bool available();
36     String gettimeString();
37     bool SDinitializing();
38     void timedif(const char *newtime);
39     bool settime(const char *str);
40     bool setdate(const char *str);
41 private:
42     String getyear();
43     String SDNameString(int addr);
44     String SDfilepath();
45     String getdateString();
46
47 const char *monthName[12] = {
48   "Jan", "Feb", "Mar", "Apr", "May", "Jun",
49   "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
50 };
```

In Listing 7 ist die SDCARD Klasse definiert. Diese besteht aus `public` sowie `private` Funktionen, welche nur aus der `SDCARD` Klasse selber aufgerufen werden können. Die `void`-Funktionen sowie `bool SDinitializing()` sind für das Benutzen der SD-Karte verantwortlich. Die restlichen Funktionen, einschließlich der Privaten, sind zur Zeitverwaltung und Interaktion mit der Real-Time-Clock zuständig. Hierfür wird der Pointer auf den unveränderlichen char: `const char *monthName[12]` eingeführt, welcher den Werten $0 - 11$ (um einen Monat verschoben) die Monatsnamen zuweist.
In der Klasse AS7264N (Listing 8) wird die Kommunikation mit dem Sensor betrieben. Diese Klasse ist gekennzeichnet durch zwei private Funktionen, welche die Byte genaue Interaktion mit den Registern des AS7264N steuern (`writeRegister() & readRegister()`) und nur aus den anderen Funktionen aufgerufen werden. Somit ist ein direktes Schreiben von Registern von außen nicht möglich und es können nur designierte Einstellungen vorgenommen werden. Die Adressen des I2C Bus, des EEPROM-Speichers und der AS7264N Register werden im Anschluss definiert.

Listing 8: AS7264N Klasse in BaLibTEE.h

```
3  class AS7264N {
4  public:
5      AS7264N();
6      void Search();
7      String getMeasurement(byte plex , byte channel);
8      String getSearchString();
9      bool i2cavailable();
10     void automatic(int para);
11     String makeautomaticmeasurement();
12     String getone(int anteil);
13     String getmanualmeasurement();
14 private:
15     void writeRegister(byte addr, byte val);
16     byte readRegister(byte addr);
17     void setupXYZ( byte gain, byte inttime, byte wtime);
18     void resetSensor();
19 };
```

4.4 BaLibTEE.cpp

Die `BaLibTEE.cpp` Datei ist in der Programmiersprache `C++` (Suffix `.cpp`) geschrieben, gehört zu der Library `BaLibTEE.h` und definiert diese. Im Aufbau folgt sie der `BaLibTEE.h` Datei und ist nach Klassen in drei Abschnitte `SDCARD)`, `Teensy (Listing 11)` und `AS7264N (Listing 12)` unterteilt.

Die `C++` Datei beginnt mit der Inklusion externer Bibliotheken, abgebildet in Listing 9. Eine Auflistung der Bibliotheken und deren Einsatzbereiche ist in Tabelle 5 dargestellt. Es werden zwei Objekte zur Verwendung der Klasse `tmElements_t` aus der Bibliothek `DS1307RTC.h` und der Klasse `File` aus der Bibliothek `SD.h` (Tab.6) erstellt.

Tabelle 6: Objekte der `BaLibTEE.cpp` Bibliothek

Bibliothek	Klasse	Objekt
DS1307RTC.h	tmElements_t	tm
SD.h	File	printfile

Die oben definierten Objekte können nun aus alles Klassen der `BaLibTEE.h` Bibliothek genutzt werden. Zusätzlich muss noch die boolesche globale Variable `newhead` als `false` initiiert werden, sodass ebenfalls aus jeder Klasse der Befehl zur neuen Ausführung der Überschriften für die csv-Datei erteilt werden kann.

Listing 9: Begin der BaLibTEE.cpp Datei

```
2  //Inclusion of the libraries
3
4  #include <BaLibTEE.h>    //<- to this c++ belonging Library
5  #include <EEPROM.h>      //library for writing to internal memory
6  #include <Wire.h>        //I2C library
7  #include <TimeLib.h>     //time managing library
8  #include <DS1307RTC.h>   //Real-Time-Clock library
9  tmElements_t tm;     //creation of tm element
10
11 //Inclusion of the SD-Card libraries
12 #include <SD.h>
13 #include <SPI.h>
14 File printfile; //creation of the file: printfile
15
16 //initiate global variable
17 boolean newhead = false;
```

Zur einfachen Betrachtung der Klassen von BaLibTEE.cpp und ihrer Funktionen sind diese nach Klassen aufgeteilt (in den Tabellen 7(SDCARD), 8(Teensy) und 9 (AS7264N) zusammengefasst dargestellt). Es werden hierbei gesondert die *Rückgabe* (Variablen, welche an die aufrufende Funktion zurückgegeben werden), und *Übergabe* (Variablen, welche an die ausführende Funktion zur Verwendung übergeben werden), betrachtet. Ebenfalls sind die Abläufe und das Ergebnis der Funktionen stichpunktartig aufgeführt und die Zeilenangaben der Funktionen in der BaLibTEE.cpp Datei aufgelistet.

Tabelle 7: Funktionen der Klasse: SDCARD

Name	Rückgabe	Übergabe / Name	Funktion	Zeile
initialising	void	-	-prüft ob SD-Karte erreichbar -schreibt Modus-Befehle seriell	23 – 45
settime	bool	const char (pointer) *str	-setzt str als neue Zeit -falls erfolgreich gibt true zurück	49 – 56
timedif	void	const char (pointer) *newtime	-berechnet Differenz aus newtime und alter Zeit -gibt Ergebnis auf seriellen Monitor aus	59 – 84
setdate	bool	const char (pointer) *str	-setzt str als neues Datum -falls erfolgreich gibt true zurück	87 – 102
gettimeString	String	-	-gibt String der aktuellen Zeit zurück	105 – 118
getdateString	String	-	-gibt String des aktuellen Datums zurück	121 – 131
available	bool	-	-prüft ob SD-Karte erreichbar ist -gibt true(erreichbar)/false zurück	135 – 145
writeprogcom	void	-	-gibt Programmierbefehle auf seriellen Monitor aus	148 – 166
SDNameString	String	int/addr	-erzeugt String aus Datenpfad, addr und Suffix '.csv' -gibt String zurück	169 – 177
SDfilepath	String	-	-erzeugt aus Jahr/Monat/Tag den Datenpfad -gibt den Datenpfad String zurück	180 – 186
getyear	String	-	-gibt aktuelles Jahr als String zurück	188 – 190
endnewheadder	void	-	-setzt boolsche Variable newhead auf false ->Überschrift der csv-Datei wird beendet	192 – 194
newheadder	void	-	-setzt boolsche Variable newhead auf true ->neue Überschrift wird benötigt	196 – 198
writeData	void	String/dataString byte/plex byte/channel	-erzeugt/öffnet Datei mit Name: SDNameString ->Datenpfad(Jahr/Monat/Tag)->Name(plex+channel) -schreibt Überschrift falls nicht vorhanden -schreibt aktuelle Zeit und dataString auf SD-Karte -gibt Sensor und dataString seriell aus	201 – 333

Tabelle 8: Funktionen der Klasse: `Teensy`

Name	Rückgabe	Übergabe / Name	Funktion	Zeile
`comptime`	`void`	`const char *t` `const char *d`	-prüft auf neu kompiliert oder alter reboot -falls neu kompiliert setzt Zeit auf `t` und Datum auf `d` ->sonst behält alte Zeit und zeigt auf seriellem Monitor an	349 – 368
`writeStringEEPROM`	`void`	`char add` `String data`	-schreibt String `data` auf EEPROM ->beginnt bei Adresse `add` und schreibt aufsteigend pro Buchstabe	371 – 380
`readStringEEPROM`	`String`	`char add`	-gibt String ausgelesen vom EEPROM zurück ->beginnt bei Adresse `add` und endet mit `'\0'`	383 – 396
`initialising`	`void`	-	-initialisiert pinMode->OUTPUT von: `datapin,ok_pin,sd_error_pin,i2c_error_pin` -initialisiert serielle Schnittstelle mit 115200 -setzt EEPROM-Register `'Error'` und `'Programm'` zu `Null` ->kein Fehler gefunden und kein Programmierbefehl eingegeben	398 – 408
`acquisition`	`bool`	-	-gibt `true` zurück falls Zeitpunkt für Datenakquise ->modulo aus Sekunden seit Mitternacht durch Sekunden ->des Intervalls muss null ergeben	410 – 422
`problem`	`bool`	-	-gibt true zurück wenn SD-Karte oder RTC nicht erreichbar ->falls SD-Problem setzt `sd_error_pin` zu `HIGH` ->falls RTC-Problem setzt `i2c_error_pin` zu `HIGH` ->sonst setzt `ok_pin` zu `HIGH`	424 – 472
`command`	`void`	`int value` `String string`	-prüft ob `string` ein Befehl ist /& führt aus -Befehle ohne Größe: ->`back,display,temperature,search` -Befehle mit einzustellender Größe (`value`) werden eingegeben: ->`interval,gain,intTime,waitTime` -...gefolgt von dem Wert (`value`)	474 – 674
`display`	`void`	-	- zeigt seriell alle Werte der einstellbaren Größen an ->`interval, gain, intTime, waitTime`	677 – 713
`reset`	`void`	-	-stellt alle einstellbaren Größen in den EEPROM-Registern zu den Werkseinstellungen nach Tabelle 4	716 – 730

Tabelle 9: Funktionen der Klasse: `AS7264N`

Name	Rückgabe	Übergabe/Name	Funktion	Zeile
getMeasurement	String	byte plex byte ch	-setzt Multiplexer auf richtigen channel -gibt makeautomatixmeasurement zurück (Automode) -gibt getmanualmeasurement zurück (sonst)	745-763
`getmanualmeasurement`	`String`	-	-gibt Messung mit eingestellten Parametern zurück	765 – 800
`Search`	`void`	-	-gibt alle gefundenen Sensoren auf Monitor aus	802 – 835
`getSearchString`	`String`	-	-gibt String mit allen gefundenen Sensoren zurück	838 – 865
`i2cavailable`	`boolean`	-	-true falls RTC erreichbar	868 – 876
`readRegister`	`byte`	byte addr	-gibt Registerinhalt der Adresse addr zurück	879 – 892
`writeRegister`	`void`	byte addr byte val	-setzt val in das Register addr des Sensors	895 – 900
`makeautomaticmeasurement`	`String`	-	-ruft automatic für X,Y,Z (0,1,2) auf -ruft getone für X,Y,Z (0,1,2) auf -gibt String alles Messungen zurück	903 – 924
`getone`	`String`	int anteil	-gibt Registerwerte der zuvor bestimmten Parameter zurück -wird pro X,Y,Z-Register einzeln aufgerufen	927 – 986
`automatic`	`void`	int para	-bestimmt die optimalen Parameter für X,Y,Z-Register -wird pro X,Y,Z-Register einzeln aufgerufen	989 – 1068
`setupXYZ`	`void`	byte gain byte inttime byte wtime	-initialisiert den Sensor mit den übergebenen Parametern	1070 – 1080
`resetSensor`	`void`	-	-startet den Sensor neu->somit alle Register leer	1082 – 1089

5 Platine

Die in Abschnitt 3 beschriebenen Bauteile müssen, um nach dem Schaltplan korrekt verbunden zu sein, auf eine Platine aufgebracht werden. Um einen möglichst freien Versuchsaufbau gewährleisten zu können, werden die Platinen in unterschiedliche Anwendungsbereiche aufgeteilt. Die Platinen wurden mit der Eagle Software erstellt. Dafür wird ein Schaltplan entwickelt, welcher dann mit dem Board-Manager entworfen wird.

5.1 Hauptplatine

Die Hauptplatine hat die Aufgabe den Teensy mit der Real-Time-Clock zu verbinden sowie eine Verbindung zu den Multiplexern zu bieten. In diesem Fall kann der erste Multiplexer mit der Adresse `[000]` → `0x70 (I2C Adresse in HEX)` ebenfalls auf die Hauptplatine gesteckt werden, sodass acht Sensoren ohne ein Multiplexer-Shield direkt angeschlossen werden können. Weiterhin kann der Aufbau der Hauptplatine auf bis zu sieben Shields erweitert werden, sodass wiederum jeweils acht weitere Sensoren ausgelesen werden können. Dieser Weg über die Multiplexer muss gewählt werden, da jeder AS7264N Sensor die selbe I^2C Adresse besitzt. Eine Dopplung auf dem Bus wird verhindert, indem die Multiplexer die Kommunikation mit nur einem Sensor gleichzeitig zulassen (Abs. 3.3).
Der Schaltplan für die Hauptplatine ist in Abbildung 11 dargestellt und zeigt den Teensy (links), die RTC (rechts unten) und die Anschlüsse für den Multiplexer (rechts oben), welcher an 8 Sensorausgänge angeschlossen ist. Wie bereits erwähnt ist der Multiplexer durch die Verbindung der Eingänge `A1-A3` mit Ground auf die Adresse `[000]` festgelegt. Ebenfalls können drei Indikations-LEDs hinzugefügt werden, welche den Status des Teensy anzeigen.
Allgemein ist zu sagen, dass alle Einzelteile, wie in Abbildung 20 zu erkennen ist, auf die Platine gesteckt werden. Im Schaltplan (Abbildung 11) sind diese farblich hinterlegt zusammengefasst. Somit wird die Fehlersuche vereinfacht und ein anspruchsvolles Anbringen der Bauelemente entfällt.

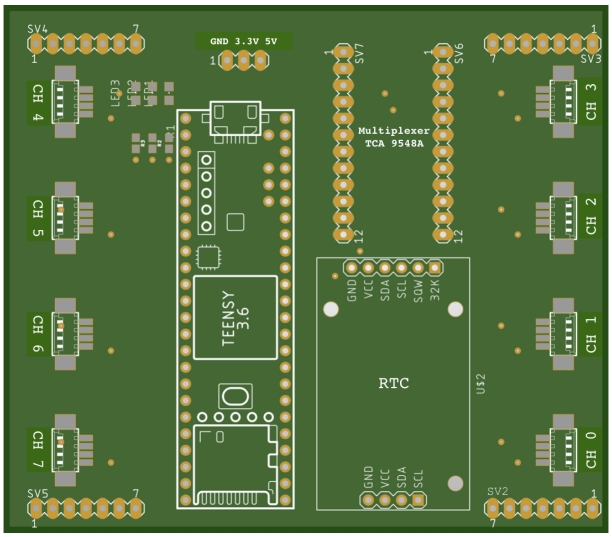


Abbildung 9: Tennsyplatine Top

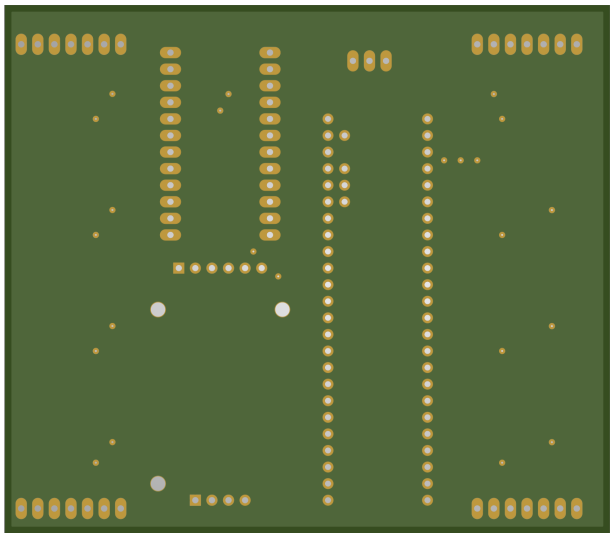


Abbildung 10: Tennsyplatine Bottom

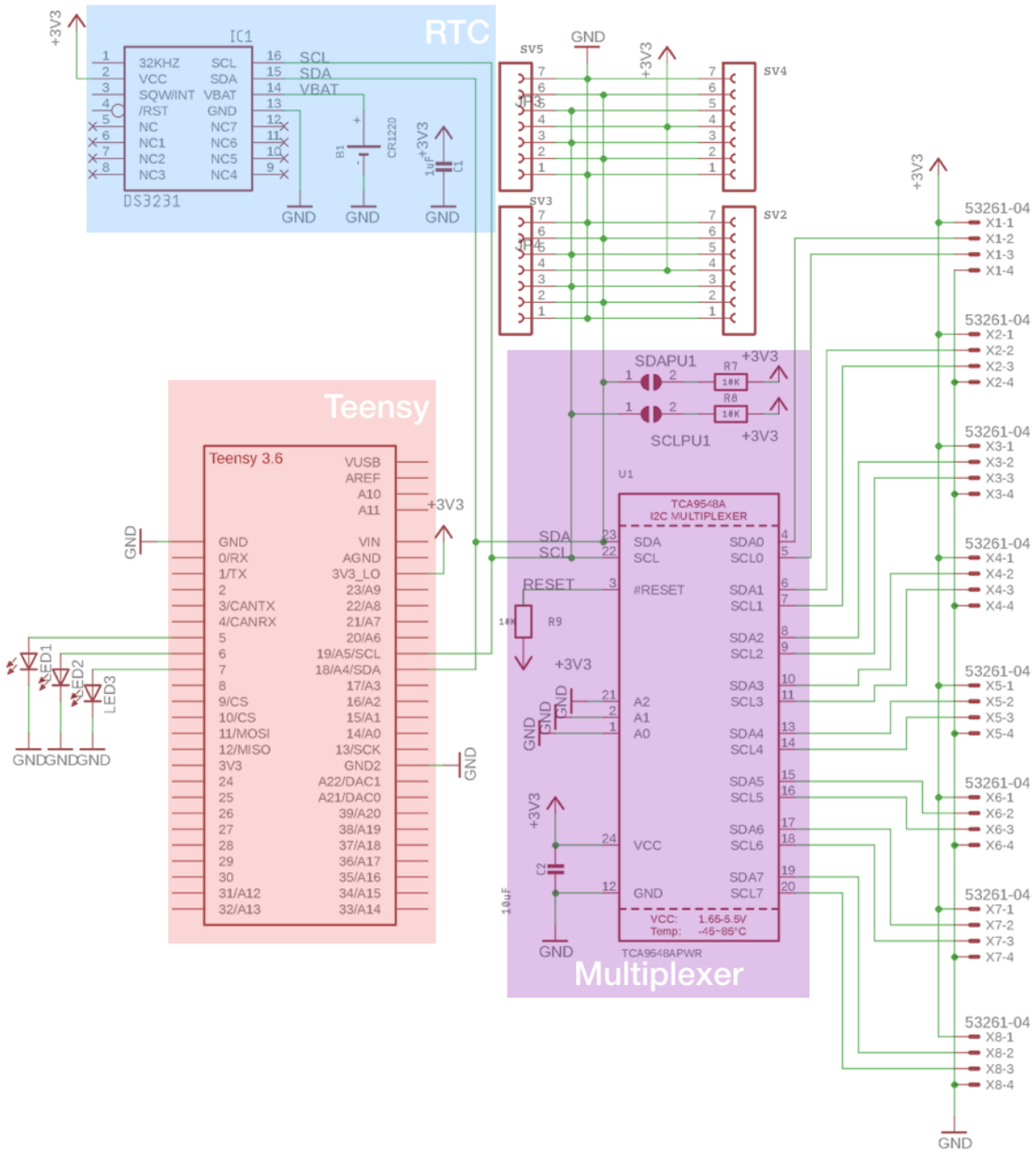


Abbildung 11: Tennsyplatine Schaltplan

Die fertig entworfene Platine ist in Abbildung 9 mit der Oberseite und in Abbildung 10 mit der Unterseite dargestellt. Es ist zu erkennen, dass der Teensy in der Mitte des Platine sitzt und rechts sowie links von ihm jeweils 4 Anschlüsse für die Sensoren angebracht sind. Ebenso in der Mitte liegt der I2C-Multiplexer (oben), die Real-Time-Clock (unten) und eine Anschlussmöglichkeit für eine externe Spannungsversorgung (oben links). An den Seiten der Sensoranschlussreihen (oben und unten) ist jeweils eine Buchse angeschlossen, welche es ermöglicht das Multiplexer Shield auf diese zu stecken und eine feste sowie korrekt verkabelte Verbindung zu schaffen. Durch die Wahl der richtigen Stecker kann dieser Vorgang auf jedweder Seite (links wie rechts), für bis zu insgesamt sieben Multiplexer, wiederholt werden.

5.2 Multiplexerplatine

Der Schaltplan der Multiplexerplatine ist identisch mit der rechten Seite des Hauptplatine-Schaltplans. Er besteht aus einem Multiplexer, den Anschlusskontakten zur Hauptplatine und den Anschlüssen für die Sensoren. Wie in 5.1 ist zu sagen, dass alle Widerstände Pullup- bzw. Pulldownwiderstände sind. Ebenso sind die Kondensatoren der Schaltung für die Glättung der Spannung eingesetzt und haben keine weitere Funktion. Da mehrere Multiplexer-Shields angeschlossen werden können, muss jedem eine eigene I2C-Adresse zugewiesen werden, indem die Jumper S1 in Abbildung 12 binär codiert von $1 \rightarrow$ [001] bis $7 \rightarrow$ [111] individuell eingestellt werden.
Die Platine für die Multiplexer ist ein Shield, was bedeutet, dass diese auf eine andere Platine aufgesteckt werden kann. Die Verbindungs-Stecker sind so zu wählen, dass auf der Unterseite Pins (male) und auf der Oberseite Buchsen (female) sind und somit eine Kaskadierung durch den Steckvorgang möglich ist.

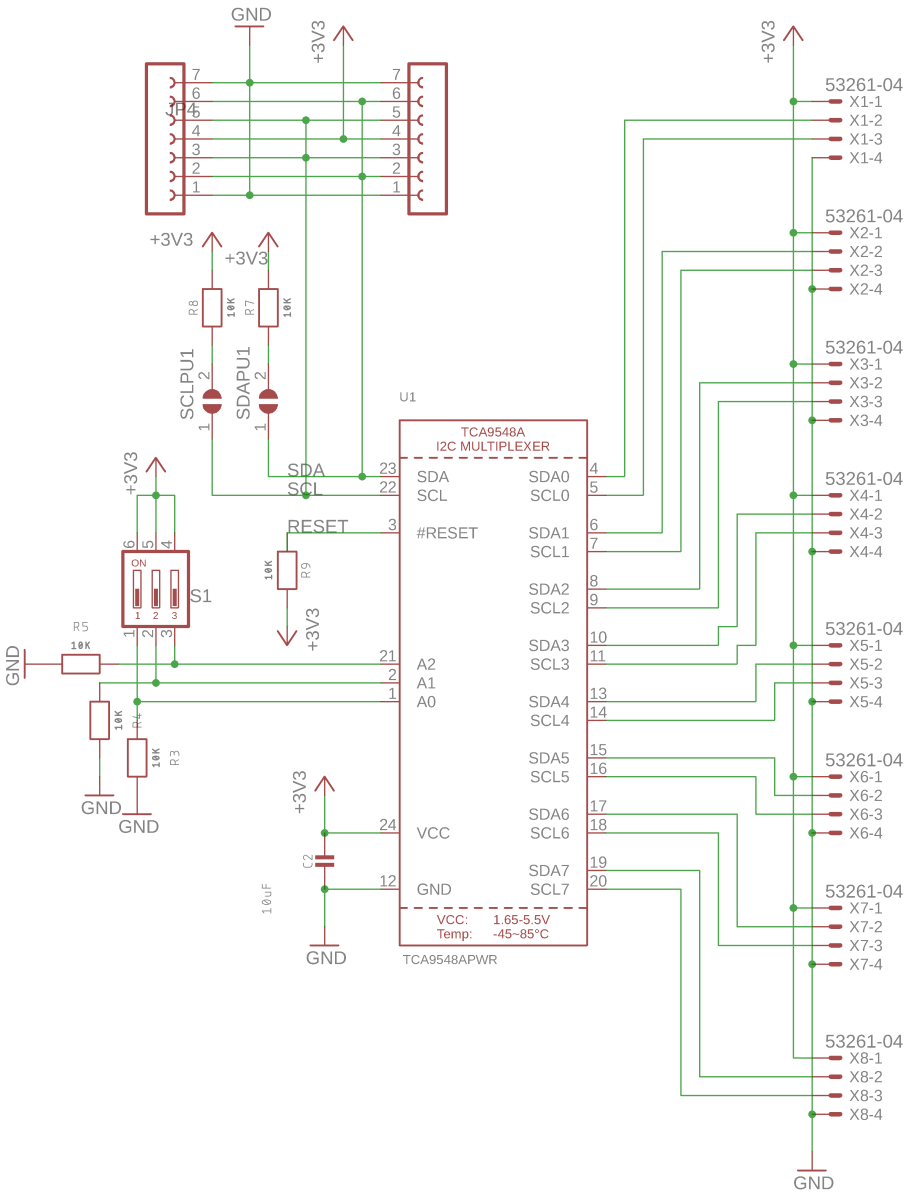


Abbildung 12: Multiplexerplatine Schaltplan

Das Shield ist in Abbildung 13 mit der Oberseite und in Abbildung 14 mit der Unterseite dargestellt. Es ist hervorzuheben, dass die acht Anschlüsse für die Sensoren auf der Ober- und Unterseite liegen und bündig mit den Anschlüssen auf der Hauptplatine sind. Die Jumper zum Einstellen der Adresse befinden sich mittig auf der Oberseite der Platine.

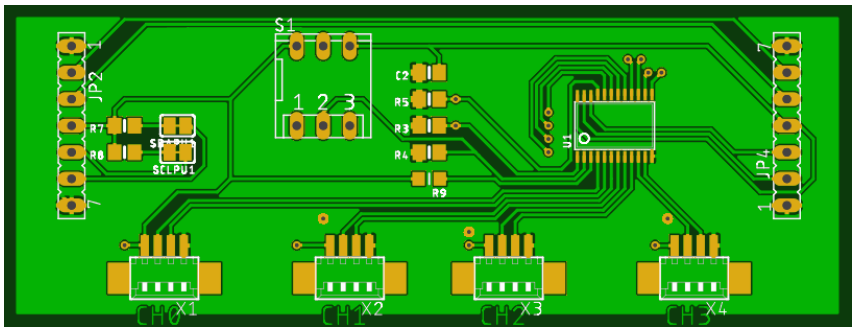


Abbildung 13: Multiplexerplatine Top Layer

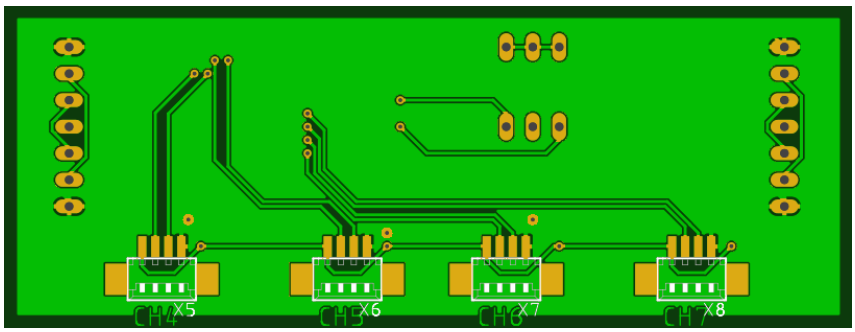


Abbildung 14: Multiplexerplatine Bottom Layer

5.3 Sensorplatine

Auf Grund der Anforderung einer maximal kompakten Bauform besteht die Sensorplatine nur aus zwei Bauteilen, dem AS7264N Sensor und der Anschlussbuchse zur Verbindung mit dem Multiplexer. Der simple Schaltplan ist auch der Grund, warum die Platine nur einseitig ist (Top Layer). Die Maße der Sensorplatine belaufen sich auf 22.84 mm x 14.91 mm.

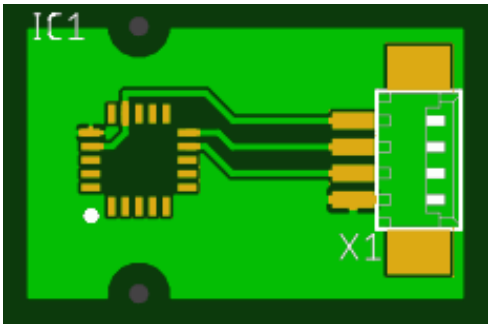


Abbildung 15: Sensorplatine Top Layer

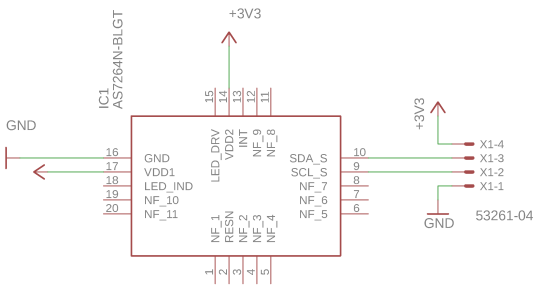


Abbildung 16: Schaltplan Sensorplatine

6 Benutzerhandbuch

6.1 Start

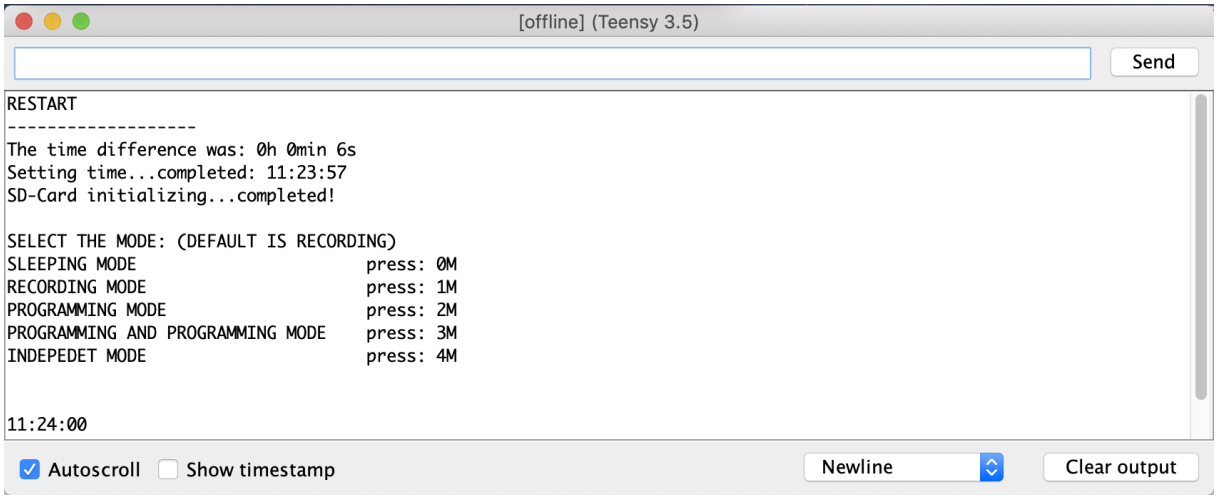


Abbildung 17: Startbildschirm nach einem Reupload

Um den Teensy benutzen zu können, muss die aktuelle Arduino IDE sowie der Teensy Loader heruntergeladen werden. Die Bibliotheken sowie das Programm des Teensy ist auf der CD bereitgestellt. Für den Zugriff auf den Teensy muss nun in der Arduino IDE im Reiter 'Tools'→'Board' auf 'Teensy 3.5' umgestellt werden und der serielle Monitor geöffnet werden. Über diesen ist eine direkte Interaktion mit dem Teensy möglich. Um sicher zu gehen, dass die Zeit richtig eingestellt ist, empfiehlt es sich, die Software neu auf den Teensy zu laden.
Nach einem Neustart meldet sich der Teensy und gibt die aktuelle Zeit sowie etwaige Fehlermeldungen aus. Der Teensy befindet sich nach dem Neustart in einem von vier Modi (dargestellt in Tabelle 3). Falls vor dem Reboot kein Modus gesetzt wurde, ist der Default-Modus 'Recording'. Aufgelesene Messdaten werden auf dem seriellen Monitor angezeigt und auf der SD-Karte abgespeichert. Dies geschieht nach der Ordner-Struktur:

$$Jahr/Monat/Tag/Sensor \tag{6.1.1}$$

Tabelle 10: URL der Software

	URL
Arduino IDE	`https://www.arduino.cc/en/Main/Software`
Teensy Loader	`https://www.pjrc.com/teensy/loader.html`

6.2 Interaktion und Einstellungen

Um Einstellungen an den Variablen vornehmen und weitere Funktionen ausführen zu können, muss in einen Recording-Modus gewechselt werden. Dies geschieht durch den Befehl `'2M'` für den reinen Programming-Modus und `'3M'` für Programming & Recording oder `'4M'` für einen autonomen Modus, welcher in das Eingabefeld eingegeben werden muss. Dabei ist darauf zu achten, dass die Einstellung 'Newline' gewählt wurde. Die darauf folgende Ansicht der Programmieroptionen ist in Abbildung 18 dargestellt und kann durch den Befehl `'repeat'` wiederholt angezeigt werden. Ebenfalls können alle derzeitigen Einstellungen mit dem Befehl `'display'` angezeigt werden. Der Befehl `'search'` sucht nach allen angeschlossenen AS7264N Sensoren und gibt diese aus. Diese Suche geschieht bei der Aufnahme von Daten automatisch und dient hier nur als Information für den Nutzer.

Für die Einstellungsmöglichkeiten gilt nun, dass alle programmierbaren Größen auf der linken Seite unter `'Programmable variables:'` aufgezählt werden und über das Eingeben ihrer entsprechenden Namen variiert werden können. Die genaue Vorgehensweise der Einstellung wird nach der Eingabe automatisch einzeln erläutert und durch den `'back'` Befehl beendet. Ein Wechsel in einen anderen Modus (Tab 3) ist aus jeder Programmebene möglich. Ebenso ist der `'reset'` Befehl immer zugelassen, welcher die Einstellungen in Tabelle 4 übernimmt.

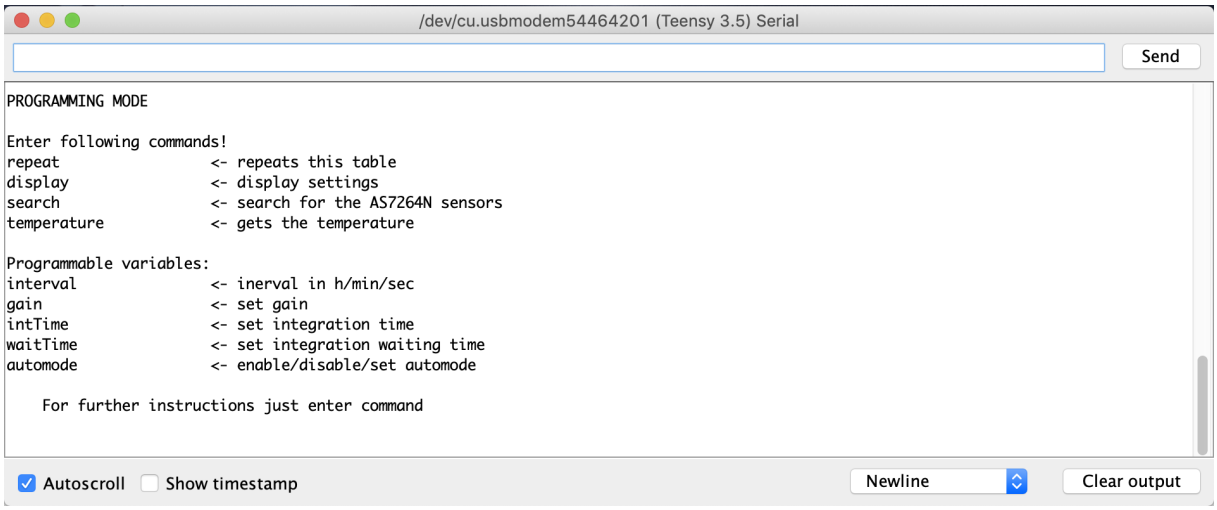


Abbildung 18: Einstellungsbildschirm

6.3 Fehlermeldung

Eine Fehlermeldung ist entweder das Fehlen der RTC oder das Fehlen einer SD-Karte. Beide Fehlermeldungen werden auf dem seriellen Monitor angezeigt, die Aufnahme von Daten gestoppt und der Fehler durch die LEDs (Tabelle 11) indiziert. Im Falle einer Fehlermeldung ist es jedoch möglich fortzufahren, indem der Befehl `'4M'` eingegeben und in den autonomen Modus gewechselt wird. Nun ist es nicht mehr möglich die Daten auf die SD-Karte zu schreiben und die Zeitangabe ist ggf. unzutreffend.

Tabelle 11: Status LEDs

LED	Fehlermeldung
Grün	kein Fehler
Blau	Fehlende SD-Karte
Rot	Fehlende RTC

7 Messungen

Um mit dem Sensor AS7264N aussagekräftige Messungen durchführen zu können, muss dieser zuvor charakterisiert werden. Hierzu werden die vom Hersteller aufgestellten Verhaltensweisen bezüglich der einstellbaren Integrationszeit und der Verstärkungsfaktors (gain) untersucht. Anschließend werden Farbort-Messungen mit dem AS7264N durchgeführt und mit simultan erhobenen Referenzwerten eines Specbos 1201 Spectroradiometers verglichen.

7.1 Aufbau

Der Versuchsaufbau ist in Abbildung 20 dargestellt. Dieser besteht aus einer frühen Version der Sensorplatine des AS7264N (oben rechts), welche durch eine Messklemme fixiert und an die Hauptplatine (unten) angeschlossen ist. Von dem quadratischen AS7264N Sensor ist nur die aufgesteckte Kappe mit Diffusorscheibe zu erkennen. Auf Höhe der Diffusorscheibe ist das Spectroradiometer installiert (oben links), welches eigene Vergleichsmessungen durchführt. Diese Versuchsapparatur ist auf einem Schienenwagen installiert, welcher eine eindimensionale Variation der Entfernung zu einer festen Lichtquelle zulässt (Abbildung 19). Diese Lichtquelle besteht aus drei individuell ansteuerbaren Lichtfarben: warm-weiß (WW), neutral-weiß (NW) und blau.

Abbildung 19: Aufbau Messung

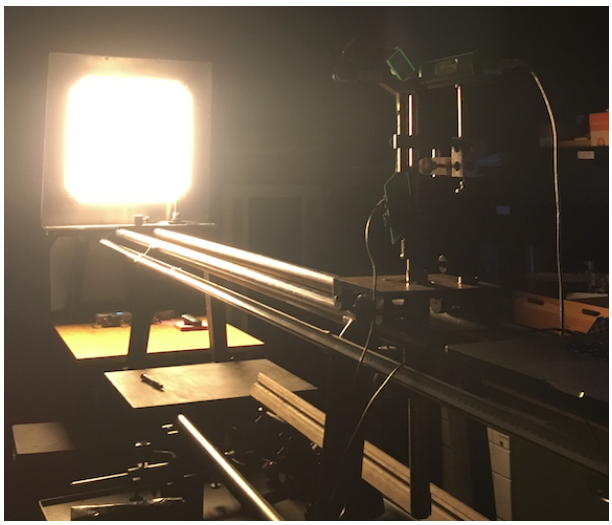
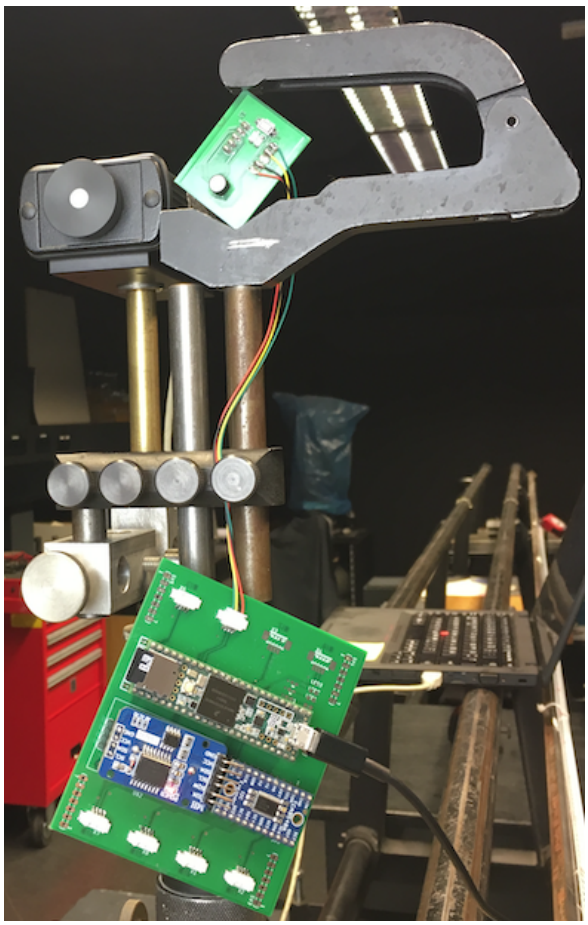


Abbildung 20: Aufbau Front



Die vor die Öffnung des Sensors aufgebrachte Diffusions-Streuscheibe hat zur Folge, dass der Sensor einen Öffnungswinkel von 90° besitzt, und somit den zur Leuchte gewandten Halbraum abdeckt. Die leuchtende Fläche liegt, unabhängig der Entfernung, vollständig im vom Öffnungswinkel aufgespannten Raumwinkel. Das Raumwinkelprojektionsgesetz $E = L \cdot \Omega_{2p}$, bei

einem projeziertem Raumwinkel von $\Omega_{2p} = \frac{A_{Kp1}}{r^2} \cdot \Omega_0$, ergibt somit, bei konstanter Leuchtdichte und Fläche, die Abhängigkeit:

$$E = L \cdot \frac{A_{Kp1}}{r^2} \cdot \Omega_0 \Rightarrow E \propto \frac{1}{r^2} \tag{7.1.1}$$

7.2 Eigenschaften

Zur Charakterisierung des Sensors werden in diesem Abschnitt zunächst die vom Hersteller AMS bereitgestellten Angaben zum Verhalten bei Variation der Integrationszeit und des Verstärkungsfaktors untersucht.

7.2.1 Lineare Integrationszeit

Der Zeitraum, in welchem ein Messvorgang stattfindet, also die Beleuchtungsstärke integriert wird, kann durch das Register `intTime` anhand der Formel 7.2.1 eingestellt werden. Alle Linearitätsmessungen werden bei voller Leistung der Leuchte (WW,KW,$blau = 100\%$) durchgeführt.

$$T_{int} = (255 - \texttt{intTime}) \cdot 2.8\,\mu s \tag{7.2.1}$$

In Abbildung 21 sind die ausgelesenen Registerwerte bei konstantem Verstärkungsfaktor in einem Abstand von einem Meter dargestellt (Tabelle 18). Zur eindeutigeren Ersichtlichkeit der Linearität sind die Registerwerte in Abbildung 22 normiert auf ihre Integrationszeit angegeben. Die Integrationszeit wird bei diesem Versuch von $28\,\mu s$ bis $4.480\,ms$ jeweils verdoppelt.

Abbildung 21: Linearität Gain-Register=0

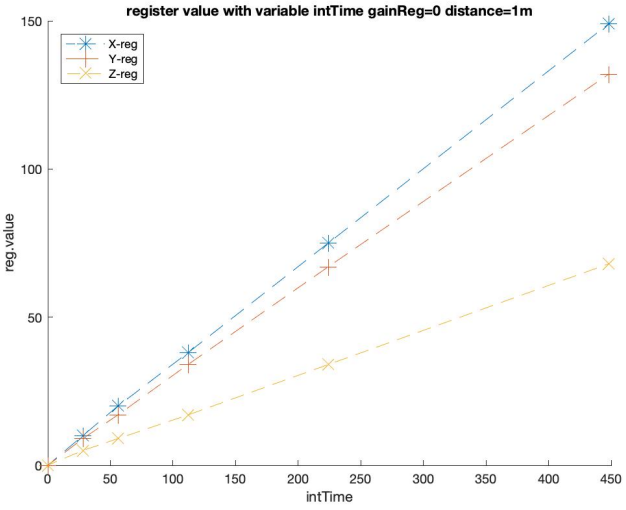
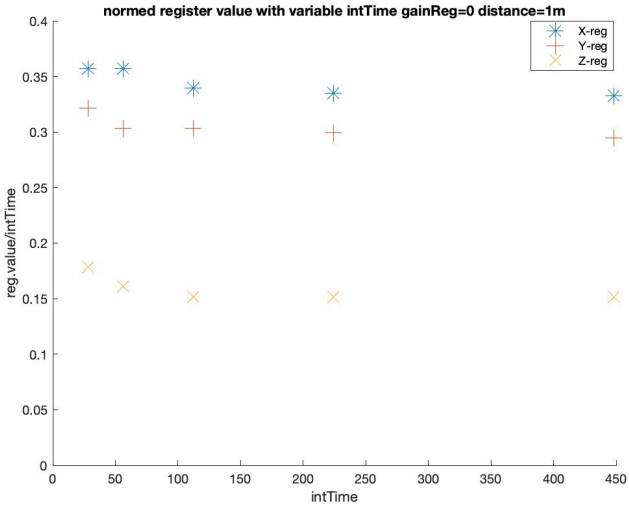


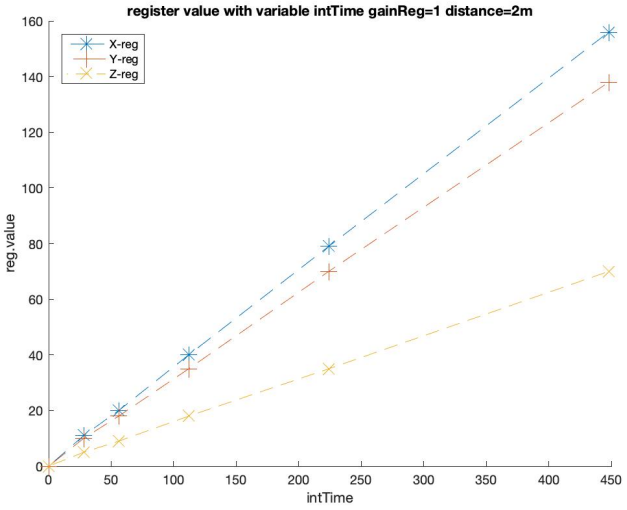
Abbildung 22: Linearität(genormt) Gain-Register=0



Ein linearer Verlauf der Registerwerte über der Integrationszeit ist in Abbildung 21 gut zu erkennen und wird durch die in Abbildung 22 konstanten Verläufe bestätigt. Lediglich bei Messwerten im Bereich von unter 10% der maximalen Registerwerte, hier bei einer Integrationszeit von unter $100\,\mu s$, treten Ungenauigkeiten auf.

Um eine Linearität auch bei den Verstärkungsfaktoren größer als Eins nachweisen zu können sind weitere Messungen mit erhöhtem Gain nach Tabelle 13 durchgeführt worden. Zur Verhinderung eines Überlaufs der Register wurde der Abstand zur Lichtquellen auf zwei Meter erhöht.

Abbildung 23: Linearität Gain-Register=1



Die Messergebnisse sind in Tabelle 19 gelistet und in Abbildung 23 dargestellt. Es fällt auf, dass nahezu die identischen Registerwerte wie in der vorherigen Messung ausgelesen wurden, sodass auch bei einem Gain-Registerwert= 1 von einem mit der Integrationszeit linearem Verlauf ausgegangen werden kann. Ebenso lässt dieses Ergebnis Rückschlüsse auf den Verstärkungsfaktor zu, auf welche in Abschnitt 7.2.3 eingegangen wird. Auch bei einem Gain-Registerwert = 2 (Abbildung 24) und = 3 (Abbildung 25) ist eine Linearität zu erkennen.

Abbildung 24: Linearität Gain-Register=2

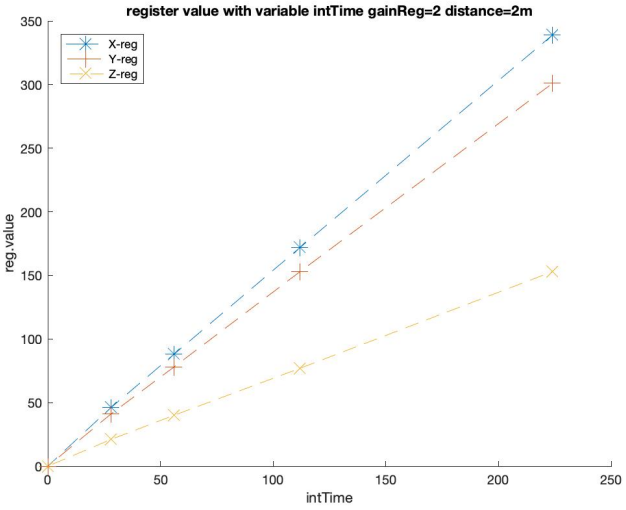
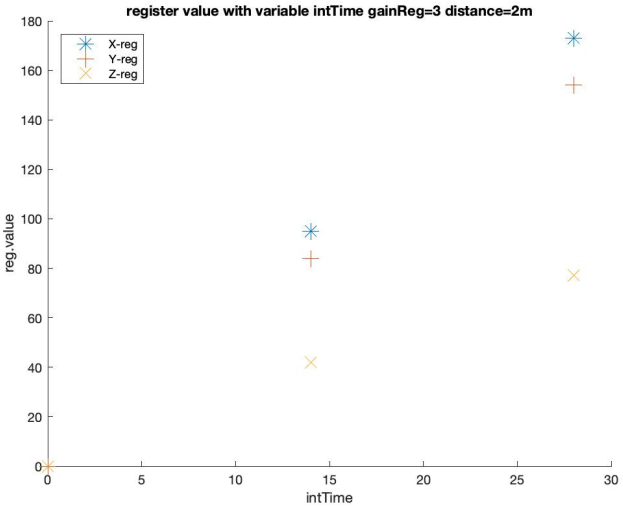


Abbildung 25: Linearität Gain-Register=3



7.2.2 Absolutwerte

Um die Absolutwerte zu vergleichen, ist es sinnvoll, die gemessenen Y-Normfarbwerte von Jeti und AS7264N zu vergleichen, da diese ein Maß für das Helligkeitsempfinden bieten (Abschnitt 2). Hierzu werden bei jedem Abstand und jeder Farbe die Jeti-Messwerte ins Verhältnis mit den AS7264N-Messwerten gestellt, um einen Skalierungsfaktor zu erhalten (Tabelle 12). Die AS7264N-Messwerte wurden gewonnen, indem die Registereinträge durch die Integrationszeit, sowie den gemessenen Verstärkungsfaktor geteilt wurden.

Tabelle 12: Normfarbwert Skalierungsfaktor

Abst./Farbe	Y (AS7264N Rohdaten)	Y (Jeti)	Skalierungsfaktor	Y (AS7264N errechnet)
30cm NW	2.641	7446	2819.4	7077
30cm WW	1.720	4923	2862.2	4609
30cm Blau	0.736	2010	2731.9	1972
60cm NW	0.940	2927	3113.4	2520
60cm WW	0.615	1936	3148.4	1648
60cm Blau	0.342	785	2293.9	917
120cm NW	0.418	886	2122.1	1120
120cm WW	0.218	585	2684.1	584
120cm Blau	0.102	238	2340.8	273

Es ist festzustellen, dass der Skalierungsfaktor den Mittelwert von 2680 bei einer Standardabweichung von 360 besitzt; dies entspricht über $15\,\%$. Die errechneten Normfarbwerte aus den Rohdaten und dem mittleren Skalierungsfaktor lassen Rückschlüsse auf den wahren Wert zu, sind aber teilweise ungenau. Um ein exaktes Ergebnis zu erhalten, wäre ein Skalierungsfaktor in Abhängigkeit des Abstandes sinnvoll, wie an den Ergebnissen für $30\,cm$ zu erkennen ist. Für diesen Anwendungsfall sind reale Tageslichtmessungen notwendig, um einen spezifischen Tageslicht-Skalierungsfaktor zu bestimmen und somit ein präziseres Ergebnis zu erhalten.

7.2.3 Linearer Verstärkungsfaktor

Die Integrationszeit kann laut Abschnitt 7.2.1 als linear unabhängig vom Verstärkungsfaktor angenommen werden. In diesem Abschnitt werden nun die vom Hersteller angegebenen Faktoren, wie in Tabelle 13 aufgelistet, überprüft. Wie im vorherigen Abschnitt angesprochen, können aus den Messwerten bei den Abständen $1\,m$ und $2\,m$ Rückschlüsse auf den Verstärkungsfaktor gezogen werden. Da, wie in Abschnitt 7.1 beschrieben, die Beleuchtungsstärke der Formel 7.1.1 folgt und somit quadratisch abnimmt, wird eine Viertelung der Beleuchtungsstärke erwartet. Durch die Erhöhung der Verstärkung auf den Registerwert von 1 (laut Hersteller Faktor von 3.7) kann nun dessen tatsächlicher Faktor bestimmt werden (Listing 14).

Der Verstärkungsfaktor mit dem Registerwert 2 wird ermittelt, indem die Messwerte im Abstand von $2\,m$ und dem Gain-Registerwert=1 ins Verhältnis gesetzt werden. Da der Registerwert bei einem Verstärkungsfaktor von im Datenblatt angegebenen 64 stark übersteuert, wird dieser indirekt über die Integrationszeit bei niedrigerem Faktor bestimmt. Dabei wird davon ausgegangen, dass sich beim Vervierfachen der Integrationszeit bei niedrigerem Gain der identische Registerwert wie für den nächst höheren Faktor mit ursprünglicher Integrationszeit ergibt. Die so ermittelten Faktoren sind in Tabelle 13 aufgeführt.

Tabelle 13: Verstärkungsfaktor laut AMS

Registerwert	Gain (AMS)	Gain (gem.)
0	1	1
1	3.7	4.17
2	16	15.58
3	64	66.68

Da diese Faktoren Mittelwerte über mehrere Messungen sowie über alle Farbwert-Register sind, werden nun die einzelnen Register auf Linearität bei Erhöhung des Verstärkungsfaktors analog zum Verfahren in Abschnitt 7.2.1 untersucht. Die in Abbildung 26 dargestellten Registerwerte lassen nur grob ein lineares Verhalten im Abstand von einem Meter erahnen. Selbiges gilt für die Messung bei einer Entfernung von $2\,m$. Bei beiden Messreihen wurde eine Integrationszeit von $28\,\mu s$ gewählt. Wegen Übersteuerns fehlt in Abbildung 26 der Wert für den höchsten Verstärkungsfaktor sowie wegen Untersteuerns der Wert für den niedrigsten Verstärkungsfaktor in Abbildung 27. Es ist festzuhalten, dass der Verstärkungsfaktor nicht für alle Register gleich ist. Dies bedeutet, dass dass die in Tabelle 13 errechneten Faktoren nur die Mittelwerte der X/Y/Z Verstärkungen darstellen, und die einzelnen Registerverstärkungen davon abweichen können. Dies führt zu einer Veränderung der Registerverhältnisse untereinander bei Variation des Gain.

Abbildung 26: Linearität intTime=μs Abstand $1\,m$

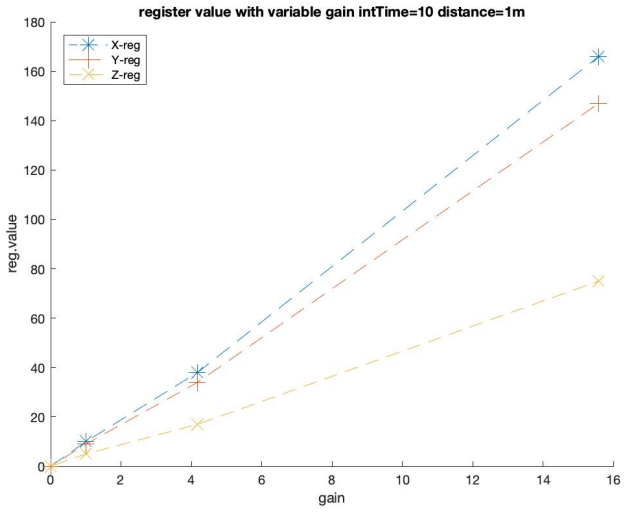
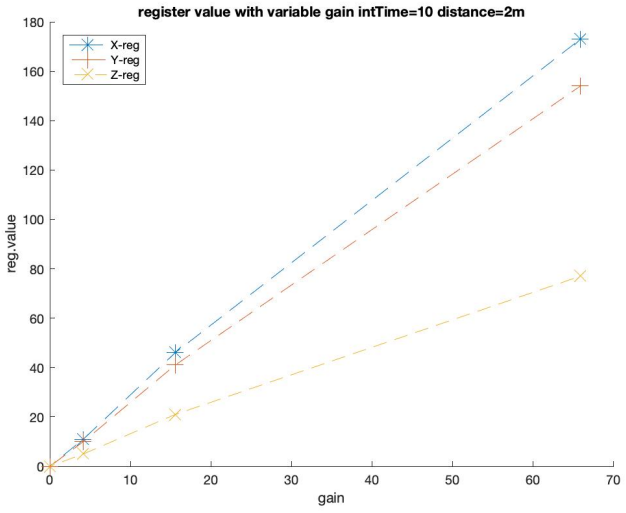


Abbildung 27: Linearität intTime=μs Abstand $2\,m$



7.3 Farbort-Messungen

Für die Bestimmung des Farbortes wurden die drei ansteuerbaren Farben warm-weiß, neutral-weiß und blau, welche von der Leuchte vorgegeben wurden, vermessen. Als Referenz wird, wie in Abschnitt 7.1 gezeigt, ein Specbos 1201 Spectroradiometer genutzt.
In dieser Messreihe wurde der Automodus des Teensy benutzt, welcher die Integrationszeit und den Gain selbstständig für einen optimalen X/Y/Z-Registerwert wählt. Die Registerwerte werden in der anschließenden Interpretation mit T_{int} und dem Gain verrechnet (RW-Tabelle in der Matlab Funktion 15). Diese Ergebnisse werden nun in x/y-Farbwertanteile nach den Formeln 2.0.4 & 2.0.5 umgerechnet, wobei der Z-Kanal eine Kanalverstärkung von 2.75 benötigt (Abschnitt 3.1).
In den Tabellen 14-16 sind die gemessenen Normfarbwertanteile, sowie die daraus berechneten u' und v' Werte im CIELUV Farbsystem (Formeln 7.3.1&7.3.2), und der Abstand zwischen den Referenz- und Messwerten, eingetragen.

$$u' = \frac{4x}{-2x + 12y + 3} \tag{7.3.1}$$

$$v' = \frac{9x}{-2x + 12y + 3} \tag{7.3.2}$$

Abbildung 28: Farbortmessung neutral-weiß

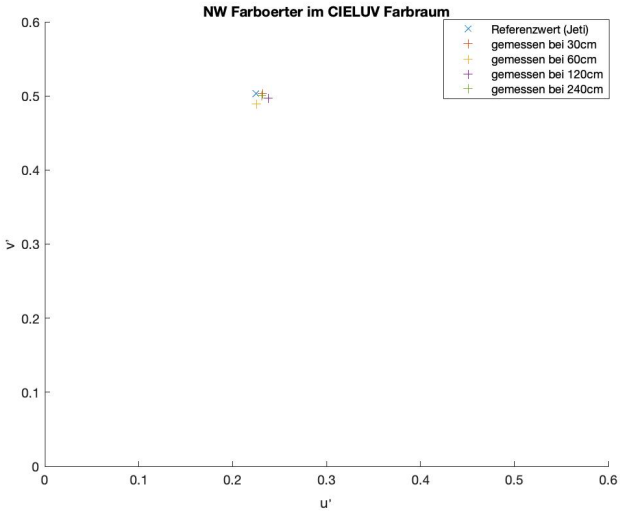
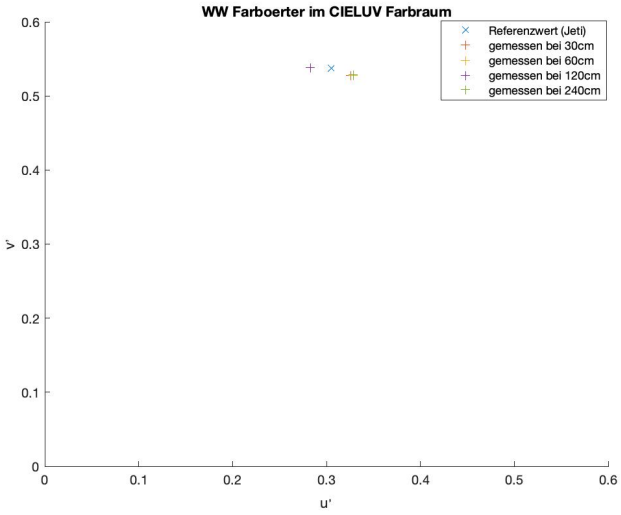


Abbildung 29: Farbortmessung warm-weiß



Die Ergebnisse der Farbortmessungen für das neutral-weiße Licht sind in der Abbildung 28 im CIELUV Farbsystem dargestellt.
In Tabelle 15 sind die gemessenen Farbörter für das warm-weiße Licht, sowie die Vergleichsmessungen des Spektroradiometers gelistet. Dargestellt sind die Messergebnisse der Farbörter des warm-weißen Lichtes in Abbildung 29.

Tabelle 14: Ergebnisse NW

Entf.[cm]	x	y	u'	v'	Δ u'v'
Referenz:	0.3818	0.3794	0.2249	0.5029	-
30	0.391	0.377	0.2319	0.5035	0.0070
60	0.367	0.353	0.2256	0.4888	0.0142
120	0.392	0.363	0.2384	0.4970	0.0147
240	0.387	0.371	0.2316	0.5002	0.0072

Tabelle 15: Ergebnisse WW

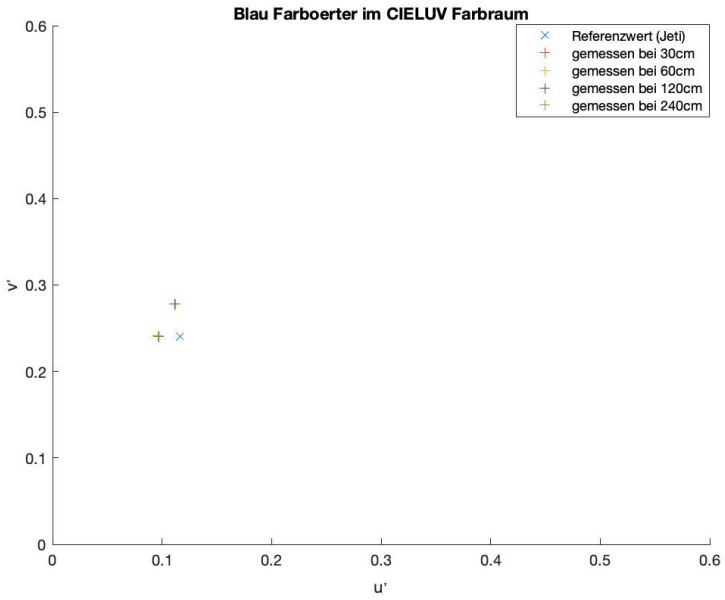
Entf.[cm]	x	y	u'	v'	Δ u'v'
Referenz:	0.5255	0.4112	0.3054	0.5376	-
30	0.532	0.383	0.3257	0.5278	0.0226
60	0.535	0.383	0.3284	0.5281	0.0249
120	0.500	0.422	0.2831	0.5379	0.0223
240	0.537	0.384	0.3290	0.5290	0.0252

Für die Blaulicht-Messung sind die Ergebnisse in der Tabelle 16 aufgeführt und in der Abbildung 30 ist der Farbort für alle Entfernungen aufgetragen.

Tabelle 16: Farbort Ergebnisse blau

Entf.[cm]	x	y	u'	v'	Δ u'v'
Referenz:	0.118	0.109	0.116	0.241	-
30	0.100	0.110	0.097	0.241	0.0189
60	0.123	0.135	0.112	0.278	0.0375
120	0.122	0.135	0.111	0.278	0.0377
240	0.099	0.110	0.096	0.241	0.0201

Abbildung 30: Farbortmessung blau



Es ist festzuhalten, dass Farbortbestimmungen mittels des AS7264N möglich sind. Jedoch treten durch das nicht konstante Verhalten des Verstärkungsfaktors Messungenauigkeiten auf. Diese sind in zukünftigen Messreihen durch alleinige Variation der Integrationszeit zu vermeiden.

Tabelle 17: Farbort Messungen (Registerwerte)

Farbe	Entf.	X	-Y	-Z	Gain	IntT	-X	Y	-Z	Gain	IntT	-X	-Y	Z	Gain	IntT
NW	30cm	208	-201	-50	0	179	-214	206	-51	0	177	-103	-70	209	1	170
WW	30cm	212	-204	-51	1	203	-220	212	-53	1	201	-138	-103	215	2	204
BLAU	30cm	207	-228	-22	1	181	-193	212	-243	1	186	-80	-88	207	0	147
BLAU	60cm	218	-240	-47	2	210	-5	5	-12	2	255	-82	-90	211	1	181
WW	60cm	212	-152	-12	1	196	-34	208	-16	1	174	-11	-208	209	3	189
NW	60cm	212	-204	-51	1	203	-220	212	-53	1	201	-137	-101	215	2	204
NW	90cm	207	-192	-41	1	145	-220	204	-43	1	138	-18	-201	205	2	127
WW	90cm	194	-139	-11	1	75	-42	214	-16	2	192	-184	-36	177	3	69
BLAU	90cm	204	-227	-12	2	112	-186	206	-221	2	125	-64	-72	166	1	59
BLAU	120cm	205	-228	-8	3	114	-184	204	-210	3	129	-73	-81	187	2	67
NW	120cm	216	-207	-50	3	216	-227	218	-52	3	214	-117	-81	205	3	92
WW	120cm	214	-154	-11	3	211	-39	212	-16	3	194	-176	-94	65	3	4

Tabelle 18: Linearitätsmessung bei 1m Abstand (Registerwerte)

X	Y	Z	intT	Gain
10	9	5	245	0
20	17	9	235	0
38	34	17	215	0
75	67	34	175	0
149	132	68	95	0
38	34	17	245	1
142	126	65	215	1
73	65	33	235	1
166	147	75	245	2
61	25	143	235	2
174	155	77	245	3

Tabelle 19: Linearitätsmessung bei 2m Abstand (Registerwerte)

X	Y	Z	intT	Gain
11	10	5	245	1
20	18	9	235	1
40	35	18	215	1
79	70	35	175	1
156	138	70	95	1
46	41	21	245	2
88	78	40	235	2
172	153	77	215	2
84	46	153	175	2
173	154	77	245	3
95	84	42	250	3

8 Zusammenfassung

Die Wahl des Teensy als Mikrocontroller hat sich im Laufe des Programmiervorgangs als richtig herausgestellt. Eine schnelle Upload-Zeit sowie der onboard SD-Slot hat die Problemsuche sehr vereinfacht. Eine ausreichend schnelle Verbindung via I2C Bus ist gegeben und die Ansteuerung mehrerer Sensoren ist über die Multiplexer zuverlässig möglich. Eine Real-Time-Clock als zeitgebendes Instrument ist jedoch sehr unzuverlässig und sollte in zukünftigen Aufbauten z. B. durch eine Internet-basierte Variante ersetzt werden.

Das Programm zum Auslesen der AS7264N Sensoren funktioniert zuverlässig. Optionen der Optimierung sind dennoch vorhanden, da insbesondere im Automodus das Finden der Parameter für eine Messung sehr viel Zeit beansprucht (ca. $20\,s$). Diese Zeit wird für jeden angeschlossenen Sensor benötigt, was einen Betrieb von vielen Sensoren behindert. Eine Lösung wäre das zeitgleiche Abfragen aller Sensoren oder ein dauerhaftes Suchen der optimalen Parameter im Hintergrund. Ebenso ist es nicht möglich während eines Messvorganges diesen abzubrechen.

Der AS7264N Sensor von AMS ist durch den in dieser Bachelorarbeit erstellten Messaufbau in der Lage Lichtmessungen durchzuführen. Da das Augenmerk auf der Bestimmung des Farbortes, also der Farbwertanteile lag, stand die Relation der Farbwerte im Fokus. Ohne Berücksichtigung von festgestellten Ungenauigkeiten und weitreichender Charakterisierungen, liegen die Messabweichungen der Farbwertanteile von weißem Licht bei maximal 6%. Diese Abweichung lässt sich in weiteren Schritten durch Anpassung der Software, in Hinblick auf Konstanthaltung des Gain für einen Farbort, sowie einer experimentellen Bestimmung der Einzelkanalverstärkungen für jede Gainstufe minimieren.

Dennoch sind die erzielten Ergebnisse des Messaufbaus überraschend genau und lassen eine Verwendung als Alternative zur konventionellen Bestimmung des Farbortes des Tageslichtes mittels Spectroradiometer zu.

9 Anhang

Listing 10: Die BaLibTEE.cpp Datei (Library)

```
1
2  //Inclusion of the libraries
3
4  #include <BaLibTEE.h>   //<- to this c++ belonging Library
5  #include <EEPROM.h>     //library for writing to internal memory
6  #include <Wire.h>       //I2C library
7  #include <TimeLib.h>    //time managing library
8  #include <DS1307RTC.h>  //Real-Time-Clock library
9  tmElements_t tm;     //creation of tm element
10
11 //Inclusion of the SD-Card libraries
12 #include <SD.h>
13 #include <SPI.h>
14 File printfile; //creation of the file: printfile
15
16 //initiate global variable
17 boolean newhead = false;
18
19
20 SDCARD::SDCARD(){}
21
22 //Function is initializing the SD-Card and printing the Mode commands
23 void SDCARD::initializing(){
24     Serial.print("SD-Card initializing...");
25
26     if (available()){
27         SD.mkdir(getyear());
28         Serial.print("completed!");     //Trying the initializing
29     }else{
30         Serial.println("failed!");       //return if failed
31     }
32
33     Serial.println();
34     Serial.println();
35     Serial.println("SELECT THE MODE: (DEFAULT IS RECORDING)");
36     Serial.println("SLEEPING MODE                    press: 0M");
37     Serial.println("RECORDING MODE                   press: 1M");
38     Serial.println("PROGRAMMING MODE                 press: 2M");
39     Serial.println("PROGRAMMING AND PROGRAMMING MODE press: 3M");
40     Serial.println("INDEPEDET MODE                   press: 4M");
41     Serial.println();
42
43     EEPROM.write(ERR,0);          //writing the ERROR byte in EEPROM to zero to reset!
44     EEPROM.write(PROG,0);         //writing the PROG byte in EEPROM to zero to reset! (no
           input yet)
45 }
46
47 //Function sets time str to tm time
48 //returns true if successful
49 bool SDCARD::settime(const char *str){
50     int Hour, Min, Sec;
51     if (sscanf(str, "%d:%d:%d", &Hour, &Min, &Sec) != 3) return false;  //char str must
           contain 3 integers
52     tm.Hour = Hour;
53     tm.Minute = Min;
54     tm.Second = Sec;
55     return true;
56 }
57
58 //Function prints the time difference between the compile time and the RTC time
59 void SDCARD::timedif(const char *newtime){
60     //define the newtime via char
61     int newHour, newMin, newSec;
62     sscanf(newtime, "%d:%d:%d", &newHour, &newMin, &newSec);
63
64     //define the oldtime via string
65     int oldHour, oldMin, oldSec;
66     String oldtimeString = gettimeString();
67     oldHour= (oldtimeString[0]-48)*10 + oldtimeString[1]-48;    // -48 because of ASCII
```

```
68      oldMin = (oldtimeString[3]-48)*10 + oldtimeString[4]-48;
69      oldSec = (oldtimeString[6]-48)*10 + oldtimeString[7]-48;
70
71      //calculate the time difference
72      int Hourdiff = newHour-oldHour;
73      int Mindiff = newMin-oldMin;
74      int Secdiff = newSec-oldSec;
75
76      //print the time difference
77      Serial.print("The time difference was: ");
78      Serial.print(Hourdiff);
79      Serial.print("h ");
80      Serial.print(Mindiff);
81      Serial.print("min ");
82      Serial.print(Secdiff);
83      Serial.println("s");
84  }
85
86  //Function sets tm date to char str
87  bool SDCARD::setdate(const char *str){
88      char Month[12];
89      int Day, Year;
90      byte monthIndex;
91
92      if (sscanf(str, "%s %d %d", Month, &Day, &Year) != 3) return false; //char must be
            Month+Day+Year!
93      for (monthIndex = 0; monthIndex < 12; monthIndex++) {               //compare to
            month char and get the index
94          if (strcmp(Month, monthName[monthIndex]) == 0) break;
95      }
96      if (monthIndex >= 12) return false;             //if month bigger than 12 return
            false
97
98      tm.Day = Day;                           //set tm.Day to Day
99      tm.Month = monthIndex + 1;              //set tm.Month to index+1
100     tm.Year = CalendarYrToTm(Year);                 //set tm.year to Year+1970
101     return true;
102 }
103
104 //Function returns time string
105 String SDCARD::gettimeString(){
106     String returnstring;
107         if (RTC.read(tm)) {
108             if(tm.Hour < 10){returnstring+= "0";}
109             returnstring+=tm.Hour;
110             returnstring+=":";
111             if(tm.Minute < 10){returnstring+= "0";}
112             returnstring+=tm.Minute;
113             returnstring+=":";
114             if(tm.Second < 10){returnstring+= "0";}
115             returnstring+=tm.Second;
116         }
117     return returnstring;
118 }
119
120 //Function returns date String
121 String SDCARD::getdateString(){
122     String returnstring;
123         if (RTC.read(tm)) {
124             returnstring+=tm.Day;
125             returnstring+=".";
126             returnstring+=tm.Month;
127             returnstring+=".";
128             returnstring+=1970+tm.Year;
129         }
130     return returnstring;
131 }
132
133
134 //Function returns if SD-Card is available
135 bool SDCARD::available(){
136     if(SD.exists(getyear()) == 1){  //if there is a folder of this year->retun true (this
            is done because it is faster than checking SD.begin)
137         return true;
```

```
138     }else{                                          //if there is no year folder ->SD.
            begin
139         if (SD.begin(BUILTIN_SDCARD) == true) {     //SD.begin only once because then the
                folder will be created
140             return true;
141         }else{
142             return false;                           //if SD.begin is false -> definitely
                    no SD Card!
143         }
144     }
145 }
146
147 //Function writes the programming commands
148 void SDCARD::writeprogcom(){
149     Serial.println();
150     Serial.println("Enter following commands!");
151     Serial.println("repeat                <- repeats this table");
152     Serial.println("display               <- display settings");
153     Serial.println("search                <- search for the AS7264N sensors");
154     Serial.println("temperature           <- gets the temperature");
155     Serial.println();
156     Serial.println("Programmable variables:");
157     Serial.println("interval              <- inerval in h/min/sec");    //1
158     Serial.println("gain                  <- set gain");                //2
159     Serial.println("intTime               <- set integration time");    //3
160     Serial.println("waitTime              <- set integration waiting time");    //4
161     Serial.println("automode              <- enable/disable automode");
162     // Serial.println("characterize          <- characterizing the sensor (X/Y/Z)");
163     Serial.println();
164     Serial.println("    For further instructions just enter command");
165     Serial.println();
166 }
167
168 //Function returns filename-string (year/month/day/plex+ch.csv)
169 String SDCARD::SDNameString(int addr){
170     String Nameplex     = "Sensor";
171     String Namech       = String(addr);
172     String NameTag      = String(tm.Day,DEC);
173     String NameMonat    = String(monthName[tm.Month-1]);
174     String NameJahr     = String(tm.Year+1970,DEC);
175     String SDName       = String(NameJahr + "/" + NameMonat + "/" + NameTag + "/" +
            Nameplex + Namech +".csv");
176     return SDName;
177 }
178
179 //Function returns filepath-string (year/month/day)
180 String SDCARD::SDfilepath(){
181     String NameTag      = String(tm.Day,DEC);
182     String NameMonat    = String(monthName[tm.Month-1]);
183     String NameJahr     = String(tm.Year+1970,DEC);
184     String SDName       = String(NameJahr + "/" + NameMonat + "/" + NameTag);
185     return SDName;
186 }
187
188 String SDCARD::getyear(){
189     return String(tm.Year+1970,DEC);
190 }
191
192 void SDCARD::endnewheadder(){
193     newhead = false;
194 }
195
196 void SDCARD::newheadder(){
197     newhead = true;
198 }
199
200 //Function writes data to SD-Card for one Sensor @(plex,ch)
201 void SDCARD::writeData(String dataString , byte plex , byte channel){
202     String TIME = gettimeString(); //get time
203     int adval = plex*10+channel;   //create Sensorname->plex+ch
204
205     if (SD.exists(SDfilepath()) == 0){  //if filepath to sensor doesn't exist (eg: next
            day)
206         SD.mkdir(SDfilepath());         //create it!
```

```
207     }
208     if (available()){ // if SD-Card is available
209     //Serial.println(SD.exists(SDNameString(adval)));
210         if (SD.exists(SDNameString(adval)) == false || newhead == true || EEPROM.read(
            automodereg+1) > 0){     //if no file or a new head required
211             if (EEPROM.read(BANK) == 128 && EEPROM.read(automodereg) == 0){
                                                    //if in mode 1 (in this usecase always)
212                 printfile = SD.open(SDNameString(adval),FILE_WRITE);        //create file
213                 Serial.print("New file: ");
214                 Serial.println(SDNameString(adval));
215                 Serial.print("Writing headers...");                          //write the
                    header...
216                 printfile.print("Time");
217                 printfile.print(" , ");
218                 printfile.print("X");
219                 printfile.print(" , ");
220                 printfile.print("Y");
221                 printfile.print(" , ");
222                 printfile.print("Z");
223                 printfile.print(" , ");
224                 printfile.print("x");
225                 printfile.print(" , ");
226                 printfile.print("y");
227                 printfile.print(" , ");
228                 printfile.print("integration-time");
229                 printfile.print(" , ");
230                 printfile.print("gain");
231                 printfile.print(" , ");
232                 printfile.println("NIR");
233                 printfile.close();
234                 Serial.println("done");                                      //...done
235             }
236             if (EEPROM.read(BANK) == 0 && EEPROM.read(automodereg) == 0){
                                                //not used mode: zero
237                 printfile = SD.open(SDNameString(adval),FILE_WRITE);//create file
238                 Serial.print("New file: ");
239                 Serial.println(SDNameString(adval));
240                 Serial.print("Writing headers...");                  //write mode zero
                    header...
241                 printfile.print("Time");
242                 printfile.print(" , ");
243                 printfile.print("X");
244                 printfile.print(" , ");
245                 printfile.print("Y");
246                 printfile.print(" , ");
247                 printfile.print("B440");
248                 printfile.print(" , ");
249                 printfile.print("-");
250                 printfile.print(" , ");
251                 printfile.print("-");
252                 printfile.print(" , ");
253                 printfile.print("integration-time");
254                 printfile.print(" , ");
255                 printfile.print("gain");
256                 printfile.print(" , ");
257                 printfile.println("B490");
258                 printfile.close();
259                 Serial.println("done");
260             }
261             if (EEPROM.read(automodereg) == 1){                             //not used
                    mode: zero
262                 printfile = SD.open(SDNameString(adval),FILE_WRITE);//create file
263             //Serial.print("New File: ");
264             //Serial.println(SDNameString(adval));
265             Serial.print("Writing automode headers...");                  //write mode
                    zero header...
266                 printfile.print("Time");
267                 printfile.print(" , ");
268                 printfile.print("X");
269                 printfile.print(" , ");
270                 printfile.print("(-Y)");
271                 printfile.print(" , ");
272                 printfile.print("(-Z)");
273                 printfile.print(" , ");
```

```
274                 printfile.print("X-gain");
275                 printfile.print("␣,␣");
276                 printfile.print("X-integration-time");
277                 printfile.print("␣,␣");
278                 printfile.print("X␣true␣value");
279                 printfile.print("␣,␣");
280                 printfile.print("|");
281                 printfile.print("␣,␣");
282                 printfile.print("(-X)");
283                 printfile.print("␣,␣");
284                 printfile.print("Y");
285                 printfile.print("␣,␣");
286                 printfile.print("(-Z)");
287                 printfile.print("␣,␣");
288                 printfile.print("Y-gain");
289                 printfile.print("␣,␣");
290                 printfile.print("Y-integration-time");
291                 printfile.print("␣,␣");
292                 printfile.print("Y␣true␣value");
293                 printfile.print("␣,␣");
294                 printfile.print("|");
295                 printfile.print("␣,␣");
296                 printfile.print("(-X)");
297                 printfile.print("␣,␣");
298                 printfile.print("(-Y)");
299                 printfile.print("␣,␣");
300                 printfile.print("Z");
301                 printfile.print("␣,␣");
302                 printfile.print("Z-gain");
303                 printfile.print("␣,␣");
304                 printfile.print("Z-integration-time");
305                 printfile.print("␣,␣");
306                 printfile.print("Z␣true␣value");
307                 printfile.print("␣,␣");
308                 printfile.print("|");
309                 printfile.print("␣,␣");
310                 printfile.print("x␣*1000");
311                 printfile.print("␣,␣");
312                 printfile.println("y␣*1000");
313                 printfile.close();
314             Serial.println("done");
315             }
316         printfile.close();
317         EEPROM.write(automodereg+1,0);
318         }
319         printfile = SD.open(SDNameString(adval),FILE_WRITE);  //open file
320         if(printfile){
321             printfile.print(TIME);          //writing time to SD-Card
322             printfile.print(",");
323             printfile.println(dataString); //writing data to SD-Card
324
325             Serial.print("Sensor:␣");
326             Serial.print(adval);
327             Serial.print("␣");
328             Serial.println(dataString);
329             printfile.close();
330         }
331     }
332 }
```

Listing 11: Teensy Klasse der BaLibTEE.cpp Datei

```
344
345 //the Teensy class
346 Teensy::Teensy(){}
347
348 //Function sets Teensy time to compile time
349 void Teensy::comptime(const char *t , const char *d){
350     SDCARD flash;
351
352     if(readStringEEPROM(70)!= t){    // if it is a reboot after new comilaion
353         flash.timedif(t);            // calculate the time difference
354         Serial.print("Setting␣time...");
```

```
355         if(flash.settime(t) && flash.setdate(d)){    // if set time and date is complete
356             if (RTC.write(tm)) {                     // if RTC is updated
357                 Serial.print("completed:␣");
358                 writeStringEEPROM(70, t);
359                 Serial.println(readStringEEPROM(70));
360             }else{
361                 Serial.println("failed!");
362             }
363         }
364     }else{                          //if it is just a reboot:
365         Serial.print("Time␣still␣is␣");
366         Serial.println(flash.gettimeString());
367     }
368 }
369
370 //Function writes String to EEPROM beginning at address add
371 void Teensy::writeStringEEPROM(char add, String data){
372     int _size = data.length();
373     int i;
374
375     for(i=0;i<_size;i++){   //for the lenght of the string
376         EEPROM.write(add+i,data[i]);// write every digit to the following address
377     }
378
379     EEPROM.write(add+_size,'\0');   //end with \0
380 }
381
382 //Function reads EEPROM Strings starting with address add
383 String Teensy::readStringEEPROM(char add){
384     char data[100];      //define maximum length to 100
385     int length=0;        //run variable = 0
386     unsigned char k = EEPROM.read(add);
387
388     while(k != '\0' && length<100){ //if add is not empty and the length is under 100
389         k=EEPROM.read(add+length);
390         data[length]=k;              //add read data to data string
391         length++;
392     }
393
394     data[length]='\0';       //end char data with \0
395     return String(data);     //return as string
396 }
397
398 void Teensy::initializing(){
399     pinMode(datapin, OUTPUT);
400     pinMode(ok_pin, OUTPUT);
401     pinMode(sd_error_pin, OUTPUT);
402     pinMode(i2c_error_pin, OUTPUT);
403     Serial.begin(115200);
404     EEPROM.write(ERR,0);         //writing the ERROR byte in EEPROM to zero to reset!
405     EEPROM.write(PROG,0);
406     Serial.println("RESTART");
407     Serial.println("--------------------");
408 }
409
410 bool Teensy::acquisition(){     //new Datapoint?
411     SDCARD flash;
412     String S =flash.gettimeString();     //setting S to current time
413     long timeinsec = ((S[0]-48)*10+(S[1]-48))*360+((S[3]-48)*10+(S[4]-48))*60+((S[6]-48)
        *10+(S[7]-48)); //calculating the time in sec since midnight
414
415     long intervalsec = EEPROM.read(inter)+EEPROM.read(inter + 1)*60+EEPROM.read(inter +
        2)*60*60;   //calculating the interval seconds
416
417     if (timeinsec%intervalsec == 0 && S !=readStringEEPROM(inter+5)){    // if time
        divided by interval is a hole number and the time is new:
418         writeStringEEPROM(inter+5,S);                                    // write newtime
            in EEPROM
419         return true;                                                     // <- new
            Datapoint!
420     }
421     return false;                                                        //else: no/old
        Datapoint!
422 }
```

```
423
424 //Function return if there is a problem with a)SD-Card b)RTC
425 bool Teensy::problem(){
426     AS7264N sensor;
427     SDCARD flash;
428
429     //turn all LED's off
430     digitalWrite(ok_pin,LOW);
431     digitalWrite(i2c_error_pin,LOW);
432     digitalWrite(sd_error_pin,LOW);
433
434     //if SD-Card is not available return true
435     if (flash.available() == false){
436         digitalWrite(sd_error_pin,HIGH);
437         if(EEPROM.read(ERR)!=1){     //<- if this is a first time error
438             flash.available();
439             Serial.println("ERROR: inject SD-CARD!");
440             flash.available();
441             Serial.println("");
442         }
443         EEPROM.write(ERR,1);
444         delay(5);
445         return true;
446     }
447
448     //if RTC is not available return true
449     if (sensor.i2cavailable() == false){
450         digitalWrite(i2c_error_pin,HIGH);
451         if(EEPROM.read(ERR)!=2){     //<-if this is a first time error
452             flash.available();
453             Serial.println("ERROR: inject RTC!");
454             Serial.println("");
455         }
456         EEPROM.write(ERR,2);
457         delay(5);
458         return true;
459     }
460
461     //if both is available return false! <- no problem!
462     if (flash.available() == true && sensor.i2cavailable() == true){
463         digitalWrite(ok_pin,HIGH);
464         if(EEPROM.read(ERR)!=0){
465             Serial.println("back to normal");
466             Serial.println();
467         }
468         EEPROM.write(ERR,0);
469         return false;
470     }
471     return false;
472 }
473
474 //Funktion compares the input to command-options:
475 //for every command input this function is only run once
476 //there are two types of commands
477 // a) with no value ("back","display","temperature","search")
478 // b) with value    ("interval","gain","intTime","waitTime" )
479 void Teensy::command(int value, String string) {
480     AS7264N sensor;
481     SDCARD flash;
482                                         //no value ---------
483     if (string.indexOf("back") >=0){    //EEPROM(PROG) = 0!
484         EEPROM.write(PROG,0);
485         flash.writeprogcom();
486     }
487
488     if (string.indexOf("display") >=0){
489         display();
490     }
491
492     if (string.indexOf("repeat") >= 0) {
493         flash.writeprogcom();
494     }
495
496     if (string.indexOf("temperature") >= 0) {
```

```
497             Serial.print("Temperature is:");
498             Serial.println(value);
499             Serial.println();
500         }
501
502         if (string.indexOf("search") >= 0) {
503             Serial.println("Searching for Sensors:");
504             sensor.Search();
505             Serial.println();
506         }
507                                                 //with value ------
508         //if EEPROM(PROG) is 1,2,3,4 you can change the values
509         if (EEPROM.read(PROG) == 1 ){
510             if (string.indexOf('s') >= 0) {
511                 EEPROM.write(inter, value);
512             }
513             if (string.indexOf("min") >= 0) {
514                 EEPROM.write(inter + 1, value);
515             }
516             if (string.indexOf("h") >= 0) {
517                 EEPROM.write(inter + 2, value);
518             }
519             if (EEPROM.read(inter)==0 && EEPROM.read(inter+1)==0 && EEPROM.read(inter+2)==0){
520                 EEPROM.write(inter, 1);
521             }
522
523             Serial.println();
524             Serial.print("Interval set to ");
525             Serial.print(EEPROM.read(inter + 2));
526             Serial.print("h ");
527             Serial.print(EEPROM.read(inter + 1));
528             Serial.print("min ");
529             Serial.print(EEPROM.read(inter));
530             Serial.println("s");
531         }
532
533         if(EEPROM.read(PROG) == 2 && value < 5){
534             if(value!=0){
535                 Serial.println();
536                 Serial.print("gain set to ");
537                 if (value ==1) {
538                     EEPROM.write(GAIN, 0);
539                     Serial.println("1 -> 1x");
540                 }
541                 if (value ==2) {
542                     EEPROM.write(GAIN, 1);
543                     Serial.println("2 -> 3.7x");
544                 }
545                 if (value == 3) {
546                     EEPROM.write(GAIN, 2);
547                     Serial.println("3 -> 16x");
548                 }
549                 if (value == 4) {
550                     EEPROM.write(GAIN, 3);
551                     Serial.println("4 -> 64x");
552                 }
553             }
554         }
555
556         if (EEPROM.read(PROG) == 3){
557             if(value < 256 && value>0){
558                 EEPROM.write(INT_T, 255-value);
559                 Serial.println();
560                 Serial.print("Integration time set to ");
561                 Serial.print((255-EEPROM.read(INT_T)) * 2.8);
562                 Serial.print(" ms -> intTime value: ");
563                 Serial.println(EEPROM.read(INT_T));
564             }
565         }
566
567         if (EEPROM.read(PROG) == 4) {
568             if(value < 256){
569                 EEPROM.write(INT_WT, 255-value);
570                 Serial.print("Integration wait time set to ");
```

```
571             Serial.print((255-EEPROM.read(INT_WT)) * 2.8);
572             Serial.print(" ms -> intTime value: ");
573             Serial.println(EEPROM.read(INT_WT));
574         }
575     }
576
577     if (EEPROM.read(PROG) == 5) {
578         if (string.indexOf("on") >= 0) {
579             EEPROM.write(automodereg, 1);
580             EEPROM.write(automodereg+1, 1);
581             Serial.println();
582             Serial.print("Automode turned on!");
583             flash.newheadder();
584             Serial.println();
585         }
586         if (string.indexOf("off") >= 0) {
587             EEPROM.write(automodereg, 0);
588             EEPROM.write(automodereg+1, 2);
589             Serial.println();
590             Serial.print("Automode turned off!");
591             flash.newheadder();
592             Serial.println();
593         }
594     }
595
596     //set EEPROM(PROG) to 1,2,3,4
597     if (string.indexOf("interval") >= 0) {       //EEPROM(PROG) = 1!
598         EEPROM.write(PROG,1);
599         Serial.println();
600         Serial.print("Interval is ");
601         Serial.print(EEPROM.read(inter + 2));
602         Serial.print("h ");
603         Serial.print(EEPROM.read(inter + 1));
604         Serial.print("min ");
605         Serial.print(EEPROM.read(inter));
606         Serial.println("s");
607         Serial.println("You can now set the interval in s/min/h !");
608         Serial.println("Type value in front then s/min/h and press enter");
609         Serial.println("If you are done type: 'back' ");
610         Serial.println("Node: You can only change one parameter (s/min/h) at a time!");
611         Serial.println();
612         return;
613     }
614
615     if (string.indexOf("gain") >= 0) {           //EEPROM(PROG) = 2!
616         EEPROM.write(PROG,2);
617
618         Serial.print("gain is ");
619         int val = EEPROM.read(GAIN);
620         if (val == 1) {
621             Serial.println("1 -> 1x");
622         }
623         if (val == 2) {
624             Serial.println("2 -> 3.7x");
625         }
626         if (val == 3) {
627             Serial.println("3 -> 16x");
628         }
629         if (val == 4) {
630             Serial.println("4 -> 64x");
631         }
632         // if (val == 5) {
633         //     Serial.println("auto");
634         // }
635         Serial.println();
636         Serial.println("You can now set the gain by typing:");
637         Serial.println("1 for 1x");
638         Serial.println("2 for 3.7x");
639         Serial.println("3 for 16x");
640         Serial.println("4 for 64x");
641         Serial.println("If you are done type: 'back' ");
642     }
643
644     if (string.indexOf("intTime") >= 0) {        //EEPROM(PROG) = 3!
```

```
645         EEPROM.write(PROG,3);
646         Serial.print("intTime is ");
647         Serial.println(EEPROM.read(INT_T));
648         Serial.println((255-EEPROM.read(INT_T)) * 2.8);
649         Serial.println();
650         Serial.println("You can now set the integration time!");
651         Serial.println("Type value 0-255 <-intTime will be value*2,8!");
652         Serial.println("If you are done type: 'back' ");
653     }
654
655     if (string.indexOf("waitTime") >= 0) {       //EEPROM(PROG) = 4!
656         EEPROM.write(PROG,4);
657
658         Serial.print("waitTime is ");
659         byte wtime = EEPROM.read(INT_WT);
660         Serial.print((225-wtime)*2.8);
661         Serial.println(" ms");
662         Serial.println("You can now set the integration waiting time!");
663         Serial.println("Type value 0-255 <-waitTime will be value*2,8!");
664         Serial.println("If you are done type: 'back' ");
665     }
666
667     if (string.indexOf("automode") >= 0) {
668         EEPROM.write(PROG,5);
669         Serial.println();
670         Serial.println("You can now enable/disable the automode by typing 'on' or 'off' "
            );
671         //Serial.println("Type value 0-20 <- Number of iteration for one measurement");
672         Serial.println("If you are done type: 'back' ");
673     }
674 }
675
676 //displays all changeable settings
677 void Teensy::display(){
678
679     Serial.println();
680     Serial.println("All changeable parameters:");
681         Serial.print("Interval is ");
682         Serial.print(EEPROM.read(inter + 2));
683         Serial.print("h ");
684         Serial.print(EEPROM.read(inter + 1));
685         Serial.print("min ");
686         Serial.print(EEPROM.read(inter));
687         Serial.println("s");
688
689         Serial.print("Gain is ");
690         int val = 1+EEPROM.read(GAIN);
691         if (val == 1) {
692             Serial.println("1 -> 1x");
693         }
694         if (val == 2) {
695             Serial.println("2 -> 3.7x");
696         }
697         if (val == 3) {
698             Serial.println("3 -> 16x");
699         }
700         if (val == 4) {
701             Serial.println("4 -> 64x");
702         }
703
704         Serial.print("Integration time is ");
705         Serial.print((255-EEPROM.read(INT_T)) * 2.8);
706         Serial.print(" ms -> intTime value: ");
707         Serial.println(EEPROM.read(INT_T));
708
709         Serial.print("Integration wait time is ");
710         Serial.print((255-EEPROM.read(INT_WT)) * 2.8);
711         Serial.print(" ms -> intTime value: ");
712         Serial.println(EEPROM.read(INT_WT));
713 }
714
715 //Function resets settings to factory settings
716 void Teensy::reset(){
717
```

```
718     EEPROM.write(BANK, 0x80);           //Bank ->addr. 10
719     EEPROM.write(GAIN, 0x03);           //Gain ->addr. 20
720     EEPROM.write(INT_WT, 0x80);         //WTime->addr. 30
721     EEPROM.write(INT_T, 0x00);          //intT ->addr. 40
722     EEPROM.write(inter, 0x05);          //interval ->addr. 50
723     EEPROM.write(inter + 1, 0x00);      //interval ->addr. 50
724     EEPROM.write(inter + 2, 0x00);      //interval ->addr. 50
725     EEPROM.write(automodereg, 0x00);    //automode off!
726
727     Serial.println();
728     Serial.println("Reset all settings!");
729     Serial.println();
730 }
```

Listing 12: AS7264N Klasse der BaLibTEE.cpp Datei

```
740 // the AS7264N class:
741 AS7264N::AS7264N(){}
742 long globalX,globalY,globalZ;
743
744 //this function returns the data of one Sensor at the multiplexer(plex) and channel(ch):
745 String AS7264N::getMeasurement( byte plex , byte ch){
746     //
747     //this part chooses the right multiplexer(plex) and channel(ch)
748     int plexaddr = TCA_ADDR;                //plexaddr set to 0x70
749     while(plexaddr < 0x78){                 //repeat until plexaddr = 0x77  (all possible
        multiplexer)
750         Wire.beginTransmission(plexaddr);
751             if(plex == plexaddr-0x70){      //if its the right plex:
752                 Wire.write(1 << ch);        //write the bit with the correct channel high
753             }else{                          //wrong plex:
754                 Wire.write(0x00);           //set all channels/bits to "0" <- disable this
                        multiplexer
755             }
756         Wire.endTransmission();
757         plexaddr++;
758     }
759     if(EEPROM.read(automodereg)==1){
760         return makeautomaticmeasurement();
761     }
762     return getmanualmeasurement();
763 }
764
765 String AS7264N::getmanualmeasurement(){
766 int X,Y,Z,NIR = 0;
767 double Xd,Yd,Zd;
768 String Sx,Sy;
769 int inttime = EEPROM.read(INT_T);
770 int gain = EEPROM.read(GAIN);
771 int wtime = 255;
772
773 resetSensor();
774 while(readRegister(byte(0xEE))+readRegister(byte(0xDE))+readRegister(byte(0xEC))==0){
775     setupXYZ(gain,inttime,wtime);
776 }
777     X   =   readRegister(byte(0xEE));
778     Y   =   readRegister(byte(0xDE));
779     Z   =   readRegister(byte(0xEC));
780     NIR =   readRegister(byte(0xDC));
781     Xd=EEPROM.read(Xcomp)*100;
782     Yd=EEPROM.read(Ycomp)*100;
783     Zd=EEPROM.read(Zcomp)*100;
784
785     if(EEPROM.read(BANK)==0x80){    //if normal mode:
786         Sx    = String((Xd/(Yd+Xd+Zd)*100));
787         Sy    = String((Yd/(Yd+Xd+Zd)*100));
788     }else{                          //is secret mode:
789         Sx    = "-";
790         Sy    = "-";
791     }
792     String SX    = String(X);   //convert to Strings
793     String SY    = String(Y);   //convert to Strings
794     String SZ    = String(Z);   //convert to Strings
```

```
795     String Sinttime = String(inttime);   //convert to relative Strings
796     String Sgain    = String(gain);   //convert to relative Strings
797
798     String all  = String( SX + "," + SY + "," + SZ + "," + Sx + "," +Sy + "," + Sinttime
            + "," + Sgain + "," + NIR);
799     return all;
800 }
801
802 void AS7264N::Search(){
803     byte val;
804     byte val2;
805     byte plexaddr=TCA_ADDR;//plexaddr set to 0x70
806
807     while(plexaddr<0x78){                    //repeat until plexaddr = 0x77  (all possible
            multiplexer)
808         Wire.beginTransmission(plexaddr);
809         val = Wire.endTransmission();
810         if(val == 0){       //if there is a plex with the plexaddr
811             int i = 1;
812             while(i < 9){   //repeat until i = 8 (channel)
813                 Wire.beginTransmission(plexaddr);    //begin writing to plexaddr
814                 Wire.write(1 << i);                  //write channel bit i to 1
815                 Wire.endTransmission();              //end
816                 Wire.beginTransmission(AS7264N_ADDR);    //try to connect a sensor
817                 val2 = Wire.endTransmission();           //disconnect
818                 if (val2 == 0){     //if there is a sensor
819                     Serial.print("Sensor␣found␣at␣multiplexer␣addr.␣");
820                     Serial.print(plexaddr , HEX);            //print the plexaddr in HEX
                            ==>DEC
821                     Serial.print("␣[");
822                     if(plexaddr-0x70 < 4){Serial.print("0");}//<-this part prints
823                     if(plexaddr-0x70 < 2){Serial.print("0");}//<-the binary address
824                     Serial.print(plexaddr-0x70,BIN);         //<-of the plex
825                     Serial.print("]␣");
826                     Serial.print("␣Gate␣no.␣");
827                     Serial.println(i , HEX);                 //print the channel
828                 }
829             i++;
830             }
831         Serial.println();
832         }
833     plexaddr++;
834     }
835 }
836
837 //Function return all found sensors in a string ->(plex1+ch1+plex2+ch2.....)
838 String AS7264N::getSearchString(){
839     String all;
840     byte val;
841     byte val2;
842     byte plexaddr=TCA_ADDR;//plexaddr set to 0x70
843
844     while(plexaddr<0x78){//repeat until plexaddr = 0x77  (all possible multiplexer)
845         Wire.beginTransmission(plexaddr);
846         val = Wire.endTransmission();
847         if(val == 0){//if there is a plex with the plexaddr
848             int i = 1;
849             while(i < 9){//repeat until i = 8 (channel)
850                 Wire.beginTransmission(plexaddr);    //begin writing to plexaddr
851                 Wire.write(1 << i);                  //write channel bit i to 1
852                 Wire.endTransmission();              //end
853                 Wire.beginTransmission(AS7264N_ADDR);    //try to connect a sensor
854                 val2 = Wire.endTransmission();           //disconnect
855                 if (val2 == 0){                      //if there is a sensor
856                     all += String(plexaddr-0x70,DEC);    //add plexaddr(single digit) to
                            string
857                     all += String(i,DEC);                //add channel i to string
858                 }
859             i++;
860             }
861         }
862     plexaddr++;
863     }
864     return all; //return the hole String!
```

```
865 }
866
867 //Function returns if RTC is available.
868 boolean AS7264N::i2cavailable(){
869     Wire.beginTransmission(DS_ADDR);
870     byte val = Wire.endTransmission();
871     if(val == 0){   //if connection to RTC:
872         return true;
873     }else{          //no connection:
874         return false;
875     }
876 }
877
878 //Function retuns byte from the register-address of the AS7264N
879 byte AS7264N::readRegister(byte addr){
880     Wire.beginTransmission(AS7264N_ADDR);    //connect to sensor
881     Wire.write(addr);                        //write register address
882     Wire.endTransmission();                  //disconnect
883
884     Wire.requestFrom(AS7264N_ADDR, 1);       //request reply
885     if (Wire.available()) {                  //if reply:
886     return (Wire.read());                    //return read byte
887     }
888     else {                                   //no reply:
889     Serial.println("I2C␣Error");
890     return (0xFF); //Error
891     }
892 }
893
894 //Funcion writes value to register-address  of the AS7264N
895 void AS7264N::writeRegister(byte addr, byte val){
896     Wire.beginTransmission(AS7264N_ADDR);    //connect to sensor
897     Wire.write(addr);                        //write register address
898     Wire.write(val);                         //write value to register
899     Wire.endTransmission();                  //disconnect
900 }
901
902 //Function returns the measurement in automode
903 String AS7264N::makeautomaticmeasurement(){
904 String all;
905     for(int i=0;i<3;i++){       //Searches for the optimal gain&intTime (each XYZ-
        parameter)
906         Serial.print("search␣parameter␣");
907         Serial.print(i);
908         Serial.print("...");
909         automatic(i);
910         Serial.println("␣␣done");
911     }
912     for(int i=0;i<3;i++){       //Measuring XYZ with the predetermined parameters
913         Serial.print("measurement␣");
914         Serial.print(i);
915         Serial.print("...");
916         Serial.println("␣␣done");
917         all+=getone(i);
918         all+=",␣|␣,";
919     }
920     all+= String(1000*globalX/(globalX+globalY+globalZ));   //adds relative value for
        easy troubleshooting
921     all+= String(",");
922     all+= String(1000*globalY/(globalX+globalY+globalZ));
923     return all;
924 }
925
926 //Function returns the Registervalues for the predetermined parameters (searched one not
    negative)
927 String AS7264N::getone(int anteil){ //anteil->X=0,Y=1,Z=2
928 int factor = 0;
929 int X,Y,Z = 0;
930 int inttime = EEPROM.read(timeValue+anteil);     //get optimal intTime for XorYorZ
931 int gain = EEPROM.read(gainValue+anteil);        //get optimal gain for XorYorZ
932 int wtime = 255;
933
934 resetSensor();
935
```

```
936 //wait for Values in Registers:
937 while(readRegister(byte(0xEE))+readRegister(byte(0xDE))+readRegister(byte(0xEC))==0){
938     setupXYZ(gain,inttime,wtime);
939 }
940     X   =   readRegister(byte(0xEE));
941     Y   =   readRegister(byte(0xDE));
942     Z   =   readRegister(byte(0xEC));
943 String SX;
944 String SY;
945 String SZ;
946 String Strue;
947
948 //the folowing necessary for unsecase
949 //(for calculating x/y)
950 switch (gain) {
951   case 0:
952     factor = 1;
953     break;
954   case 1:
955     factor = 3.7;
956     break;
957   case 2:
958     factor = 16;
959     break;
960   case 3:
961     factor = 64;
962     break;
963 }
964         //inverts X/Y/Z that is not looked vor and possibly wrong
965         SX = String(X);
966     if(anteil==0){
967         globalX=1000*X/(factor*(255-inttime));
968         Strue = String(10000*X/(factor*(255-inttime)));
969     }else{ SX = String(-X);}
970     if(anteil==1){
971         SY = String(Y);
972         globalY=1000*Y/(factor*(255-inttime));
973         Strue = String(10000*Y/(factor*(255-inttime)));
974     }else{ SY = String(-Y);}
975     if(anteil==2){
976         SZ = String(Z);
977         globalZ=1000*Z/(factor*(255-inttime));
978         Strue = String(10000*Z/(factor*(255-inttime)));
979     }else{ SZ = String(-Z);}
980
981         //returns the string for ONE of X/Y/Z
982         String Sinttime = String(inttime);
983         String Sgain = String(gain);
984         String all  = String( SX + "," + SY + "," + SZ + "," + Sgain + "," + Sinttime+ ",
              " + Strue);
985         return all;
986 }
987
988 //Function finds the optimal parameters for automode!
989 void AS7264N::automatic(int para){
990 int val=0;
991 int val2=0;
992 int repeatcounter = 0;
993 int X,Y,Z;
994 int Xr,Yr,Zr;
995 int maxvalue = 200;
996 int jumpgain = 30;
997 int inttime = 255;
998 int gain = 0;
999 int wtime = 100;
```

Listing 13: BachelorTeensy

```
1 /* This is the Teensy scatch using the Arduino IDE for the Bachelor thesis
2  * "Entwicklung einer Messeinrichtung auf Basis eines Einplatinencomputers und des
       Sensors AS7264N von ams"
3  * by Matthias Schaale-Segeroth
4  *
```

```
 * For using this scribt the following libraries are needed
 * and can be downloaded from this github page:
 * - BaLibTEE.h     <-this Library is written by me for this occasion
 * - EEPROM.h
 * - Arduino.h
 * - Wire.h
 * - SD.h
 * - SPI.h
 * - TimeLib.h
 * - DS1307RTC.h
 */


//Beginning of the Sketch:

//inclusion of the libraries
#include <BaLibTEE.h> //my lib
#include <EEPROM.h>     //library for writing to internal memory

// Create instance of the classes provided by the 'BaLibTEE.h' library
AS7264N sensor;
SDCARD flash;
Teensy interface;

// introduction of the variables
int modus = 1;               //defauld mode of the Teensy
String inputString = "";     //inputString is empty
String commandString = "";   //commandString is empty
boolean newcommand = false; //no new command
bool erroverwrite = false;//no data while error

// Begin setup  (only run once at the beginning)
void setup() {
  delay(1000);
    pinMode(datapin, OUTPUT);    //LED pin init.
    pinMode(ok_pin, OUTPUT);
    pinMode(sd_error_pin, OUTPUT);
    pinMode(i2c_error_pin, OUTPUT);

    Serial.begin(115200);        //begin serial connection

    Serial.println("RESTART");
    Serial.println("--------------------");
  interface.comptime(__TIME__,__DATE__);  //if newly uploaded sets to compiler time and
      calculates time difference
  flash.initializing();                   //initializing of the SD-Card
}


// continues loop (repeats itself)
void loop() {

  if (interface.problem() && !erroverwrite){
      goto skiped;  //if there is a problem -> start again
  }
  //this part collects the data (RECORDING MODE)
  if (modus == 1 || modus == 3 || modus == 4) {
    if (interface.acquisition()|| modus == 4){  //if its the correct interval
      Serial.println();
      allSensorData();          //record the data
    }
  }

  //this part enables the commands (PROGRAMMING MODE)
  if (modus == 2 || modus == 3) {
    if (newcommand == true) {   //  if a new command is entered
      interface.command(commandString.toInt(), commandString);  //run the command ->(
          value,command)
      commandString = ""; //erase command
      newcommand = false; //no new comamnd
    }
  }

  skiped:
```

```
77    delay(100);
78  }
79
80  // After every void loop run the seialEvent function is called
81  void serialEvent() {
82    while (Serial.available()) {        //if a new command is entered
83      char inChar = (char)Serial.read();
84      inputString += inChar;            //add all letters to inputString
85      if (inChar == '\n') {                 //if inputString is complete
86        if (inputString.indexOf("M") > 0) {   //searches for the 'M'
87          modus = inputString.toInt();        //the value in front of the M is set to mode
88          if (modus != 1 && modus != 2 && modus != 3 && modus != 4) {  //if modus is not
                1,2,3 go to sleep mode
89            Serial.println();
90            Serial.println("SLEEPING");
91            Serial.println();
92            erroverwrite=false;
93          }
94          if (modus == 1) {     //if modus is 1 go to recording mode (default)
95            Serial.println();
96            Serial.println("RECORDING MODE");
97            Serial.println();
98            erroverwrite=false;
99          }
100         if (modus == 2) {     //if modus is 2 go to programming mode
101           Serial.println();
102           Serial.println("PROGRAMMING MODE");
103           flash.writeprogcom();   //prints the programming commands
104           erroverwrite=false;
105         }
106         if (modus == 3) {     //if modus is 3 go to recording and programming mode
107           Serial.println();
108           Serial.println("RECORDING + PROGRAMMING MODE");
109           flash.writeprogcom();   //prints the programming commands
110           erroverwrite=false;
111         }
112         if (modus == 4) {     //if modus is 4 go to independent mode
113           Serial.println();
114           Serial.println("INDEPEDET MODE");
115           erroverwrite=true;
116         }
117         inputString = "";
118       }
119       // reset
120       if (inputString.indexOf("reset") >= 0) {  //if "reset" is written reset all
            parameters
121         interface.reset();  //reset function
122         erroverwrite=false;
123       }
124       newcommand = true;          //there is a new command
125       commandString = inputString;//define commandString
126       inputString = "";           //erase inputString
127     }
128   }
129 }
130
131
132 //----------------- the following functios are only run if triggert in other functions
      -----------------
133
134 // function saves and prints all data from every sensor
135 void allSensorData() {
136     Serial.println(flash.gettimeString());  //print timestamp
137
138     String sstr = sensor.getSearchString(); //gets adresses(plex'+'ch) of all conected
          sensors
139     for (int i=0; i< sstr.length();i=i+2) { //every adress has two digits -> repeats for
          each found sensor
140         byte plex = sstr[i] - 48;     //first digit of every adress is plex no. (-48
              because of ASCII)
141         byte ch = sstr[i + 1] - 48;   //second digit is channel number (-48 because of
              ASCII)
142         collectSensorData(plex, ch);  //run function below for every sensor
143     }
```

```
144     flash.endnewheadder(); // if all csv-headders are written the first time, stop
            writing them for next data aquision loop
145 }
146
147 // function writes sensor data of one sensor (at channel->ch and Miltiplexer->plex) on
        the SD-Card
148 void collectSensorData(byte plex, byte ch) {
149   if(flash.available()||erroverwrite){      // if SD-available
150       digitalWrite(datapin, HIGH);  // set indicator LED on
151       if(!erroverwrite){
152         flash.writeData(sensor.getMeasurement(plex, ch), plex, ch); //(Data, Multiplexer,
                Sensor) -> write Data to the SD-Card with the filepath: year/month/date/plex
              +ch
153       }
154       if(erroverwrite){
155         Serial.print("Sensor ");//prints sensor and channel
156         Serial.print(plex);
157         Serial.print(ch);
158         Serial.print(": ");
159         Serial.println(sensor.getMeasurement(plex, ch));//prints Data ->no SD-Card
              writing!
160       }
161       digitalWrite(datapin, LOW);    // set indicator LED off
162   }
163 }
```

Listing 14: Matlab

```
1  %%Variablen:
2  MP=[69 107 111 120 129 144 355 366 371 375 382 390];
3  MW=[1 6 7 11];
4  WW=[2 5 8 12];
5  BLUE=[3 4 9 10];
6
7  fak=2.75;%<- Z-Kanalverstaerkung
8
9  i=1;
10 while i<=length(MP)%%<-erzeugt MW-Tabelle mit allen 12 Messwerten
11     MW(i,:)=SENSOR2(MP(i),:);
12      i=i+1;
13 end
14 i=1;
15
16 while i<=length(MW)/2 %%<-erzeugt XYZ-Tabellen mit allen 3 Messwerten
17     X(i,1)=MW(i,2)/(gainhB(MW(i,5))*(255-MW(i,6)));%<- eigentliche X-Messung
18     Y(i,1)=MW(i,3)/(gainhB(MW(i,5))*(255-MW(i,6)));
19     Z(i,1)=MW(i,4)/(gainhB(MW(i,5))*(255-MW(i,6)));
20     X(i,2)=MW(i,9)/(gainhB(MW(i,12))*(255-MW(i,13)));%<- eigentliche Y-Messung
21     Y(i,2)=MW(i,10)/(gainhB(MW(i,12))*(255-MW(i,13)));
22     Z(i,2)=MW(i,11)/(gainhB(MW(i,12))*(255-MW(i,13)));
23     X(i,3)=MW(i,2)/(gainhB(MW(i,5))*(255-MW(i,6)));%<- eigentliche maximale Aufl?sung
24     Y(i,3)=MW(i,10)/(gainhB(MW(i,12))*(255-MW(i,13)));
25      if i==4
26     Y(i,3)=MW(i,3)/(gainhB(MW(i,5))*(255-MW(i,6)));
27     end
28     Z(i,3)=MW(i,18)/(gainhB(MW(i,19))*(255-MW(i,20)));
29      i=i+1;
30 end
31 i=1;
32
33 while i<=length(MW)/2    %<- erzeugt real Werte
34          RW(i,1)=X(i,3);
35          RW(i,2)=Y(i,3);
36          RW(i,3)=Z(i,3);
37       i=i+1;
38  end
39
40 i=1;
41 Z=Z*fak; %%<- Anpassung des Z-Kanals
42
43 while i<=length(MW)/2    %<- erzeugt Farborte
44        FO(i,1)=X(i,1)/(X(i,1)+Y(i,1)+Z(i,1));
45        FO(i,2)=Y(i,1)/(X(i,1)+Y(i,1)+Z(i,1));
```

```
46          FO(i,4)=X(i,2)/(X(i,2)+Y(i,2)+Z(i,2));
47          FO(i,5)=Y(i,2)/(X(i,2)+Y(i,2)+Z(i,2));
48          FO(i,7)=X(i,3)/(X(i,3)+Y(i,3)+Z(i,3));
49          FO(i,8)=Y(i,3)/(X(i,3)+Y(i,3)+Z(i,3));
50      i=i+1;
51  end
52  i=1;
53
54  ANW=returncolor(NW,FO);%1
55  AWW=returncolor(WW,FO);%2
56  Abl=returncolor(BLUE,FO);%3
57  while i<5
58  gemFO(i,1)=ANW(i,7);
59  gemFO(i,2)=ANW(i,8);
60  gemFO(i,4)=AWW(i,7);
61  gemFO(i,5)=AWW(i,8);
62  gemFO(i,7)=Abl(i,7);
63  gemFO(i,8)=Abl(i,8);
64     i=i+1;
65  end
66  i=1;
67
68  % plotFO(ANW,1,1)
69  % title('x-Farbwertanteil NW');
70  % xlabel('Abstand');
71  % ylabel('Farbwertanteil');
72  % legend({'Referenzwert (Jeti)','gemessen bei 30cm','gemessen bei 60cm','gemessen bei 120
        cm','gemessen bei 240cm'},'Location','northwest');
73  %
74  % plotFO(ANW,1,2)
75  % title('y-Farbwertanteil NW');
76  % xlabel('Abstand');
77  % ylabel('Farbwertanteil');
78  % legend({'Referenzwert (Jeti)','gemessen bei 30cm','gemessen bei 60cm','gemessen bei 120
        cm','gemessen bei 240cm'},'Location','northwest');
79  %
80  % plotFO(AWW,2,1)
81  % title('x-Farbwertanteil WW');
82  % xlabel('Abstand');
83  % ylabel('Farbwertanteil');
84  % legend({'Referenzwert (Jeti)','gemessen bei 30cm','gemessen bei 60cm','gemessen bei 120
        cm','gemessen bei 240cm'},'Location','northwest');
85  %
86  % plotFO(AWW,2,2)
87  % title('y-Farbwertanteil WW');
88  % xlabel('Abstand');
89  % ylabel('Farbwertanteil');
90  % legend({'Referenzwert (Jeti)','gemessen bei 30cm','gemessen bei 60cm','gemessen bei 120
        cm','gemessen bei 240cm'},'Location','northwest');
91  %
92  % plotFO(Abl,3,1)
93  % title('x-Farbwertanteil blue');
94  % xlabel('Abstand');
95  % ylabel('Farbwertanteil');
96  % legend({'Referenzwert (Jeti)','gemessen bei 30cm','gemessen bei 60cm','gemessen bei 120
        cm','gemessen bei 240cm'},'Location','northwest');
97  %
98  % plotFO(Abl,3,2)
99  % title('y-Farbwertanteil blue');
100 % xlabel('Abstand');
101 % ylabel('Farbwertanteil');
102 % legend({'Referenzwert (Jeti)','gemessen bei 30cm','gemessen bei 60cm','gemessen bei 120
        cm','gemessen bei 240cm'},'Location','northwest');
103
104
105 refFO=[0.3818  0.3794
106         0.5255  0.4112
107         0.1183  0.1089];
108 Yref=[7446
109     4923
110     2010
111     785
112     1936
113     2927
```

```
    886
    585
    238];



v=zeros(3);
while i<5
    v(1,1)=v(1,1)+(refFO(1,1)-ANW(i,7))^2;
    v(1,2)=v(1,2)+(refFO(1,2)-ANW(i,8))^2;
    v(2,1)=v(2,1)+(refFO(2,1)-AWW(i,7))^2;
    v(2,2)=v(2,2)+(refFO(2,2)-AWW(i,8))^2;
    v(3,1)=v(3,1)+(refFO(3,1)-Abl(i,7))^2;
    v(3,2)=v(3,2)+(refFO(3,2)-Abl(i,8))^2;

   i=i+1;
end
v=sqrt(v/4);
i=1;
while(i<=9)
    Yfak(i,1)=Y(i,3);
    Yfak(i,2)=Yref(i,1);
    Yfak(i,3)=Yref(i,1)/Y(i,3);
    i=i+1;
end
i=1;
mit=mean(Yfak)
sqrt(var(Yfak))./mean(Yfak)


function plotFO(FOS,color,val)
refFO=[0.3818  0.3794
       0.5255  0.4112
       0.1183  0.1089];
    figure()
    hold on
    lineY(refFO(color,val));%<-(color,X/Y)
    i=1;
    while i<5
        plot(30*2^(i-1),FOS(i,6+val),'*','MarkerSize', 15);%<-1/2NW 4/5WW 7/8BLUE
        i=i+1;
    end
        plot(0);
    plot(1);
    hold off
end
function reture = gainhB(g) %gibt gain laut HandBuch zur?ck
if g == 0
    reture=1;
elseif g==1
    reture=4.176;
elseif g==2
    reture=15.585;
else
    reture=65.9292;
end
end
function reture = returncolor(color,tab)
l=1;
while l<=length(color)
    reture(l,:) = tab(color(l),:);
    reture(l,:) = tab(color(l),:);
    l=l+1;
end
end
function lineY(y)
line([0,240],[y,y],'MarkerSize', 15)
end
```

Listing 15: Matlab2

```

```

```
3
4  %%Variablen:
5  MP=[69 135 111 120 129 144 355 366 371 375 382 390];
6  MP_GT_1=[515 521 525 529 535 540 550 555 566 577 593];
7  MP_GT_2=[598 605 610 620 631 640 645 658 663 674 681];
8
9
10 i=1;
11 while i<=length(MP_GT_1)%%<-erzeugt MW-Tabellen
12     MW1(i,:)=SENSOR2(MP_GT_1(i),:);
13     MW2(i,:)=SENSOR2(MP_GT_2(i),:);
14     i=i+1;
15 end
16 i=1;
17
18
19 %%intTime Linear???
20
21 % plotLinear(SENSOR2,MP_GT_1,0); %<-(SENSOR2,Messpunkte,gain)
22 % title('register value with variable intTime gainReg=0 distance=1m');
23 % xlabel('intTime');
24 % ylabel('reg.value');
25 % legend({'X-reg','Y-reg','Z-reg'},'Location','northwest');
26 %
27 % plotNormed(SENSOR2,MP_GT_1,0); %<-(SENSOR2,Messpunkte,gain)
28 % title('normed register value with variable intTime gainReg=0 distance=1m');
29 % xlabel('intTime');
30 % ylabel('reg.value/intTime');
31 % legend({'X-reg','Y-reg','Z-reg'},'Location','northwest');
32 %
33 % plotLinear(SENSOR2,MP_GT_2,1); %<-(SENSOR2,Messpunkte,gain)
34 % title('register value with variable intTime gainReg=1 distance=2m');
35 % xlabel('intTime');
36 % ylabel('reg.value');
37 % legend({'X-reg','Y-reg','Z-reg'},'Location','northwest');
38
39 % plotLinear(SENSOR2,MP_GT_2,2); %<-(SENSOR2,Messpunkte,gain)
40 % title('register value with variable intTime gain=16 distance=2m');
41 % xlabel('intTime');
42 % ylabel('reg.value');
43 % legend({'X-reg','Y-reg','Z-reg'},'Location','northwest');
44 %
45 % plotLinearcorr(SENSOR2,MP_GT_2,2); %<-(SENSOR2,Messpunkte,gain)
46 % title('register value with variable intTime gainReg=2 distance=2m');
47 % xlabel('intTime');
48 % ylabel('reg.value');
49 % legend({'X-reg','Y-reg','Z-reg'},'Location','northwest');
50 %%
51 % plotLinear(SENSOR2,MP_GT_2,3); %<-(SENSOR2,Messpunkte,gain)
52 % title('register value with variable intTime gainReg=3 distance=2m');
53 % xlabel('intTime');
54 % ylabel('reg.value');
55 % legend({'X-reg','Y-reg','Z-reg'},'Location','northwest');
56
57
58
59
60 %%%gain Linear???
61
62 % plotgainLinear(SENSOR2,MP_GT_1,10)
63 % title('register value with variable gain intTime=10 distance=1m');
64 % xlabel('gain');
65 % ylabel('reg.value');
66 % legend({'X-reg','Y-reg','Z-reg'},'Location','northwest');
67 %
68 % plotgainNormed(SENSOR2,MP_GT_1,20)
69 % title('normed register value with variable gain intTime=20 distance=1m');
70 % xlabel('gain');
71 % ylabel('reg.value/gain');
72 % legend({'X-reg','Y-reg','Z-reg'},'Location','northwest');
73
74 % plotgainLinear(SENSOR2,MP_GT_2,10)
75 % title('register value with variable gain intTime=10 distance=2m');
76 % xlabel('gain');
```

```
% ylabel('reg.value');
% legend({'X-reg','Y-reg','Z-reg'},'Location','northwest');
%
% plotgainNormed(SENSOR2,MP_GT_2,10)
% title('normed register value with variable gain intTime=10 distance=2m');
% xlabel('gain');
% ylabel('reg.value/gain');
% legend({'X-reg','Y-reg','Z-reg'},'Location','northwest');

while MW1(i,8) == 0
k(1,i)=4*MW2(i,2)./MW1(i,2);
k(2,i)=4*MW2(i,3)./MW1(i,3);
k(3,i)=4*MW2(i,4)./MW1(i,4);
mean(k);
Vf1=mean(mean(k));
i=i+1;
end
i=1;
while i <= 3
kl(1,i)=16*MW2(i,2)./MW2(i+5,2);
kl(2,i)=16*MW2(i,3)./MW2(i+5,3);
kl(3,i)=16*MW2(i,4)./MW2(i+5,4);
kl;
mean(kl);
Vf2=mean(mean(kl))*Vf1;
i=i+1;
end
i=1;

h(1,1)=MW2(10,2)/MW2(1,2);
h(2,1)=MW2(10,3)/MW2(1,3);
h(3,1)=MW2(10,4)/MW2(1,4);
h(1,2)=Vf2*MW2(10,2)/MW2(8,2);
h(2,2)=Vf2*MW2(10,3)/MW2(8,3);
h(3,2)=Vf2*MW2(10,4)/MW2(8,4);
h(1,3)=Vf2*MW2(11,2)/MW2(7,2);
h(2,3)=Vf2*MW2(11,3)/MW2(7,3);
h(3,3)=Vf2*MW2(11,4)/MW2(7,4);
h
Vf3=mean(mean(h))*Vf1;

function plotLinear(SENSOR2,MP,g)

figure()
hold on
Val = 2;
 while Val < 5
 l=1;
 k=2;
  while l<=length(MP)
    if SENSOR2(MP(l),8)== g
        if Val == 2
            %plot(255-SENSOR2(MP(l),7),SENSOR2(MP(l),Val),'*','MarkerSize', 12);
            m(k,1)=(255-SENSOR2(MP(l),7))*2.8;
            m(k,2)=SENSOR2(MP(l),Val);
        end
        if Val == 3
            %plot(255-SENSOR2(MP(l),7),SENSOR2(MP(l),Val),'+','MarkerSize', 12);
            m(k,3)=(255-SENSOR2(MP(l),7))*2.8;
            m(k,4)=SENSOR2(MP(l),Val);
        end
        if Val == 4
            %plot(255-SENSOR2(MP(l),7),SENSOR2(MP(l),Val),'x','MarkerSize', 12);
            m(k,5)=(255-SENSOR2(MP(l),7))*2.8;
            m(k,6)=SENSOR2(MP(l),Val);
        end
        k=k+1;
    end

    l=l+1;
  end
  Val=Val+1;
 end
  plot(m(:,1),m(:,2),'*','MarkerSize', 12);
```

```
151     plot(m(:,3),m(:,4),'+','MarkerSize', 12);
152     plot(m(:,5),m(:,6),'x','MarkerSize', 12);
153   plot(0,0);
154   hold off
155 end
156 function plotNormed(SENSOR2,MP,g)
157 figure()
158 hold on
159 Val = 2;
160   while Val < 5
161   l=1;
162     k=1;
163     while l<=length(MP)
164       if SENSOR2(MP(l),8)== g
165           if Val == 2
166               %plot(255-SENSOR2(MP(l),7),SENSOR2(MP(l),Val)/(255-SENSOR2(MP(l),7)),'*','
                      MarkerSize', 12);
167               m(k,1)=(255-SENSOR2(MP(l),7))*2.8;
168               m(k,2)=SENSOR2(MP(l),Val)/((255-SENSOR2(MP(l),7))*2.8);
169           end
170           if Val == 3
171               %plot(255-SENSOR2(MP(l),7),SENSOR2(MP(l),Val)/(255-SENSOR2(MP(l),7)),'+','
                      MarkerSize', 12);
172               m(k,3)=(255-SENSOR2(MP(l),7))*2.8;
173               m(k,4)=SENSOR2(MP(l),Val)/((255-SENSOR2(MP(l),7))*2.8);
174           end
175           if Val == 4
176               %plot(255-SENSOR2(MP(l),7),SENSOR2(MP(l),Val)/(255-SENSOR2(MP(l),7)),'x','
                      MarkerSize', 12);
177               m(k,5)=(255-SENSOR2(MP(l),7))*2.8;
178               m(k,6)=SENSOR2(MP(l),Val)/((255-SENSOR2(MP(l),7))*2.8);
179           end
180           k=k+1;
181       end
182
183       l=l+1;
184     end
185     Val=Val+1;
186   end
187     plot(m(:,1),m(:,2),'*','MarkerSize', 12);
188     plot(m(:,3),m(:,4),'+','MarkerSize', 12);
189     plot(m(:,5),m(:,6),'x','MarkerSize', 12);
190   plot(0,0);
191   hold off
192 end
193 function plotLinearcorr(SENSOR2,MP,g)
194 figure()
195 hold on
196 Val = 2;
197   while Val < 5
198   l=1;
199 k=2;
200     while l<=length(MP)
201       if SENSOR2(MP(l),8)== g
202           if Val == 2
203
204               if SENSOR2(MP(l),Val) > SENSOR2(MP(l-1),Val) || SENSOR2(MP(l-1),8)~=g
205                   %plot(255-SENSOR2(MP(l),7),SENSOR2(MP(l),Val),'*','MarkerSize', 12);
206                   m(k,1)=(255-SENSOR2(MP(l),7))*2.8;
207                   m(k,2)=SENSOR2(MP(l),Val);
208               else
209                   %plot(255-SENSOR2(MP(l),7),SENSOR2(MP(l),Val)+255,'*','MarkerSize', 12);
210                   m(k,1)=(255-SENSOR2(MP(l),7))*2.8;
211                   m(k,2)=SENSOR2(MP(l),Val)+255;
212               end
213           end
214           if Val == 3
215               if SENSOR2(MP(l),Val) > SENSOR2(MP(l-1),Val) || SENSOR2(MP(l-1),8)~=g
216                   %plot(255-SENSOR2(MP(l),7),SENSOR2(MP(l),Val),'+','MarkerSize', 12);
217                   m(k,3)=(255-SENSOR2(MP(l),7))*2.8;
218                   m(k,4)=SENSOR2(MP(l),Val);
219               else
220                   %plot(255-SENSOR2(MP(l),7),SENSOR2(MP(l),Val)+255,'+','MarkerSize', 12);
221                   m(k,3)=(255-SENSOR2(MP(l),7))*2.8;
```

```
                m(k,4)=SENSOR2(MP(l),Val)+255;
            end
        end
        if Val == 4
            if SENSOR2(MP(l),Val) > SENSOR2(MP(l-1),Val)|| SENSOR2(MP(l-1),8)~=g
                %plot(255-SENSOR2(MP(l),7),SENSOR2(MP(l),Val),'x','MarkerSize', 12);
                m(k,5)=(255-SENSOR2(MP(l),7))*2.8;
                m(k,6)=SENSOR2(MP(l),Val);
            else
                %plot(255-SENSOR2(MP(l),7),SENSOR2(MP(l),Val)+255,'x','MarkerSize', 12);
                m(k,5)=(255-SENSOR2(MP(l),7))*2.8;
                m(k,6)=SENSOR2(MP(l),Val)+255;
            end
        end
        k=k+1;
    end
    l=l+1;
  end
  Val=Val+1;
 end
  plot(m(:,1),m(:,2),'--*','MarkerSize', 12);
  plot(m(:,3),m(:,4),'--+','MarkerSize', 12);
  plot(m(:,5),m(:,6),'--x','MarkerSize', 12);
  plot(0,0);
 hold off
end

function plotgainLinear(SENSOR2,MP,iT)
iT=255-iT;
figure()
hold on
Val = 2;
 while Val < 5
 l=1;
 k=2;
  while l<=length(MP)-1
    if SENSOR2(MP(l),7)== iT
        if Val == 2
            %plot(gainhB(SENSOR2(MP(l),8)),SENSOR2(MP(l),Val),'*','MarkerSize', 12);
            m(k,1)=gainhB(SENSOR2(MP(l),8));
            m(k,2)=SENSOR2(MP(l),Val);
        end
        if Val == 3
            %plot(gainhB(SENSOR2(MP(l),8)),SENSOR2(MP(l),Val),'+','MarkerSize', 12);
            m(k,3)=gainhB(SENSOR2(MP(l),8));
            m(k,4)=SENSOR2(MP(l),Val);
        end
        if Val == 4
            %plot(gainhB(SENSOR2(MP(l),8)),SENSOR2(MP(l),Val),'x','MarkerSize', 12);
            m(k,5)=gainhB(SENSOR2(MP(l),8));
            m(k,6)=SENSOR2(MP(l),Val);
        end
    k=k+1;
    end

    l=l+1;
  end
  Val=Val+1;
 end
  plot(m(:,1),m(:,2),'--*','MarkerSize', 12);
  plot(m(:,3),m(:,4),'--+','MarkerSize', 12);
  plot(m(:,5),m(:,6),'--x','MarkerSize', 12);
  plot(0,0);
 hold off
end
function plotgainNormed(SENSOR2,MP,iT)
iT=255-iT;
figure()
hold on
Val = 2;
 while Val < 5
 l=1;
 k=1;
  while l<=length(MP)
```

```
296     if SENSOR2(MP(l),7)== iT
297         if Val == 2
298             %plot(gainhB(SENSOR2(MP(l),8)),SENSOR2(MP(l),Val)/gainhB(SENSOR2(MP(l),8))
                    ,'*','MarkerSize', 12);
299             m(k,1)=gainhB(SENSOR2(MP(l),8));
300             m(k,2)=SENSOR2(MP(l),Val)/gainhB(SENSOR2(MP(l),8));
301         end
302         if Val == 3
303             %plot(gainhB(SENSOR2(MP(l),8)),SENSOR2(MP(l),Val)/gainhB(SENSOR2(MP(l),8))
                    ,'+','MarkerSize', 12);
304             m(k,3)=gainhB(SENSOR2(MP(l),8));
305             m(k,4)=SENSOR2(MP(l),Val)/gainhB(SENSOR2(MP(l),8));
306         end
307         if Val == 4
308             %plot(gainhB(SENSOR2(MP(l),8)),SENSOR2(MP(l),Val)/gainhB(SENSOR2(MP(l),8)),'x
                    ','MarkerSize', 12);
309             m(k,5)=gainhB(SENSOR2(MP(l),8));
310             m(k,6)=SENSOR2(MP(l),Val)/gainhB(SENSOR2(MP(l),8));
311         end
312         k=k+1;
313     end
314
315     l=l+1;
316   end
317   Val=Val+1;
318  end
319   plot(m(:,1),m(:,2),'*','MarkerSize', 12);
320   plot(m(:,3),m(:,4),'+','MarkerSize', 12);
321   plot(m(:,5),m(:,6),'x','MarkerSize', 12);
322   plot(0,0);
323  hold off
324 end
325 function plotgaincorr(SENSOR2,MP,iT)
326
327 iT=255-iT;
328 figure()
329 hold on
330 Val = 2;
331  while Val < 5
332  l=1;
333
334   while l<=length(MP)
335     if SENSOR2(MP(l),7)== iT
336         if Val == 2
337                 plot(gainhB(SENSOR2(MP(l),8)),SENSOR2(MP(l),Val)/gainhB(SENSOR2(MP(l),8))
                        ,'*','MarkerSize', 12);
338         end
339         if Val == 3
340             plot(gainhB(SENSOR2(MP(l),8)),SENSOR2(MP(l),Val)/gainhB(SENSOR2(MP(l),8)),'+'
                    ,'MarkerSize', 12);
341         end
342         if Val == 4
343             plot(gainhB(SENSOR2(MP(l),8)),SENSOR2(MP(l),Val)/gainhB(SENSOR2(MP(l),8)),'x'
                    ,'MarkerSize', 12);
344         end
345     end
346
347     l=l+1 ;
348   end
349   Val=Val+1;
350  end
351   plot(0,0);
352  hold off
353 end
354
355
356 function reture = gainhB(g) %gibt gain laut HandBuch zurueck
357 if g == 0
358     reture=1;
359 elseif g==1
360     reture=4.176;
361 elseif g==2
362     reture=15.585;
363 else
```

```
364     reture=65.9292;
365 end
366 end
```

Literatur

Bear/Barfuss/Seifert. *Beleuchtungstechnik - Grundlagen*. 4. Auflage. Berlin: HUSS-Medien GMBH, 2016. ISBN: 978-3-341-01634-3.

Judd/MacAdam/Wyszecki. *Spectral Distribution of Typical Daylight as a Function of Correlated Color Temperature*. Aug. 1964.