

Elmar Wings

AI with an Arduino Nano 33 BLE Sense Realtime Object Detection with the ArduCAM

Version: 1765
May 20, 2025

Contents

Contents	3
List of Figures	23
List of Listings	29
1. Introduction	35
1.1. Introduction	35
1.2. Challenges and Solutions	35
1.3. Structure	35
I. Arduino IDE	37
2. Arduino IDE 2.3.x	39
2.1. Arduino IDE Description	39
2.2. Installation of the Arduino IDE	40
2.2.1. Download of the Arduino IDE	40
2.2.2. Summary of Options	40
2.2.3. Installation on Windows	41
2.2.4. Installation (MacOS)	43
2.3. Overview	45
2.4. Features	46
2.4.1. Sketchbook	46
2.4.2. Boards Manager	47
2.4.3. Library Manager	47
2.5. Integrating and Using a Custom Library in Arduino IDE	48
2.5.1. Creating the Library Files	48
2.5.2. Installation and Integration of the Library	48
2.5.3. Using Symbolic Links for Library Integration	48
2.5.4. Windows Tool <code>mklink</code> to Create Symbolic Links	48
2.5.5. MacOS Command <code>ln -s</code> to Create Symbolic Links	50
2.5.6. Verifying Library Availability in the Arduino IDE	50
2.5.7. Serial Monitor	51
2.5.8. Serial Plotter	51
2.6. Examples	51
2.7. Debugging	52
2.8. Autocompletion	52
2.9. Conclusion	53
3. To-Do	55
4. Setup for the Arduino Nano 33 BLE Sense	57
4.1. Introduction	57
4.2. Configuration for the Arduino Nano 33 BLE Sense	57
4.2.1. Installation of the driver	57

4.2.2. Connection and Configuration of the Arduino Board to the Computer	58
4.2.3. Select the Appropriate Port	58
4.2.4. Upload and Verify a Sketch	60
4.3. Test the Configuration	61
4.3.1. Testing the Steps by an Example Sketch <code>blink.ino</code>	61
4.3.2. Description of Basic Sketch for Printing 'Hello'	62
5. Comments with doxygen	65
5.1. Installation	65
5.1.1. doxygen	65
5.1.2. Graphviz	67
5.2. Configuration of Graphviz and doxygen with doxywizard	70
5.2.1. Graphviz	70
5.2.2. DoxyWizard	70
5.2.3. Saving the Configuration	76
5.3. Run of doxygen	76
5.3.1. Run of doxygen using DoxyWizard	76
5.3.2. Run of doxygen using a Command Shell	76
5.4. Syntax and Keywords	77
5.4.1. Keywords for Files	78
5.4.2. Keywords for Functions	79
5.4.3. Keywords for Definitions	79
5.5. Use of Documentation	79
5.6. Add Ons	79
5.6.1. Mainpage	79
5.6.2. Use of simple HTML Commands	80
5.6.3. Inserting Enumerations	80
5.6.4. Inserting Images	81
5.6.5. Inserting HTML pages	81
II. Arduino Nano 33 BLE Sense - Onboard Sensoren	83
6. Arduino Nano 33 BLE Sense	85
6.1. Arduino Nano 33 BLE Sense	85
6.2. Unterschied zwischen dem Arduino Nano 33 BLE Sense Rev1, dem Arduino Nano 33 BLE Sense Rev2 und dem Arduino Nano 33 BLE Sense Lite	86
6.2.1. What is the difference between Rev1 and Rev2?	86
6.2.2. Changes in the Sketch	87
6.2.3. Arduino Nano 33 BLE Sense Lite	88
6.3. On-Board Sensor Description	88
6.3.1. Gesture, Proximity, and Color Detection Sensor ADPS-9960	90
6.3.2. Accelerometer, Gyroscope, and Magnetometre Sensor LSM9DS1	91
6.3.3. Pressure Sensor LPS22HB	92
6.3.4. Relative Humidity and Temperature Sensor HTS221	93
6.3.5. Digital Microphone MP34DT05-A	93
6.3.6. Bluetooth Module nRF52840	95
6.4. Bluetooth Module INA B306	96
6.5. Arduino Nano 33 BLE Pin Configuration	97
6.6. Was fehlt	97

7. Reset Button on the Arduino Nano 33 BLE Sense	99
7.1. Purpose and Functionality	99
7.1.1. Functions of the Reset Button	99
7.2. Timing and Sequence	100
7.3. Use Cases	100
7.4. Conclusion	100
8. Built-inLED	101
8.1. General Information	101
8.2. Built-in LED	101
8.3. Specification	101
8.3.1. Pin Assignment	101
8.3.2. Power Comsumption	102
8.4. Simple Code	102
8.4.1. Simple Code for the Built-in LED	102
8.4.2. Simple Code for Testing the Built-in LED	103
8.5. Tests	103
8.5.1. Simple Function Test	103
8.5.2. Test all Functions	104
8.6. Simple Application	104
8.7. Further Readings	104
9. Power LED	105
9.1. General Information	105
9.2. Power LED	105
9.3. Specification	106
9.3.1. Pin Assignment	106
9.3.2. Power Comsumption	106
9.4. Simple Code	106
9.4.1. Simple Code for LEDs	106
9.4.2. Simple Code for the Power LED	108
9.4.3. Simple Code for Testing the Power LED	108
9.4.4. Test all Functions	109
9.5. Simple Application	109
9.6. Further Readings	112
10. Built-in RGB-LED	113
10.1. General Information	113
10.2. Specific Sensor	113
10.3. Specification	114
10.3.1. Pin Assignment	114
10.3.2. Power Comsumption	114
10.4. Simple Code	114
10.5. Tests	115
10.5.1. Simple Function Test	115
10.5.2. Test all Functions	116
10.6. Simple Application	120
10.7. Further Readings	121
11. TinyML Shield: Built-in Push Button	123
11.1. General	123
11.2. Built-in Push Button	123
11.3. Specification	124
11.4. Simple Code	124
11.5. Tests	124

11.6. Interrupt Function on the Arduino Nano 33 BLE Sense	126
11.7. Simple Application	126
11.8. Further Readings	128
12. Pressure and Temperature Sensor LPS22HB	129
12.1. General	130
12.2. Specific Sensor	130
12.3. Specification	131
12.4. Library	132
12.4.1. Description	132
12.4.2. Installation	132
12.4.3. Functions	133
12.4.4. Example - Manual	134
12.4.5. Example	134
12.4.6. Example - Code	134
12.4.7. Example - Files	134
12.5. Calibration	134
12.6. Simple Code	136
12.7. Sleep Mode	136
12.8. Simple Application	138
12.9. Tests	138
12.9.1. Simple Function Test	138
12.9.2. Test all Functions	138
12.10. Simple Application	138
12.11. Further Readings	138
13. Sensormodul APDS-9960 for Gesture, Proximity, and Color Detection	139
13.1. Functionality of the APDS-9960	139
13.2. Key Technical Specifications	139
13.3. Library Arduino_APDS9960	141
13.3.1. Installation of APDS-9960 Library	141
13.3.2. Functions	141
13.4. Simple Function Tests	143
13.4.1. Example Color Detection	143
13.4.2. Example Color Detection - Manual	144
13.4.3. Setting Up the Color Sensor Measurement Program	144
13.4.4. Example	144
13.4.5. Example Color Detection - File	144
13.4.6. Proximity	145
13.4.7. Proximity - Manual	145
13.4.8. Example Proximity - Code	145
13.4.9. Example Proximity - File	145
13.4.10. Gesture Detection	146
13.4.11. Example Gesture Detection	146
13.4.12. Example Gesture Detection - Manual	146
13.4.13. Example Gesture Detection - Code	146
13.4.14. Example Gesture Detection - File	147
13.4.15. Troubleshoot	148
13.5. Calibration Color Detection	149
13.6. Calibration Color Detection	149
13.6.1. Calibration Setup	149
13.6.2. Step 1: Measuring Reference Values	149
13.6.3. Step 2: Normalizing the Sensor Values	149
13.6.4. Step 3: Implementing the Calibration in Code	150
13.6.5. Calibration File	150

13.6.6. Step 4: Validating the Calibration	154
13.7. Tests	154
13.7.1. Simple Function Test	154
13.7.2. Test all Functions	154
13.8. Simple Application	154
13.9. Further Readings	158
14. Sensor APDS 9960 Gesture	159
14.0.1. Sensormodul APDS 9960 for Gesture	159
14.1. General	159
14.2. Specific Sensor	159
14.3. Specification	159
14.4. Bibliothek	160
14.4.1. Description	160
14.4.2. Installation	160
14.4.3. Functions	160
14.4.4. Example - Manual	160
14.4.5. Example	160
14.4.6. Example - Code	160
14.4.7. Example - Files	160
14.5. Calibration	160
14.6. Simple Code	160
14.7. Simple Application	160
14.8. Tests	160
14.8.1. Simple Function Test	160
14.8.2. Test all Functions	160
14.9. Simple Application	160
14.10. Further Readings	160
15. Sensor APDS 9960 Color	161
15.1. General	161
15.2. Specific Sensor	163
15.3. Specification	163
15.4. Bibliothek	163
15.4.1. Description	163
15.4.2. Installation	163
15.4.3. Functions	163
15.4.4. Example - Manual	163
15.4.5. Example	163
15.4.6. Example - Code	163
15.4.7. Example - Files	163
15.5. Calibration	163
15.6. Simple Code	164
15.7. Simple Application	164
15.8. Tests	164
15.8.1. Simple Function Test	164
15.8.2. Test all Functions	164
15.9. Simple Application	164
15.10. Further Readings	164
16. Sensor APDS 9960 Proximity	165
16.1. General	165
16.2. Specific Sensor	165
16.3. Specification	165

16.4. Bibliothek	165
16.4.1. Description	165
16.4.2. Installation	165
16.4.3. Functions	165
16.4.4. Example - Manual	165
16.4.5. Example	165
16.4.6. Example - Code	165
16.4.7. Example - Files	165
16.5. Calibration	165
16.6. Simple Code	166
16.7. Simple Application	166
16.8. Tests	166
16.8.1. Simple Function Test	166
16.8.2. Test all Functions	166
16.9. Simple Application	166
16.10 Further Readings	166
17. Sensor Ambient Light	167
17.1. General	167
17.2. Specific Sensor	167
17.3. Specification	167
17.4. Bibliothek	167
17.4.1. Description	167
17.4.2. Installation	167
17.4.3. Functions	167
17.4.4. Example - Manual	167
17.4.5. Example	167
17.4.6. Example - Code	167
17.4.7. Example - Files	167
17.5. Calibration	167
17.6. Simple Code	168
17.7. Simple Application	168
17.8. Tests	168
17.8.1. Simple Function Test	168
17.8.2. Test all Functions	168
17.9. Simple Application	168
17.10 Further Readings	168
18. Application of the Color Sensor with OLED Display	169
18.1. Manual	169
18.2. Code Explanation	170
18.3. Application Test	173
18.4. Improvement Suggestions	173
19. Mikrophon MP34DT05-A	175
19.0.1. Ton und Frequenzen	175
19.0.2. Unterschied Ton, Klang und Geräusch	176
19.0.3. Mikrofone	176
19.0.4. Wavedatei	178
19.0.5. Package <code>PyAudio</code> Version 0.2.14	181
19.0.6. Package <code>Wave</code> Version 3.12	181
19.0.7. Package <code>NumPy</code> Version 1.26	182
19.0.8. Package <code>Guizero</code> Version 1.5.0	183
19.0.9. Package <code>Threading</code> (Version 3.7)	184
19.0.10. Package <code>Librosa</code> Version 0.10.2	185

19.1. Funktionen	185
19.1.1. Laden	185
19.1.2. Abspielen	186
19.1.3. Speichern	186
19.1.4. Anzeigen von zwei Audio-Dateien	188
19.1.5. Glättung von Audio-Dateien	189
19.1.6. Ausschnitt	190
19.1.7. Zoomen in einer Matplotlib	191
19.1.8. Änderung der Tonhöhe und Geschwindigkeit	193
19.1.9. Frequenzanalyse	196
19.1.10. Overlay	198
19.2. General	199
19.2.1. Decibels	199
19.2.2. Puls-Dichtemodulation	200
19.3. Specific Sensor	200
19.4. Specification	200
19.5. Bibliothek <code>pdm.h</code>	201
19.5.1. Description	201
19.5.2. Installation	201
19.5.3. Functions	201
19.5.4. Example - Manual	202
19.5.5. Example	202
19.5.6. Example - Code	202
19.5.7. Example - Files	203
19.6. Calibration	203
19.7. Simple Code	203
19.8. Simple Application	203
19.8.1. Pin-Konfiguration	203
19.8.2. Initialisierung und Setup	203
19.8.3. Messungssteuerung	203
19.8.4. Mikrofondatenverarbeitung	205
19.8.5. Anzeige der Ergebnisse	205
19.8.6. Umrechnung auf den Wert dB SPL	205
19.8.7. Benutzeroberfläche	206
19.9. Tests	206
19.9.1. Simple Function Test	207
19.9.2. Test all Functions	207
19.10. Specific Sensor	207
19.11. Specification	207
19.12. PDM Library	207
19.13. Calibration	208
19.14. Simple Code	208
19.15. Simple Application	208
19.16. Tests	209
19.17. Further Readings	209
20. Inertial Measurement Unit	211
20.1. Introduction	211
20.2. 6-Axis IMU LSM6DSOX	212
20.3. IMU LSM6DSOX Features	212
20.4. IMU LSM6DSOX Data	214
20.4.1. Accelerometer:	214
20.4.2. Gyroscope	215
20.5. Library setup in Arduino IDE	215
20.6. Applications	216

20.7. LSM9DS1(IMU)	216
20.8. Features	216
20.9. Pin description LSM9DS1	217
20.10Pin connections LSM9DS1	217
20.10.1Module specifications	219
20.10.2Block Diagram	220
20.10.3System Requirements	222
20.10.4Precautions to be taken	222
20.11Bibliotheken	223
20.12Beispiele auf dem Mikrocontroller	223
20.12.1.Testen des Sensors LSM9DS1	223
20.13Programmierung	223
20.13.1.Programmablaufplan	223
20.13.2.Programmcode und Dokumentation	223
20.13.3.Definition Echtzeit	229
20.14Kalibrierung	229
20.15Probleme	230
21. Calibration	231
21.1. Introduction	231
21.2. Standard Operating Procedure	231
21.2.1. Low and high limit method	234
21.2.2. FreeIMU Calibration Application Magnetometer	234
21.2.3. Example	234
22. Errors	237
22.1. Introduction	237
22.2. Affected Parameters	237
22.2.1. Accelerometer:	238
22.2.2. Gyroscope:	238
23. IMU LSM6DSOX - Libraries and Functions	239
23.1. Libraries	239
23.1.1. Wire.h	239
23.1.2. Kalman.h	239
23.1.3. Arduino_LSM6DSOX.h	239
23.1.4. LSM6DSOXSensor.h	239
23.2. Functions	240
23.2.1. setup()	240
23.2.2. loop()	240
23.2.3. IMU.begin()	240
23.2.4. IMU.setAccelerometerRange()	240
23.2.5. IMU.setGyroscopeRange()	240
23.2.6. IMU.setAccelerometerDataRate()	241
23.2.7. IMU.setGyroscopeDataRate()	241
23.2.8. IMU.accelerationSampleRate()	241
23.2.9. IMU.gyroscopeSampleRate()	241
23.2.10. IMU.temperatureAvailable()	241
23.2.11. IMU.readTemperature()	241
23.2.12. IMU.accelerationAvailable()	241
23.2.13. IMU.readAcceleration()	241
23.2.14. IMU.gyroscopeAvailable()	242
23.2.15. IMU.readGyroscope()	242
23.2.16. randomGaussian()	242
23.2.17. kalmanX.setQ(), kalmanY.setQ(), kalmanZ.setQ()	242

23.2.18.kalmanX.setR(), kalmanY.setR(), kalmanZ.setR()	242
23.2.19.kalmanX.update(), kalmanY.update(), kalmanZ.update()	242
23.2.20.kalmanX.getSigma(), kalmanY.getSigma(), kalmanZ.getSigma()	243
23.2.21.delay()	243
23.2.22.lsm6dsoxSensor.Set_X_FS()	243
23.2.23.lsm6dsoxSensor.Set_G_FS()	243
23.2.24.Initialize_IMU()	244
23.2.25.Factors_Calculation()	244
23.2.26.Factors_Calculation()	244
24. Sensor Inertial Mesure Unit	245
24.1. General Description	245
24.1.1. Always On Mode	245
24.1.2. Tilt detection	245
24.1.3. Significant Motion Detection	246
24.2. Specific Sensor	246
24.2.1. LSM9DS1	246
24.2.2. Accelerometer	246
24.2.3. Gyroscope	247
24.3. Specification	248
24.3.1. Accelerometer Sensor	248
24.3.2. PIN Connction	248
24.4. Library	249
24.4.1. Library Description	249
24.4.2. Installation	250
24.5. Function	250
24.5.1. Library <code>Wire.h</code>	251
24.5.2. Function <code>IMU.begin()</code>	252
24.5.3. Function <code>IMU.end()</code>	252
24.5.4. Function <code>IMU.readAcceleration(x,y,z)</code>	252
24.5.5. Function <code>IMU.readGyroscope()</code>	253
24.5.6. Function <code>IMU.accelerationAvailable()</code>	253
24.5.7. Function <code>IMU.gyroscopeAvailable()</code>	253
24.5.8. Function <code>IMU.accelerationSampleRate()</code>	254
24.5.9. Function <code>IMU.gyroscopeSampleRate()</code>	254
25. Distance, Speed and Acceleration Detection Algorithms	255
25.1. Review of Distance Measurement Methods	255
25.2. Review of Speed and Acceleration Detection Algorithms	256
III. Arduino Nano 33 BLE Sense - External Sensors and Actors	259
26. Tiny Machine Learning Kit	261
26.1. Das Arduino Tiny Machine Learning Shield	261
26.2. Die OV7675 Kamera	262
26.3. USB-Micro-B- auf USB-A-Kabel	263
27. Powering the TinyML Shield	265
27.0.1. USB Power Delivery	265
27.1. Battery	265
27.2. Checking the Battery Voltage	266
27.3. Batterieclip	267
27.4. Spannungssensor	267
27.4.1. Durchführung	268

27.4.2. Ergebnisse	270
28. TinyML Shield: Built-in Push Button	271
28.1. General	271
28.2. Built-in Push Button	271
28.3. Specification	272
28.4. Simple Code	272
28.5. Tests	272
28.6. Interrupt Function on the Arduino Nano 33 BLE Sense	274
28.7. Simple Application	274
28.8. Further Readings	276
29. Connectors of Type JST	277
29.1. Connector Series	277
29.2. JST VH	277
29.2.1. Product Profile	277
29.2.2. Specification	277
29.3. JST PH	278
29.3.1. Product Profile	278
29.3.2. Specification	278
29.3.3. JST PHR-2	279
30. Grove	281
30.1. Grove-Universal-Kabel	282
30.2. Verbindung Grove-Schnittstelle zu 4-Pin-Lösungen	282
30.3. Pinbelegung	282
31. Zwei Taster mit Grove-Anschluss von M5 Stack	285
31.0.1. Pin-Konfiguration	285
31.0.2. Initialisierung und Setup	285
31.1. 2.Beschreibung	285
32. External Standard LED	289
32.1. General	289
32.2. Specification	290
32.3. Using Standard External LED	291
32.3.1. Connecting a LED to the Arduino Board	291
32.3.2. Pins	291
32.4. Bibliothek	291
32.5. Simple Code	292
32.6. Tests	292
32.6.1. Simple Function Test	292
32.6.2. Test all Functions	293
32.7. Simple Application	294
32.8. Further Readings	296
33. External RGB-LED	297
33.1. Standard RGB-LED	297
33.2. Specific Sensor	298
33.2.1. Pins	298
33.3. Specification	298
33.4. Calibration	299
33.5. Simple Code	299
33.6. Simple Application	299

33.7. Tests	299
33.7.1. Simple Function Test	299
33.7.2. Test all Functions	299
33.7.3. Brightness of the RGB-LED	299
33.8. Simple Application	301
33.9. Further Readings	301
34. Sensor BME280 für Temperatur, Luftfeuchtigkeit und den Luftdruck	303
34.1. Beschreibung der Hardware	303
34.2. Schaltplan des Aufbaus	304
34.2.1. Bibliothek <code>cactus_io_BME280</code> für den Sensor BME280	304
34.2.2. Testen des OLED-Displays	304
35. Servomotor JAMARA 033212	307
35.1. Datenblatt JAMARA 033212	308
35.2. Schaltplan	308
36. OLED-Display	311
36.1. OLED	311
36.2. Anschluss	311
36.3. Programmierung	311
36.3.1. <code>Wire.h</code>	311
36.3.2. OLED-Display	313
36.3.3. Testen des OLED-Displays	313
36.4. Software	313
36.4.1. Verwendete Bibliotheken	313
36.4.2. OLED-Display	314
36.4.3. Initialisierung und Setup	314
36.4.4. Messungssteuerung	315
36.4.5. Mikrofondatenverarbeitung	316
36.4.6. Anzeige der Ergebnisse	316
36.4.7. Umrechnung auf den Wert dB SPL	316
36.4.8. Benutzeroberfläche	316
36.5. Das OLED-Display SSD1306	316
36.6. Schaltplan des Aufbaus	317
36.7. Genutzte Bibliotheken	317
36.7.1. Bibliothek <code>Wire.h</code>	318
36.7.2. Bibliothek <code>SSD1306Ascii.h</code> für das Testprogramm	318
36.7.3. Testen des OLED-Displays	318
36.8. Beispiel eines OLED-Displays	320
36.8.1. OLED	321
37. Kamera-Modul ArduCAM OV2640	325
37.1. OV7675 Camera Module	325
37.2. Indroduction	325
37.3. Produktbeschreibung	326
37.3.1. Pin Configuration of Arducam OV2640 2MP Mini	327
37.4. ArduCAM Interface with Arduino	328
37.5. ArduCAM Library Introduction	328
37.6. Libraries Structure	329
37.7. How to use	330
37.7.1. Edit <code>memoriesaver.h</code> file	330
37.7.2. Choose correct CS pin for your camera	330
37.7.3. Upload the examples	330
37.7.4. How To Connect Bluetooth Module	330

38. Sensor ov7675	333
38.1. Code	333
38.1.1. OV7670 Camera and Image Sensor with Nano 33 BLE	336
38.1.2. Connection	338
38.1.3. Arduino OV7670 library	340
38.1.4. Testing the OV7670	342
38.1.5. Final Step	343
38.2. Kameramodul OV7675	343
38.3. Kamera Modul OV7675	343
38.4. header	344
38.5. <code>void setup()</code>	344
38.6. <code>void loop()</code>	345
38.7. Test der Bildaufnahme	345
38.8. Bildverarbeitung	345
38.9. Produktbeschreibung	349
38.10. ArduCAM Library Introduction	350
38.11. Libraries Structure	351
38.12. How to use	352
38.12.11. Edit <code>memoriesaver.h</code> file	352
38.12.22. Choose correct CS pin for your camera	352
38.12.33. Upload the examples	352
38.12.44. How To Connect Bluetooth Module	352
39. Lens Calibration Tool	355
39.1. Circle Grid Camera Calibration Patterns	355
39.2. Siemens Star	355
39.3. Enjoyyourcamera Testkarte für Kameras und Objektive 30 × 45 cm	355
39.3.1. Testkarte zum Prüfen von Kameras und Objektiven in Bezug auf ihre Qualität und mögliche Defekte	356
39.3.2. Zahlreiche Testkriterien: z.B. Schärfe in der Mitte und an den Bildrändern, Randabschattungen u.v.m.	356
39.3.3. Mehrere Testkarten zu einer großen Testkarte zusammenfügbar - ideal für Weitwinkelobjektive	356
39.3.4. Eine Testkarte für Normal- und Teleobjektive, vier Testkarten für Weitwinkelobjektive usw.	357
39.3.5. Festes Papier, hohe Druckqualität, extrem feine Detailelemente für besonders genaues Testen	357
39.3.6. Handhabung	357
39.4. B.I.G. RES7 Testtafel für Kameras und Objektive	358
39.4.1. BIG RES7 Test Board für Kameras und Objektive	358
39.4.2. Merkmale des BIG RES7 Test Boards für Kameras und Objektive	358
39.5. Pattern Generator	359
39.6. Chess Board	359
39.7. To Do	359
39.8. Übersicht	362
39.9. Applications	362
39.10. Package Contents	362
39.10.1. Licht, Farben und Farbmodelle	362
39.10.2. Texterkennung	365
39.10.3. Kamerakalibrierung mit Schachbrett	368
39.10.4. Kamerakalibrierung mit Farbpaletten	370
39.10.5. Bibliotheken in der Programmierung	371
40. Grundlagen der Kamerakalibrierung	381
40.1. Funktionsweise einer Kamera	381

40.2. Mathematische Grundlagen: Lösung des nichtlinearen Kleinste-Quadrat-Problems	384
41. Farbkalibrierung und Farberkennung	385
41.1. Farbkalibrierung durch Farbkarten	385
41.2. Berechnung des Weißabgleichs	386
41.3. Berechnung der Farbkorrektur-Matrix	387
42. Graustufenkalibrierung und Graustufenerkennung	389
42.1. Graustufenkalibrierung durch Graukarten	389
43. Bestimmung des Fokus	391
44. Kalibrierung mit Kalibrierkörper	393
44.1. Extrinsische Parameter	394
44.2. Intrinsische Parameter	395
44.3. Stetiges Kameramodell der zentralen Projektion	395
44.4. Diskretes Kameramodell der zentralen Projektion	397
44.5. Berechnung der Kalibrierung	399
45. BlueTooth	403
45.1. Quick Start	403
45.2. Supplies	403
45.3. Step 1: Introduction	403
45.4. How pfodApp is optimised for short BLE style messages	405
45.5. Step 2: Creating the Custom Android Menus and Generating the Code	405
45.6. BLE Mobile App “Nordic Semiconductors nRF Connect”	406
45.7. Arduino BLE Example – Explained Step by Step	408
45.7.1. Arduino BLE Example Code Explained	408
45.7.2. Arduino BLE – Bluetooth Low Energy Introduction	408
45.7.3. Arduino BLE Example 1 – Battery Level Indicator	408
45.7.4. Arduino BLE Tutorial Battery Level Indicator Code	410
45.7.5. Arduino Bluetooth Battery Level Indicator Code Explained	410
45.7.6. Installing the App for Android	411
45.7.7. Example 2 – Arduino BLE Accelerometer Tutorial	411
45.8. Arduino Library BLEAK	417
45.9. Test	417
45.10. Test with Bluetooth Module Connection	417
45.11. BLEStack	418
45.12. Control Arduino Nano BLE with Bluetooth & Python	419
45.13. Peripheral Device aka Arduino Nano	419
45.13.1. Taking the Project Further	422
46. Ethernetschnittstelle ENC28J60	423
46.1. Netzadapter	423
46.2. Test des Ethernet-Shield ENC28J60	424
46.3. Test des Web Servers	424
46.4. Datenübertragung	424
47. Heart Rate Sensor	429
47.1. The Heartbeat	429
47.1.1. Introduction	429
47.1.2. Cardiac Cycle	429
47.1.3. Electrical Activity	429
47.1.4. Heart Rate	429
47.1.5. Regulation of the Heartbeat	430

47.1.6. Heartbeat Deviations	430
47.2. Operation of the Heartbeat Sensor	430
47.2.1. The Photodetector	431
47.2.2. Voltage Data Processing	431
47.2.3. Drift and Measurement Inaccuracies	433
47.3. Heart Rate Sensor with Grove Interface and Ear Clip	433
47.3.1. Key Technical Specifications	434
47.4. Simple Function Test	434
47.4.1. Heart Rate Detection - Manual	434
47.4.2. Heart Rate Detection - Code	435
47.4.3. Heart Rate Detection - File	435
47.5. Kalibrierung	436
47.5.1. Calibration	436
47.6. Further Readings	436
48. Application of Heart Rate Monitor with OLED Display	437
48.1. Overview	437
48.2. Functionality	437
48.3. Manual	437
48.3.1. Setup of the Device	437
48.3.2. Programming the Device	438
48.3.3. Using the Heart Rate Monitor	442
48.3.4. Mobile Operation	442
48.4. Code Explanation	442
48.5. Application Test	448
48.6. Improvement Suggestions	448
49. Ultraschallsensor HC-SR04	449
49.1. Domänenwissen	449
49.2. Grundlagen zur Verwendung eines Ultraschallsensors	449
49.2.1. Grundlagen Schall	449
49.2.2. Schallgeschwindigkeit und Wellenlänge	449
49.2.3. Absorption und Reflexion	450
49.2.4. Weg- und Abstandsmessung mit Ultraschall	451
49.2.5. Anwendungen	452
49.2.6. Herausforderungen	452
49.2.7. Lösungsansätze	453
49.3. Ultraschall Abstandssensor	453
49.4. Specification	453
49.5. Bibliothek	454
49.5.1. Description	454
49.5.2. Installation	454
49.5.3. Functions	454
49.5.4. Example - Manual	454
49.5.5. Example	454
49.5.6. Example - Code	454
49.5.7. Example - Files	454
49.6. Calibration	454
49.7. Kalibrierung	454
49.8. Simple Code	454
49.9. Einfacher Beispiel zur Verwendung des Ultraschallabstandssensors	454
49.9.1. Durchführung	454
49.9.2. Ergebnisse	456
49.10 Simple Application	456

49.11Tests	456
49.11.1Simple Function Test	456
49.11.2Test all Functions	456
49.12Simple Application	456
49.13Further Readings	456
50. Hypatia - Li-Po batteries, Pins and board LEDs	457
50.1. Battery capacity	457
50.2. Battery connector	457
50.3. 5V	457
50.4. VCC	457
50.5. LED ON	457
50.6. CHARGE LED	458
50.7. Onboard LED	458
50.8. Pin Assignment	458
50.9. Checking the Battery Voltage with Hypatia	458
50.10Battery Example	458
51. Battery	461
51.1. Appendix - Powering Your TinyML System	461
51.1.1. USB Power Delivery	461
51.2. Battery	461
51.3. Checking the Battery Voltage	462
51.4. Batterieclip	463
51.5. Spannungssensor	463
51.6. Spannungssensor	464
51.6.1. Durchführung	464
51.6.2. Ergebnisse	464
51.7. Lithium Battery	465
51.7.1. Attention	465
51.7.2. Battery	466
51.7.3. Jumper	466
52. Powerbank	471
53. Entwicklerboard - Motorsteuerung DEBO MotoDriver2 L298N	473
54. Terminal Adapter Board mit Schraubklemmen kompatibel mit Nano V3 und Arduino	475
55. Getriebemotor mit Rad	477
55.1. Funktion Motoren	477
56. Drehwinkel-Encoder	481
56.1. Encoder	481
56.2. Drehwinkel-Encoder	481
57. Arduino - SD-Karten-Modul	483
57.1. Eigenschaften	483
57.2. Technische Spezifikationen	484
57.3. Bibliothek <code>SD.h</code>	484
57.4. Testen des SD-Karten Moduls	484
58. Sensor DEBO DHT22	487
58.1. Luftdruck	487
58.1.1. Resistive Druckmessung	487

58.1.2. Piezoresistive Druckmessung	488
58.2. Temperatur	489
58.3. Feuchtigkeit	490
58.3.1. Kapazitive Feuchtigkeitssensoren	490
58.3.2. Resistive Feuchtigkeitssensoren	490
58.3.3. Wärmeleitfähigkeits Feuchtigkeitssensor	490
58.3.4. Psychrometrische Feuchtigkeitssensoren	490
58.4. Sensor DHT22	490
58.5. Schaltplan	491
58.6. Testen des Sensors DHT22	491
58.7. Kalibrierung	492
58.7.1. Opus20 THI	492
58.7.2. Funktionen	493
58.7.3. Schnittstellen	493
58.7.4. Technische Daten	493
58.7.5. Anwendung	494
58.7.6. Kalibrierung	494
59. Geschwindigkeitssensor LM393	497
59.1. Allgemeine Beschreibung des Sensors	497
59.2. Spezifische Beschreibung des Sensors	497
59.2.1. Funktionsweise der Lichtschranke	497
59.2.2. Signalverarbeitung mit dem LM393 Komparator	498
59.2.3. Anschluss des Sensors mit dem Arduino Nano 33 BLE Sense .	498
59.3. Spezifikationen	499
59.4. Bibliothek	499
59.4.1. Installation	499
59.4.2. Code-Beispiel	500
59.5. Kalibrierung	500
59.6. Einfaches Beispiel mit Code	501
59.7. Tests	503
59.8. Weiterführende Literatur	503
60. Sensor XY	505
60.1. General	505
60.2. Specific Sensor/Actor	505
60.3. Specification	505
60.4. Library	505
60.4.1. Description	505
60.4.2. Installation	505
60.4.3. Functions	505
60.5. Example	505
60.5.1. Manual	506
60.5.2. Inside the Example	506
60.5.3. Code	506
60.5.4. Files	506
60.6. Calibration	506
60.7. Simple Code	506
60.8. Simple Application	506
60.9. Tests	506
60.9.1. Simple Function Test	506
60.9.2. Test all Functions	506
60.10 Further Readings	506

61. Actor XY	507
61.1. General	507
61.2. Specific Sensor/Actor	507
61.3. Specification	507
61.4. Library	507
61.4.1. Description	507
61.4.2. Installation	507
61.4.3. Functions	507
61.5. Example	507
61.5.1. Manual	508
61.5.2. Inside the Example	508
61.5.3. Code	508
61.5.4. Files	508
61.6. Calibration	508
61.7. Simple Code	508
61.8. Simple Application	508
61.9. Tests	508
61.9.1. Simple Function Test	508
61.9.2. Test all Functions	508
61.10 Further Readings	508
62. Getting Started	509
62.1. Download and Install the NRF Connect App on Your Mobile	509
62.1.1. For Android	509
62.1.2. For iOS	509
62.2. Power ON the Arduino Nano 33 BLE Sense Lite	510
62.2.1. Connect to Arduino Nano 33 BLE Sense Lite through NRFconnect	510
62.2.2. For Android	510
62.2.3. For iOS	510
IV. Applikation <Title>	513
63. Application	515
64. Conclusion XY	517
65. To DoX Y	519
V. Templates	521
66. Sensor	523
66.1. General	523
66.2. Specific Sensor	523
66.3. Specification	523
66.4. Library	523
66.4.1. Description	523
66.4.2. Installation	523
66.4.3. Functions	523
66.4.4. Example's Manual	523
66.4.5. Inside the Example	523
66.4.6. Example's Code	523
66.4.7. Example's Files	523
66.5. Calibration	523

66.6. Simple Code	524
66.7. Simple Application	524
66.8. Tests	524
66.8.1. Simple Function Test	524
66.8.2. Test all Functions	524
66.9. Simple Application	524
66.10 Further Readings	524
67. Package Example	525
67.1. Introduction	525
67.2. Description	525
67.3. Installation	525
67.4. Example - Manual	525
67.5. Example	525
67.6. Example - Code	525
67.7. Example - Files	525
67.8. Further Reading	525
VI. Anhang	527
68. Materialliste	529
VII. Hints - Examples for LaTeX - Read and Use	531
69. Hints	533
69.1. Allgemeine mathematische Beschreibung Bézier-Kurve	534
70. Verrundung mit einem Kreisbogen	537
70.1. Gleichungen	537
70.2. Bewertung	539
70.3. Examples	540
71. Maple files	543
71.1. Geometries with Maple	543
71.2. Geometry elements used	543
71.3. Building the data structure	544
71.4. Structure of a module	544
71.4.1. General structure of a module	546
71.4.2. Saving a module	547
71.4.3. Verwendung eines Moduls	547
71.4.4. Creating a Maple module	547
71.5. Functions of the modules	548
71.5.1. Module MPoint	548
71.5.2. Module MLine	549
71.5.3. Module MArc	549
71.5.4. module MBezier	550
71.5.5. Module MPolygon	550
71.5.6. module MGeoList	550
71.5.7. Module MHermiteProblem	551
71.5.8. Module MHermiteProblemSym	551
71.5.9. Module MConstant	552
71.5.10 Module MGeneralMath	552
71.5.11 Module Biarc	553

Contents	21
71.5.12 Module MBiarc	553
71.6. Programme Flowchart	554
71.6.1. Overall Flow	554
71.6.2. Verrundung der Kurve	554
72. Representation of Python Programs	557
73. Representation of Programs written for Arduino Boards	559
74. First Chapter	561
75. CAGD	563
A. drawings with tikz	565
B. Criteria for a good L^AT_EX project	569
Literaturverzeichnis	571
Stichwortverzeichnis	583
Index	583

List of Figures

2.1. Menu Button	39
2.2. Arduino IDE 2.3.3 Download	40
2.3. Summary of DownloadOptions	41
2.4. Arduino IDE Create Agent Installation	41
2.5. Arduino Setup Installation options	42
2.6. Arduino Setup Installation Folder	42
2.7. Arduino IDE Sketch	42
2.8. Steps for Installation	43
2.9. ArduinoIDE Create Agent Installation	43
2.10. Open the Download folder	44
2.11. Copy to the Applications Folder	44
2.12. Arduino IDE Sketch	45
2.13. Overview	46
2.14. Sketchbook	46
2.15. Board Manager	47
2.16. Library Manager	47
2.17. Command Prompt with Administrator Privileges	49
2.18. Command Prompt	49
2.19. Including the Nano33BLESenseLED Library	50
2.20. Serial Monitor	51
2.21. Examples	52
2.22. Debugging Example	52
2.23. Autocomplete	53
2.24. Menu Bar Option	53
4.1. Arduino Mbed OS Nano Boards Installation	58
4.2. Select the Connected board - here Arduino Nano 33 BLE Sense	58
4.3. Arduino Nano 33 BLE Sense Reset Button	59
4.4. green and orange Builtin-LED	60
4.5. Select Available Port for Uploading Arduino Sketch	60
4.6. Upload the Program in Arduino board	61
4.7. Setting the Port	61
4.8. LED Example Sketch	62
4.9. LED-Built Example	62
5.1. Exmaple output of Graphviz	65
5.2. Start image from the website https://doxygen.nl/	66
5.3. Download area from the website https://doxygen.nl/	66
5.4. Query after the complete installation during the installation process of doxygen	67
5.5. Query after the complete installation during the installation process of doxygen	67
5.6. Start image from the website https://www.graphviz.org/	68
5.7. Download area from the website https://www.graphviz.org/ download/	68
5.8. Query for adding the Graphviz path to the system variable <code>PATH</code>	69
5.9. Query for the path to Graphviz	69

5.10. Query for the start menu of Graphviz	70
5.11. DoxyWizard tab “Wizard” - Project	71
5.12. DoxyWizard tab “Wizard” - Mode	72
5.13. DoxyWizard tab “Wizard” - Diagrams	72
5.14. DoxyWizard tab “Wizard” - Output	73
5.15. DoxyWizard tab “Wizard” - Expert - Build	74
5.16. DoxyWizard tab “Wizard” - Expert - Dot	75
5.17. DoxyWizard tab “Wizard” - Run	76
 6.1. Arduino Nano 33 BLE Sense Rev. 2, see Arduino Store	86
6.2. Arduino Nano 33 BLE Sense Rev. 1	87
6.3. Arduino Nano 33 BLE Sense Lite	88
6.4.	89
6.5. Pin assignment of the Arduino Nano 33 BLE Sense	90
6.6. Circuit diagram microphone	95
6.7. Connection with BLE and smartphone	96
6.8. Simple sketch to seen Arduino battery	96
6.9. Arduino Nano 33 BLE Pin Configuration	98
 7.1. Arduino Nano 33 BLE Sense’s Reset Button	99
 8.1. Arduino Nano 33 BLE Sense’s built-in LED with Pin 13	101
9.1. Arduino Nano 33 BLE Sense’s Power LED with Pin 25	105
10.1. Arduino Nano 33 BLE Sense’s RGB LED with Pin 22, Pin 23, and Pin 24	113
11.1. The Tiny Machine Learning Shield	123
12.1. Arduino Nano 33 BLE Sense’s pressure and temperature sensor LPS22HB	129
13.1. Arduino Nano 33 BLE Sense’s APDS-9960	139
13.2. APDS-9960 Library Instalation	141
13.3. Gesture, Proximity, Color Sensor Output Window	154
 15.1. Schematic of the main function of a color sensor	161
15.2. Functional Block Diagram	162
 18.1. Circuit diagram for connecting the Arduino Nano 33 BLE Sense with a 1.30” IIC OLED display.	169
 19.1. Arduino Nano 33 BLE Sense’s RGB LED with Pin 22, Pin 23, and Pin 24	175
19.2. Liste verschiedener Mikrofonarten mit Funktionsprinzip	177
19.3. Datenblatt des betrachteten Mikrofons	177
19.4. Aufbau eines Kondensator Mikrofons	178
19.5. Funktionsprinzip eines Analog zu Digital Wandlers	178
19.6. Symbole und numerische Werte des ASCII Formats	179
19.7. Aufbau des Wave Dateiformats mit Erklärung [Wil03]	180
19.8. Beispiel Anzeige von zwei Audio-Dateien	189
19.9. Window-Funktion	194
19.10Ausgabe Sample Frequenzanalyse	198
19.11MP34DT05, Digital Microphone	206
19.12MP34DT05, Serial Plotter	207
 20.1. Examples in the library Arduino_LSM9DS1	211
20.2. <i>Axis Acceleration of Accelerometer Sensor</i>	214

20.3. Angular Speed of Gyroscope Sensor	215
20.4. Library setup in Arduino IDE	215
20.5. Pin connections LSM9DS1	217
20.6. Pin description LSM9DS1	218
20.7. Sensor Characteristics	219
20.8. Temperature Sensor Characteristics	219
20.9. Absolute Maximum Temperature	220
20.10 Accelerometer and gyroscope block diagram	220
20.11 Magnetometer block diagram	221
20.12 LSM9DS1 electrical connections	222
20.13 Output des Testprogramms	223
20.14 Der Programmablaufplan	224
20.15 Anzeige des Arduino OLED Displays sobald der Arduino eingeschaltet ist	229
20.16 Ausgabe von Messdaten	229
20.17 Der Reset Button des Arduino	230
24.1. LSM9DS1 axis on this card	247
24.2. LSM9DS1 axis on this card	247
24.3. Explication of pin mode	249
24.4. Wire include	250
24.5. Board card installation	250
24.6. Library choice	250
25.1. Design of OAK-D Pro camera	256
26.1. Das Tiny Machine Learning Kit	261
26.2. Das Tiny Machine Learning Shield	262
26.3. Pin-Belegung des Tiny Machine Learning Shields	262
26.4. Das OV7675 Kameramodul	263
26.5. Ein USB-A auf Micro-USB Kabel	263
27.1. Montage des Clips	266
27.2. Komplettes System mit Batterie	266
27.3. Batterieclip für 9-Volt-Block[Rei24b]	268
27.4. Voltage Sensor 170640 von dem Hersteller <i>Shenzhen Global Technology Co., Ltd</i> [She]	268
27.5. Testoutput des Spannungssensors	270
28.1. The Tiny Machine Learning Shield	271
29.1. JST connector type VH https://www.jst-mfg.com/product/index.php?series=262	278
29.2. JST connector type VH [Jstc]	279
29.3. JST connector type PHR-2 [Jstc]	279
30.1. Mikrocontroller gesteckt auf dem Shield	281
30.2. Grove-Universal-Kabel[Rei24a]	282
30.3. Grove Jumper zu Grove 4 Pin-Kabel[Rei24c]	282
31.1. Prinzipieller Aufbau Pull-down Schaltung (Eigene Darstellung)	286
31.2. Dual Button oben [M5S21]	287
31.3. Dual Button unten [M5S21]	287
32.1. Resistor with 220 Ohm for an external LED.	290
32.2. Parts of a LED.	291
32.3. Schematic Symbol for a LED.	291

33.1. External RGB-LED with Resistors [Simd]	297
33.2. Connectors of a RGB-LED.	298
34.1. Sensor BME280	303
34.2. Stationärer Aufbau der Wetterstation	304
34.3. Pfad des Testprogramms.	305
34.4. Erste Ausgabe Display	306
35.1. Aufbau eines Servomotors Quelle	308
35.2. PWM am Servomotor Quelle	309
35.3. Auszug aus Gebrauchsanleitung JAMARA 033212, Seite 1 [Jam]	309
35.4. Schaltplan zum Anschluss eines Servomotors JAMARA 033212	310
36.1. Gesamter Schaltplan	312
36.2. Beispiele in der OLED Bibliothek	313
36.3. Testausgabe des OLED Display	314
36.4. Code Einlesen/Zählen	315
36.5. Code Messungssteuerung	315
36.6. Code	317
36.7. Pins des OLED-Displays.	317
36.8. Stationärer Aufbau der Wetterstation	318
36.9. Pfad des Testprogramms.	319
36.10Erste Ausgabe Display	320
36.11Das Display zur Ausgabe der Messwerte	321
37.1. Test program	325
37.2. Test program	326
37.3. Kamera IMX477 der Firma Arducam; [Ard21]	327
37.4. ArduCAM Pin Config	327
37.5. ArduCAM Interface with Arduin Mega 2560	328
38.1. Blockdiagramm, der Kamera OV7675	337
38.2. Camera Module OV7675	338
38.3. Connection Table of the Camera Module OV7675	339
38.4. Camera Module OV7675 connected to the Arduino Nano 33 BLE Sense	339
38.5. RGB565	340
38.6. Installtion of the Arduino Library	341
38.7. Dialogue of RawPixel	342
38.8. Kameramodul OV7675	343
38.9. Bildschirmaufnahme	349
38.10Kamera IMX477 der Firma Arducam; [Ard21]	350
39.1. Circle Grid Camera Calibration Patterns	355
39.2. Siemens Star	356
39.3. Siemens Star	356
39.4. Siemens Star	359
39.5. Siemens Star	360
39.6. Siemens Star	360
39.7. Siemens Star	361
39.8. Kalibrierungswerzeug der Firma Arducam; [Ard21]	363
39.9. Zeilenerkennung	365
39.10Buchstaben in binäre Matrix umwandeln	365
39.11Sektionierung der Buchstabenmatrix	366
39.12Unterteilung der Buchstabenmatrix in Quadranten	366
39.13Radiale Verzeichnung	368

39.14 Tangentiale Verzeichnung	369
39.15 Farbpalette	370
40.1. Prinzipdarstellung der Funktionsweise einer Kamera	381
40.2. Darstellung des Bayer-Filters Prinzips	382
40.3. Darstellung der Bayer-Mosaik-Farbinterpolation einzelner Farben [Vis24]	383
41.1. Farbkarte GretagMacbeth ColorChecker [Bal24]	385
41.2. Optimierung in unterschiedlichen Farbräumen [Kue08, S. 24]	388
42.1. Graukarte von Datacolor Spyder Checkr 24 [Dat24]	390
43.1. Darstellung des Fokus mit Tiefenschärfe [Koe23]	391
43.2. Darstellung der Testschablone Arducam Lens Calibration Tool [Ard24a]	392
43.3. Darstellung der Testschablone SpyderLensCal von Datacolor [Dat23] .	392
44.1. 3D-Kalibrierkörper mit eingezeichneten Achsen nach [CJP23]	393
44.2. 2D-Kalibrierkörper Schachbrett Testschablone [Kru23]	394
44.3. Transformation des Weltkoordinatensystems in das Kamerakoordinatensystem [SR14, S. 294]	394
44.4. Schematische Darstellung der Linsenverzeichnungsarten [Iac16, S. 8] .	396
44.5. Kameramodell der zentralen Projektion [CJP23, S. 546]	396
44.6. Diskretes Kameramodell der zentralen Projektion [CJP23, S. 550]	398
45.1. BLE Devices – Peripheral and Central Devices	404
45.2. Step 2: Creating the Custom Android Menus and Generating the Code	405
45.3. App nRF is searching the Arduino Nano BLE Sense	406
45.4. App nRF is searching the Arduino Nano BLE Sense	407
45.5. App nRF is getting data the Arduino Nano BLE Sense	407
45.6. BLE Devices – Peripheral and Central Devices	410
45.7. Battery app “nRF Connect”	412
45.8. nRF Connect Screen	415
45.9. Accelerometer Values Read from Arduino Using BLE	416
45.10 Bluetooth Connection	417
46.2. Netzadapter PA0217	423
46.1. Netzwerkschnittstelle ENC28J60	424
46.3. GUI	425
47.1.]	430
47.2. Photoplethysmography	431
47.3. The Photoelectric Effect	432
47.4. Structure of a Sensor	432
47.5. Heart Rate Sensor with Grove Interface and Ear Clip [See15]	434
48.1. Circuit Diagramm of Application	438
49.1. Ausbreitungsgeschwindigkeit von Schall in Luft	450
49.2. Abnahme der Schallintensität in Abhängigkeit der Frequenz bei einem Meter, nach [HS23]	451
49.3. Spannungsverlauf Schallwandler bei einer Echo-Laufzeit-Messung, nach [HS23]	452
49.4. Ultraschallabstandssensor der Marke JOY-IT [Sime]	453
49.5. Testoutput des Ultraschallsensors	456

51.1. Batterieclip für 9-Volt-Block[Rei24b]	464
51.2. Testoutput des Spannungssensors	465
51.3. Example with Battery and Jumper	466
51.4. Jumper connection Arduino card	467
51.5. Photo of the jumper on the project	467
52.1. Die verwendete Powerbank	471
53.1. Motorsteuerung DEBO MotoDriver2 L298N	474
53.2. Treiber mit Beschriftung	474
54.1. Terminal Adapter Board mit Schraubklemmen kompatibel mit Nano V3 und Arduino	476
55.1. Getriebemotor	477
57.1. SD/TF-Card-Shield-Modul	483
57.2. Ausgabe SD-Kartentest Serieller Monitor	485
58.1. Sensor DHT22	487
58.2. Positionierung der DMS auf Membran	488
58.3. Aufbau des piezoresistiven Sensorchips	489
58.4. Schaltplan des Raumklimamessgeräts	491
58.5. Opus 20 THI	492
58.6. Graph für Abweichung der Luftfeuchtigkeit	495
58.7. Graph für Abweichung der Temperatur	495
58.8. Graph für Abweichung des Luftdrucks	495
59.1. Anschluss des Sensors ans den Arduino Nano 33 BLE Sense	498
59.2. Aufrufen der Bibliotheken in der Arduino IDE	499
59.3. Installation der TimerOne Bibliothek	500
59.4. Aufrufen der Beispiele für die TimerOne Bibliothek	501
59.5. wird noch in Code umgewandelt!	502
62.1. NRF Connect Application	509
62.2. Landing page	510
62.3. List of devices	511
69.1. Bernstein polynomial of degree 3	535
69.2. Bézier curve for example 69.1	536
70.1. Smoothing a corner with the help of an arc - triangle	537
70.2. Angular change with blending arc	539
70.3. Curvature progression for a blending arc	539
70.4. Maximum range for a blending arc of a circle - $L(\varepsilon = \text{const}, \alpha)$	540
70.5. Deviation when specifying the distance L when rounding with a circular arc	540
71.1. Programme flow chart „Corner rounding“	555
71.2. Programme flowchart „Selection of the rounding strategies“	556

List of Listings

6.1. Simple example using of the builtin microphone of the Arduino Nano 33 BLE Sense	94
8.1. Header file without comments for using the Built-in LED	102
8.2. Code file without comments for using the Built-in LED	102
8.3. Simple sketch without comments to control the built-in LED	103
8.4. Simple sketch to test the built-in LED	103
8.5. A simple watch dog: A simple watchdog: the built-in LED is switched on for 1 second every 30 seconds.	104
9.1. Header file without comments for using a LED	107
9.2. Code file without comments for using a LED	107
9.3. Header file with doxygen comments for using the Power LED	108
9.4. Code file with doxygen comments for using the Power LED	108
9.5. Simple sketch to control the power LED	108
9.6. Simple sketch to check the battery state using the power LED	109
9.7. Simple code for signs of life	110
9.8. Simple code for signs of life	110
9.9. Simple sketch to check the battery state using the power LED	111
10.1. Header file without comments for using the RGB-LED	114
10.2. Code file without comments for using the RGB-LED	115
10.3. Simple sketch to test the RGB LED	115
10.4. value between 0 and 255 to write to the RGB LED	117
10.5. Different brightness levels for the RGB-LED colors	119
10.6. A simple watch dog: A simple watchdog: the built-in RGB-LED is switched on for 1 second every 30 seconds.	120
11.1. Defining the built-in button's pin as an input.	124
11.2. Read the built-in button's state	124
11.3. Simple sketch to test the push button and the built-in LED	124
11.4. Simple sketch connects the push button with an interrupt.	126
12.1. Example code for the sensor LPS22HB on the Arduino Nano 33 BLE Sense	132
12.2. Simple sketch calibrating the sensor LPS22HB	134
12.3. Sketch for the Arduino Nano 33 BLE Sense to switch the sensor LPS22HB into sleep mode	136
13.1. Simple sketch using the sensor APDS9960 for colors	144
13.2. Simple sketch using the sensor APDS9960 for measuring the proximity	146
13.3. Simple sketch using the sensor APDS9960 for gesture detection	147
13.4. APDS-9960: Example Calibration	150
13.5. Simple sketch using the sensor APDS9960	154
18.1. Application of the sensor APDS9960: Setup	170
18.2. Application of the sensor APDS9960: Loop	171
18.3. Application of the sensor APDS9960: Main function	172

19.1. Simple sketch to control the built-in LED	202
19.2. Einlesen der Daten	203
19.3. Messungssteuerung	204
19.4. Überwachung	204
19.5. Initialisierung der Datenverarbeitung	205
19.6. Umrechnung des Mikrofonsignals	205
19.7. Aktualisierung des Bildschirms	206
19.8. PDM Microphone Example Code	208
21.1. Example Microphone	235
21.2. Example IMU (Accelerometer and Gyroscope)	235
21.3. Example ADPS-9960	236
24.1. Simple sketch using the sensor LSM9DS1 detection	251
27.1. Beispiel-Sketch zum Testen der Batterie	269
28.1. Defining the built-in button's pin as an input.	272
28.2. Read the built-in button's state	272
28.3. Simple sketch to test the push button and the built-in LED	272
28.4. Simple sketch connects the push button with an interrupt.	274
32.1. Defining the pin for an external LED	292
32.2. Defining the pin for an external LED	292
32.3. Swichting On the LED	292
32.4. Simple sketch to test a LED	292
32.5. Simple sketch to test a LED with pulse width modulation	293
32.6. Simple sketch to control an external LED. Here, pushing the built-in button is handled by an interrupt. Then the LED switch on for 2 sec.	295
33.1. Swichting On the LEDs	299
33.2. Swichting Off the LEDs	299
33.3. value between 255 - 0 to write to the RGB-LED	299
33.4. Different brightness levels for the RGB-LED colors	300
34.1. Testprogramm für ein OLED-Display	306
36.1. Monitoring	315
36.2. Start and stop sampling	316
36.3. Conversion of the microphone signal	316
36.4. OLED-Aktualisierung	316
36.5. Simple program for OLED displays	320
38.1. Sketch <code>TestENHeader.ino</code>	344
38.2. Sketch <code>TestENCVoidSetup.ino</code>	345
38.3. Sketch <code>TestENCVoidLoop.ino</code>	346
38.4. Sketch <code>TestCameraRawBites320x240x2.ino</code>	347
38.5. Sketch <code>CameraVisualizerHochkant320x240x2.ino</code>	348
45.1. Arduino BLE Tutorial Battery Level Indicator Code	409
46.1. Sketch <code>LinkStatus.ino</code>	425
46.2. Sketch <code>TestENC.ino</code>	426
46.3. Sketch <code>CaptureSingleHexImage.ino</code>	427
47.1. Test sketch for the heart rate sensor	436

48.1. Appliction Heart Rate Sensor	438
48.2. Appliction Heart Rate Sensor - Initialization	443
48.3. Appliction Heart Rate Sensor - Setup	444
48.4. Appliction Heart Rate Sensor - Loop	445
48.5. Appliction Heart Rate Sensor - Functions	447
56.1. Testprogramm für den Encoder	482
57.1. Testprogramm SD-Karten-Modul	486
72.1.,„Hello World“ in Python – Variant 1	558
73.1.,„Hello World“ in Python – Variant 1	560

Acronyms

- ADC** Analog to Digital Converter
- AOP** Acoustic Overload Point
- bpm** beats per minute
- CCD-Sensor** Charge-Coupled-Device
- CIE** Commission Internationale de l'Eclairage
- CMOS-Sensor** Complementary Metal-Oxide-Semiconductor
- CNC** Computerized Numerical Control
- GND** Ground
- GPIO** General Purpose Input Output
- HR** heart rate
- I²C** Inter-Integrated Circuit
- IDE** Integrated Development Environment
- IMU** Inertial Measurement Unit
- IoT** Internet of Things
- JST** Japan Solderless Terminal
- LED** Light Emitting Diode
- mcd** Millicandela
- PCB** Printed Circuit Board
- PDM** Pulse Density Modulation
- PPG** Photoplethysmographie
- PWM** Pulse with Modulation
- RGB** Rot-Grün-Blau
- SCL** Serial Clock
- SD** Secure Digital
- SDA** Serial Data
- SPI** Serial Peripheral Interface
- SPS** Speicherprogrammierbare Steuerung
- tf** TensorFlow
- UART** Universal Asynchronous Receiver Transmitter
- VCC** Voltage at the Common Collector
- WPPLS** White-point Preserving Least Square

1. Introduction

1.1. Introduction

Bézier curves are parameter curves that can be used to represent free-form curves. They are named after the French engineer Pierre Bézier, who developed them in the 1960s at Renault to design car body shapes. During the same period, French physicist and mathematician Paul de Casteljau developed these curves independently of Pierre Bézier at Citroën. Paul de Casteljau's results were available earlier, but they were not published. This is the reason why Pierre Bézier is the namesake of these parameter curves.[Far02]

The Bézier curves are the basis for computer-aided design of models. This is referred to either as Computer Aided Design (CAD) or, when the emphasis is more on a mathematical view, Computer Aided Geometric Design (CAGD). [BN11] Computer requires a mathematical description of shapes to represent them. The most suitable description method for this is the use of parametric curves and surfaces. Here Bézier curves play the central role, because they are the most numerically stable polynomial bases used in CAD/CAGD software.[Far02]

Typography on the computer also uses Bézier curves. There are two ways of representing type with the computer. The simplest way is to save each individual letter with a fixed resolution and size as a bitmap and copy it to the memory area of the screen as needed. These so-called bitmap fonts can be displayed quickly but require a lot of memory if the characters are to be available in different sizes. In this case, an extra bitmap must be created for each size. The quality also decreases when these bitmap fonts are scaled in size and resolution. Vector fonts are an alternative. Their name is derived from the fact that they use curves defined in a two-dimensional vector space to represent the characters. The advantage here is that the characters displayed in this way can be scaled without loss of quality. Two standards have developed for vector fonts. One is the TrueType font and the other is the PostScript font. The TrueType fonts use square Bézier curves while the PostScript fonts use cubic Bézier curves. The cubic Bézier curves have more control points which leads to a better quality.

Another field of application is the control of machine tools. When moving to a corner, the axis must be braked to a standstill at the corner point and then accelerated again after direction correction. This procedure ensures that it is not possible to move at a constant speed, which has negative consequences for the cycle time and quality. A possible solution to this problem is to place a curve between the two sections instead of a sharp corner, which can be run at a constant speed. Bézier curves are suitable for this purpose because only two points and two tangents are needed to form them. In the case of a corner, the points as well as the tangents would lie on the sections that form the corner. The resulting error would be analytically controllable and would allow an adjustment of the curve tolerance.[SS14]

1.2. Challenges and Solutions

1.3. Structure

Part I.

Arduino IDE

2. Arduino IDE 2.3.x

This chapter provides a comprehensive examination of the Arduino IDE, covering its features, interface options, and functionality. It includes a detailed explanation of each component's purpose and usability, a step-by-step installation guide, and instructions for initializing and configuring the environment. This foundational overview equips readers to effectively use the IDE for programming and troubleshooting Arduino-compatible hardware.

2.1. Arduino IDE Description

The Arduino IDE is an open-source official software designed for editing, compiling, and uploading code to Arduino modules. It is cross-platform and compatible with operating systems such as Windows, Linux, and macOS. Built on the Java platform, the IDE supports various Arduino modules and programming in C and C++.

The microcontrollers on Arduino boards are programmed to process information provided in the form of code. Programs written in the IDE are referred to as sketches. These sketches generate a Hex file, which is then transferred and uploaded to the microcontroller.

The IDE environment consists of two primary components: the editor and the compiler. The editor is used to write code, while the compiler compiles and uploads the code to the Arduino module.[FAD18] In version 2.3.3 of the Arduino IDE, the menu bar, located at the top of the interface, plays a crucial role by providing access to important commands such as Upload, Verify/Compile, and Save. Additionally, it includes the File menu for opening new or existing files and the Examples section, which offers prewritten sketches for various applications like Blink and Fade.

The 6 buttons are present on top of the screen are as follows:



Figure 2.1.: Menu Button

- ➊ The icon check mark is used to **Verify** the written code. After the code is entered, selecting this icon checks for any errors and confirms that the code is properly structured. This step ensures the code is ready for use with the hardware.
- ➋ The icon **Upload** in the Arduino IDE compiles your code and uploads it to the connected board, making it ready to run.
- ➌ The icon **Start Debugging** icon in the Arduino IDE initiates a debugging session, enabling you to analyze your code by setting breakpoints, stepping through execution, and inspecting variables on compatible boards.
- ➍ The icon **Serial Plotter** in the Arduino IDE opens a tool that visualizes real-time data sent from the board over the serial connection, displaying it as graphs for easier analysis.

The icon **Serial Monitor** in the Arduino IDE opens a terminal to view and send text-based data over the serial connection, enabling communication with the connected board in real time.

VS:hier noch einmal die letzten Grafiken in der Auflistung nach links verschieben

2.2. Installation of the Arduino IDE

2.2.1. Download of the Arduino IDE

The Arduino Nano 33 BLE Sense utilizes the Arduino Software **Integrated Development Environment (IDE)** for programming, which is the most widely used IDE for all Arduino boards and can be run both online and offline. This open-source Arduino IDE simplifies the process of writing code and uploading it to the board. Various versions of the software are available for each operating system (OS), including macOS, Linux, and Windows. The Arduino community also offers an online platform for coding and saving sketches in the cloud. This online Arduino editor is the latest version of the IDE, featuring all libraries and support for new Arduino boards. To access these software packages, visit the following Website: [Arduino-Software](#), to stay up to date, as there are daily updates available on the mentioned link.

The editor can be downloaded directly from the Arduino Software page with ease. The download options can be seen in Figure 2.2.

Downloads

The screenshot shows the Arduino IDE 2.3.3 download page. On the left, there's a logo of two overlapping circles with a plus sign and minus sign inside. Below it, the text "Arduino IDE 2.3.3" is displayed. A brief description follows: "The new major release of the Arduino IDE is faster and even more powerful! In addition to a more modern editor and a more responsive interface it features autocompletion, code navigation, and even a live debugger." It also mentions "Nightly builds with the latest bugfixes are available through the section below." At the bottom, there's a "SOURCE CODE" link and a note that the IDE is open source and its source code is hosted on [GitHub](#). On the right, a teal sidebar titled "DOWNLOAD OPTIONS" lists download links for various platforms:

- Windows**: Win 10 and newer, 64 bits
- Windows**: MSI installer
- Windows**: ZIP file
- Linux**: AppImage 64 bits (X86-64)
- Linux**: ZIP file 64 bits (X86-64)
- macOS**: Intel, 10.15: "Catalina" or newer, 64 bits
- macOS**: Apple Silicon, 11: "Big Sur" or newer, 64 bits

Below the download links, there's a "Release Notes" link.

Figure 2.2.: Arduino IDE 2.3.3 Download

2.2.2. Summary of Options

Below is a summary 2.3 of the available download options for the Arduino IDE, tailored to different operating systems and individual needs. Choose the appropriate version for the device to ensure compatibility and access to the latest features.

Summary of Options:

Operating System	Option	Description
Windows	Windows 10 and newer (64-Bit)	Direct download for Windows 10 and newer 64-bit versions, without using the MSI installer.
	MSI Installer	Easy installation through an .msi package.
	ZIP File	Portable, no installation required, just extract and run directly.
Linux	AppImage 64-Bit (x86-64)	Portable, no installation required.
	ZIP File 64-Bit (x86-64)	Portable, extract and run directly.
macOS	macOS Intel (10.15: Catalina or newer)	For Intel Macs with macOS 10.15 or newer.
	macOS Apple Silicon (11: Big Sur or newer)	For Apple Silicon Macs (M1, M2) with macOS Big Sur 11 or newer.

Figure 2.3.: Summary of DownloadOptions

2.2.3. Installation on Windows

The installation process for Arduino IDE 2 on a Windows computer will be described step by step in the upcoming section. Following the installation instructions, the subsequent chapter will provide an overview of the IDE's features, including the Board Manager, Library Manager, Sketchbook, and more.

To install the Arduino IDE 2 on a Windows computer, simply run the file downloaded from the software page Arduino Software [Arduino-Software](#). Follow the installation steps, as shown in the Figure ??, to ensure correct installation.

Downloads

Figure 2.4.: Arduino IDE Create Agent Installation

After the download is complete, open the **file setup** and proceed with the installation. Select all components as shown in Figure 2.5 in the dialog box, and then click **Next**.

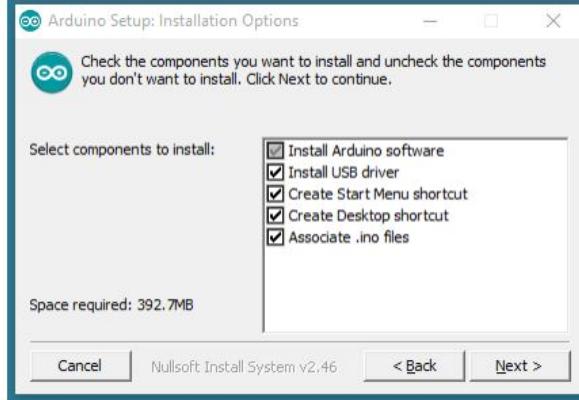


Figure 2.5.: Arduino Setup Installation options

Select the destination folder as seen in Figure 2.6 and click **Install**.

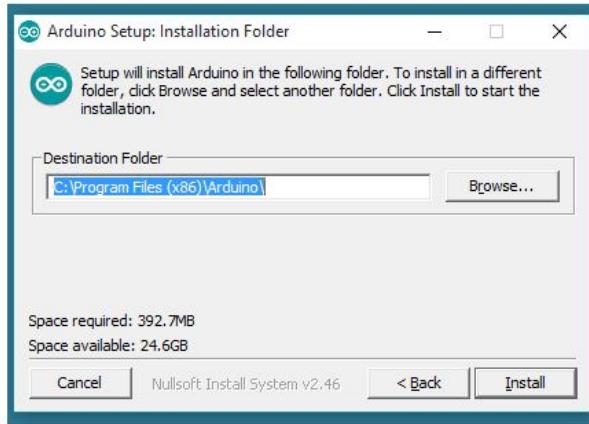


Figure 2.6.: Arduino Setup Installation Folder

Once the installation is complete, open the Arduino IDE. A default sketch will appear on the screen, as shown in Figure 2.7.

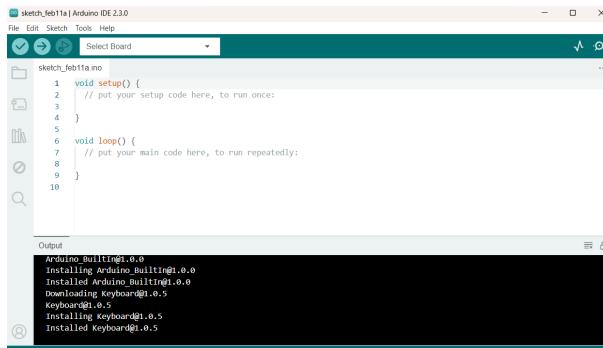


Figure 2.7.: Arduino IDE Sketch

It can be seen from the above figure 2.7 that the basic arduino sketch has two parts. The first part is the function `void setup()` which returns void and we do the initialization such as the output LED color, specifying the core etc. The second part is the function `void loop()` where we define functions which are to be performed

through out the loop. These codes are placed between paranthesis {} and each function has a return type, here it has void return type.

Below in Figure 2.8 is a summary of the installation steps for the Windows version, outlining the key actions required to properly install the software.

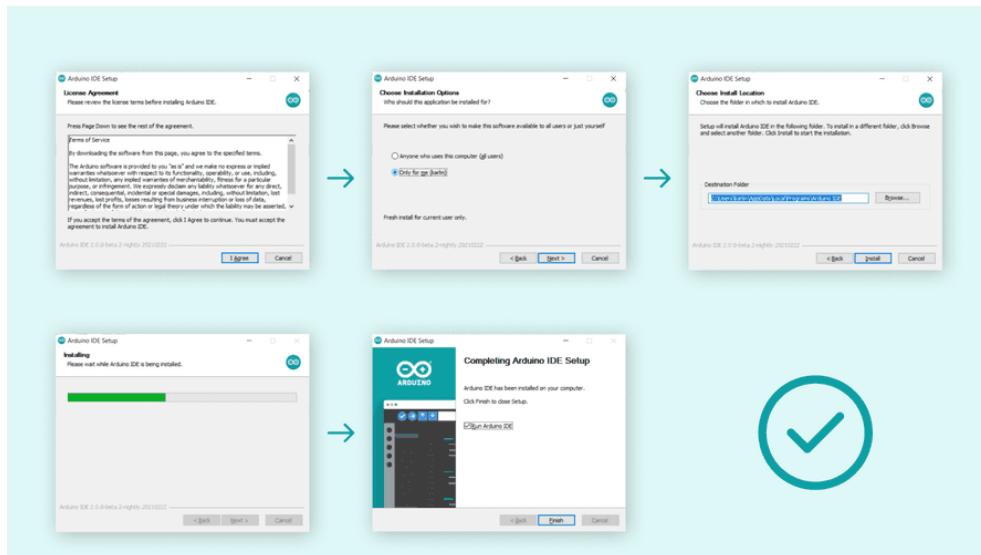


Figure 2.8.: Steps for Installation

2.2.4. Installation (MacOS)

To install the Arduino IDE, begin by downloading the latest version from the Arduino website Arduino Software [Arduino-Software](#). Select the version compatible with the specific operating system in use. In this case, Arduino 2.3.2 is being installed for macOS (Sonoma 14.4.1). The setup file is named [Arduino-Software](#) and has a size of 193,600 KB. This file is in Zip format. For those downloading Arduino IDE 2.3.X with Safari, the file will automatically extract upon download completion, while other browsers may require manual extraction. The figure below displays the latest offline version of Arduino IDE 2.3.3 2.9, which is also compatible with all operating systems.

Downloads



Arduino IDE 2.3.3

The new major release of the Arduino IDE is faster and even more powerful! In addition to a more modern editor and a more responsive interface it features autocompletion, code navigation, and even a live debugger.

For more details, please refer to the [Arduino IDE 2.0 documentation](#).

Nightly builds with the latest bugfixes are available through the section below.

SOURCE CODE

The Arduino IDE 2.0 is open source and its source code is hosted on [GitHub](#).

DOWNLOAD OPTIONS

Windows	Win 10 and newer, 64 bits
Windows	MSI installer
Windows	ZIP file
Linux	AppImage 64 bits (X86-64)
Linux	ZIP file 64 bits (X86-64)
macOS	Intel, 10.15: "Catalina" or newer, 64 bits
macOS	Apple Silicon, 11: "Big Sur" or newer, 64 bits

[Release Notes](#)

Figure 2.9.: ArduinoIDE Create Agent Installation

S:Die Installation muss aktualisiert werden, die folgenden Bilder müssen mit einem Mac ebenso aktualisiert werden

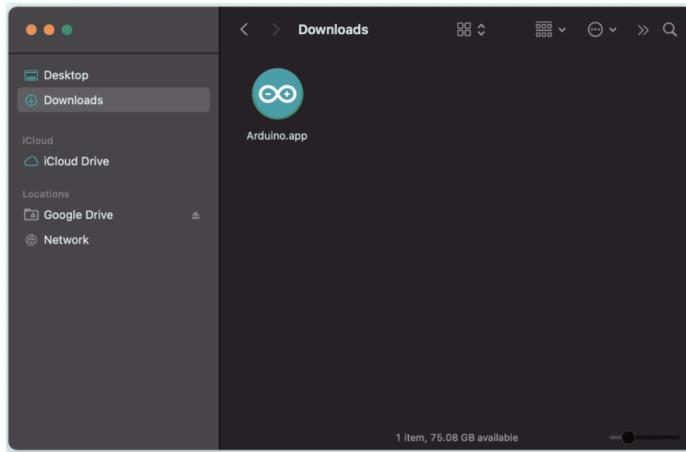


Figure 2.10.: Open the Download folder

Copy the Arduino application bundle into the application's folder (or elsewhere on the computer) then it looks like Figure 2.11.

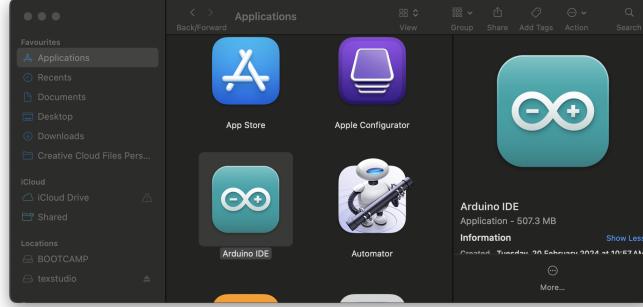


Figure 2.11.: Copy to the Applications Folder

It can be seen from the Figure 2.12 that the basic Arduino sketch has two parts.

- `void setup()`: This function returns void and performs initializations such as setting the output LED color and specifying the core.
- `void loop()`: In this function, specific operations to be executed within the loop are defined. The code for each operation is enclosed within curly braces `{}`, and each function has a return type. In this case, the return type is `void`.

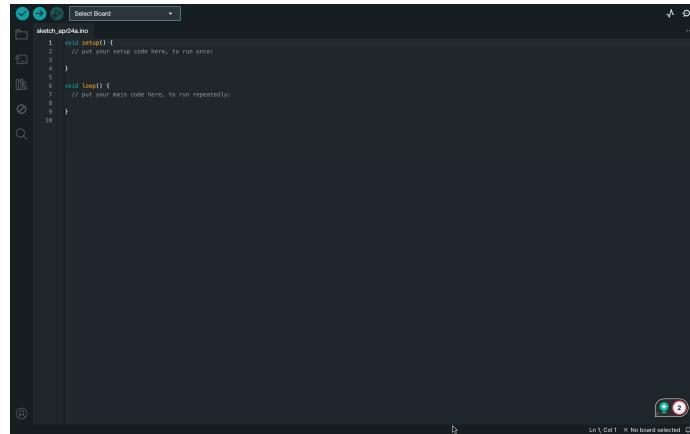


Figure 2.12.: Arduino IDE Sketch

WS:Figure 1.22 muss
größer und alle Bilder i
diesem Kapitel ohne
Schwarzmodus

2.3. Overview

The Arduino IDE 2 features a new sidebar, making the most commonly used tools more accessible. 2.13 [Ard24b]

- **Verify / Upload** These functions are used to compile and upload code to an Arduino board.
- **Select Board and Port** detected Arduino boards automatically show up here, along with the port number.
- **Sketchbook** This section serves as a centralized repository for all sketches that are stored locally on the user's computer. The Sketchbook is designed to provide a structured, easily accessible space for managing, organizing, and editing code developed within the Arduino environment. Additionally, the Sketchbook offers a synchronization feature with the Arduino Cloud, enabling seamless access to stored sketches across devices. This cloud integration allows users to retrieve and edit sketches from the online Arduino environment
- **Boards Manager** browse through Arduino and third party packages that can be installed. For example, using a MKR WiFi 1010 board requires the Arduino SAMD Boards package installed.
- **Library Manager** browse through thousands of Arduino libraries, made by Arduino and its community.
- **Debugger** test and debug programs in real time.
- **Search** search for keywords in the written code.
- **Open Serial Monitor** opens the Serial Monitor tool, as a new tab in the console.



Figure 2.13.: Overview

2.4. Features

The Arduino IDE 2 is a versatile development tool offering features like direct library installation, cloud synchronization for sketches, and built-in debugging tools. This section highlights some of its core features, with links to more detailed resources.

2.4.1. Sketchbook

The Sketchbook is where Arduino code files, known as sketches, are stored. These files are saved with the extension .ino. To maintain proper organization, each sketch must be placed in a folder named exactly the same as the sketch file. For example, a sketch named `MySketch.ino` must be saved in a folder named `MySketch`. This naming convention is essential for the Arduino IDE to correctly identify and manage the sketches. As seen in Figure 2.14.

Typically, sketches are stored in a folder named `Arduino` located within the Documents directory of the system.

To access the Sketchbook, the icon folder in the sidebar of the Arduino IDE can be selected. This action opens the directory where the sketches are stored, allowing for efficient management and access to the sketches.

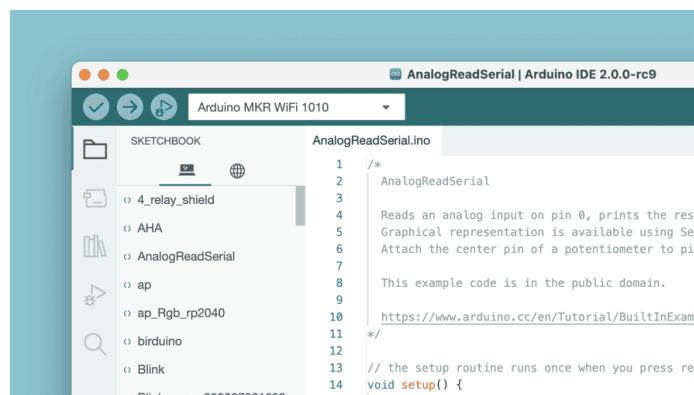


Figure 2.14.: Sketchbook

2.4.2. Boards Manager

The Board Manager can be found in the following steps: **Tools** **Board** **Board Manager**. The Boards Manager in the Arduino IDE allows the installation of different board packages, as shown in Figure 2.15. A board package provides the necessary files and instructions to compile and upload code to specific types of Arduino boards.

Various board packages are available, such as avr, samd and megaavr. Each package supports different Arduino board families. For example, avr is for older boards like the Arduino Uno, while samd is used for newer boards like the Arduino Zero. Using the Boards Manager ensures that the right tools and files are installed for programming the selected board.



Figure 2.15.: Board Manager

2.4.3. Library Manager

The Library Manager can be found in the following steps: **Tools** **Manage Libraries**.

The Library Manager in the Arduino IDE allows users to browse and install a wide range of libraries. Libraries extend the core Arduino functions, making it easier to perform tasks such as controlling servo motors, reading data from specific sensors, or working with modules like Wi-Fi. These libraries provide prewritten code to handle specific hardware components and simplify complex tasks, allowing for faster and more efficient development in Arduino projects. As seen in Figure 2.16.

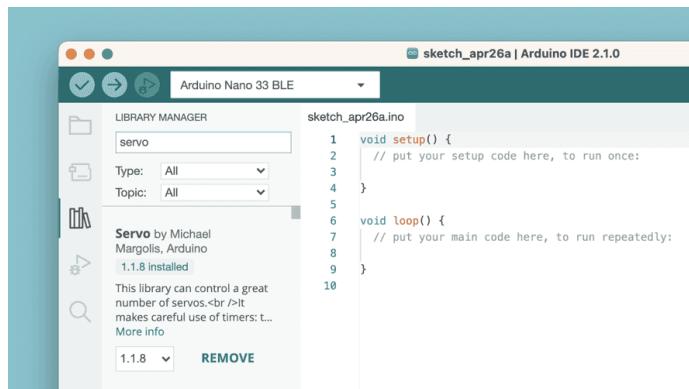


Figure 2.16.: Library Manager

2.5. Integrating and Using a Custom Library in Arduino IDE

In the development of embedded systems using the Arduino platform, libraries are essential for abstracting and simplifying complex functionality. A custom library is particularly beneficial when a specific functionality or hardware interface is repeatedly used across different projects. This section outlines the process of creating, installing, and using a custom library within an Arduino project, in the Arduino Integrated Development Environment (IDE).

2.5.1. Creating the Library Files

A well-structured custom Arduino library typically consists of at least two key components:

Header File (.h): This file contains the declarations of classes, functions, and variables that define the interface of the library. It provides the necessary definitions for the functions and classes to be used by external programs.

Source File (.cpp): This file contains the actual implementation of the functions and methods declared in the header file. It defines the behavior of the functions and classes. The library structure is organized in a way that allows for clear separation between the declaration and implementation of its functionalities.

2.5.2. Installation and Integration of the Library

Once the custom library has been created, it must be properly installed and integrated into the Arduino IDE to be used in projects.

To install a custom library in Arduino follow these steps:

1. Locate the Libraries Folder: The Arduino IDE searches for libraries in the libraries folder, which is located within the Arduino sketchbook directory. The typical path to this directory is:

C:/Users/<username>/Documents/Arduino/libraries

2. Copy the custom library folder, e.g., Nano33BLESenseLED, into the directory `libraries`. The final path should look like this:

C:/Users/<username>/Documents/Arduino/libraries/Nano33BLESenseLED

3. Restart the Arduino IDE: After placing the library in the correct directory, restart the Arduino IDE to ensure that it recognizes the newly added library.

2.5.3. Using Symbolic Links for Library Integration

In some cases, it may be more convenient or necessary to store the library files outside the default library directory. For example, if the libraries are stored in a GitHub repository or for better version control, the command `mklink` can be used to create a symbolic link. This allows the Arduino IDE to access the library files as if they were in the default directory, without duplicating the files.

2.5.4. Windows Tool `mklink` to Create Symbolic Links

Symbolic links allow libraries to be stored in a different location while still being treated by the Arduino IDE as if they reside in the standard directory `libraries`. This can be particularly useful for version-controlled environments, such as when using Git, or to organize libraries without duplicating files.

Steps for Creating a Symbolic Link:

- Open Command Prompt with Administrator Privileges:** Press **Win + S** and search for **cmd**, click on **Command Prompt** and select **Run as Administrator** as seen in Figure 2.17

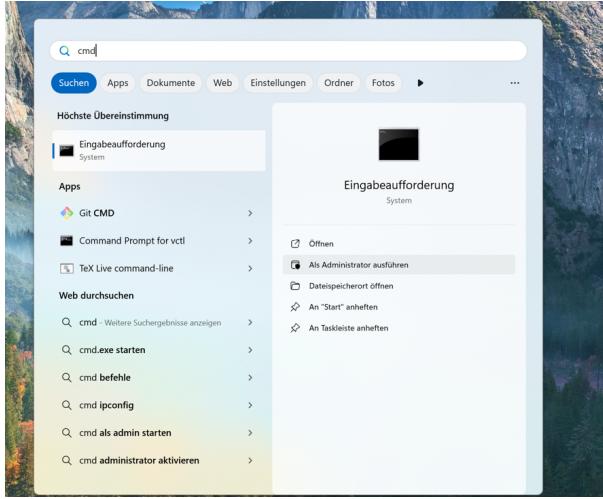


Figure 2.17.: Command Prompt with Administrator Privileges

WS:Figure 1.17, 1.18 w
an english system, so the
figures are on english

- Execute the Command mklink :** The command for creating a symbolic link is: `mklink /D "TargetPath" "SourcePath"`

- TargetPath:** The location where the Arduino IDE expects the library to be, here `libraries`.
- SourcePath:** The actual location of the library.

For this example, the library is located at:

`C:/Users/MSRLabor/Documents/GitHub/241031ArduinoNano33BLESense/report/Code/Nano33BLESense`

The target path in the Arduino IDE's libraries folder is:

`C:/Users/MSRLabor/Documents/Arduino/libraries/Nano33BLESenseLED`

The full command can be seen in Figure 2.18.

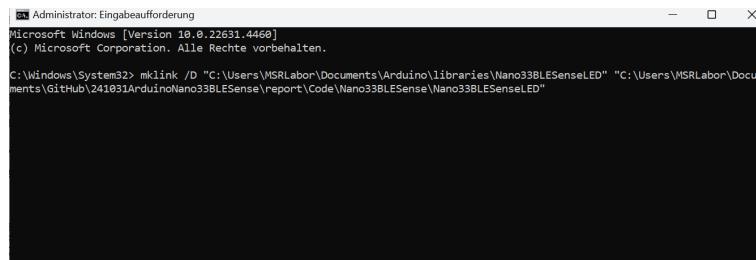


Figure 2.18.: Command Prompt

Verifying the Symbolic Link: After executing the command, navigate to the folder `libraries`. A folder named `Nano33BLESenseLED` should be present, acting as a symbolic link pointing to the actual library location.

2.5.5. MacOS Command `ln -s` to Create Symbolic Links

1. Open Terminal: Open the `Terminal` application by searching for it in `Applications`, go to `Utilities`.
2. Execute the Command `ln -s` : The command for creating a symbolic link is:
`ln -s "SourcePath" "TargetPath"`
 - **TargetPath:** The location where the Arduino IDE expects the library to be, here `libraries`.
 - **SourcePath:** The actual location of the library.

For this example, the library is located at:

```
C:/Users/MSRLabor/Documents/GitHub/241031ArduinoNano33BLESense/report/Code/Nano33BLESense/
```

The target path in the Arduino IDE's libraries folder is:

```
C:/Users/MSRLabor/Documents/Arduino/libraries/Nano33BLESenseLED
```

The full command would be:

```
ln -s "/Users/username/Documents/GitHub/241031ArduinoNano33BLESense/report/Code/Nano33BLESense/" "/Users/username/Documents/Arduino/libraries/Nano33BLESenseLED"
```

Verifying the Symbolic Link: After executing the command, navigate to the folder `libraries`. A folder named `Nano33BLESenseLED` should be present, acting as a symbolic link pointing to the actual library location.

WS:here are some pictures necessary as in
1.5.4

2.5.6. Verifying Library Availability in the Arduino IDE

To ensure that the custom library has been successfully integrated and is recognized by the Arduino IDE, follow these steps:

1. Restart the Arduino IDE: If the Arduino IDE was open during the library installation or after creating the symbolic link, close and reopen it. This step ensures the IDE reloads its list of available libraries.
2. Check the Library Menu: Navigate to `Sketch` go to `[Include Library]` and select the Library `Nano33BLESenseLED`, as seen in Figure 2.19
3. Verify Inclusion in the List: If the library is listed, it indicates successful recognition by the IDE. If not, double-check the directory structure and symbolic link to confirm they are correctly configured.
4. Compile a Test Program: Write a simple test program using functions or classes from the library. Compile the sketch to ensure there are no errors, which confirms that the library has been successfully integrated.

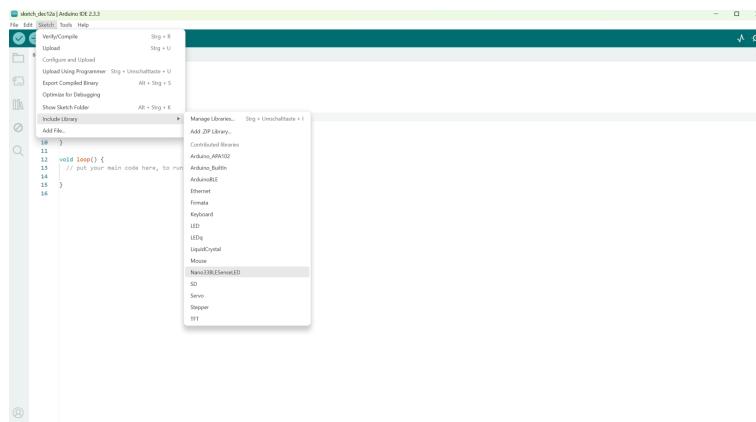


Figure 2.19.: Including the Nano33BLESenseLED Library

Once the program compiles without errors, the library is ready to use, and you can confidently include it in your Arduino projects.

2.5.7. Serial Monitor

The Serial Monitor can be found in the following steps: [Tool](#) [Serial Monitor](#)

The Serial Monitor is a tool that allows to view data streaming from the board, via for example the command `Serial.print()`.

Historically, this tool was located in a separate window but is now integrated with the editor, making it easy to run multiple instances simultaneously on your computer. The Serial Monitor can be seen in Figure 2.20.

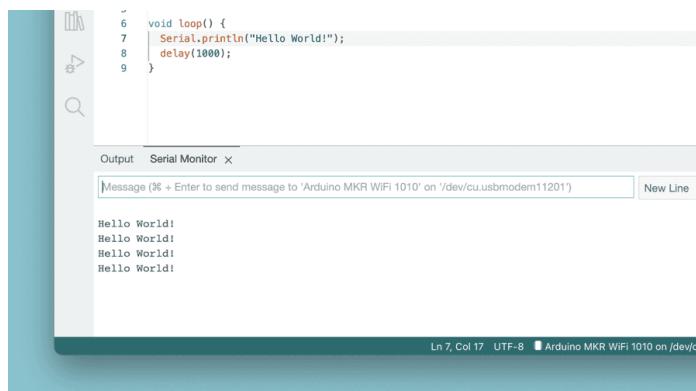


Figure 2.20.: Serial Monitor

2.5.8. Serial Plotter

The tool Serial Plotter is great for visualizing data with graphs and monitoring, such as voltage peaks. It can monitor multiple variables simultaneously, with the option to enable specific types.

2.6. Examples

A significant component of the Arduino Documentation is the set of example sketches bundled with various libraries. These examples demonstrate the practical application of library functions, showcasing their intended uses and main features. Libraries included with certain board packages may also contain their own example sketches. To access these example sketches whether from libraries installed manually or those included with board packages navigate to [File](#) [Examples](#), and locate the desired library from the list.

For instance, when an UNO R4 WiFi board is connected, the examples list appears with options specific to that board. An example sketch demonstrating a pre-loaded Tetris animation can be accessed by navigating to [File](#) [Examples](#) [LED Matrix](#) [MatrixIntro](#) and uploading it to the connected board. This example shows how the board's bundled code can be used in practice, assisting users in learning how to control various hardware functions directly.

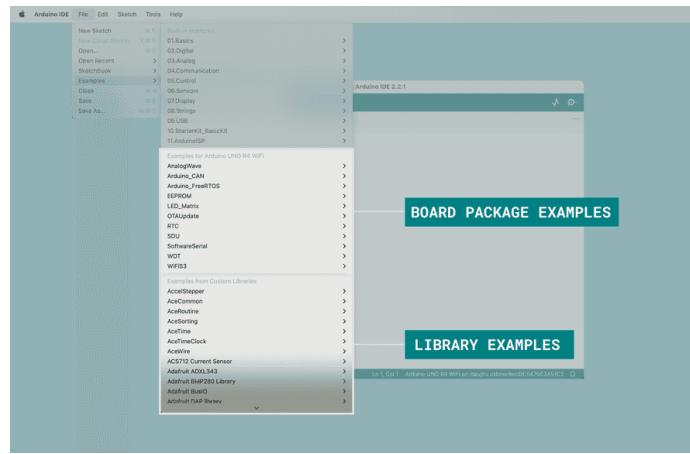


Figure 2.21.: Examples

In Figure 2.21 above, the examples list is shown when a UNO R4 WiFi board is connected to the computer.

2.7. Debugging

The tool Debugger in Arduino IDE 2.3.3 is designed to assist in testing and debugging programs by providing the ability to step through code execution in a controlled manner. This tool allows users to set breakpoints, step through code line-by-line, and inspect variables, helping to identify issues such as logic errors or incorrect variable values. The debugger enables users to pause execution at specific points in the program, allowing for a detailed examination of the program's behavior and memory state. This feature enhances the development process by making it easier to locate and fix errors during runtime, rather than relying solely on print statements or trial and error. 2.22

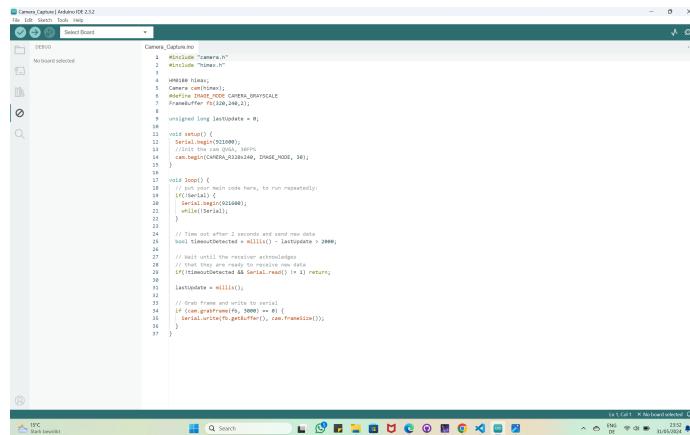


Figure 2.22.: Debugging Example

S:Debugging Bild muss
ersetzt werden, weil nicht
zu lesen

2.8. Autocompletion

Autocompletion is a key feature in Arduino IDE version 2, making it easier to code by suggesting functions and elements from the Arduino API. This feature helps identify and complete code more quickly, improving efficiency and accuracy.

For autocompletion to work, the board must be selected in the IDE. This ensures that the editor recognizes the correct functions and libraries for the specific board, allowing accurate suggestions. 2.23

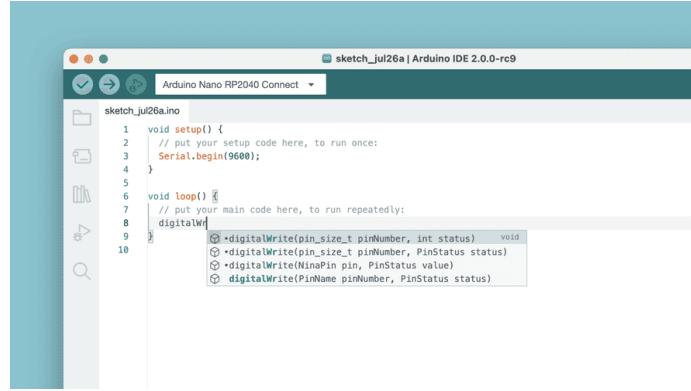


Figure 2.23.: Autocomplete

2.9. Conclusion

This guide provides an overview of key features in Arduino IDE 2, along with references to detailed articles for further exploration. Each section aims to enhance understanding and enjoyment of the wide range of functionalities included in the IDE.

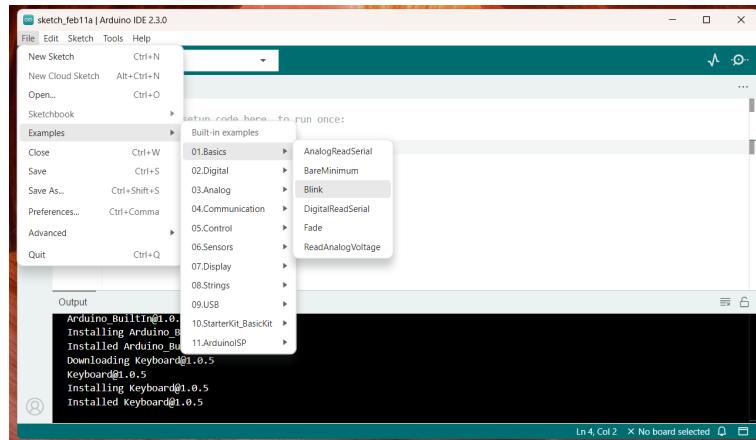


Figure 2.24.: Menu Bar Option

3. To-Do

- Mac Installation nochmal durchführen und aktualisieren (gilt auch für die Bilder)
- In Description: Item-Punkte anpassen mit den Bildsymbolen
- MyNote Kommentare beachten

4. Setup for the Arduino Nano 33 BLE Sense

4.1. Introduction

In this chapter, the process of connecting the Arduino Nano 33 BLE Sense to a computer or laptop and configuring essential settings to get started is explored. The steps for installing the necessary tools and ensuring the board is correctly set up in the Arduino IDE are detailed. Finally, a simple example sketch is demonstrated to verify that the board is working as expected and ready for further development.

There are set of examples which are build in Arduino (IDE) for the testing purpose, for checking all the configuration and setting up the board we can open one of the basic example `blnik.ino` first.

4.2. Configuration for the Arduino Nano 33 BLE Sense

To program the **Arduino Nano 33 BLE Sense** in an offline environment, follow these steps:

4.2.1. Installation of the driver

1. **Installation of the driver:** Begin by installing the latest version of the Arduino IDE on your computer. Refer to Section ?? for more details.
2. **Installation of the packages:** Once the IDE installation is complete, open the `Arduino IDE` and navigate to the menu `Tools`, located in the upper-left corner.
3. In the menu `Tools`, select the `Board Manager`.
4. In the window `Board Manager`, use the search bar to locate the **Arduino Nano 33 BLE Sense** by typing its name as shown in Figure 4.1 .
5. From the search results, select `Arduino Mbed OS Nano Boards` and click `Install` to install the required package.

The Mbed OS Nano Board package includes support for several Arduino Nano family boards, including the Arduino Nano 33 BLE Sense. After completing the installation, connect the Arduino Nano 33 BLE Sense to your computer using a USB-A to USB-Micro to begin programming.

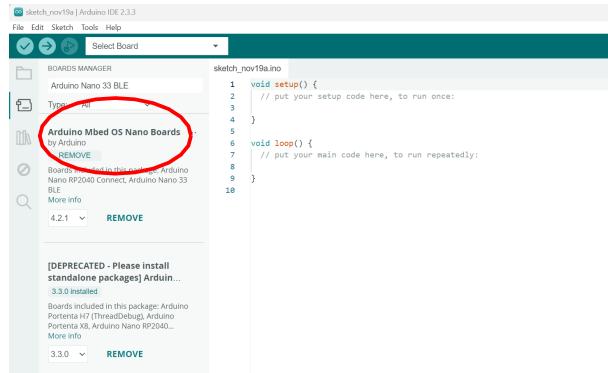


Figure 4.1.: Arduino Mbed OS Nano Boards Installation

4.2.2. Connection and Configuration of the Arduino Board to the Computer

To run either a built-in example or a custom program on the Arduino Nano 33 BLE Sense, follow these initial steps:

1. Connect the Arduino board to the computer using a USB-A-USB-micro-cable.
2. Open the Arduino IDE on the computer. A blank environment page will appear, showing the default `void setup()` and `void loop()` functions.
3. Navigate to the menu **Tools**, select **Board**, and choose the connected board, which is the **Arduino Nano 33 BLE Sense**, as shown in Figure 4.2.

This setup prepares the Arduino IDE to recognize and communicate with the Arduino Nano 33 BLE Sense.

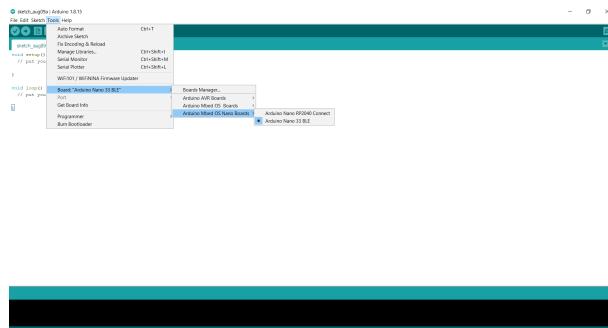


Figure 4.2.: Select the Connected board - here Arduino Nano 33 BLE Sense

4.2.3. Select the Appropriate Port

After selecting the Arduino Nano 33 BLE Sense board, the next step is to verify the connected port. To do this, the Arduino board must be set into Bootloader mode by pressing the white reset button on the board, as shown in Figure 4.4.

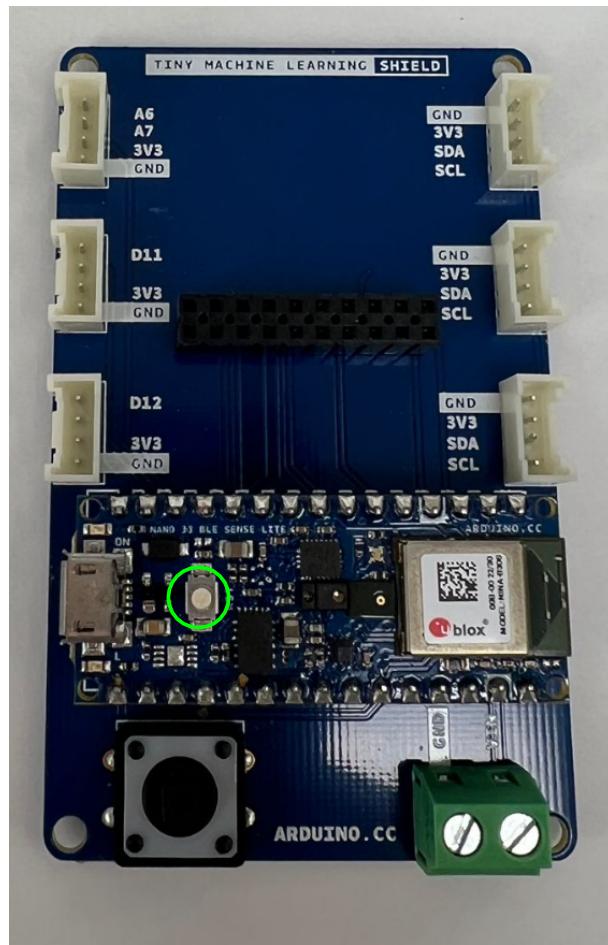


Figure 4.3.: Arduino Nano 33 BLE Sense Reset Button

By pressing the white reset button, the Arduino board will enter Bootloader mode. It is important to verify that the orange Built-in-LED is illuminated, as shown in Figure 4.4.

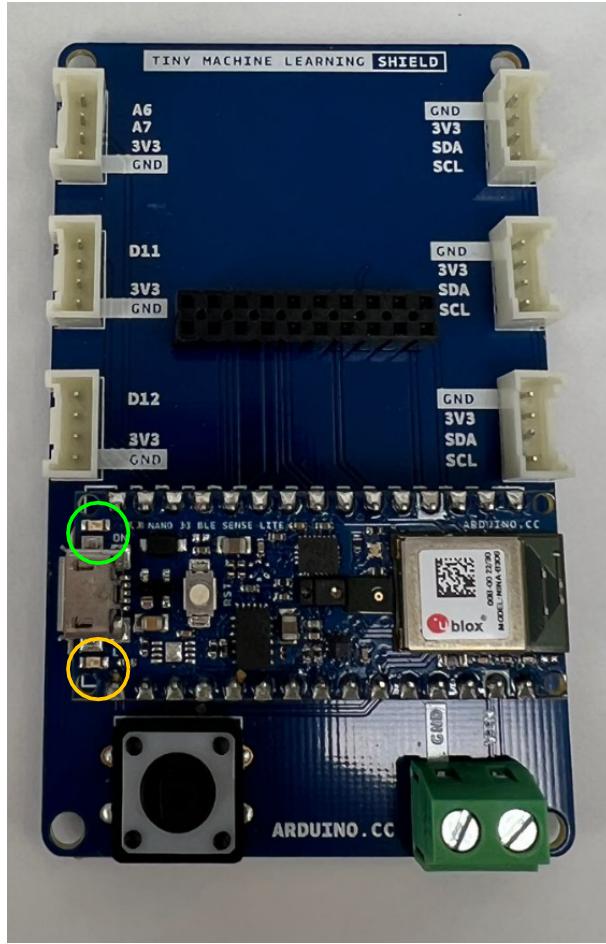


Figure 4.4.: green and orange Builtin-LED

After successfully completing the previously mentioned steps, the next task is to select the connected port before uploading the program. To do this, navigate to the menu **Tools**, select **Port**, and ensure that the available port for uploading the program is properly selected, as shown in Figure 4.5

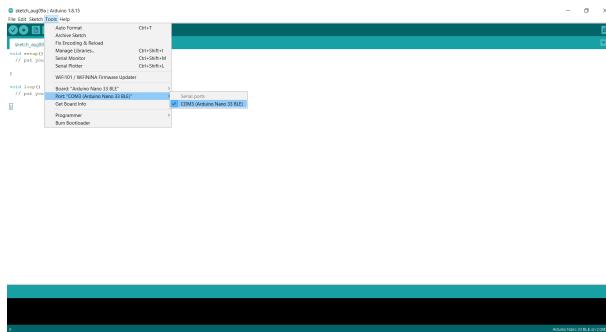


Figure 4.5.: Select Available Port for Uploading Arduino Sketch

4.2.4. Upload and Verify a Sketch

After ensuring that the appropriate port is selected, the next step is to upload the Arduino program. **Before** uploading the program, it is considered best practice to

verify it first. This process will indicate whether any errors or warnings exist in the program. Once the program is successfully verified, it can be safely uploaded by clicking the button **Upload** located below the file section, as shown in Figure 4.6

After selecting the board and port, confirm that the Arduino Nano 33 BLE Sense is properly connected to the Arduino IDE. This can be verified by checking the message in the bottom right corner of the IDE window, which should display the connected board and port.



Figure 4.6.: Upload the Program in Arduino board

After uploading, the code will be compiled, and any issues in the program will be displayed in the bottom black window. Once the code is successfully uploaded and compiled on the Arduino board, it is necessary to reselect the port, as done previously. Navigate to the **Tools menu**, select **Port**, and ensure the correct port is selected, as shown in Figure 4.7, in order to view the output in the Serial Monitor.

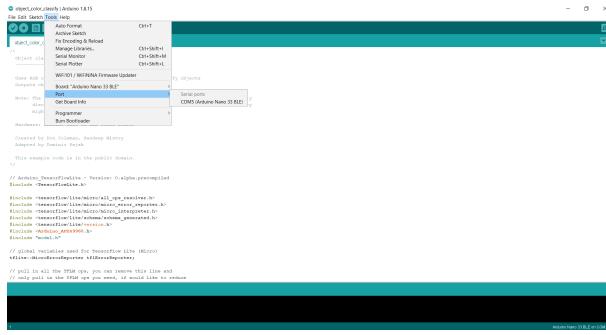


Figure 4.7.: Setting the Port

4.3. Test the Configuration

In this subsection, we will test whether the board is properly connected to the Arduino IDE by running an Example Blink Sketch. This simple test will make an onboard LED light up, verifying both the connection and basic functionality of the Arduino Nano 33 BLE Sense. By the end of this section, you will have confirmed that the hardware and software are working together seamlessly.

4.3.1. Testing the Steps by an Example Sketch `blink.ino`

The Arduino IDE contains a set of built-in examples for testing purposes. To verify the configuration and set up the board, the basic example `blink.ino` can be opened first. After uploading the example `blink.ino` the Built-in LED blinks with 2 Hz. To begin, open the Arduino IDE on the computer and follow the steps below:

1. **Open the Examples:** Once it's open, click on the **File** menu and hover over **Examples** in the dropdown menu.
 2. **Selecting the Example Sketch:** A list of example categories will appear. Under the Built-in Examples, go to **01.Basics** and click on **Blink** as shown in Figure 4.8.
 3. **Blink Example Sketch:** After following steps 1 and 2, the Example Sketch **blink.ino** will open in a new window within the IDE as seen in Figure 4.9.
 4. **Upload Example Sketch `blink.ino`:** Click the **Upload** button (right arrow icon) in the Arduino IDE toolbar to compile and upload the example sketch to the Arduino Nano 33 BLE Sense board. Once the upload is complete, the orange built-in LED starts blinking.

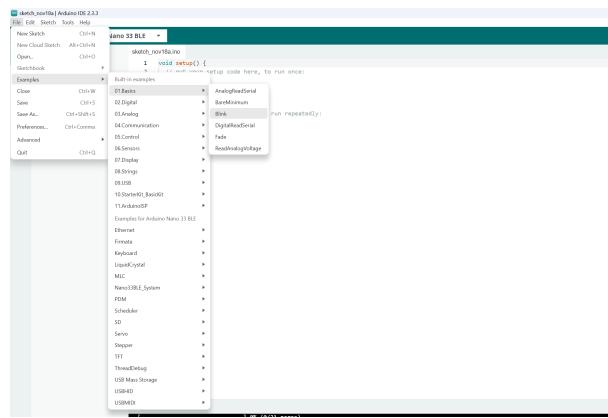


Figure 4.8.: LED Example Sketch

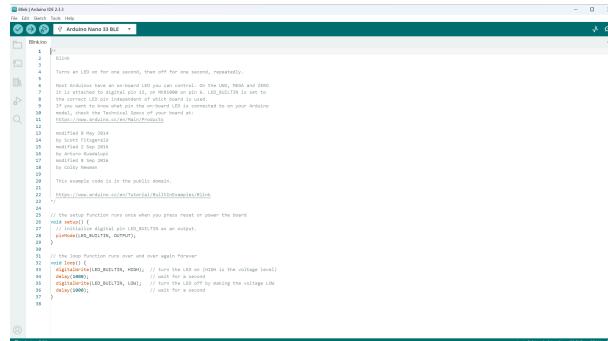


Figure 4.9.: LED-Built Example

With the successful blinking of the built-in orange LED, it is confirmed that the Arduino Nano 33 BLE Sense is properly connected to the Arduino IDE and its basic functionality has been verified. The board is now ready for further development and projects.

4.3.2. Description of Basic Sketch for Printing 'Hello'

To test the development environment and the basic functionality of the hardware, a simple example sketch that controls only one LED is suitable. This sketch provides the Arduino Integrated Development Environment (IDE). Under the path `File/Examples/01.Basics`, small example sketches can be selected. Here, the example `Fade` is used. In this case, the built-in LED, which is an RGB LED, is utilized.

```
int brightness = 0; // how bright the LED is
int fadeAmount = 5; // how many points to fade the LED by

// the setup routine runs once when you press reset:
void setup() {
    // declare LED_BUILTIN to be an output:
    pinMode(LED_BUILTIN, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
    // set the brightness of LED_BUILTIN:
    analogWrite(LED_BUILTIN, brightness);

    // change the brightness for next time through
    // the loop:
    brightness = brightness + fadeAmount;

    // reverse the direction of the fading at the ends
    // of the fade:
    if (brightness <= 0 || brightness >= 255) {
        fadeAmount = -fadeAmount;
    }
    // wait for 30 milliseconds to see the dimming
    // effect
    delay(30);
}
```

As a result, the brightness of the built-in LED should gradually fade in and out.

5. Comments with doxygen

Source code must always be documented. This includes a flowchart as well as the documentation of individual functions. The idea and structure of the software is documented in the developer documentation. The documentation of details such as constants and functions is best done in the software itself. There are various tools for this, e.g. Sphinx and Doxygen. [Hee24b; Ben17; Dev24; Ous18]

This chapter describes the use of Doxygen. Doxygen can visualise relationships between classes and their instances (inheritance hierarchy) and dependencies between methods, which is particularly useful for object-oriented projects.

doxygen is a cross-platform which is used for Linux x86-64 since kernel 3.2.0 and gcc in version 4.6.3, Windows x86-64 since Windows XP, Mac OS X x86-64 since version 10.5, and Oracle Solaris under the general public license. doxygen is a tool which generates software documentation intended for programmers from comments of source code. It supports multiple programming languages as C++, C, C#, Objective-C, Java, Python, IDL, VHDL, Fortran, PHP, ... and generates output in different output formats, HTML, CHM, RTF, PDF, LaTeX, XML, man page. It takes account the syntax and the structure of the language. Using doxygen, it is easy to keep up to date the documentation because of writing within code and systematizing the behavior of developers for they document their code.

The graph visualization software Graphviz is integrated in doxygen. Graphviz is a set of open-source tools. Graphviz creates graphs defined through the scripts DOT language which is a graph description language in text format. The figure 5.1 presents an example output of Graphviz.

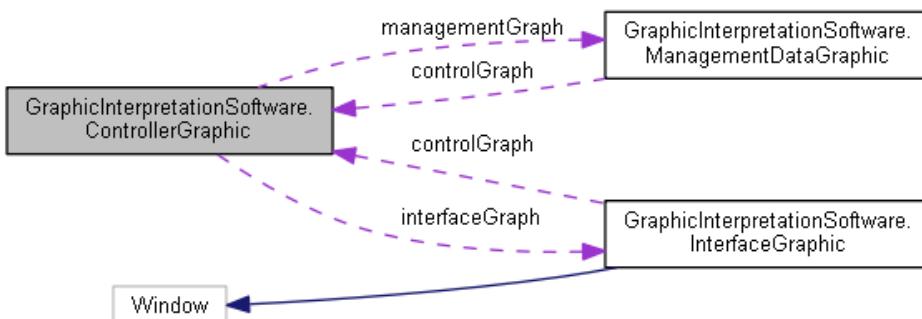


Figure 5.1.: Example output of Graphviz

5.1. Installation

To use all the possibilities of doxygen, two programmes must be installed:

- doxygen
- Graphviz

5.1.1. doxygen

Firstly, the installation file must be downloaded from the website. To do this, click the green button [download](#) on the website <https://doxygen.nl/>, see image 5.2.

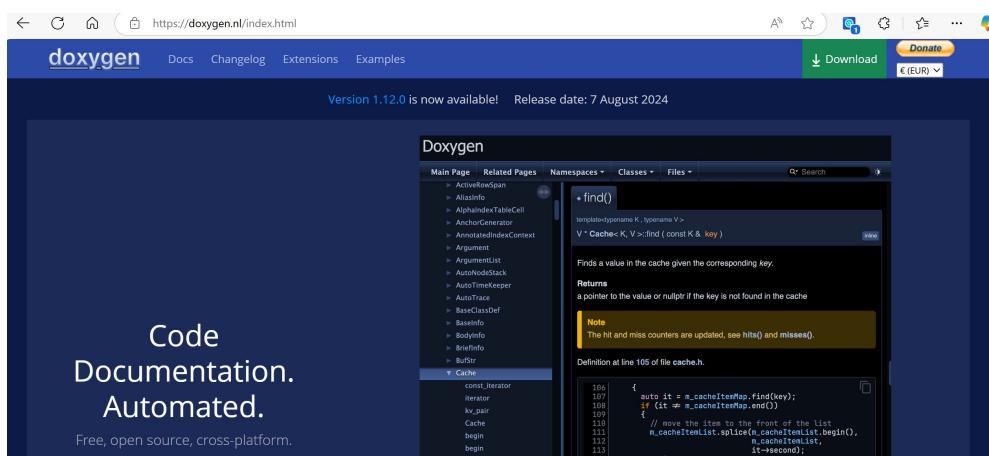


Figure 5.2.: Start image from the website <https://doxygen.nl/>

The download area appears, see image 5.3. In this area, now the version can be selected that matches the operating system of the target system.

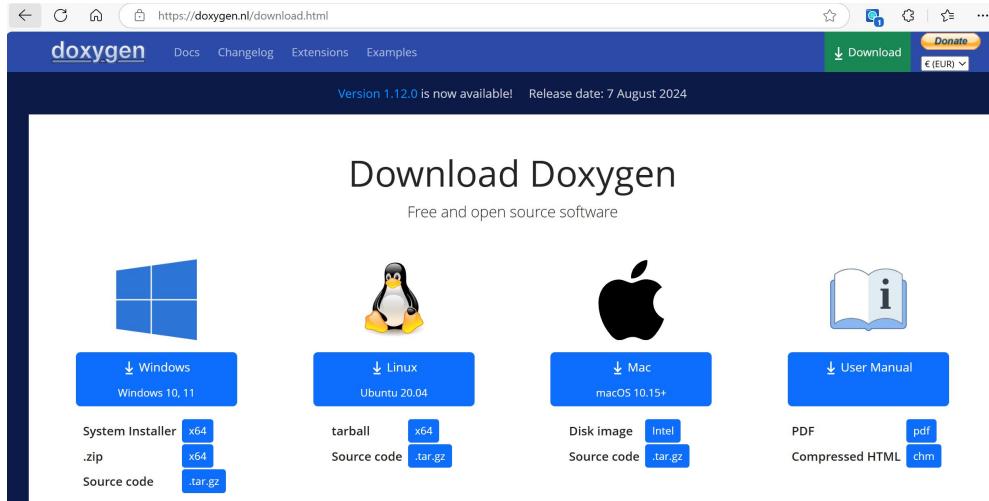


Figure 5.3.: Download area from the website <https://doxygen.nl/>

The appropriate manual [Hee24a] should also be downloaded. The system installer file [doxygen-1.12.0-setup.exe](#) is available for Windows.

After downloading the file, it must be called up. Before the installation begins, several queries are made. Firstly, the installation routine asks whether a complete installation, see image 5.4, should be carried out. Beginners should agree to this.

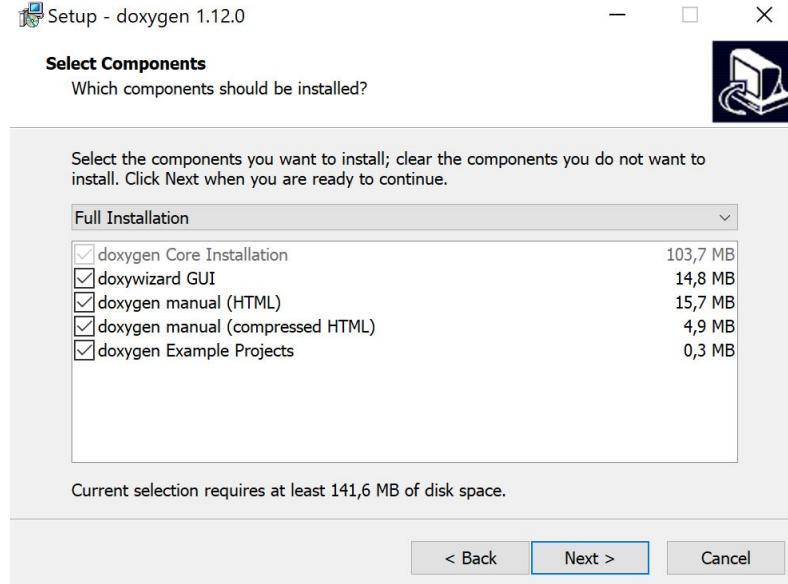


Figure 5.4.: Query after the complete installation during the installation process of doxygen

In the next step, the installation process can be started by clicking the **Install** button, see figure 5.5

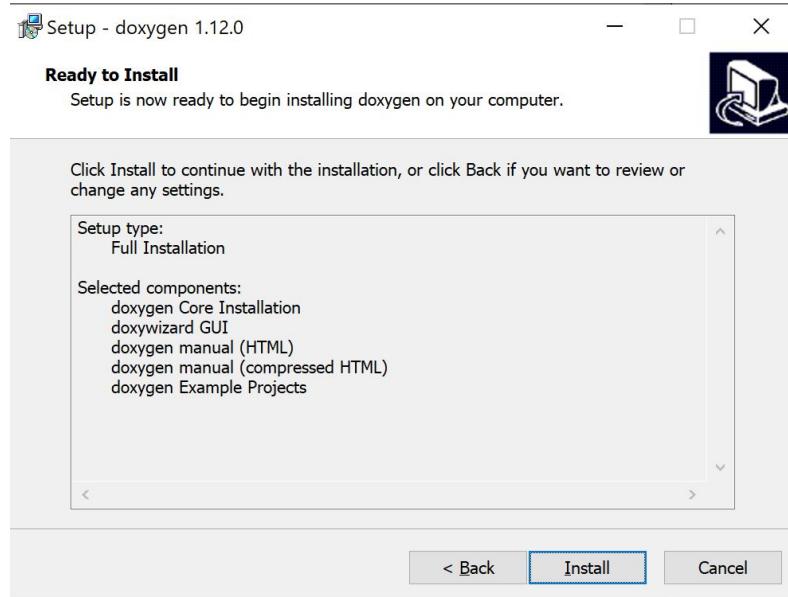


Figure 5.5.: Query after the complete installation during the installation process of doxygen

5.1.2. Graphviz

Firstly, the installation file must be downloaded from the website. To do this, click the button **download** on the website <https://www.graphviz.org/>, see image 5.6.

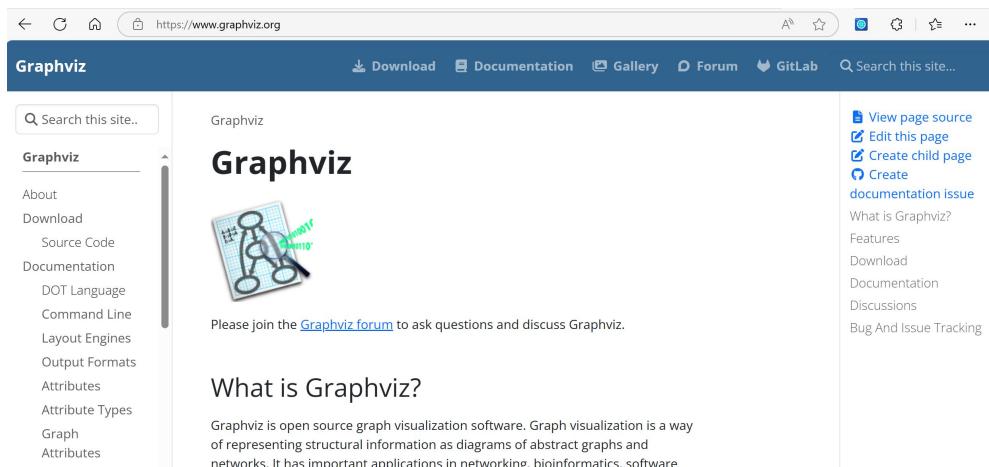


Figure 5.6.: Start image from the website <https://www.graphviz.org/>

The download area appears, see image 5.7. In this area, the version can now selected that matches the operating system of the target system.

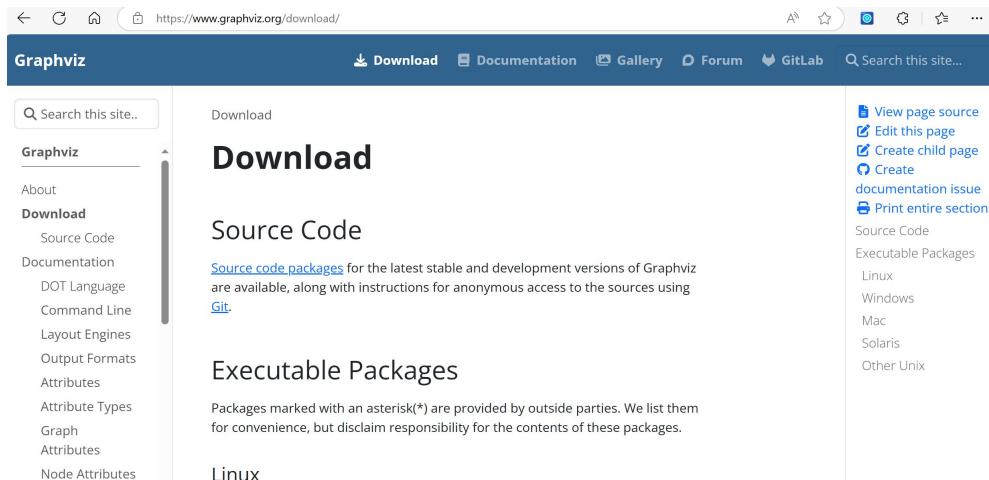


Figure 5.7.: Download area from the website <https://www.graphviz.org/download/>

The system installer file [graphviz-12.2.1 \(64-bit\) EXE installer](#) is available for Windows.

After downloading the file, it must be called up. Before the installation begins, several queries are made. First, the system asks whether the path to Graphviz should be added to the system variable `PATH`. As a beginner, add the path to all users, see figure 5.8.



Figure 5.8.: Query for adding the Graphviz path to the system variable `PATH`

The next step asks for the path for the tool Graphviz tool, see figure 5.10.



Figure 5.9.: Query for the path to Graphviz

Finally, the folder for the start menu must be set. You can select doxygen here, see figure ???. After clicking the button `[Install]`, the installation process starts.

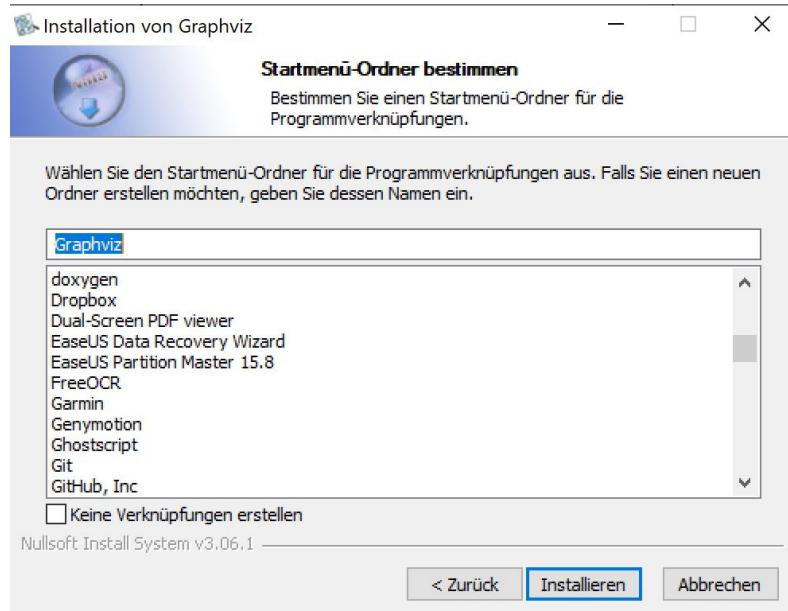


Figure 5.10.: Query for the start menu of Graphviz

5.2. Configuration of Graphviz and doxygen with doxywizard

5.2.1. Graphviz

Graphviz is configured by configuring doxygen with doxywizard.

5.2.2. DoxyWizard

DoxyWizard is a frontend for using doxygen. DoxyWizard configures and saves options of generation of Doxygen. It allows to run easily the extraction the documentation from the source and is available on different platforms.

Das Werkzeug DoxyWizard har 3 tab-Reiter:

- “Wizard”
- “Expert”
- “Run”

In the following, only the tab “Wizard” and then the tab “Expert/Build” and “Expert/Dotare considered. After this, the tab “Run” is described. The tab “Expert” contains more settings which can be selected at a later stage.

DoxyWizard tab “Wizard” - Project

The following settings can be made in the tab window “Wizard/Project”, see figure 5.11:

- Directory of doxygen in which the programme `doxygen.exe` is located.
- Name of the project
- Short description of the project

- Version number of the project
- Logo of the project
- Directory of the sources and, if applicable, its subfolders
- Output directory

When specifying the path, instead of the Windows-typical backslashes “\” the normal slashes “/” have to be used.

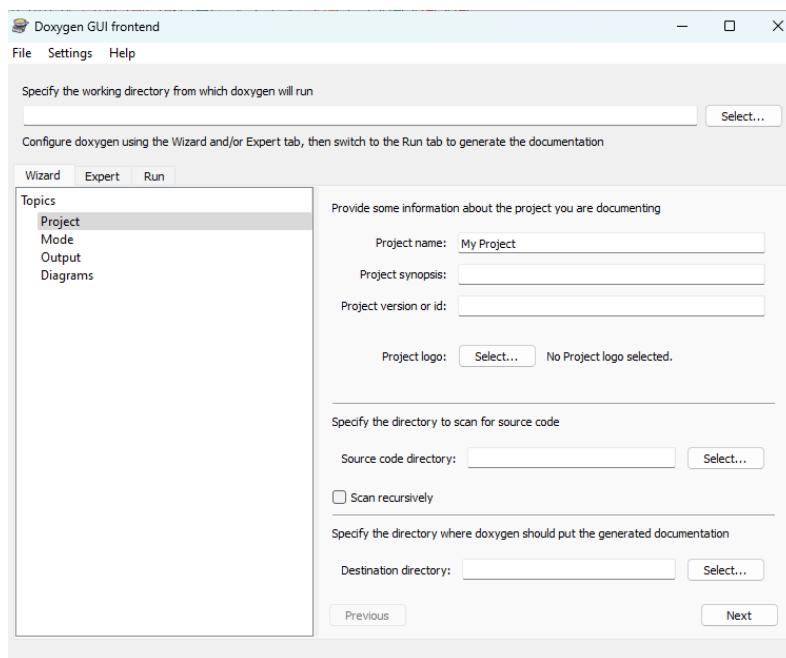


Figure 5.11.: DoxyWizard tab “Wizard” - Project

DoxyWizard tab “Wizard” - Mode

The following settings can be made in the tab window “Wizard/Mode”, see figure 5.12:

- All entities, which recommended, or documented entities only
- Optimization for the language
 - The best choice for Arduino C is C++
 - The best choice for Python is C++, too.

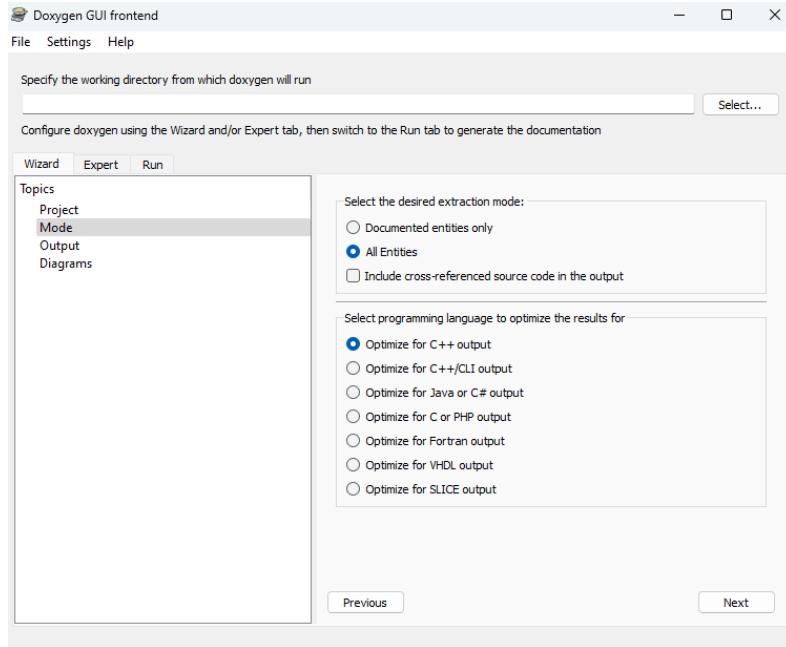


Figure 5.12.: DoxyWizard tab “Wizard” - Mode

DoxyWizard tab “Wizard” - Output

In the tab window “Wizard/Output”, see figure 5.13, the graphic support can be configured. Because of Graphviz’ installation, the support by “Use dot tool from Graphvizpackage” can be used.

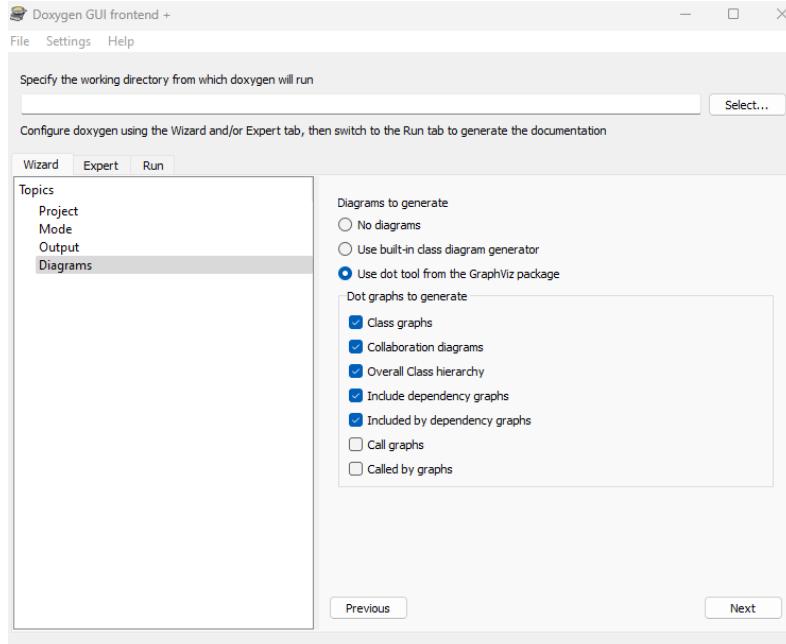


Figure 5.13.: DoxyWizard tab “Wizard” - Diagrams

DoxyWizard tab “Wizard” - Diagrams

In the tab window “Wizard/Diagrams”, see figure 5.13, different output formats can be chosen. For Beginner, the format plain HTML is a good choice. LaTex or other formats can be configured later.

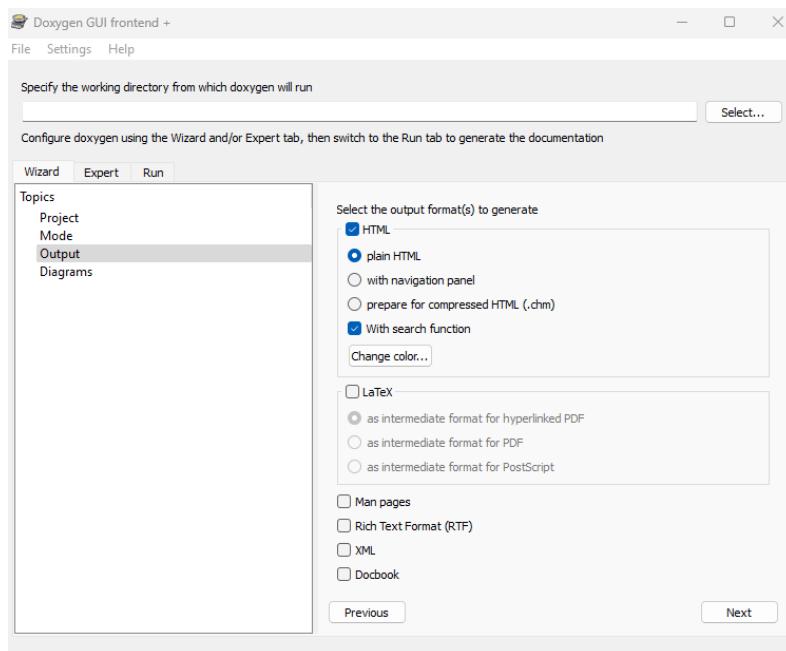


Figure 5.14.: DoxyWizard tab “Wizard” - Output

DoxyWizard tab “Wizard” - Expert - Build

In the tab window “Wizard/Expert/Build”, see figure 5.15, can select what is to be included in the documentation. The items “Extract_All” and “Extract_Private” should also be selected.

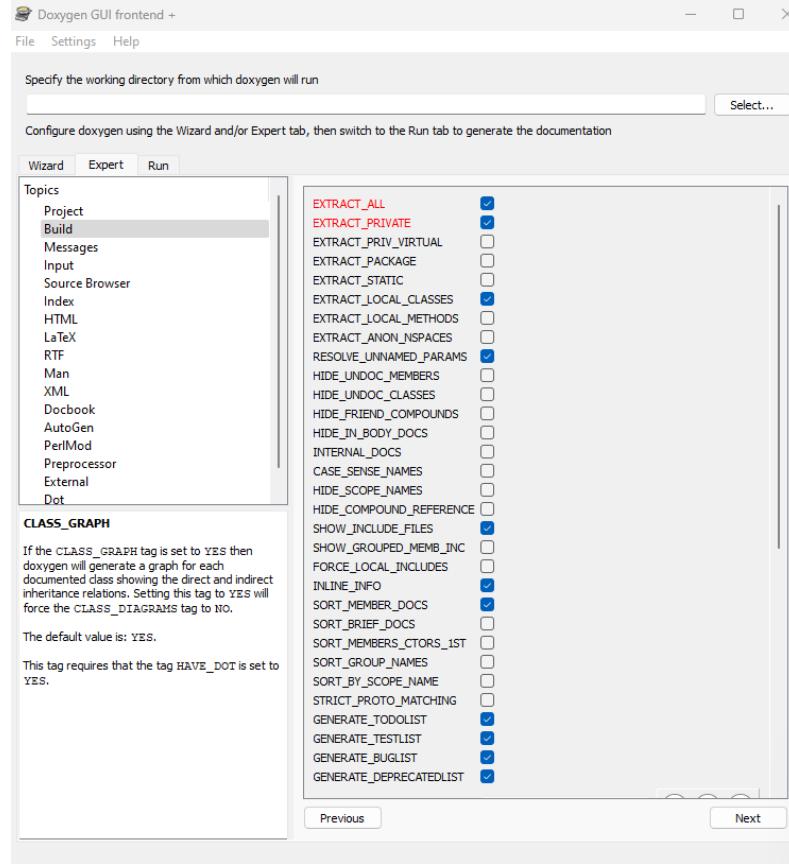


Figure 5.15.: DoxyWizard tab “Wizard” - Expert - Build

DoxyWizard tab “Wizard” - Expert - Dot

In the tab window “Wizard/Expert/Diagrams”, see figure 5.16, the option CLASS_DIAGRAMMS must be selected and the path for the tool specified. When specifying the path to Dot, instead of the Windows-typical backslashes “\” the normal slashes “/” have to used.

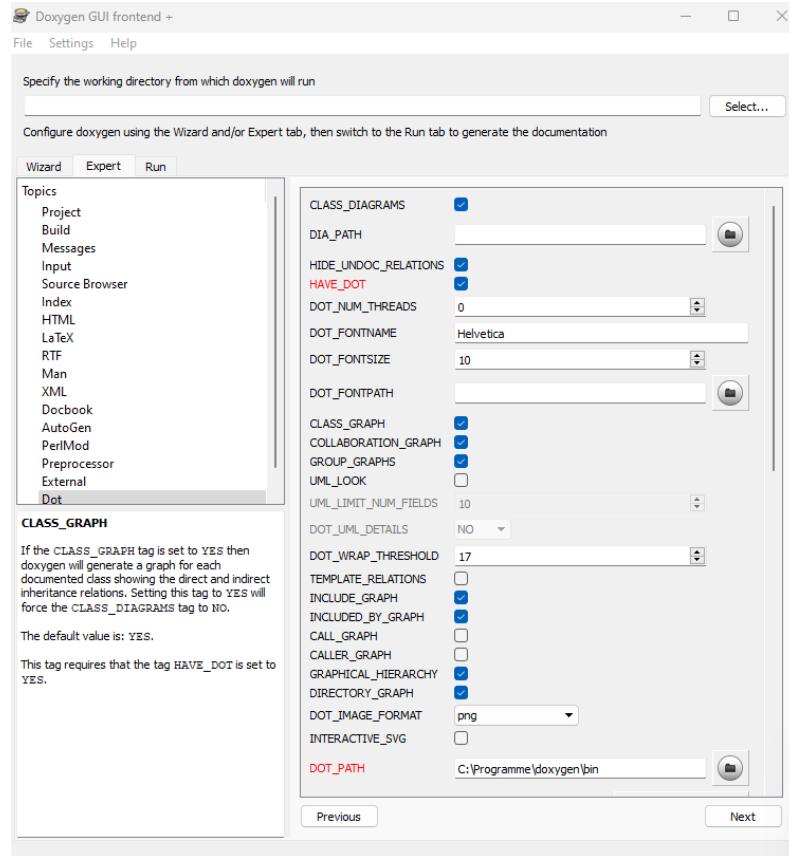


Figure 5.16.: DoxyWizard tab “Wizard” - Expert - Dot

DoxyWizard tab “Wizard” - Run

In the tab window “Wizard/Run”, see figure 5.17, the complete configuration can be shown and the doxygen can be started. The output of doxygen is shown in the window.

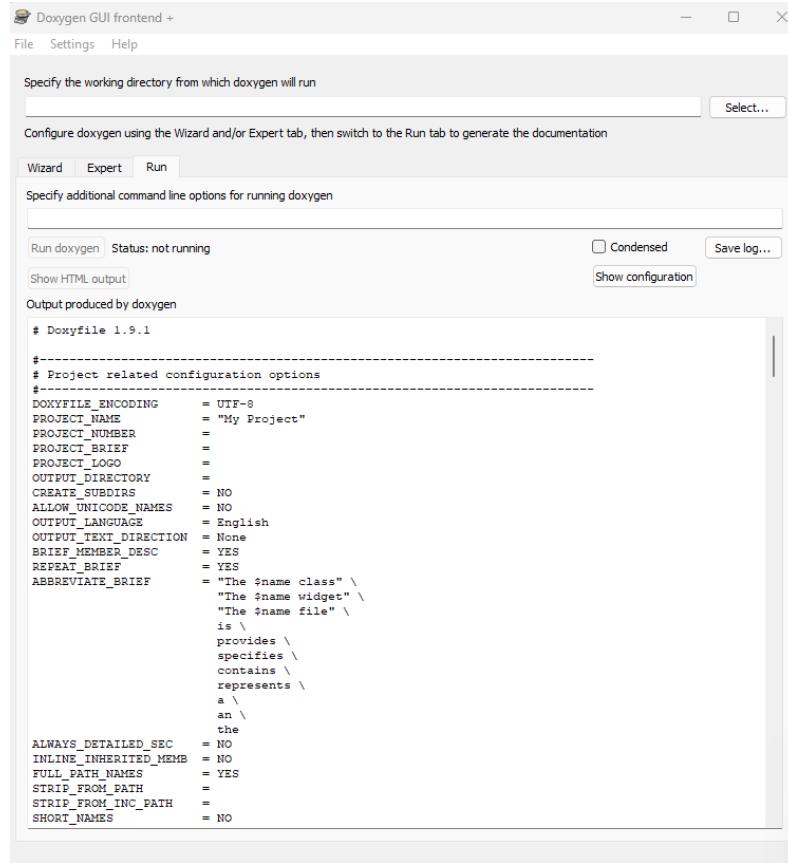


Figure 5.17.: DoxyWizard tab “Wizard” - Run

5.2.3. Saving the Configuration

The configuration of DoxyWizard can be saved in a file. To do this, call the menu **File** → **Save as ...**. The file name can be freely selected. A specific extension is not required. A good choice would be the project name with the extension **.doxyfile**.

5.3. Run of doxygen

To generate the doxygen documentation, there are two possibilities:

- DoxyWizard
- Command shell

5.3.1. Run of doxygen using DoxyWizard

In the tab window “Wizard/Run”, see figure 5.17, the complete configuration can be shown and the doxygen can be started. The output of doxygen is shown in the window.

5.3.2. Run of doxygen using a Command Shell

This doxyfile can be used to generate the documentation by calling the shell command

doxygen -g <config_file>

If the project name is “Nano33BLESense” and the doxyfile has the name “Nano33BLESense.doxyfile”, then the command

```
doxygen -g Nano33BLESense.doxyfile
```

generates the documentation. The structure of the doxyfile is an ASCII file with a lot of keywords. One keyword is **INPUT**. Every file, which contains some documentation, can be added.

```
INPUT = mainpage.dox \
Arduino.dox \
Test \
Test/TestLED.ino \
Test/TestLEDBuiltinApplication.ino \
Test/TestLEDPowerBrightness.ino \
Test/TestLEDPowerBattery.ino \
Test/TestLEDPower.ino \
Test/TestLEDBuiltin.ino \
SensorLPS22HB \
LEDs/LED.h \
LEDs/LED.cpp \
LEDs/PowerLED.h \
LEDs/PowerLED.cpp \
LEDs/BuiltinLED.h \
LEDs/BuiltinLED.cpp \
LEDs/SignsOfLife.h \
LEDs/SignsOfLife.cpp
```

It is also possible to define a logo for the project. Here, the keyword “PROJECT_-LOGO” is used for this.

```
PROJECT_LOGO = LogoDoxyGen.jpg
```

Filetype .ino

Für die Programmierung von Arduino-Mikrokontroller werden ino-Dateien verwendet. Die Syntax ist C++. Aufgrund der Extension müssen folgende Einstellungen gemacht werden:

```
OPTIMIZE_OUTPUT_JAVA = NO
EXTENSION_MAPPING = ino=C++
FILE_PATTERNS = *.c \
*.cc \
*.cxx \
*.cxxm \
*.cpp \
*.pppm \
*.ino \
*.ixx \
...
```

5.4. Syntax and Keywords

Generally, only minor changes need to be made to the documentation. There are various options; if a variant is chosen, the convention must be implemented consistently.

The following options exist for the C++ programming language:

```
/**  
 * comments  
 */  
  
/*!  
 * some comments  
 */  
  
///!  
/// some other comments  
///!  
  
///  
/// yet other comments  
///
```

The following options are available for the Python programming language:

```
## @package pyexample  
# Documentation for this module.  
#  
# More details.  
  
## Documentation for a function.  
#  
# More details.  
  
"""! Documentation for a class.  
  
More details.  
"""
```

doxygen evaluates predefined keywords. The syntax for this is as follows:

```
/**  
 * @KEYWORD DESCRIPTION  
 */
```

The order of the tags has no importance. doxygen creates documentation even if the code is not completely commented. It indicates warning during compilation:

```
warning: The following parameters of <name class>  
are not documented: parameter 'dv'
```

Some keywords are described in the following sections.

5.4.1. Keywords for Files

```
/** @file TestLEDPower.ino  
*  
* @date 10.12.2024  
*  
* @brief Simple program for testing the power LED  
*  
* Turns the power LED on for one second, then off for one second, repeatedly.  
*  
* The LED is switched on for 1 second and switched off  
* for 1 second so that the LED flashes accordingly.
```

```

/*
* On the Arduino Nano 33 BLE Sense, it is attached to digital pin 25
*/

```

5.4.2. Keywords for Functions

```

/***
* @fn Name
*
* @brief the setup function runs once when reset or power the board is pressed
*
* standard function of Arduino sketches
*
* Initialization of the pin LED_BUILTIN as output
*
* @param ---
*
* @return void
*/

```

5.4.3. Keywords for Definitions

```
#define SET_ON true /*< Define flag for switching on */
#define SET_OFF false /*< Define flag for switching off */
```

5.5. Use of Documentation

If doxygen generates HTML documentation, there is a file [index.html](#). Calling up the file [index.html](#) displays the documentation in a browser.

5.6. Add Ons

5.6.1. Mainpage

Basically, these steps are already sufficient for documentation. However, there is not much on the start page apart from the title and version number. A customised start page can be inserted here. The keyword here is [@mainpage](#) and is followed by a description of the project.

In the HTML documentation, the start page corresponds to the file [index.html](#).

```

/***
* @mainpage Example
*
* Description of the project <br>
*
* With the keyword <img>an image can be included.
* <img src = "../images/application_screenshot.jpg" alt = "Screenshot">
*
* @author Elmar Wings
*/

```

```

/***
* @file example.ino

```

```
*  
* @brief A short description of the file – what does it contain, what is it for,  
*  
**/  
  
/**  
* @class MyExampleClass  
*  
* @brief A brief description of the class  
*  
* A more detailed description of the class  
*/  
class MyExampleClass  
{  
    /**  
     * @brief A brief description of the method  
     *  
     * A more detailed functional description  
     */  
    public static void main(String [] args)  
    {  
        System.out.println("HelloWorld!");  
    }  
}
```

5.6.2. Use of simple HTML Commands

Es ist möglich, HTML Kommandos einzufügen. So ergeben sich beispielsweise folgende Möglichkeiten:

- The command `
` forces a new line paragraph.
- The command `` introduces boldface and ends with the command ``.
- The command `<i>` introduces italics and ends with command `</i>`.

5.6.3. Inserting Enumerations

Enumerations can be integrated with the command ``. The complete syntax is:

```
<ul>  
    <li>First</li>  
    <li>Second</li>  
    <li>Third</li>  
</ul>
```

This then leads to an output of the following type:

- First
- Second
- Third

5.6.4. Inserting Images

The `` keyword `` can be used to integrate images. For the start page in particular, it makes sense to include a screenshot of the application, which can be done with:

```

```

The path `../images/` indicates to Doxygen that the image file is located in the project directory in the subfolder `images`.

5.6.5. Inserting HTML pages

It is possible to include complete HTML pages using the command `@page`. The following code is an example.

```
/** @page Arduino Nano 33 BLE Sense

<p>
<ul>
    <li>Date created: 28.8.2024</li>
    <li>Path: Code/Arduino/Blink.ino</li>
    <li>Version: 2.0</li>
    <li>Author: </li>
    <li>Reviewed by: </li>
    <li>Review Date: </li>
</ul>
</p>

<h2>Description</h2>

<p>
The Arduino Nano 33 BLE Sense is a compact, low-power microcontroller board that combines a 32-bit ARM Cortex-M4 microcontroller, a 2.4 GHz BLE module, and a range of sensors including a 3-axis accelerometer, a 3-axis gyroscope, a magnetometer, and a temperature sensor. The board features a USB-C connector for power and data transfer and a microSD card slot for data storage. The board is designed to be battery-powered and has a low power consumption, making it suitable for battery-powered applications. The Arduino Nano 33 BLE Sense is a versatile and powerful board that can be used for a wide variety of projects, from simple IoT devices to more complex wearables.

</p>

<h2>History</h2>

<p>
The Arduino Nano 33 BLE Sense was first announced by Arduino in 2020 as a new member of the Arduino family. It provides a compact and powerful solution for IoT and wearable device applications. The Arduino Nano 33 BLE Sense is based on the Arduino Uno R3 hardware platform. The board was designed to be compatible with the Arduino IDE and other development tools.
*/
```


Part II.

Arduino Nano 33 BLE Sense - Onboard Sensoren

6. Arduino Nano 33 BLE Sense

Arduino is an open-source electronics platform based on flexible, easy-to-use hardware and software. It provides us on board microcontroller and microprocessor kits for making digital and analogue devices. The microcontroller, often called as tiny computer embed on the arduino board, normally in order to run these microcontroller we need some type of electronics e.g; diodes, resistors, capacitors, and transistors for making the voltage and current balancing. But the arduino team make a user friendly environment for getting rid of these electronics complication to run the hardware on software, just power the board as per the required voltage write the desired program and upload it in a few seconds, it will bring everything on the board and make us independent from any worry about the electronics complication. By the technological advancement in semiconductor and electronics industry, the control problems are now being solved by using these small size microcontroller instead of mechanical and electrical switches. All Arduino boards have one thing in common which is a microcontroller, it is basically a really small computer, which help us to make a edge computing application.[Ard21]

WS:Advertisement!
Rewrite more as an engineer!

6.1. Arduino Nano 33 BLE Sense

The Arduino Nano Family is a set of boards with a tiny footprint, packed with features. It ranges from the inexpensive, entry model Nano , to the more feature-packed Nano BLE Sense / Nano RP2040 Connect which comprises of Bluetooth / Wi-Fi radio modules. These boards also have a set of embedded sensors, such as temperature, humidity, pressure, gesture, microphone and more. They can also be programmed with MicroPython and supports Machine Learning. [Raj19].

Arduino Nano 33 BLE Sense is one type of arduino family board, which is come up with Bluetooth Low Energy (BLE) capability for communication and set of sensors. This compact and reliable Nano board is built around the NINA B306 module for BLE and Bluetooth 5.0 communication; Bluetooth 5.0 is the latest version of the Bluetooth wireless communication standard. Use of microcontrollers has become inevitable in almost every field of engineering. The Arduino Nano 33 BLE Sense module is based on Nordic nRF52480 processor that contains a powerful Cortex M4F and the board has a rich set of sensors that allow the creation of innovative and highly interactive designs. Its reduced power consumption, compared to other same size boards, together with the Nano form factor opens up a wide range of applications.

The model name Arduino Nano 33 BLE Sense, itself puts out some important information. It is called “Nano” because of its compact nano size. “33” is included in the model name to indicate that the board operates on 3.3V. Then the name “BLE” indicates that the module supports Bluetooth Low Energy and the name “Sense” indicates that it has on-board sensors like accelerometer, gyroscope, magnetometer, temperature and humidity sensor, Pressure sensor, Proximity sensor, Colour sensor, Gesture sensor, and even a built-in microphone. [Raj19].

Also, for making the RGB color and person detection, we need to make the interface of BLE 33 Sense with camera shield Arducam OV2640. The Arducam OV2640 camera use to detect the RGB, object detection and also the gesture too. Arduino Nano 33 BLE Sense and camera shield Arducam is a perfect match for making ML and AI application, by having the set of sensors on board we just need to install the respective

WS:Advertisement!
Rewrite more as an
engineer!

library on Arduino board and it supports the functionality of sensors and Machine learning application.

The following figure 6.1 shows an Arduino Nano 33 BLE Sense Revision 2, showing how compact it is and small in size.

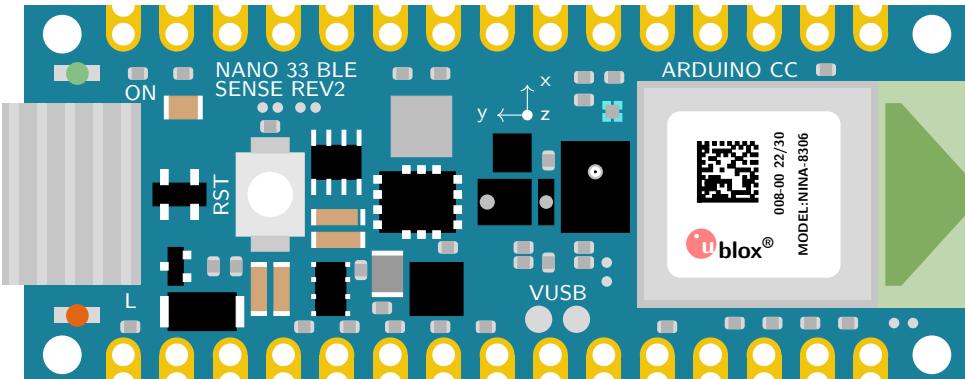


Figure 6.1.: Arduino Nano 33 BLE Sense Rev. 2, see Arduino Store

The BLE (Bluetooth Low Energy) compact and reliable Nano board are built on NINA B306 module for BLE and Bluetooth 5 communication; the NINA B306 module based on Nordic nRF52480 processor that contains a powerful Cortex M4F CPU. Its architecture is fully compatible with Arduino IDE Online and Offline. The Arduino Nano BLE 33 Sense have the following set of sensors on board, ADPS-9960, LPS22HB, HTS221, LSM9DS1, and MP34DT05-A. It is small in size, having all the required sensor on board. [Ard21b]

The Arduino Nano 33 BLE Sense have the following set of Sensors, BLE module and its functionality below.

- The Bluetooth is managed by a NINA B306 module.
- The ADPS-9960 is a digital proximity, ambient light, RGB and gesture sensor.
- The LSM9DS1 is a system-in-package featuring a 3D digital linear acceleration sensor, a 3D digital angular rate sensor, and a 3D digital magnetic sensor.
- The LPS22HB reads barometric pressure and environmental temperature.
- The HTS221 senses relative humidity.
- The MP34DT05 is support the sound detection.

6.2. Unterschied zwischen dem Arduino Nano 33 BLE Sense Rev1, dem Arduino Nano 33 BLE Sense Rev2 und dem Arduino Nano 33 BLE Sense Lite

6.2.1. What is the difference between Rev1 and Rev2?

The differences between the Arduino Nano 33 BLE Sense and the Arduino Nano 33 BLE Sense Rev2 concern the IMU, the temperature and humidity sensor, the microphone, the crypto chip and the voltage converter. In the second revision, the LSM9DS1 IMU has been replaced by two modules: the BMI270 acceleration and angular rate sensor and the BMI150 magnetometer. The HTS221 humidity sensor has been replaced by the HS3003, with the latter promising greater accuracy. The values

of the microphones have remained the same despite the change from the MP34DT05 to the MP34DT06JTR. Similarly, only the component designations of the voltage converters differ. The first generation uses the MPM3610, the Rev2 uses the MP2322. However, the crypto chip is no longer present in the Rev2.

The changes are summarized below:

- Replacement of the IMU of LSM9DS1 (9 axes) with a combination of two IMUs (BMI270 - 6 axes IMU and BMM150 - 3 axes IMU).
- Replacement of the temperature and humidity sensor from HTS221 to HS3003.
- Replacing the microphone from MP34DT05 to MP34DT06JTR.

In addition, some components and changes have been made to increase user-friendliness:

- Replacing the power supply from MPM3610 to MP2322.
- Add a VUSB solder pad to the top of the board.
- New test points for USB, SWDIO and SWCLK.

The figure 6.2 presents the layout of the first version of the Arduino Nano 33 BLE,

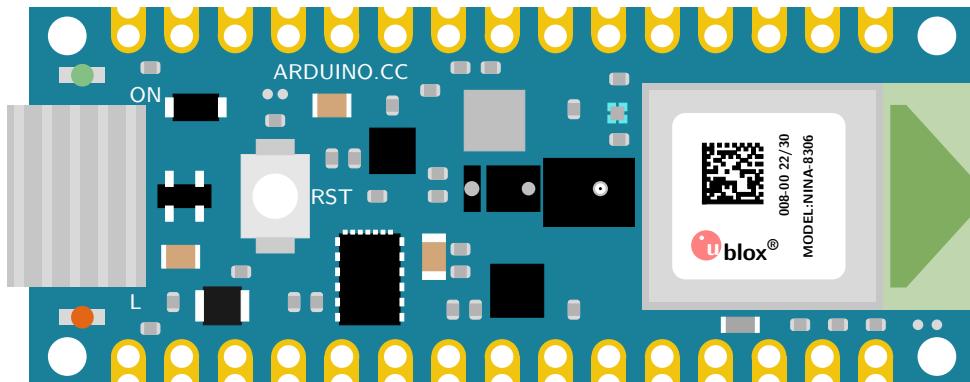


Figure 6.2.: Arduino Nano 33 BLE Sense Rev. 1

6.2.2. Changes in the Sketch

For sketches that were created with libraries such as LSM9DS1 for the IMU or HTS221 for the temperature and humidity sensor, these libraries must be changed to the following for the new revision:

- Arduino_BMI270_BMM150 for the new combined IMU, and
- Arduino_HS300x for the new temperature and humidity sensor.

Rev 1 [TensorFlow Lite lib](#)

6.2.3. Arduino Nano 33 BLE Sense Lite

WS:citation The Tiny Machine Learning Kit contains only the Arduino Nano 33 BLE Sense Lite.

The Arduino Nano 33 BLE Sense Lite differs only slightly from the Arduino Nano 33 BLE Sense Revision 1. The only difference between the two models is that the Lite version does not have the HTS221, the sensor for relative humidity and temperature. The LPS22HB sensor, which is on the board, is a pressure sensor, but it can also be used to measure the temperature. It is therefore not possible to measure humidity with the Arduino Nano 33 BLE Lite. To measure relative humidity, a separate sensor must be connected. The reason for this decision is that the Arduino company has difficulties maintaining the stock. [FD22]

The figure 6.3 presents the layout of the Arduino Nano 33 BLE Lite,

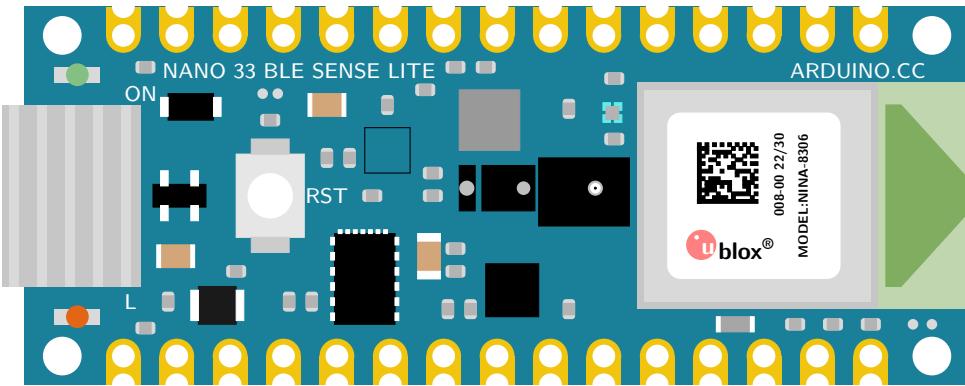


Figure 6.3.: Arduino Nano 33 BLE Sense Lite; the empty blank square marks the position of the missing sensor HTS221

6.3. On-Board Sensor Description

Arduino Nano 33 BLE sense come up with the set of embed sensor on the board. The available embed sensors are commonly use for measuring both the analog and digital values around the sorrounding. Arduino Nano 33 BLE sense is very small 45mm × 18mm in size, which makes it very usefull for Internet of things (IOT) and Artificial intelligence (AI) application as a embed device where space is the main constrained issue. It is low power consumption board and operate normally on 3.3 V, we can say that this small size low power consumption board can operate on small batteries even for many months. Due to on-board available sensor, the low power consumption and mini architecture we can use this nano board anywhere. The Arduino Nano 33 BLE Sense is a completely new board on a well-known form factor. For getting detail information about each component of Arduino Nano 33 BLE Sense and data sheets of each sensor the following links give us a detail information [Ard21b]. The short description of each sensor are as follow.

- The ADPS-9960 is a digital proximity, ambient light, RGB and gesture sensor. it can measure the proximity distance, light, color and gestures when moving close with the borad.
- The sensor LSM9DS1 is a 9 axis Inertial Measurement Unit (IMU) use as a accelerometre, gyroscope, and magnatometre, this 9 axis sensor is ideal for wearable devices.

- The sensor LPS22HB is a barometric pressure sensor, it measures the environmental pressure which is useful for simple weather station monitoring.
- The sensor HTS221 senses the relative humidity, and temperature, to get highly accurate measurements of the environmental conditions.
- The sensor MP34DT05 is the digital microphone. It is useful for capturing, analyzing and detecting the sound in real time.
- The USB port allows you to connect Arduino Nano 33 BLE Sense to your machine.
- There are 3 different LEDs that can be accessed on the Nano BLE Sense: RGB Programmable LED, the built-in orange Programmable LED and the Power LED.

The figure 6.4 shows the embedded sensors on the board, with a powerful processor as compared to other Arduino boards the nRF52840 from Nordic Semiconductors, a 32-bit ARM® Cortex™-M4 CPU processor running at 64 MHz are as follows.

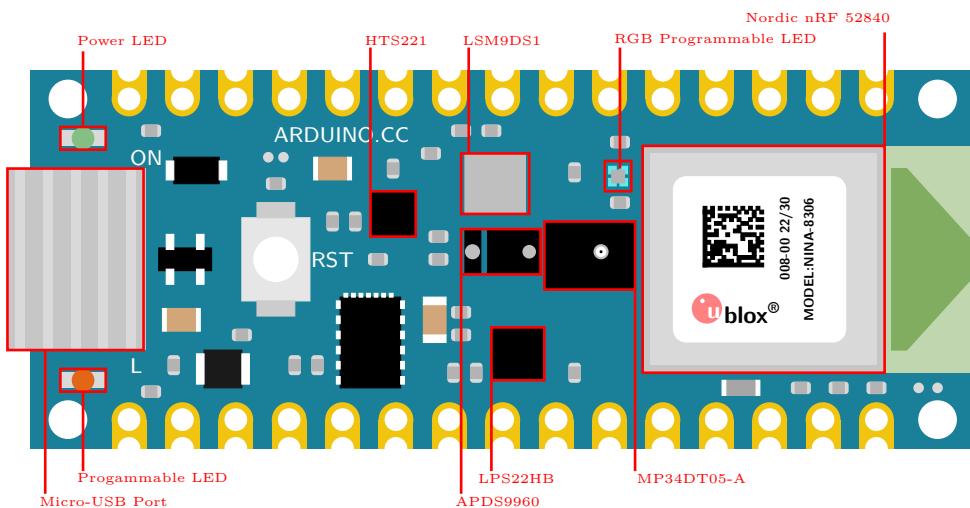


Figure 6.4.

Another point to bear in mind is the overall 'trueness' of sensor readings based on where you place the actual sensors in the environment, as this could be quite critical. The operating temperature should not exceed 85°C and not lower than -40°C. Other factors like humidity level and air pressure values should also be kept in check.

WS:Auch mit dem Rev

MICROCONTROLLER	nRF52840
OPERATING VOLTAGE	3.3V
INPUT VOLTAGE (LIMIT)	21V
DC CURRENT PER I/O PIN	15 mA
CLOCK SPEED	64MHz
CPU FLASH MEMORY	1MB
SRAM	256KB
LED_BUILTIN	13
IMU (Accelerometer, Gyroscope, Magnetometer)	LSM9DS1
MICROPHONE	MP34DT05
GESTURE, LIGHT, PROXIMITY, COLOUR	APDS9960
BAROMETRIC PRESSURE	LPS22HB
TEMPERATURE, HUMIDITY	HTS221

Table 6.1.: Technical Specifications of Arduino Nano 33 BLE Sense [Ard21b]

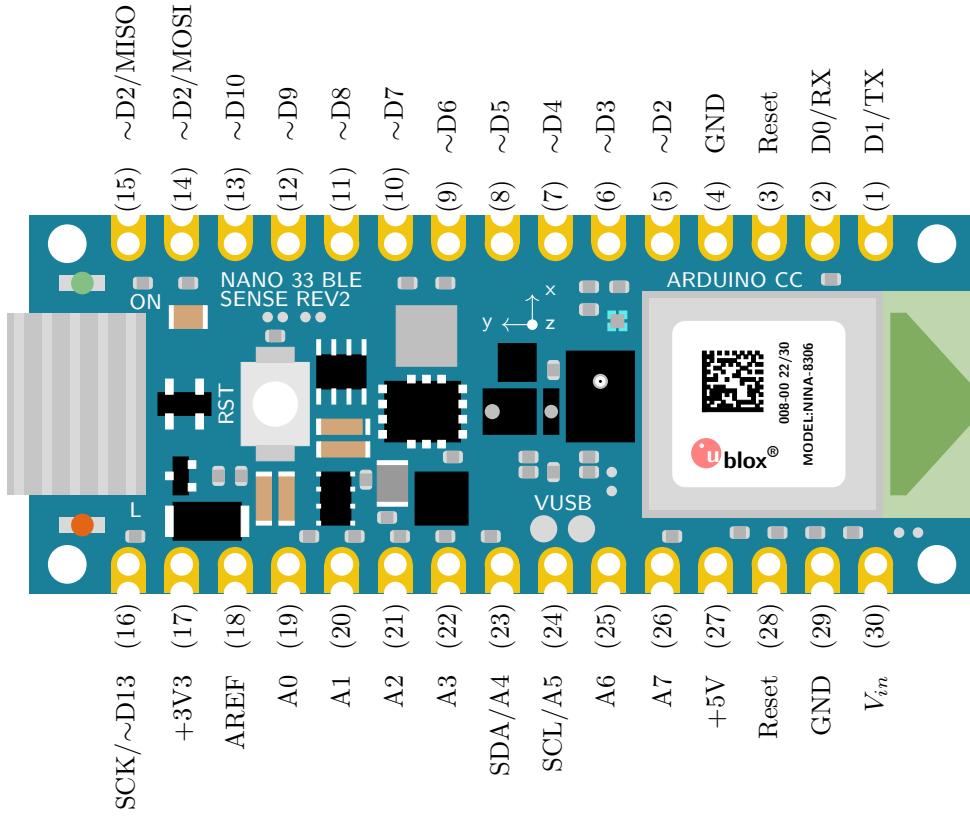


Figure 6.5.: Pin assignment of the Arduino Nano 33 BLE Sense and for the Arduino Nano 33 BLE Sense Rev 2; note that the orange built-in LED is connected to pin D13 and the power LED to pin D1. The built-in RGB LED occupies pins D18, D19 and D20.

WS:Beide Diagramme

6.3.1. Gesture, Proximity, and Color Detection Sensor ADPS-9960

The APDS-9960 device features advanced Gesture detection, Proximity detection, Digital Ambient Light Sense (ALS) and Color Sense (RGB). [Ard21b] Gesture detection

utilizes four directional photodiodes to sense reflected IR energy (sourced by the integrated LED) to convert physical motion information (i.e. velocity, direction and distance) to a digital information.

For the following Applications the sensor is in use:

- Gesture Detection
- Color Sense
- Ambient Light Sensing
- Proximity Sensing

6.3.2. Accelerometer, Gyroscope, and Magnetometre Sensor LSM9DS1

The LSM9DS1 is a system-in-package featuring a 3D digital linear acceleration sensor, a 3D digital angular rate sensor, and a 3D digital magnetic sensor. IMU's work by detecting rotational movements across the 3 axis known as Pitch, Roll and Yaw. To achieve the same, it depends on Accelerometer, Gyroscope and Magnetometer. The accelerometer gives the velocity at which the IMU module moves. The gyroscope measure the rotational movement rate on the IMU. Magnetometer measures the force of gravity acting on the IMU.

For the following Applications the IMU is in use:

- Indoor navigation
- Advanced gesture recognition
- Gaming and virtual reality input devices
- Display/map orientation and browsing
- Consumer electronics; Smartphones, tablets, fitness trackers for motion sensing and orientation.
- Compact transportation solutions like Segway.
- Sports Technology - helping athletes to know how they can improve their movements.

There are also certain disadvantages of using the IMU and points that need to be kept in mind while using an IMU sensor. Accumulated error or '*Drift*' is the main disadvantage of IMUs, present due to its constant measuring of changes and rounding off its calculated values off. When such a process happens for a prolonged period of time, it can lead to significant errors. The best way to avoid the *drift* factor is to use a good quality IMU Sensor and make sure the IMU sensor is calibrated. [Stmb]

WS:The explanation of drift is to small; citation

Calibration of an IMU

On research it was found that there are various methods to calibrate the sensors involved, the time period between each calibration is also not defined specifically, however it is advised that regular calibration is done especially when there are strange outputs noticed. Few methods of calibration are briefed below:

WS:What is calibration citations!

Low and High Limit Method

In this method the sensor is rotated in circles along each axis a few times. The midpoint is then found between the two extremes. If there is no offset, the midpoint is close to zero, but if there is a slight deviation from zero, this figure is the hard iron offset, which is the result of the distortion caused by the Earth's magnetic field. This method is mainly used to calibrate the Magnetometer [Mal15]

Magneto V1.2

In this method, the raw magnetometer data is pre-processed with axis specific gain correction to convert the raw output into nanoTesla:

```
Xm_nanoTesla = rawCompass.m.x*(100000.0/1100.0);
Gain X [LSB/Gauss] for selected input field range
Ym_nanoTesla = rawCompass.m.y*(100000.0/1100.0);
Zm_nanoTesla = rawCompass.m.z*(100000.0/980.0);
```

This converted data is saved into the file `Mag_raw.txt` that you open with the Magneto program. To start using this method, we first need to replace the (100000.0/1100.0) scaling factors with values that convert your specific sensors output into nanoTesla. Rather than simply finding an offset and scale factor for each axis, Magneto creates twelve different calibration values that correct for a whole set of errors: bias, hard iron, scale factor, soft iron and misalignment.

A side benefit of this is that it can be used to calibrate accelerometers as well. You might again need to pre-process your specific raw accelerometer output, taking into account the bit depth and G sensitivity, to convert the data into milliGalileo. Then enter a value of 1000 milliGalileo as the “norm” for the gravitational field. [Mal15]

VS:Here, we need more information because we want to use it:

- General infomration about imus
- angle to roation matrix with example!
- calibration
- drift
- ...
- specials for this imu
- description of the parameters for coding
- example code, see Code/- Nano33BLESense/IMU/Ardunio

6.3.3. Pressure Sensor LPS22HB

The LPS22HB is an ultra-compact piezoresistive absolute pressure sensor which functions as a digital output barometer.

For the following Applications the sensor is in use:

- Altimeters and barometers for portable devices
- Weather station equipment
- Sports watchs

A sensor element is installed as well as an IC interface that communicates via an I²C or SPI bus. The function is given in a temperature range from -40°C to +85°C. [STM17]

- Absolutdruckbereich: 260 bis 1260 hPa
- Versorgungsspannung: 1,7 bis 3,6 Volt
- 24-bit Druckdatenausgabe
- 16-bit Temperaturdatenausgabe

6.3.4. Relative Humidity and Temperature Sensor HTS221

The HTS221 is an ultra-compact sensor for relative humidity and temperature. It includes a sensing element and a mixed signal to provide the measurement information through digital serial interfaces.

For the following Applications the sensor is in use:

- Air conditioning, heating and ventilation
- Air humidifiers
- Refrigerators
- Smart home automation
- Industrial automation

Dieser Sensor kommuniziert über den I²C- und SPI-Bus. Dieser Sensor ist einsetzbar in einem Temperaturbereich von $-40^{\circ}C$ bis $+120^{\circ}C$. Zur Spannungsversorgung werden 1,7 bis 3,3 Volt benötigt. Eine Temperaturmessung erfolgt mit einer Genauigkeit von $\pm 5^{\circ}C$. [STM23]

- GND - Ground
- DRDY - Data ready output signal
- SCL/SPC - I2C serial clocl (SCL) & SPI serial port clock (SPC)
- VDD - Stromversorgung
- SDA/SDI/SDO - I²C serial Data (SDA) & 3 wire-SPI serial data input/output (SDI/SDO)
- SPI enable - I²C/SPI mode selection

6.3.5. Digital Microphone MP34DT05-A

The MP34DT05-A is an ultra-compact, low-power, omni directional, digital microphone built with a capacitive sensing element and an IC interface. The sensing element, capable of detecting acoustic waves, is manufactured using a specialized silicon micromachining process dedicated to producing audio sensors. The MP34DT05 is a low-distortion microphone with a signal-to-noise ratio of 64 dB. The sensitivity is -26 dBFS ± 3 dB. The Acoustic Overload Point (AOP) is 122.5 dB SPL.

The output signal of the microphone is a PDM signal. This signal is a binary signal modulated by Pulse Density Modulation (PDM) from the analogue signal. [Stmc]

For the following Applications the sensor is in use:

- Speech recognition
- Portable media player
- Mobile Terminal

The Arduino nano 33 BLE Sense has a built-in microphone that uses PDM (Pulse-Density Modulation) to convert sound into digital data. You can use the PDM library to access the microphone data and perform various tasks with it. Here is a simple example of using the builtin microphone for an Arduino nano 33 BLE Sense:

The code 6.1 will print the sound level of the microphone to the serial monitor every 100 milliseconds.

```

1000 // Include the PDM library
1001 #include <PDM.h>
1002
1003 // Buffer to store the microphone data
1004 short sampleBuffer[256];
1005
1006 // Variable to store the sound level
1007 int soundLevel = 0;
1008
1009 // Callback function for PDM data
1010 void onPDMdata() {
1011     // Read the PDM data
1012     int bytesAvailable = PDM.available();
1013     PDM.read(sampleBuffer, bytesAvailable);
1014
1015     // Calculate the sound level
1016     soundLevel = 0;
1017     for (int i = 0; i < bytesAvailable / 2; i++) {
1018         soundLevel += abs(sampleBuffer[i]);
1019     }
1020     soundLevel /= bytesAvailable / 2;
1021 }
1022
1023 void setup() {
1024     // Initialize serial communication
1025     Serial.begin(9600);
1026     while (!Serial);
1027
1028     // Initialize PDM with a sample rate of 16 kHz and 16-bit
1029     // resolution
1030     PDM.begin(1, 16000);
1031     PDM.onReceive(onPDMdata);
1032 }
1033
1034 void loop() {
1035     // Print the sound level to the serial monitor
1036     Serial.println(soundLevel);
1037     delay(100);
1038 }
```

Listing 6.1.: Simple example using of the builtin microphone of the Arduino Nano 33 BLE Sense

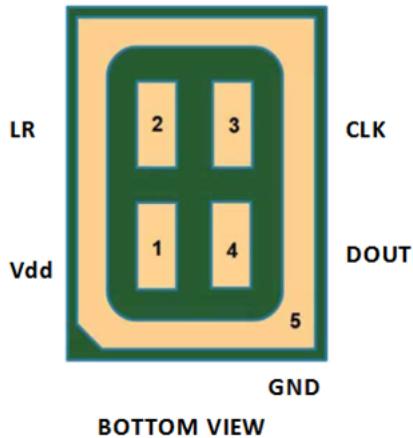


Table 1. Pin description

Pin #	Pin name	Function
1	Vdd	Power supply
2	LR	Left/Right channel selection
3	CLK	Synchronization input clock
4	DOUT	Left/Right PDM data output
5 (ground ring)	GND	Ground

Figure 6.6.: Circuit diagram microphone [Stmc]

You can modify this code to perform other tasks with the microphone data, such as controlling LEDs, playing sounds, or sending data to other devices. For more information and examples, you can check out the PDM library documentation or the Arduino nano 33 BLE Sense page.

WS:How to install the library PDM
description of the lib functions

6.3.6. Bluetooth Module nRF52840

The nRF52840 is an advanced, highly flexible single chip solution for today's increasingly demanding Ultra Low Power (ULP) wireless applications for connected devices on our person, connected living environments and the Internet of Things (IoT) at large. It is designed ready for the major feature advancements of Bluetooth 5 and takes advantage of Bluetooth 5's increased performance capabilities. [Ard21]

Applications

- Smart Home products
- Industrial mesh networks
- Smart city infrastructure
- Connected watches
- Advanced personal fitness devices
- Wearables with wireless payment
- Connected Health

6.4. Bluetooth Module INA B306

The module is a Bluetooth Low Energy (BLE). The connection with our smartphone is possible, to do this we have to use application “Nordic Semiconductors nRF Connect” They used 5.0 generation bluetooth. 6.4



Figure 6.7.: Connection with BLE and smartphone

We can use for example to share Arduino batterie on smartphone, 6.8 here you see an code example for how we can do this.

Figure 6.8.: Simple sketch to seen Arduino battery

```

1000 #include <ArduinoBLE.h>
1002 BLEService batteryService("1101");
1003 BLEUnsignedCharCharacteristic batteryLevelChar("2101"
1004 BLERead | BLENotify);
1006 void setup() {
1007   Serial.begin(9600);
1008   while (!Serial);
1010   pinMode(LED_BUILTIN, OUTPUT);
1011   if (!BLE.begin()) { //Search connection but they wasn't function
1012     Serial.println("Starting BLE failed!");
1013     while (1);
1014   }
1016   BLE.setLocalName("BatteryMonitor");
1017   BLE.setAdvertisedService(batteryService);
1018   batteryService.addCharacteristic(batteryLevelChar);
1019   BLE.addService(batteryService);
1020   BLE.advertise();
1021   Serial.println("Bluetooth device active, waiting for connections... ")
1022 ;
1024 void loop() {
1025   BLEDevice central = BLE.central();
1026   if (central) {
1027     Serial.print("Connected to central: ");
1028     Serial.println(central.address());
1029     digitalWrite(LED_BUILTIN, HIGH);
1030     while (central.connected()) {
1031       int battery = analogRead(A0);
1032       int batteryLevel = map(battery, 0, 1023, 0, 100);
1033       Serial.print("Battery Level is now: ");
1034       Serial.println(batteryLevel);
1035       batteryLevelChar.writeValue(batteryLevel);
1036       delay(200);
1037     }
1038     digitalWrite(LED_BUILTIN, LOW);
1039     Serial.print("Disconnected from central: ");
1040     Serial.println(central.address());
1041   }
1042 }
```

..../Code/Nano33BLESense/Bluetooth/bluetooth.ino

6.5. Arduino Nano 33 BLE Pin Configuration

Arduino Nano 33 BLE is an advanced version of Arduino Nano board that is based on a powerful processor the nRF52840. The figure 6.9 shows that the board has the following pin configuration. Pin Configuration

Digital pin The number of digital I/O pins are 14 which receive only two values HIGH or LOW. These pins can either be used as an input or output based on the requirement. When these pins receive 5V, they are in a HIGH state and when they receive 0V they are in a LOW state.

Analog pin Total 8 analog pins available on the board A0 – A7. These pins get any value as opposed to digital pins that only receive two values HIGH or LOW. These pins are used to measure the analog voltage ranging between 0 to 5V.

PWM pin All digital pins can be used as PWM pins. These pins generate analog results with digital means.

SPI pin The board supports serial peripheral interface (SPI) communication protocol. This protocol is employed to develop communication between a controller and other peripheral devices like shift registers and sensors. Two pins are used for SPI communication i.e. Master Input Slave Output (MISO) and Master Output Slave Input (MOSI) are used for SPI communication. These pins are used to send or receive data by the controller.

I2C pin The board carries the I2C communication protocol which is a two-wire protocol. It comes with two pins SDA and SCL.

UART pin The board features a UART communication protocol that is used for serial communication and carries two pins Rx and Tx. The Rx is a receiving pin used to receive the serial data while Tx is a transmission pin used to transmit the serial data.

External Interrupts pin All digital pins can be used as external interrupts. This feature is used in case of emergency to interrupt the main running program with the inclusion of important instructions at that point.

LED at Pin 13 and AREF pin There is an LED connected to pin 13 of the board. And AREF is a pin used as a reference voltage for the input voltage.

6.6. Was fehlt

- Beschreibung der Sensoren
 - Hintergrundwissen
 - Beschreibung
 - Eigenschaften
 - Kalibrierung
- Beschreibung der Bibliothek
 - Beschreibung
 - Installation
 - Funktionen

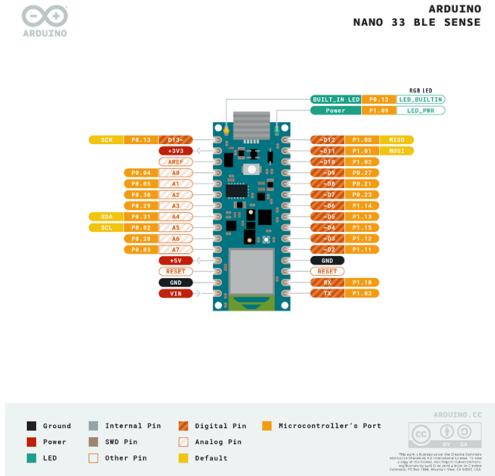


Figure 6.9.: Arduino Nano 33 BLE Pin Configuration

- Laufzeit, Speicherplatz, ...
- Software-Dokumentation
- Verwendung von Werkzeugen, z.B. Doxygen
 - Beschreibung, inklusive im Quellcode; z.B. Header
 - Handbuch
 - Ablaufdiagramm
 - ...
- Interpretation der Ergebnisse
- Literatur, Bilder
- ...

7. Reset Button on the Arduino Nano 33 BLE Sense

The reset button on the Arduino Nano 33 BLE Sense plays a crucial role in managing the board's operation. It allows for restarting the board, entering Bootloader Mode, and addressing issues related to the program or functionality. The following section outlines its functions, appropriate use cases, and technical specifications.

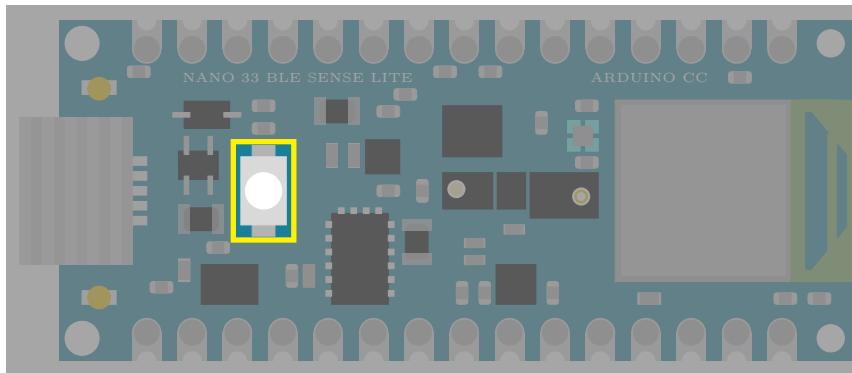


Figure 7.1.: Arduino Nano 33 BLE Sense's Reset Button

7.1. Purpose and Functionality

The reset button on the Arduino Nano 33 BLE Sense is positioned on the top side of the board, close to the USB connector. This small tactile push button communicates with the nRF52840 microcontroller, performing specific actions depending on the timing and sequence of the presses.

7.1.1. Functions of the Reset Button

- System Reset: A single press of the reset button triggers a system reset. This action momentarily interrupts the power to the microcontroller, restarting the board and reinitializing the uploaded program.
- Bootloader Mode: The reset button also allows entry into Bootloader Mode, which is essential for uploading new sketches or recovering the board when it becomes unresponsive due to a faulty program.
 - Activation: To activate Bootloader Mode, press the reset button twice in quick succession (within approximately 1 second). The double-press sequence triggers the bootloader, which prepares the board to receive new firmware or sketches via the USB interface.
 - Indication: When Bootloader Mode is active, the built-in LED blinks quickly, indicating that the board is ready for a firmware upload.
 - Purpose: Bootloader Mode is particularly helpful for uploading sketches when the Arduino IDE cannot recognize the board and for recovering from unresponsive or faulty programs that disrupt normal functionality.

•

7.2. Timing and Sequence

The reset button's behavior depends on the timing and sequence of presses:

- Single Press: Executes a system reset, restarting the microcontroller and the loaded sketch.
- Double Press: Activates Bootloader Mode, readying the board for new firmware uploads.
 - The timing of the double-press must be precise (quickly within 1 second); otherwise, the board will perform only a standard reset.

7.3. Use Cases

The reset button provides critical functionality in several scenarios:

1. Restarting the Board: During development, a single press allows developers to restart the board and observe the program from its initial state.
2. Uploading Sketches: When the Arduino IDE fails to detect the board or cannot upload a sketch, entering Bootloader Mode manually resolves the issue.
3. Recovering from Errors: If a faulty sketch causes the board to freeze or become unresponsive, Bootloader Mode enables users to bypass the issue and upload a new program.

7.4. Conclusion

The reset button on the Arduino Nano 33 BLE Sense is a vital feature for developers, providing tools to restart the board, enter Bootloader Mode, and recover from programming errors. Its dual functionality—system reset and Bootloader Mode activation—ensures flexibility and reliability during development and debugging. The requirement for a precise double-press to activate Bootloader Mode enhances user control, making it a robust tool for managing the board.

8. Built-inLED

Some Arduino boards have a LED on board which can be used for the application. [Ard24a; Ard23a; Ard23b]

8.1. General Information

LEDs are used in the applications. Depending on the action performed by a user or the sketch, corresponding feedback can be provided. This can take the form of the LED switching on, switching off or flashing. [Kur21]

8.2. Built-in LED

The built-in LED of the Arduino Nano 33 BLE Sense is a single orange LED that is connected to pin 13 on the board. The built-in LED is active-high, which means that setting the pin to `HIGH` will turn the LED on, and setting the pin to `LOW` will turn the LED off. The built-in LED can be used for various purposes, such as indicating the status of the board, displaying sensor data, creating visual effects, or debugging. [Ard23a; Ard23b; Arda]

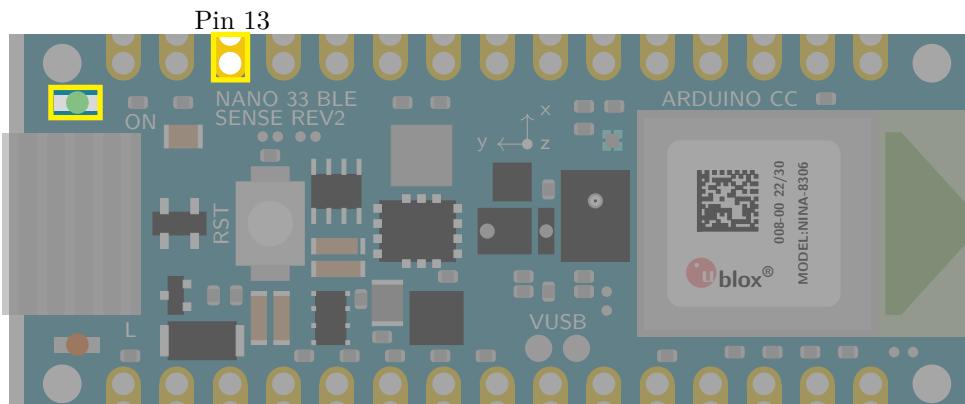


Figure 8.1.: Arduino Nano 33 BLE Sense's built-in LED with Pin 13

8.3. Specification

8.3.1. Pin Assignment

The built-in LED is an orange LED and connected to pin 13. The brightness can not be controlled. [Ard23a; Ard23b]

Built-in LED: `LED_BUILTIN = 13u`

Built-in LED is active-high and connected to pin 13.

The built-in LED can be controlled programmatically by setting the pin to `HIGH` or `LOW`.

The pin 13 must be defined as an output in the function `setup` by setting `pinMode(13, OUTPUT)`, otherwise the LED cannot be switched on.

The pin 13 can also be used otherwise. Then the LED is not in use.

8.3.2. Power Consumption

The power consumption has to be considered. It is the same value as for the power LED, see section 9.3.2.

8.4. Simple Code

8.4.1. Simple Code for the Built-in LED

For using the built-in LED, a simple library is introduced here.

In the header file `BuiltinLED.h`, see code ??, are the declaration of the variables and the functions. A variable is connected to pin 13. The pin 13 is defined as an output in the function `BuiltinLEDinit`. There are 2 functions:

1. `BuiltinLEDinit`: This function initializes the digital pin 13 of the built-in LED as an output.
2. `BuiltinLED`: This function switches the built-in LED on or off.

Listing 8.1.: Header file without comments for using the Built-in LED

```

1000 #ifndef LEDBuiltin_h
1001 #define LEDBuiltin_h
1002
1003 /**
1004 */
1005 /**
1006 */
1007 #endif
1008
1009 ..../Code/Nano33BLESense/LEDs/LEDBuiltin.h

```

In the file `BuiltinLED.cpp`, see code 8.2, the functions are implemented.

Listing 8.2.: Code file without comments for using the Built-in LED

```

1000 #include <LEDGeneral.h>
1001 #include <LEDBuiltin.h>
1002 * @return 0 – success
1003 * @return 1 – error
1004 */
1005 int LEDBuiltinsetup()
1006 {
1007     // initialize digital pin LED_BUILTIN as an output.
1008     LEDSetup(LED_BUILTINLED);
1009 *
1010 * @return 0 – success
1011 * @return 1 – error
1012 */
1013 int LEDBuiltin(bool On)
1014 {
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3020
3021
3022
3023
3024
3025
3026
3027
3028
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3039
30
```

8.4.2. Simple Code for Testing the Built-in LED

In the sketch 19.1, a variable is connected to pin 13. The pin is defined as an output in the function `setup`. In the function `loop`, the LED is switched on for 1 second and switched off for 1 second so that the LED flashes accordingly.

Listing 8.3.: Simple sketch without comments to control the built-in LED

```

1000 * Turns the built-in LED on for one second , then off for one second ,
  repeatedly .
*
1002 * The LED is switched on for 1 second and switched off
*
1004 * Initialization of the pin LED_BUILTIN as output
*
1006 * @param ——
*
1008 * swichting the led on / off for 1sec .
*
1010 * @param ——
*
1012 * @return void
*/
1014 void loop() {
    // Turn the LED on
    LEDBuiltin(false);
    // Wait for one second

```

..../Code/Nano33BLESense/LEDs/examples/TestLEDBuiltin/TestLEDBuiltin.ino

This is just a simple example. The variable `LED_BUILTIN` is already defined, so the assignment is not necessary. The command `delay` should be avoided in an Arduino sketch. Instead, variables of the type `elapsedMillis` should be used.

8.5. Tests

8.5.1. Simple Function Test

The simplest test is the flashing of the LED at 2 Hz, see sketch 8.4.

Listing 8.4.: Simple sketch to test the built-in LED

```

1000 * Turns the built-in LED on for one second , then off for one second ,
  repeatedly .
*
1002 * The LED is switched on for 1 second and switched off
*
1004 * Initialization of the pin LED_BUILTIN as output
*
1006 * @param ——
*
1008 * swichting the led on / off for 1sec .
*
1010 * @param ——
*
1012 * @return void
*/
1014 void loop() {
    // Turn the LED on
    LEDBuiltin(false);
    // Wait for one second

```

..../Code/Nano33BLESense/LEDs/examples/TestLEDBuiltin/TestLEDBuiltin.ino

8.5.2. Test all Functions

As the built-in does not allow any special manipulations, all functions are covered with the sketch 8.4.

8.6. Simple Application

In many applications, it is necessary to save power so that the LED is not switched on continuously. Nevertheless, feedback is required as to whether the system, e.g. a fire detector, is switched on and functional. In this case, the LED is switched on for 1 second at a defined time interval, in this case 30 seconds. This is implemented in the example 8.5.

Listing 8.5.: A simple watch dog: A simple watchdog: the built-in LED is switched on for 1 second every 30 seconds.

```

1000 #include <../LEDs/BuiltinLED.h>
1001 #include <../LEDs/LED.h>
1002 #include <../LEDs/SignsOfLife.h>

1004
1005 #define CycleTimeOn 1000 /*< Duty cycle [ms] */
1006 #define CycleTimeOff 29000 /*< Switch-off time [ms] */
1007 void setup() {
1008     // Initialize the function SignsOfLife
1009     SignsOfLifeInit(LED_BUILTIN, CycleTimeOn, CycleTimeOff)
1010
1011     // Application
1012     // ...
1013 }
1014 void loop() {
1015     // Switch builtin LED on/off
1016     SignsOfLife();
1017
1018     // Application
1019     // ...
1020 }
```

..../Code/Nano33BLESense/Test/TestLEDBuiltinApplication.ino

8.7. Further Readings

- Babu, Neelakandan Ramesh and Chenchireddy, Kalagotla and Reddy, V. Harsha Vardhan and Samhitha, Dr. and Apparao, P. and Kalyan, Ch. Pavan: *Case Study on Ni-MH Battery*. 2023 2nd International Conference on Applied Artificial Intelligence and Computing (ICAAIC), 2023. [Bab+23]
- Scherer, Thomas: *Elektor special: Stromversorgung und Batterien*. Elektor Special. 2022. [Sch22]

9. Power LED

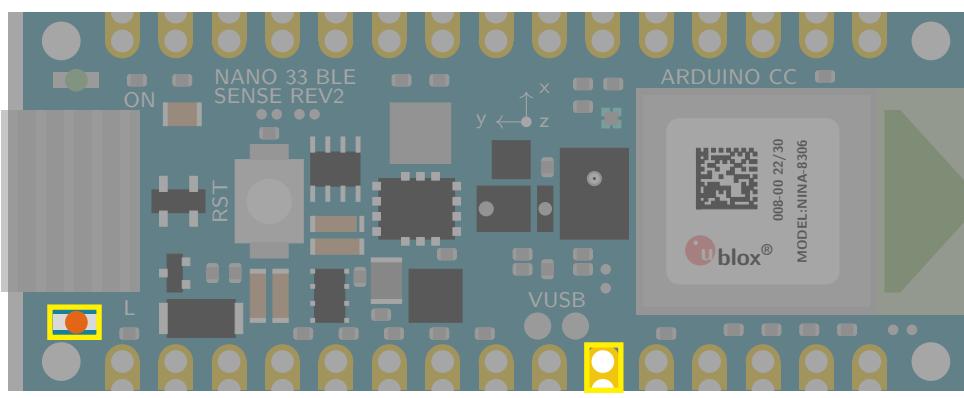
Arduino boards have a power LED on board. Normally, the power LED indicates the status of the board.

9.1. General Information

The power LED on an Arduino board is a small, usually red or green, Light Emitting Diode (LED) that indicates whether the board is receiving power. It is typically located near the USB connector on the board. When the board is powered on, the LED will illuminate. The specific color and behavior of the power LED may vary depending on the Arduino board model. For example, on the Arduino Uno, the power LED is red and illuminates steadily when the board is powered on. On the Arduino Nano, the power LED is green and blinks slowly when the board is powered on. The power LED is a helpful visual indicator that can be used to troubleshoot power supply issues. If the power LED is not illuminated, it means that the board is not receiving power. This could be due to a number of reasons, such as a loose USB connection, a damaged power supply, or a problem with the board itself. By checking the power LED, you can quickly identify and resolve power supply problems with your Arduino board. [Ard24a; Ard23a; Ard23b; Arda; Kur21]

9.2. Power LED

While labeled as a power LED, the single green LED on the Nano 33 BLE Sense isn't solely for power indication. It has multi-functionality depending on board state and user code interaction. During typical operation, it lights steadily green when powered, similar to other Arduino models. However, it flashes rapidly during serial communication and bootloader mode. Notably, when user code enters deep sleep mode, the LED turns off entirely for power saving. This includes power status, communication activity, and sleep mode activation. Understanding these LED behaviors can aid in troubleshooting and code debugging. The green power LED's primary function remains indicating power and basic board states. [Ard24a; Ard23a; Ard23b; Arda]



Pin 25

Figure 9.1.: Arduino Nano 33 BLE Sense's Power LED with Pin 25

9.3. Specification

9.3.1. Pin Assignment

The power LED is a green LED and connected to pin 25. The brightness can be controlled via Pulse with Modulation (PWM). [Ard23a; Ard23b]

Power LED: `LED_PWR = 25u`

Power LED is active-high and connected to pin 25.

The power LED can also be controlled programmatically by setting the pin to `HIGH` or `LOW`.

The pin must be defined as an output in the function `setup` by setting `pinMode (LED_PWR, OUTPUT)`, otherwise the LED cannot be switched on.

9.3.2. Power Consumption

The power consumption has to be considered. There, the power and current in a simple circuit using a resistor and an LED has to be calculated.

The led onboard works in a circuit with an 200 Ohm resistor, a 5V power source from an Arduino, and an LED with 2V across it. To find out how much power is being used by the LED and the resistor, the current must be calculated in the first step. Using Ohm's Law $V = I \cdot R$ we can find the current flowing through the resistor. Since 3V is across the resistor, we can plug in the values:

$$\frac{3V}{200\Omega} = 0.015A = 15mA$$

This means that 15mA of current is flowing through the resistor and the LED. In the next step, we calculate the power. Now that we know the current, we can calculate the power dissipated in the LED and the resistor.

- For the LED: $2V \times 15mA = 30mW$
- For the resistor: $3V \times 15mA = 45mW$
- Total Power:

To find the total power coming out of the Arduino, we multiply the voltage by the current:

$$5V \times 15mA = 75mW$$

Notice that the total power (75mW) is equal to the sum of the power dissipated in the LED (30mW) and the resistor (45mW). This makes sense, since all the power coming out of the Arduino is being used by either the LED or the resistor.

The pin 25 can also be used otherwise. Then the LED is just switched on if the board is connected to a power source.

9.4. Simple Code

9.4.1. Simple Code for LEDs

Using of LEDs is a standard problem for Arduino's programmer. Therefore, a simple library is introduced here.

In the header file `LED.h`, see code 9.1, are the declaration of the functions. There are 3 functions:

1. `LEDinit`: This function initializes a digital pin of the LED as an output.
2. `LED`: This function switches the LED on or off.
3. `LEDBrightness`: This function sets the LED's brightness.

Listing 9.1.: Header file without comments for using a LED

```

1000 #define LED_h
1002
1004 /**
1005 * @brief initialize digital pin of the LED as an output.
1006 * @brief function for switching on or off the LED
1007 */
1008 /**
1009 * @brief setting the brightness
1010 */
1011 #endif

```

..../Code/Nano33BLESense/LEDs/LEDGeneral.h

In the file `LED.cpp`, see code 9.2, the functions are implemented.

Listing 9.2.: Code file without comments for using a LED

```

1000 #include "LEDGeneral.h"
1001 #include <Arduino.h>
1002 *
1003 * @return 0 – success
1004 * @return 1 – error
1005 */
1006 int LEDSetup(int PinNo) {
1007     // initialize digital pin LED_BUILTIN as an output.
1008     pinMode(PinNo, OUTPUT);
1009 *
1010 * @return 0 – success
1011 * @return 1 – error
1012 */
1013 int LED(int PinNo, bool On)
1014 {
1015     if (On)
1016     {
1017         digitalWrite(PinNo, HIGH);    // turn the LED on (HIGH is the voltage
1018         level)
1019     } else
1020     {
1021         digitalWrite(PinNo, LOW);   // turn the LED off by making the
1022         voltage LOW
1023     }
1024 *
1025 * @return 0 – success
1026 * @return 1 – error
1027 */
1028 int LEDBusiness(int PinNo, int setBrightness)
1029 {
1030     int b;
1031
1032     if (setBrightness <= 0) {
1033         b = 0;
1034     } else
1035     if (setBrightness >= 255) {
1036         b = 255;
1037     } else
1038     {
1039         b = setBrightness;
1040     }

```

```
1040     analogWrite(PinNo, b);
..../Code/Nano33BLESense/LEDs/LEDGeneral.cpp
```

9.4.2. Simple Code for the Power LED

For using the power LED, a simple library is introduced here.

In the header file `PowerLED.h`, see code 9.3, are the declaration of the variables and the functions. A variable is connected to pin 25. The pin 25 is defined as an output in the function `PowerLEDinit`. There are 2 functions:

1. `PowerLEDinit`: This function initializes the digital pin 25 of the power LED as an output.
2. `PowerLED`: This function switches the power LED on or off.

Listing 9.3.: Header file with doxygen comments for using the Power LED

```
1000 #ifndef LEDPower_h
# define LEDPower_h
1002
1004 /**
1006 /**
1008 #endif
..../Code/Nano33BLESense/Nano33BLESenseLED/LEDPower.h
```

In the file `PowerLED.cpp`, see code 9.4, the functions are implemented.

Listing 9.4.: Code file with doxygen comments for using the Power LED

```
1000 #include <LEDGeneral.h>
# include <LEDPower.h>
1002
1004 #define LED_PWR 25 /*< Define the pin for the power LED */
* @return 1 - error
*/
1006 int LEDPowersetup()
{
    // initialize digital pin LED_BUILTIN as an output.
    LEDSetup(LED_PWR);
* @return 0 - success
* @return 1 - error
*/
1012 int LEDPower(bool On)
{
    LED(LED_PWR, On); // turn the LED on = true or off = false
..../Code/Nano33BLESense/Nano33BLESenseLED/LEDPower.cpp
```

9.4.3. Simple Code for Testing the Power LED

In the sketch 9.5, the power LED is tested. First, the function `PowerLEDinit` is called in the function `setup`. In the function `loop`, the power LED is switched on for 1 second and switched off for 1 second so that the power LED flashes accordingly.

Listing 9.5.: Simple sketch to control the power LED

```

1000 /*
1001 */
1002 * @brief Setup function runs once when the sketch starts.
1003 *
1004 * Initializes the power LED for use in the main loop. The pin for the
1005 * LED is configured here.
1006 */
1007 void setup() {
1008     LEDPower.turnOff(); //< Turns off the power LED
1009     delay(500); //< Wait for 500 milliseconds (LED is OFF)
1010 }

```

..../Code/Nano33BLESense/LEDs/examples/TestLEDPower/TestLEDPower.ino

This is just a simple example. The variable `LED_PWR` is already defined, so the assignment is not necessary. The command `delay` should be avoided in an Arduino sketch. Instead, variables of the type `elapsedMillis` should be used. [Ard24b]

9.4.4. Test all Functions

The brightness of the power LED can be controlled. This is demonstrated in the example sketch 9.6.

Using the pulse width modulation, the brightness is gradually increased to the maximum value and then gradually reduced to 0 again.

Listing 9.6.: Simple sketch to check the battery state using the power LED

```

1000 #include <../LEDs/PowerLED.h>
1001 #include <../LEDs/LED.h>
1002
1003 int Brightness = 0; /*< Define the initial brightness values (0-255) */
1004 int redStep = 5; /*< Define the increment/decrement value */
1005 void setup() {
1006     // Initialize the pin as an output
1007     PowerLEDinit();
1008 }
1009 void loop() {
1010     // Write the PWM values to the LED pin
1011     LEDBrightness(LED_PWR, redBrightness);
1012
1013     // Update the brightness values
1014     Brightness += Step;
1015
1016     // Check if the brightness values are out of range and reverse the
1017     // direction
1018     if (Brightness <= 0 || Brightness >= 255) {
1019         Step = -Step;
1020     }
1021
1022     // Wait for 10 milliseconds
1023     delay(10);
1024 }

```

..../Code/Nano33BLESense/Test/TestLEDPowerBrightness.ino

9.5. Simple Application

There are different situations where it might be useful to program the power LED of the Arduino Nano. For example, you could use it to:

- Indicate the status of the board, such as whether it is connected to a power source, a computer, or a sensor.
- Display the battery level of the board, by changing the brightness or color of the power LED.
- Create a visual alarm or notification, by making the power LED blink or flash in a certain pattern.

The next sketch is a frame for applications with a sign of life. The power LED is switched on for 1 sec and off for 29 sec. The file `SignsOfLife.h`, see 9.8, contains the declarations and the file `SignsOfLife.cpp` contains the implementations.

Listing 9.7.: Simple code for signs of life

```

1000 #ifndef LEDSignsOfLife_h
1001 #define LEDSignsOfLife_h
1002
1004 /**
1005 * @brief initialize digital pin of the LED as an output.
1006 *
1007 * call the function once, in the function setup, when you press reset
1008 * or power the board
1009 *
1010 * Initialization of the pin as an output
1011 *
1012 * The program doesn't check if the parameter PinNo is reasonable.
1013 *
1014 * @param PinNo - number of pin for the LED
1015 * @return false - LED's actual state is off
1016 */
1017 extern bool SignsOfLife();

```

..../Code/Nano33BLESense/LEDs/LEDSignsOfLife.h

The library contains one function for initialization and one function `SignsOfLife`, which controls the LED.

Listing 9.8.: Simple code for signs of life

```

1000 #include "LEDGeneral.h"
1001 #include "LEDSignsOfLife.h"
1002 #include <Arduino.h>
1003
1004 /**
1005 * @brief initialize digital pin of the LED as an output.
1006 *
1007 * call the function once, in the function setup, when you press reset
1008 * or power the board
1009 *
1010 * Initialization of the pin as an output
1011 *
1012 * The program doesn't check if the parameter PinNo is reasonable.
1013 *
1014 * @param PinNo - number of pin for the LED
1015 * @param CycleTimeOn - duration of the LED is on
1016 * @param CycleTimeOff - duration of the LED is off
1017 *
1018 * @return true - LED's actual state is on
1019 * @return false - LED's actual state is off
1020 */
1021 bool SignsOfLifeSetup( int PinNo, int CycleTimeOn, int CycleTimeOff)
1022 {
1023     LEDSetup(PinNo);

```

```

1024     SignsOfLifePinNo = PinNo;
1026     LED( SignsOfLifePinNo , SignsOfLifeState );
1028     return SignsOfLifeState;
1029 }
1030 /**
1031 * @brief function for blinking the LED
1032 *
1033 *
1034 * The function checks the duration of the state. If the state has to
1035 * change, then the function changes the state.
1036 *
1037 *
1038 * @return true - LED's actual state is on
1039 * @return false - LED's actual state is off
1040 */
1041 bool SignsOfLife()
1042 {
1043     // Check if CycleTimeOff have passed since the last LED turn on

```

..../Code/Nano33BLESense/LEDs/LEDSignsOfLife.cpp

A simple application is to check the condition of the battery. The sktech 9.9 demonstrates, if the voltage drops too low, the power LED flashes, see [Sch22] for more information.

Listing 9.9.: Simple sketch to check the battery state using the power LED

```

1000 #include <Arduino.h>
1001 #include <../LEDs/PowerLED.h>
1002 #include <../LEDs/LED.h>
1003
1004 const int BATTERY_PIN = A0; /*< Battery measurement pin */
1005
1006 const float REFERENCE_VOLTAGE = 3.3; /*< Reference voltage for 3.3V (
1007     adjust if using external power) */
1008 void setup() {
1009
1010     SignsOfLifeInit(LED_PWR, 500, 500);
1011     PowerLED(true);
1012 }
1013 int BatteryState(bool SendMessages)
1014 {
1015     int ret;
1016     // Read battery voltage from analog pin
1017     float rawVoltage = analogRead(BATTERY_PIN) * (REFERENCE_VOLTAGE /
1018         1023.0);
1019
1020     // Calculate percentage based on reference voltage (adjust based on
1021     // battery specs)
1022     float batteryPercentage = (rawVoltage / 4.2) * 100.0;
1023
1024     if (SendMessages)
1025     {
1026         // Print battery voltage and percentage (for debugging)
1027         Serial.print("Battery voltage: ");
1028         Serial.print(rawVoltage);
1029         Serial.println(" V");
1030         Serial.print("Battery percentage: ");
1031         Serial.print(batteryPercentage);
1032         Serial.println("%");
1033     }
1034
1035     // Optional: blink LED based on battery level (adjust thresholds)

```

```

1034   if (LED_PWR >= 0) {
1035     if (batteryPercentage < 20) {
1036       digitalWrite(LED_PWR, HIGH);
1037       delay(500);
1038       digitalWrite(LED_PWR, LOW);
1039       delay(500);
1040       ret = 1;
1041     } else {
1042       digitalWrite(LED_PWR, HIGH);
1043       ret = 0;
1044     }
1045   } else {
1046   {
1047     ret = 0;
1048   }
1049   return ret;
1050 }
1051
1052 void loop() {
1053
1054   BatteryState(false);
1055
1056   // Delay between measurements
1057   delay(5000);
1058 }
```

..../Code/Nano33BLESense/Test/TestLEDPowerBattery.ino

9.6. Further Readings

- Kurniawan, Agus: *IoT Projects with Arduino Nano BLE Sense: Step-By-Step Projects for Beginners*. Apress, 2021. [Kur21]
- Kühnel, Claus: *Arduino - Das umfassende Handbuch*. Rheinwerk Verlag GmbH, 2024. [Küh24]
- Smythe, Richard J.: *Advanced Arduino Techniques in Science - Refine Your Skills and Projects with PCs or Python-Tkinter*. Apress, 2021. [Smy21a]
- Smythe, Richard J.: *Arduino in Science - Collecting, Displaying, and Manipulation Sensor Data*. Apress, 2021. [Smy21b]

10. Built-in RGB-LED

Some Arduino boards have a RGB-LED on board which can be used for the application. ARG-LED is at least three LEDs in one.

10.1. General Information

An RGB LED is a type of LED that can emit different colors of light. RGB stands for red, green, and blue, which are the three primary colors of light. An RGB LED consists of three individual LEDs inside a single package, each with its own color and pin. By varying the brightness of each LED using PWM signals, an RGB LED can produce a wide range of colors by mixing the primary colors. An RGB LED is usually active-low, which means that setting the pin to LOW will turn the LED on, and setting the pin to HIGH will turn the LED off. [Ber18; HEG21; KKV14]

10.2. Specific Sensor

The onboard LED of the Arduino Nano 33 BLE Sense is a RGB-LED that consists of three individual LEDs: red, green, and blue. Each LED is connected to a different pin on the board:

- Pin 22 for red,
- Pin 23 for green, and
- Pin 24 for blue.

The RGB-LED can produce different colors by varying the brightness of each LED using PWM signals. The RGB-LED is active-low, which means that setting the pin to `LOW` will turn the LED on, and setting the pin to `HIGH` will turn the LED off. This is different from the power LED and the built-in LED, which are both active-high. [Ard24a; Ard23a; Ard23b]

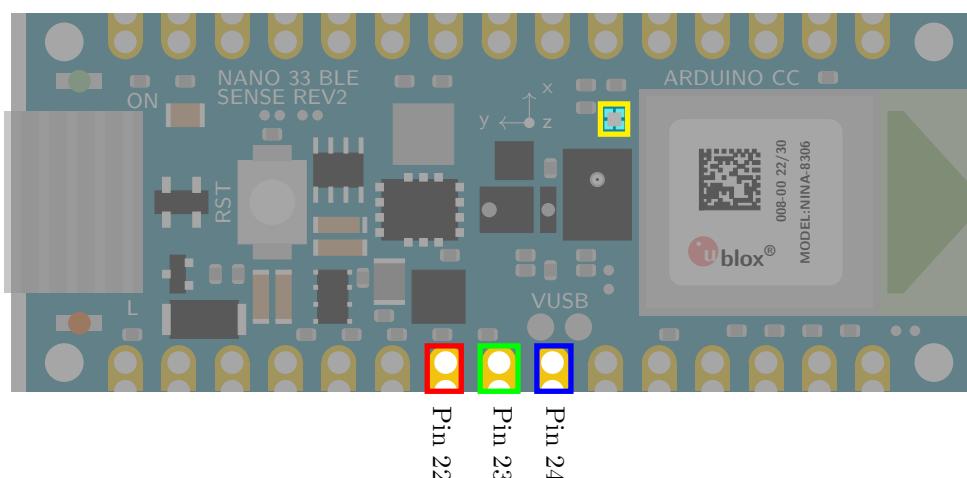


Figure 10.1.: Arduino Nano 33 BLE Sense's RGB LED with Pin 22, Pin 23, and Pin 24

The brightness of the LED varies between 5000-6000 Millicandela (mcd) at a current of 20 mA. The color temperature is controllable and can be set in a range of 5000-6000 Kelvin. The LED should be stored and used between -40°C and +85°C.

10.3. Specification

10.3.1. Pin Assignment

The RGB LED occupies pins on the board internally. These pins are defined via variable names `LEDR`, `LEDG`, and `LEDB`. [Ard23a; Ard23b]
This must be observed when using the pins.

Red LED: `LEDR` = Pin 22

Green LED: `LEDG` = Pin 23

Blue LED: `LEDB` = Pin 24

RGB LED is active-low and connected to pin 22, 23, and 24.

The RGB LED can be controlled programmatically by setting the pin to `LOW` or `HIGH`. The pins 22, 23, and 24 must be defined as an output in the function `setup` by setting `pinMode (22, OUTPUT)`, `pinMode (23, OUTPUT)` and `pinMode (22, OUTPUT)`, otherwise the LED cannot be switched on.

The pins 22, 23, 24 can also be used otherwise. Then the LED is not in use.

10.3.2. Power Consumption

The power consumption has to be considered. It is the same value as for the power LED, see section 9.3.2. Here, a RGB-LED has 3 LEDs inside, therefore the poswer consumption must be consiedered for each color.

10.4. Simple Code

In the header file `RGBLED.h`, see code 10.1, are the declaration of the variables and the functions. Variables are connected to pin 22, pin 23, and pin 24. The pins are defined as output in the function `RGBLEDinit`. There are the following functions:

1. `RGBLEDinit`: This funtion initializes the digital pin 22, pin 23, and pin 24 of the RGB-LED as output.
2. `RGBLED_Red`: This funtion switches the red part of the RGB-LED on or off.
3. `RGBLED_Green`: This funtion switches the green part of the RGB-LED on or off.
4. `RGBLED_Blue`: This funtion switches the blue part of the RGB-LED on or off.
5. `RGBLED_Color`: This functions combines the colors.

Listing 10.1.: Header file without comments for using the RGB-LED

```
1000 #ifndef LEDRGB_h
1001 #define LEDRGB_h
1002
1003
1004 /**
1005 */
1006 /**
1007 */
1008 /**
1009 */
1010 /**
1011 */
```

In the file `RGBLED.cpp`, see code 10.2, the functions are implemented.

Listing 10.2.: Code file without comments for using the RGB-LED

```
1000 #include <LEDGeneral.h>
1001 #include <LEDRGB.h>
1002 #include <Arduino.h>
1003
1004 /*
1005 *   @param —
1006 *   @return 0 — success
1007 *   @return 1 — error
1008 */
1009 int LEDRGBsetup()
1010 {
1011 /*
1012 *   swichting the red led on / off
1013 *
1014 *   @param On — true: switch on, false: switch off
1015 *
1016 *   @return 0 — success
```

..../..//Code/Nano33BLESense/LEDs/LEDRGB.cpp

10.5. Tests

10.5.1. Simple Function Test

The simplest test is the flashing of the LEDs at 2 Hz, see sketch ??.

Listing 10.3.: Simple sketch to test the RGB LED

```
1000 /**
* @file TestLEDRGB.ino
1002 *
* @brief Simple program for testing the RGB-LED
1004 *
*
1006 * Turns the RGB-LED on for one second, then off for one second,
* repeatedly.
*
1008 * The LED is switched on for 1 second and switched off
* for 1 second so that the LED flashes accordingly.
1010 *
* On the Arduino Nano 33 BLE Sense, it is attached to digital pin 22,
* 23, 24
1012 *
*/
1014
1016 #include <../LEDs/RGBLED.h>
#include <../LEDs/LED.h>
```

```

1018 /**
1020 * @brief the setup function runs once when you press reset or power the
1021 * board
1022 *
1023 * Standard function of Arduino sketches
1024 *
1025 * Initialization of the pin 22, pin 23, and 24 as output
1026 *
1027 * @param —
1028 *
1029 * @return void
1030 */
1031 void setup() {
1032     // Initialize the pin as an output
1033     RGBLEDinit();
1034 }
1035
1036 /**
1037 * @brief the setup function runs once when you press reset or power the
1038 * board
1039 *
1040 * Standard function of Arduino sketches
1041 *
1042 * In each loop , the red part is switched on for 1 sec ,
1043 * then the green part is switched on for 1 sec , and
1044 * at last the blue part is switched on.
1045 *
1046 * @param —
1047 *
1048 * @return void
1049 */
1050 void loop() {
1051     // Turn the red LED on
1052     RGBLED_Red(SET_ON);
1053     // Wait for one second
1054     delay(1000);
1055     // Turn the LED off
1056     RGBLED_Red(SET_OFF);
1057     // Turn the green LED on
1058     RGBLED_Green(SET_ON);
1059     // Wait for one second
1060     delay(1000);
1061     // Turn the LED off
1062     RGBLED_Green(SET_OFF);
1063     // Turn the blue LED on
1064     RGBLED_Blue(SET_ON);
1065     // Wait for one second
1066     delay(1000);
1067     // Turn the LED off
1068     RGBLED_Blue(SET_OFF);
1069     // Wait for one second
1070     delay(1000);
1071 }

```

..../Code/Nano33BLESense/Test/TestLEDRGB.ino

10.5.2. Test all Functions

Different Colors

Colors

A RGB LED is a device that can emit light of different colors by mixing the primary colors of red, green, and blue. The color of the light depends on the relative brightness

of each LED, which can be controlled by PWM signals. By varying the brightness of each LED, the RGB LED can produce a wide range of colors, such as yellow, cyan, magenta, white, and more. Some examples of the colors and their corresponding brightness values are:

Red: red = 255, green = 0, blue = 0

Green: red = 0, green = 255, blue = 0

Blue: red = 0, green = 0, blue = 255

Yellow: red = 255, green = 255, blue = 0

Cyan: red = 0, green = 255, blue = 255

Magenta: red = 255, green = 0, blue = 255

White: red = 255, green = 255, blue = 255

Black: red = 0, green = 0, blue = 0

The RGB LED can also create intermediate colors by using different brightness values for each LED. For example, to create

- **orange**, one can use red = 255, green = 127, blue = 0.
- To create **pink**, one can use red = 255, green = 192, blue = 203.
- To create **purple**, one can use red = 128, green = 0, blue = 128.

The sketch 10.4 tests different colors of the LED. The color of the LED changes every 1 second.

Listing 10.4.: value between 0 and 255 to write to the RGB LED

```

1000 /**
1001 * @file TestLEDRGBColors.ino
1002 *
1003 * @brief Simple program for testing the RGB-LED
1004 *
1005 *
1006 * Turns the RGB-LED on for one second, then off for one second,
1007 * repeatedly.
1008 *
1009 * The LED is switched on for 1 second and switched off
1010 * for 1 second so that the LED flashes accordingly.
1011 *
1012 * On the Arduino Nano 33 BLE Sense, it is attached to digital pin 22,
1013 * 23, 24
1014 */
1015
1016 #include <../LEDs/RGBLED.h>
1017 #include <../LEDs/LED.h>
1018
1019 /**
1020 * @brief the setup function runs once when you press reset or power the
1021 * board
1022 *
1023 * Standard function of Arduino sketches
1024 *
1025 * Initialization of the pin 22, pin 23, and 24 as output
1026 */

```

```

1026 *  @param ——
1027 *
1028 *  @return void
1029 */
1030 void setup() {
1031     // Set the LED pins as outputs
1032     RGBLEDinit();
1033 }
1034
1036 /**
1037 *  @brief the setup function runs once when you press reset or power the
1038 *         board
1039 *
1040 * Standard function of Arduino sketches
1041 *
1042 * In each loop , different colors are tested
1043 *
1044 *  @param ——
1045 *
1046 *  @return void
1047 */
1048 void loop() {
1049     // red
1050     RGBLED_Color(255,0,0);
1051     // Wait for one second
1052     delay(1000);
1053     // green
1054     RGBLED_Color(0,255,0);
1055     // Wait for one second
1056     delay(1000);
1057     // blue
1058     RGBLED_Color(0,0,255);
1059     // Wait for one second
1060     delay(1000);
1061     // yellow
1062     RGBLED_Color(255,255,0);
1063     // Wait for one second
1064     delay(1000);
1065     // cyan
1066     RGBLED_Color(0,255,255);
1067     // Wait for one second
1068     delay(1000);
1069     // magenta
1070     RGBLED_Color(255,0,255);
1071     // Wait for one second
1072     delay(1000);
1073     // white
1074     RGBLED_Color(255,255,255);
1075     // Wait for one second
1076     delay(1000);
1077     // black
1078     RGBLED_Color(0,0,0);
1079     // Wait for one second
1080     delay(1000);
1081     // orange
1082     RGBLED_Color(255,127,0);
1083     // Wait for one second
1084     delay(1000);
1085     // pink
1086     RGBLED_Color(255,192,203);
1087     // Wait for one second
1088     delay(1000);
1089     // purple
1090     RGBLED_Color(120,0,128);
1091     // Wait for one second
1092     delay(1000);
1093 }
```

.../Code/Nano33BLESense/Test/TestLEDRGBColors.ino

Brightness of the RGB-LED

The RGB-LED can also create gradients of colors by changing the brightness values gradually over time. This can create a smooth transition from one color to another, such as from red to green to blue and back to red.

This sketch 10.5 will make the RGB-LED change colors smoothly by varying the brightness of each LED with different speeds. You can adjust the initial brightness values and the increment/decrement values to get different effects.

Listing 10.5.: Different brightness levels for the RGB-LED colors

```

1000 /**
*  @file TestLEDRGBBrightness.ino
*
*  @brief Simple program for testing the RGB-LED
*
*
1006 * Turns the RGB-LED on for one second, then off for one second,
*   repeatedly.
*
1008 * The LED is switched on for 1 second and switched off
*   for 1 second so that the LED flashes accordingly.
1010 *
* On the Arduino Nano 33 BLE Sense, it is attached to digital pin 22,
*   23, 24
1012 *
*/
1014
1016 #include <../LEDs/RGBLED.h>
#incude <../LEDs/LED.h>
1018
1019 int redBrightness = 0; /*< Define the initial brightness values for
*   red (0-255) */
1020 int greenBrightness = 0; /*< Define the initial brightness values for
*   green (0-255) */
1021 int blueBrightness = 0; /*< Define the initial brightness values for
*   blue (0-255) */
1022
1023
1024 int redStep = 5; /*< Define the increment/decrement value for red */
int greenStep = 3; /*< Define the increment/decrement value for green */
int blueStep = 7; /*< Define the increment/decrement value for blue */
1026
1028
1029 /**
1030 * @brief the setup function runs once when you press reset or power the
*   board
*
1032 * Standard function of Arduino sketches
*
1034 * Initialization of the pin 22, pin 23, and 24 as output
*
1036 * @param —
*
1038 * @return void
*/
1040 void setup() {
    // Set the LED pins as outputs
    RGBLEDinit();
}
1042
1044

```

```

1046     /**
1047 * @brief the setup function runs once when you press reset or power the
1048 * board
1049 *
1050 * Standard function of Arduino sketches
1051 *
1052 * In each loop , the color is changing
1053 *
1054 * @param —
1055 *
1056 */
1057 void loop() {
1058     // Write the PWM values to the LED pins
1059     RGBLED_Colors(redBrightness, greenBrightness, blueBrightness);
1060
1061     // Update the brightness values for each color
1062     redBrightness += redStep;
1063     greenBrightness += greenStep;
1064     blueBrightness += blueStep;
1065
1066     // Check if the brightness values are out of range and reverse the
1067     // direction
1068     if (redBrightness <= 0 || redBrightness >= 255) {
1069         redStep = -redStep;
1070     }
1071     if (greenBrightness <= 0 || greenBrightness >= 255) {
1072         greenStep = -greenStep;
1073     }
1074     if (blueBrightness <= 0 || blueBrightness >= 255) {
1075         blueStep = -blueStep;
1076     }
1077
1078     // Wait for 10 milliseconds
1079     delay(10);
1080 }
```

..../Code/Nano33BLESense/Test/TestLEDRGBBrightness.ino

10.6. Simple Application

In many applications, it is necessary to save power so that the LED is not switched on continuously. Nevertheless, feedback is required as to whether the system, e.g. a fire detector, is switched on and functional. In this case, the LED is switched on for 1 second at a defined time interval, in this case 30 seconds. This is implemented in the example ??.

Listing 10.6.: A simple watch dog: A simple watchdog: the built-in RGB-LED is switched on for 1 second every 30 seconds.

```

1000 /**
1001 *
1002 * @file TestLEDRGBApplication.ino
1003 *
1004 * @brief Simple application using the built-in RGB-LED of the Arduino
1005 * Nano 33 BLE Sense
1006 *
1007 * @details The red part of the RGB-LED is switched on for 1 second and
1008 * switched off for 29 second so that red part of the LED flashes
1009 * accordingly .
1010 */
```

```

1012 #include <../LEDs/RGBLED.h>
1013 #include <../LEDs/LED.h>
1014 #include <../LEDs/SignsOfLife.h>

1016 #define CycleTimeOn 1000 /*< Duty cycle [ms] */
1018 #define CycleTimeOff 29000 /*< Switch-off time [ms] */

1022 /**
1024 * @brief the setup function runs once when you press reset or power the
1025 * board
1026 *
1027 * standard function of Arduino sketches
1028 *
1029 * Initialization of the pin LED_RED, the red part of the builtin RGB-
1030 * LED for the signs of life
1031 *
1032 * @param —
1033 *
1034 * @return void
1035 */
1036 void setup() {
    // Initialize the function SignsOfLife
    SignsOfLifeInit(LED_RED, CycleTimeOn, CycleTimeOff)

1038 // Application
1039 // ...
1040 }

1042

1044 /**
1045 * @brief the loop function runs over and over again forever
1046 *
1047 * standard function of Arduino sketches
1048 *
1049 * switching the led on / off for the signs of life
1050 *
1051 * @param —
1052 *
1053 * @return void
1054 */
1055 void loop() {
    // Switch red part of the RGB LED on/off
    SignsOfLife();

1057 // Application
1058 // ...
1059 }

```

..../Code/Nano33BLESense/Test/TestLEDRGBApplication.ino

10.7. Further Readings

- Schanda, Janos: *Colorimetry: Understanding the CIE System*. Wiley, 2007. [Sch07]
- Hiller, Gabrielle: *Color measurement - the CIE color space*. Datacolor, 2019. [Hil22]
- Lukac, Rastislav and Plataniotis, Konstantinos N.: *Color Image Processing: Methods and Applications*. CTC Press, 2018. [LP18]

11. TinyML Shield: Built-in Push Button

11.1. General

A push button is a simple switch mechanism used to control various devices and processes. It is typically made of hard materials like plastic or metal. The surface of a push button is designed to be easily depressed or pushed by the human finger or hand. When you press a push button, it either closes or opens an electrical circuit.

In industrial and commercial applications, push buttons can be linked together so that pressing one button releases another. Emergency stop buttons, often with large mushroom-shaped heads, enhance safety in machines and equipment. Pilot lights are sometimes added to push buttons to draw attention and provide feedback when the button is pressed. Color-coding is common to associate push buttons with their specific functions (e.g., red for stopping, green for starting). [Din]

11.2. Built-in Push Button

The Arduino Nano 33 BLE Sense features an onboard push button. This button is a simple electrical switch that can be activated by pressing it. When you press the button, it completes an electrical circuit. The push button is designed for user interaction and can be used for various purposes.

The built-in button `BUTTON_B` is connected with pin 11. Using the function `pinMode(BUTTON_PIN, INPUT_PULLUP)` the pin is declared as an input. As can be seen in the sketch, pressing the button can be used to trigger actions; typical actions include switching on an LED, changing modes, or initiating sensor readings. Overall, the push button provides a convenient way to interact with the Arduino Nano 33 BLE Sense and create responsive projects. [Ard23a; Ard23b; Arda]

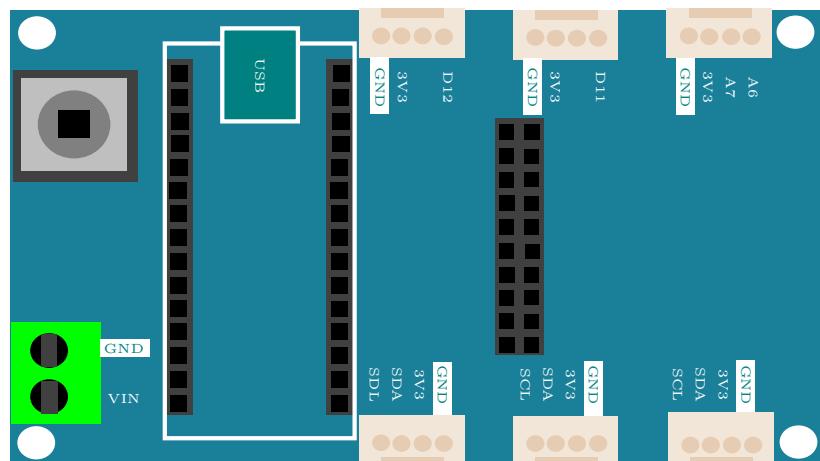


Figure 11.1.: Tiny Machine Learning Shield [Gua21]

11.3. Specification

The built-in button is a small white button and connected to pin 11.

Built-in Button: `BUTTON_B = 11u`

If the pin is declared as input in the function `setup`, then it can be used.

The pin 11 must be defined as an input in the function `setup` by setting `pinMode(11, INPUT_PULLUP)`, otherwise the button cannot be read.

The pin 11 can also be used otherwise. Then the button is not in use. [Ard23a; Ard23b; Arda]

WS:

- cite data sheet
- Circuit Diagram

11.4. Simple Code

As soon as the button is connected, it can be used. It is not necessary to install a special library. Programming takes place in two steps:

1. In the first step, the pin is configured in the function `setup`:

Listing 11.1.: Defining the built-in button's pin as an input.

```
1000  pinMode(BUTTON_B, INPUT_PULLUP)
```

2. In the second step, the button can be used in the function `loop`. To read in the value, use the function `digitalRead`:

Listing 11.2.: Read the built-in button's state

```
1000  buttonState = digitalRead(BUTTON_B);
```

11.5. Tests

The simplest test is the flashing of an LED for 2 seconds, if the button is pressed, see sketch 28.3.

Listing 11.3.: Simple sketch to test the push button and the built-in LED

```
1000 #include <LEDGeneral.h>
1001 #include <LEDPower.h>
1002 #include <LEDRGB.h>
1003 #include <LEDSignsOfLife.h>
1004 #include <LEDbuiltin.h>
1005 #include <Arduino.h>
1006
1007 /**
1008 *  @file TestPushButton.ino
1009 *
1010 *  @brief Simple application reading in the built-in push button states.
1011 *
1012 *
1013 *  @details If the built-in push button is pressed, the the internal
1014 *          built-in LED is turn on for 2 seconds
```

```
1016 *
1018 */
1019 #define BUTTON_B 11 /*< Button_B is here defined as pin 11 */
1020 #define BUTTON_PIN BUTTON_B /*< Use the onboard push button (BUTTON_B)
1021 */
1022 #define SET_ON true /*< Define flag for switching on */
1023 #define SET_OFF false /*< Define flag for switching off */
1024 int buttonState = 0; /*< state of the built-in push button */
1026 /**
1027 * @brief the setup function runs once when you press reset or power the
1028 * board
1029 *
1030 * standard function of Arduino sketches
1031 *
1032 * Initialization of the built-in LED and the push button
1033 *
1034 * @param —
1035 *
1036 * @return void
1037 */
1038 void setup() {
1039     // Initialize the pin as an output
1040     LEDBuiltinsetup();
1041     // Initialize the pin as an input
1042     pinMode(BUTTON_PIN, INPUT_PULLUP);
1043 }
1044
1046 /**
1047 * @brief the loop function runs over and over again forever
1048 *
1049 * standard function of Arduino sketches
1050 *
1051 * switching the built-in led on for 2 sec, if the push button is
1052 * pressed
1053 *
1054 * @param —
1055 *
1056 * @return void
1057 */
1058 void loop()
1059 {
1060     buttonState = digitalRead(BUTTON_PIN);
1061     if (buttonState == HIGH)
1062     {
1063         // Turn the LED on
1064         LEDBuiltin(SET_ON);
1065         // Wait for two second
1066         delay(2000);
1067         // Turn the LED off
1068         LEDBuiltin(SET_OFF);
1069         // Wait for one second
1070         delay(1000);
1071         buttonState = LOW;
1072     }
1073 }
```

..../Code/Nano33BLESense/PushButton/TestPushButton/TestPushButton.ino

11.6. Interrupt Function on the Arduino Nano 33 BLE Sense

An interrupt is a function that allows the microcontroller on the Arduino Nano 33 BLE Sense to immediately respond to an external event, such as pressing a button, without the need for the program to continuously check for that event. Normally, in a program, the microcontroller would repeatedly check if a button is pressed by running through a loop, which consumes processing power and can slow down other tasks. This is not efficient, especially when the program needs to perform other actions at the same time.

With an interrupt, the program can continue running other tasks, and only when the specified event (like a button press) occurs, the program is temporarily paused to execute a special function known as the **Interrupt Service Routine (ISR)**. The ISR is a small, efficient function designed to handle the event, such as changing a variable or triggering another action. After the ISR is executed, the program returns to where it left off, continuing its normal operation.

Using interrupts in this way helps make the program more efficient and responsive, as it doesn't waste time constantly checking for events and can focus on other tasks until something important happens. This is especially useful for real-time applications where quick responses to external events are necessary, such as in embedded systems, robotics, or sensor monitoring.

11.7. Simple Application

This code sketch 28.4 shows how to use an interrupt to control the built-in LED on the Arduino Nano 33 BLE Sense when the built-in push button is pressed.

The program sets up the built-in LED and push button. When the button is pressed, an interrupt runs a special function called an Interrupt Service Routine. The ISR is very short and only sets a flag variable `pushPressed` to `true`, indicating that the button has been pressed.

The main program `loop` checks this flag. If it is set to `true`, the program turns on the LED for 2 seconds, then turns it off and resets the flag to `false`. This ensures the system is ready to detect the next button press. Using the `true` and `false` values makes it easy to manage the button's state.

This method avoids constantly checking the button's state, making the program more efficient. Additionally, the Arduino Nano 33 BLE Sense allows all its pins to be used for interrupts. This makes it easy to attach interrupt functions to any pin, allowing quick and efficient responses to inputs like buttons or sensors. [Ardf]

WS:andere Überschrift
für simple Application

Listing 11.4.: Simple sketch connects the push button with an interrupt. Here, pushing the built-in button is handled by an interrupt. Then the built-in LED switch on for 2 sec.

```

1000 #include <LEDGeneral.h>
1001 #include <LEDPower.h>
1002 #include <LEDRGB.h>
1003 #include <LEDSignsOfLife.h>
1004 #include <LEDbuiltin.h>

1006 #include <LEDGeneral.h>
1007 #include <LEDPower.h>
1008 #include <LEDRGB.h>
1009 #include <LEDSignsOfLife.h>
1010 #include <LEDbuiltin.h>

1012 /**

```

```
*  
1014 * @file TestPushButtonInterrupt.ino  
*  
1016 * @brief Simple application reading in the built-in push button states  
*       using an interrupt  
*  
1018 *  
1019 * @details If the builtin push button is pressed, the built-in LED is  
*       switched on for 2 second and switched off again.  
1020 * But in this example, an interrupt is used.  
*  
1022 */  
  
1024 #include <LEDGeneral.h>  
1025 #include <LEDBuiltin.h>  
1026  
  
1028 #define BUTTON_B 11 /*< Button_B is here defined as pin 11 */  
1029 #define BUTTON_PIN BUTTON_B  
  
1032 #define SET_ON true /*< Define flag for switching on */  
1033 #define SET_OFF false /*< Define flag for switching off */  
  
1036  
  
1038 // Initialize variables  
1039 //  
1040 volatile bool pushPressed = false; /*< // Flag, whether the button is  
*       pressed. Declare as volatile for interrupt. safety */  
1041  
1042 int ledState = 0; /*< LED>Status zur Verarbeitng */  
1043  
1044 /**  
1045 * @brief the setup function runs once when you press reset or power the  
*       board  
*  
1047 * standard function of Arduino sketches  
*  
1049 * Initialization of the built-in LED and the push button  
*  
1051 * @param —  
*  
1053 * @return void  
*/  
1055 void setup() {  
    // Initialize the pin as an output  
    LEDBuiltinsetup();  
    // Initialize the pin as an input  
    pinMode(BUTTON_PIN, INPUT_PULLUP);  
    // Initialize the interrupt function  
    attachInterrupt(digitalPinToInterrupt(BUTTON_PIN), buttonPressed,  
                    CHANGE);  
}  
1064  
  
1066 /**  
1067 * @brief Interrupt service function  
*  
1069 * Attention: as short as possible  
*  
1071 * The function changes just one flag.  
*/  
1073 void buttonPressed()  
{  
    if (pushPressed == false)  
    {
```

```

1078     pushPressed = true;
1079 }
1080 /**
1082 * @brief the loop function runs over and over again forever
1084 *
1086 * standard function of Arduino sketches
1087 *
1088 * switching the built-in led on for 2 sec, if the push button is
1089 * pressed
1090 *
1091 * @param —
1092 * @return void
1093 */
1094 void loop()
1095 {
1096     if (pushPressed)
1097     {
1098         // Turn the LED on
1099         LEDBuiltin(SET_ON);
1100         // Wait for one second
1101         delay(2000);
1102         // Turn the LED off
1103         LEDBuiltin(SET_OFF);
1104         pushPressed = false;
1105     }
1106     // ...
1107 }
```

..../Code/Nano33BLESense/Button/TestPushButtonInterrupt/TestPushButtonInterrupt.ino

This is just a simple example. The variable `BUTTON_B` is already defined, so the assignment is not necessary. The command `delay` should be avoided in an Arduino sketch. Instead, variables of the type `elapsedMillis` should be used.

11.8. Further Readings

- Boxall, John: *Arduino Workshop - A Hands-On Introduction with 65 Projects*. No Starch Press, 2021. [Box21]
- Voš, Andreas: *Volumio mit Drehgebern erweitern*. Make Magazin, 2024. [Voš24]
- Kühnel, Claus: *Arduino - Das umfassende Handbuch*. Rheinwerk Verlag GmbH, 2024. [Küh24]

12. Pressure and Temperature Sensor LPS22HB

The sensor LPS22HB is a piezoresistive absolute pressure sensor that is integrated into the Arduino Nano 33 BLE Sense board. It's a digital sensor that measures atmospheric pressure and temperature. The sensor uses a piezoresistive technology to detect changes in pressure. The LPS22HB has an accuracy of ± 1.5 hPa and a resolution of 0.01 hPa. It can measure pressures from 260 to 1260 hPa. The sensor also measures temperature with an accuracy of $\pm 0.12^\circ\text{C}$ and a resolution of 0.01°C . The temperature range is from -40°C to 85°C .

The LPS22HB communicates with the Arduino board using the protocol I²C. It has a low power consumption of $1.5 \mu\text{A}$ in sleep mode and 1.5 mA in active mode. The sensor is also resistant to shock and vibration. It's a sensor for projects that require accurate pressure and temperature measurements, such as weather stations, altimeters. [Ard23a; Ard23b]

Arduino Nano 33 BLE Sense Lite hat keine HTS221-Temperatur- und Feuchtigkeitssensoren, sondern nur den LPS22HB-Drucksensor, der allerdings auch die Temperatur messen kann

Arduino Nano 33 BLE Sense Rev2 hat einen HTS221-Temperatur- und Feuchtigkeitssensor und den LPS22HB-Drucksensor, der allerdings auch die Temperatur messen kann. Unterschiede:

Inertiale Messeinheit (IMU): Rev 1: Verfügt über eine einzelne 9-Achsen-IMU. Rev 2: Hat eine Kombination aus zwei IMUs: eine 6-Achsen-IMU (BMI270) und eine 3-Achsen-IMU (BMM150), was die Möglichkeiten zur Bewegungserkennung erweitert1.

Design und Zugänglichkeit: Rev 2: Integriert neue Pads und Testpunkte für USB, SWDIO und SWCLK, was den Zugang zu diesen wichtigen Punkten auf der Platine erleichtert1. Rev 1: Hat diese zusätzlichen Pads und Testpunkte nicht. Stromversorgung:

Rev 2: Verwendet den MP2322 als Stromversorgungskomponente, was die Leistung verbessert1.

Rev 1: Nutzt eine andere Stromversorgungskomponente. Der Arduino Nano 33 BLE Sense Rev 1 verwendet den MPS MP2144 als Stromversorgungskomponente

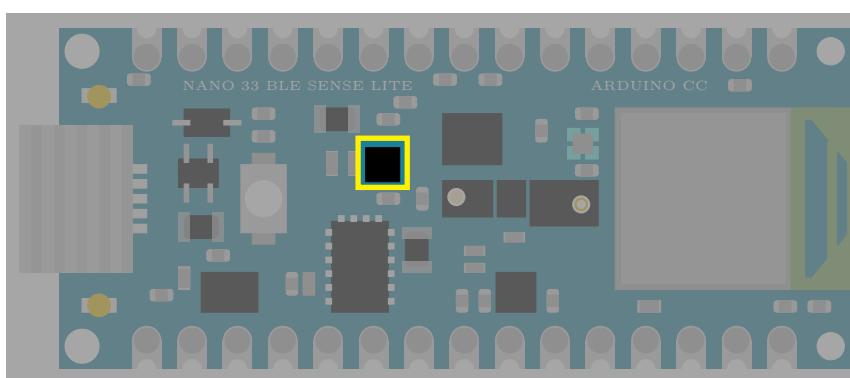


Figure 12.1.: Arduino Nano 33 BLE Sense's pressure and temperature sensor LPS22HB

12.1. General

A pressure sensor is a device that measures the pressure of its surroundings. It works by converting the mechanical energy of the environment into an electrical signal. The sensor typically consists of a diaphragm or membrane, which is a thin, flexible material that changes its shape in response to changes in pressure. The diaphragm or membrane is usually connected to a microcontroller or other electronic device, which reads the electrical signal and converts it into a pressure reading. The pressure sensor can be calibrated to provide accurate readings over a specific pressure range. The sensor can be used in a variety of applications, including industrial control systems, medical devices, and consumer electronics.

Pressure sensors can be classified into two main types: absolute and differential. Absolute pressure sensors measure the absolute pressure of the environment, while differential pressure sensors measure the difference in pressure between two points. Pressure sensors can be used to measure a wide range of pressures, from very low pressures to very high pressures. The accuracy of a pressure sensor depends on its calibration and the quality of its components. Pressure sensors can be affected by various factors, including temperature, humidity, and air flow. Pressure sensors can be used to monitor and control pressure in a variety of applications, including heating and cooling systems, refrigeration systems, and food processing systems. Pressure sensors can be used to detect pressure anomalies and alert users to potential problems. [Gan24]

A temperature sensor is a device that measures the temperature of its surroundings. It works by converting the thermal energy of the environment into an electrical signal. The sensor typically consists of a thermistor or thermocouple, which is a device that changes its electrical resistance or voltage in response to changes in temperature. The thermistor or thermocouple is usually connected to a microcontroller or other electronic device, which reads the electrical signal and converts it into a temperature reading. The temperature sensor can be calibrated to provide accurate readings over a specific temperature range. The sensor can be used in a variety of applications, including industrial control systems, medical devices, and consumer electronics.

Temperature sensors can be classified into two main types: contact and non-contact. Contact temperature sensors, such as thermocouples, come into direct contact with the object being measured. Non-contact temperature sensors, such as infrared sensors, do not come into contact with the object being measured.

Temperature sensors can be used to measure a wide range of temperatures, from very low temperatures to very high temperatures. The accuracy of a temperature sensor depends on its calibration and the quality of its components. Temperature sensors can be affected by various factors, including humidity, air flow, and radiation. Temperature sensors can be used to monitor and control temperature in a variety of applications, including heating and cooling systems, refrigeration systems, and food processing systems. Temperature sensors can be used to detect temperature anomalies and alert users to potential problems.

WS:cite books,
applications, board

12.2. Specific Sensor

The pressure sensor of the LPS22HB is a piezoresistive sensor that changes its electrical resistance in response to changes in pressure. The pressure sensor of the LPS22HB is connected to the Arduino Nano 33 BLE Sense, which reads the electrical signal and converts it into a pressure reading. The pressure sensor of the LPS22HB is calibrated to provide accurate readings over a pressure range of 260 to 1260 mbar. The pressure sensor of the LPS22HB is a non-contact sensor, meaning that it does not come into

direct contact with the object being measured. The pressure sensor of the LPS22HB is affected by various factors, including temperature and humidity. The pressure sensor of the LPS22HB can be used to monitor and control pressure in a variety of applications, including industrial control systems and consumer electronics. The pressure sensor of the LPS22HB can be used to detect pressure anomalies and alert users to potential problems. The pressure sensor of the LPS22HB has an accuracy of ± 0.12 mbar. The pressure sensor of the LPS22HB is a low-power sensor, making it suitable for use in battery-powered devices.

The LPS22HB is a digital pressure sensor that also includes a temperature sensor. The temperature sensor of the LPS22HB is a thermistor that changes its electrical resistance in response to changes in temperature. The thermistor is connected to the microcontroller Arduino Nano 33 BLE Sense, which reads the electrical signal and converts it into a temperature reading. The temperature sensor of the LPS22HB is calibrated to provide accurate readings over a temperature range of -40°C to 85°C . The temperature sensor of the LPS22HB is a non-contact sensor, meaning that it does not come into direct contact with the object being measured.

The temperature sensor of the LPS22HB is affected by various factors, including humidity and air flow. The temperature sensor of the LPS22HB can be used to monitor and control temperature in a variety of applications, including industrial control systems and consumer electronics. The temperature sensor of the LPS22HB can be used to detect temperature anomalies and alert users to potential problems. The temperature sensor of the LPS22HB is a high-accuracy sensor, with an accuracy of $\pm 0.12^{\circ}\text{C}$ and a resolution of 0.01°C .

The temperature sensor of the LPS22HB is a low-power sensor, making it suitable for use in battery-powered devices.

WS:cite board

12.3. Specification

The LPS22HB is a digital pressure sensor that measures pressure in the range of 260 to 1260 mbar. It has a high accuracy of ± 0.12 mbar and a resolution of 0.01 mbar. The sensor is calibrated to provide accurate readings over a temperature range of -40°C to 85°C . It has a non-contact measurement principle, which means that it does not come into direct contact with the object being measured. The LPS22HB is a compact sensor, measuring $3.5 \times 3.5 \times 1.1$ mm in size.

It has a low power consumption of $1.5\mu\text{A}$ in active mode and $0.1\mu\text{A}$ in sleep mode. The sensor has a sampling rate of up to 100 Hz and a data transfer rate of up to 400 kbps. Note that the sampling rate of the sensor LPS22HB can be adjusted using the function `setSamplingRate()` in the Arduino library. The default sampling rate is 100 Hz, but it can be set to 50 Hz, 25 Hz, or 10 Hz using the following code:

```
lps22hb.setSamplingRate(50); // 50 Hz
lps22hb.setSamplingRate(25); // 25 Hz
lps22hb.setSamplingRate(10); // 10 Hz
```

It's worth noting that the sampling rate of the sensor LPS22HB can affect the accuracy and reliability of the measurements. A higher sampling rate can provide more accurate measurements, but it can also increase the power consumption of the sensor.

The sensor is also resistant to shock, vibration, and temperature changes. It has a high sensitivity and a low noise level, making it suitable for use in applications where high accuracy is required.

- cite data sheet
- Circuit Diagram

12.4. Library

To program the sensor LPS22HB on the Arduino Nano 33 BLE Sense, you will need to use the library LPS22HB. [Ardd]

12.4.1. Description

The library LPS22HB is a library for the Arduino platform that provides a simple interface for interacting with the sensor LPS22HB. It allows you to read temperature and pressure values from the sensor, as well as set the sensor's mode and power mode. [Ardd]

12.4.2. Installation

To use the library LPS22HB, you will need to install it on your Arduino IDE. You can do this by following these steps:

- Open the Arduino IDE and navigate to the menu **Sketch**.
- Select **Sketch -> Include Library -> Manage Libraries**.
- Search for “LPS22HB” in the library search bar.
- Click on the library “LPS22HB” and then click on the button “Install”.
- Wait for the library to install and then restart the Arduino IDE.

Once you have installed the library LPS22HB, you can use it to program the sensor LPS22HB on the Arduino Nano 33 BLE Sense. Here is an example ?? of how you can use the library to read temperature and pressure values from the sensor:

Listing 12.1.: Example code for the sensor LPS22HB on the Arduino Nano 33 BLE Sense

```

1000 /**
1001 * @file LPS22HBSimpleExample.ino
1002 *
1003 * @brief Example code for the sensor LPS22HB on the Arduino Nano 33 BLE
1004 * Sense
1005 *
1006 * @author Elmar Wings
1007 *
1008 * @version 1.0
1009 */
1010
1011 #include <LPS22HB.h>
1012
1013 /**
1014 * @brief LPS22HB sensor object
1015 *
1016 * @details This object represents the LPS22HB sensor and provides
1017 * methods for reading temperature and pressure values.
1018 */
1019 LPS22HB lps22hb;
1020
1021 /**
1022 * @brief Setup function
1023 *
1024 * @details This function is called once at the beginning of the program
1025 * and is used to initialize the sensor and serial communication.
1026 */
1027 void setup() {
1028     // Initialize serial communication

```

```

1026   Serial.begin(9600);

1028   // Initialize the LPS22HB sensor
1029   lps22hb.begin();
1030 }

1032 /**
1033 * @brief Loop function
1034 *
1035 * @details This function is called repeatedly after the setup function
1036 *          and is used to read temperature and pressure values from the sensor.
1037 */
1038 void loop() {
1039   // Read temperature value from the sensor
1040   int16_t temp = lps22hb.readTemperature();

1041   // Read pressure value from the sensor
1042   int32_t pressure = lps22hb.readPressure();

1043   // Print temperature and pressure values to the serial console
1044   Serial.print("Temperature: ");
1045   Serial.print(temp);
1046   Serial.println(" C");
1047   Serial.print("Pressure: ");
1048   Serial.print(pressure);
1049   Serial.println(" mbar");

1050   // Wait for 1 second before taking the next reading
1051   delay(1000);
1052 }

```

..../Code/Nano33BLESense/SensorLPS22HB/LPS22HBSimpleExample.ino

This sketch 12.1 uses the library LPS22HB to read temperature and pressure values from the sensor LPS22HB and print them to the serial console.

Note that you will need to have the Arduino Nano 33 BLE Sense board connected to your computer and the Arduino IDE installed on your computer in order to use the LPS22HB library.

12.4.3. Functions

Here are the functions of the Arduino library LPS22HB:

- Initialization Functions
 - `begin()`: Initializes the sensor LPS22HB and sets it up for use.
 - `reset()`: Resets the sensor LPS22HB to its default state.
- Reading Functions
 - `readTemperature()`: Reads the temperature value from the sensor LPS22HB.
 - `readPressure()`: Reads the pressure value from the sensor LPS22HB.
 - `readAltitude()`: Reads the altitude value from the sensor LPS22HB.
 - `readTemperatureAndPressure()`: Reads both the temperature and pressure values from the sensor LPS22HB.
- Setting Functions
 - `setMode()`: Sets the mode of the sensor LPS22HB.
 - `setPowerMode()`: Sets the power mode of the sensor LPS22HB.
 - `setSamplingRate()`: Sets the sampling rate of the sensor LPS22HB.

- `setFilter()`: Sets the filter of the sensor LPS22HB.
- Getting Functions
 - `getMode()`: Gets the current mode of the sensor LPS22HB.
 - `getPowerMode()`: Gets the current power mode of the sensor LPS22HB.
 - `getSamplingRate()`: Gets the current sampling rate of the sensor LPS22HB.
 - `getFilter()`: Gets the current filter of the sensor LPS22HB.
- Sleep Functions
 - `sleep()`: Puts the sensor LPS22HB into sleep mode.
 - `wakeUp()`: Wakes up the sensor LPS22HB from sleep mode.
- Other Functions
 - `checkStatus()`: Checks the status of the sensor LPS22HB.
 - `getError()`: Gets the error code of the sensor LPS22HB.
 - `resetError()`: Resets the error code of the sensor LPS22HB.

Note that this list may not be exhaustive, and the library may have additional functions not listed here.

12.4.4. Example - Manual

12.4.5. Example

12.4.6. Example - Code

12.4.7. Example - Files

12.5. Calibration

To calibrate the sensor LPS22HB on the Arduino Nano 33 BLE Sense, you will need to follow these steps. First, upload the calibration code to the Arduino Nano 33 BLE Sense. The calibration code will prompt you to enter the calibration parameters, such as the temperature and pressure values. Enter the calibration parameters, and the code will store them in the sensor's memory. The calibration process will take a few minutes to complete. During the calibration process, the sensor will take multiple readings of the temperature and pressure values. The sensor will then calculate the average of the readings and store it as the calibration value. Once the calibration process is complete, the sensor will be ready to use. To verify the calibration, you can use the calibration code to read the temperature and pressure values from the sensor. Compare the readings with the expected values to ensure that the sensor is calibrated correctly. If the readings are not accurate, you may need to repeat the calibration process. The calibration process can be repeated as many times as necessary to achieve accurate readings. The calibration values can be stored in the sensor's memory and retrieved later. By following these steps, you can calibrate the sensor LPS22HB on the Arduino Nano 33 BLE Sense and ensure accurate readings.

The code 12.2 reads the temperature and pressure values from the sensor LPS22HB, stores them in the sensor's memory, and prints them to the serial console. The `storeCalibration` function is used to store the calibration values in the sensor's memory.

Listing 12.2.: Simple sketch calibrating the sensor LPS22HB

WS:cite method, more mathematics!

```
1000 /**
1001 * @file LPS22HBCalibration.ino
1002 *
1003 * @brief Calibration code for the sensor LPS22HB on the Arduino Nano 33
1004 * BLE Sense
1005 *
1006 * @details This code reads the temperature and pressure values from the
1007 * LPS22HB sensor, stores them in the sensor's memory, and prints them
1008 * to the serial console. The storeCalibration function is used to
1009 * store the calibration values in the sensor's memory.
1010 *
1011 * Note that this is just an example code and you may need to modify it
1012 * to suit your specific needs. Additionally, you will need to make
1013 * sure that the LPS22HB sensor is properly connected to the Arduino
1014 * Nano 33 BLE Sense and that the I2C interface is enabled.
1015 *
1016 * @author Elmar Wings
1017 *
1018 * @version 1.0
1019 */
1020
1021 #include <Wire.h>
1022 #include <LPS22HB.h>
1023
1024 /**
1025 * @brief LPS22HB sensor object
1026 */
1027 LPS22HB lps22hb;
1028
1029 /**
1030 * @brief Setup function
1031 */
1032 void setup() {
1033     Serial.begin(9600);
1034     Wire.begin();
1035     lps22hb.begin();
1036 }
1037
1038 /**
1039 * @brief Loop function
1040 */
1041 void loop() {
1042     // Read the temperature and pressure values from the sensor
1043     int16_t temp = lps22hb.readTemperature();
1044     int32_t pressure = lps22hb.readPressure();
1045
1046     // Print the temperature and pressure values to the serial console
1047     Serial.print("Temperature: ");
1048     Serial.print(temp);
1049     Serial.println(" C");
1050     Serial.print("Pressure: ");
1051     Serial.print(pressure);
1052     Serial.println(" mbar");
1053
1054     // Store the calibration values in the sensor's memory
1055     lps22hb.storeCalibration(temp, pressure);
1056
1057     // Wait for 1 second before taking the next reading
1058     delay(1000);
1059 }
1060
1061 /**
1062 * @brief Store calibration values in the sensor's memory
1063 * @param temp Temperature value
1064 * @param pressure Pressure value
1065 */
1066 void storeCalibration(int16_t temp, int32_t pressure) {
```

```

1062 // Store the calibration values in the sensor's memory
1063 Wire.beginTransmission(0x5C); // LPS22HB I2C address
1064 Wire.write(0x00); // Calibration register
1065 Wire.write(temp >> 8); // High byte of temperature value
1066 Wire.write(temp & 0xFF); // Low byte of temperature value
1067 Wire.write(pressure >> 24); // High byte of pressure value
1068 Wire.write(pressure >> 16); // Middle byte of pressure value
1069 Wire.write(pressure >> 8); // Low byte of pressure value
1070 }

```

..../Code/Nano33BLESense/SensorLPS22HB/LPS22HBCalibration.ino

Note that this is just an example code and you may need to modify it to suit your specific needs. Additionally, you will need to make sure that the sensor LPS22HB is properly connected to the Arduino Nano 33 BLE Sense and that the I2C interface is enabled.

12.6. Simple Code

12.7. Sleep Mode

Listing 12.3.: Sketch for the Arduino Nano 33 BLE Sense to switch the sensor LPS22HB into sleep mode

```

1000 /**
1001 * @file LPS22HBSleep.ino
1002 *
1003 * @brief Sketch to switch the sensor LPS22HB into sleep mode
1004 *
1005 * @details Sketch for the Arduino Nano 33 BLE Sense to switch the
1006 *         sensor LPS22HB into sleep mode
1007 *
1008 * @author Elmar Wings
1009 *
1010 * @version 1.0
1011 */
1012
1013 #include <Wire.h>
1014 #include <LPS22HB.h>
1015
1016 /**
1017 * @brief LPS22HB sensor object
1018 * @details This object represents the LPS22HB sensor and provides
1019 *         methods for reading temperature and pressure values.
1020 */
1021 LPS22HB lps22hb;
1022
1023 /**
1024 * @brief Setup function
1025 *
1026 * @details This function is called once at the beginning of the program
1027 *         and is used to
1028 *         - Initialize I2C communication
1029 *         - initialize the sensor, and
1030 *         - initialize serial communication.
1031 */
1032 void setup() {
1033     // initialize serial communication
1034     Serial.begin(9600);
1035
1036     // Initialize I2C communication
1037     Wire.begin();

```

```

1036 // This line initializes the LPS22HB sensor and sets it up for use.
1037 lps22hb.begin();
1038 }

1040 /**
1041 * @brief Loop function
1042 *
1043 * @details This function is called repeatedly after the setup function
1044 *          and is used to switch the sensor into sleep mode and wake it up.
1045 */
1046 void loop() {
1047     // Switch the sensor into sleep mode
1048     lps22hb.sleep();

1049     // Wait for 1 second
1050     delay(1000);

1051     // Switch the sensor out of sleep mode
1052     lps22hb.wakeUp();

1053     // Wait for 1 second
1054     delay(1000);
1055 }

```

..../Code/Nano33BLESense/SensorLPS22HB/LPS22HBSleep.ino

This sketch 12.3 uses the function `sleep()` to switch the sensor into sleep mode, and the function `wakeUp()` to switch the sensor out of sleep mode. The function `sleep()` puts the sensor into a low-power state, and the function `wakeUp()` restores the sensor to its normal operating state.

Note that the function `sleep()` must be called before the sensor can be put into sleep mode, and the function `wakeUp()` must be called before the sensor can be restored to its normal operating state.

Also, note that the function `sleep()` can only be called when the sensor is in a valid state, and the function `wakeUp()` can only be called when the sensor is in a valid state.

You can also use the function `setMode()` to set the sensor to sleep mode, and the function `getMode()` to get the current mode of the sensor.

```

lps22hb.setMode(LPS22HB_MODE_SLEEP);
lps22hb.getMode();

```

This will set the sensor to sleep mode and get the current mode of the sensor.

You can also use the function `setPowerMode()` to set the power mode of the sensor, and the function `getPowerMode()` to get the current power mode of the sensor.

```

lps22hb.setPowerMode(LPS22HB_POWER_MODE_SLEEP);
lps22hb.getPowerMode();

```

This will set the power mode of the sensor to sleep mode and get the current power mode of the sensor.

12.8. Simple Application**12.9. Tests****12.9.1. Simple Function Test****12.9.2. Test all Functions****12.10. Simple Application****12.11. Further Readings**

13. Sensormodul APDS-9960 for Gesture, Proximity, and Color Detection

The APDS-9960 sensor, integrated on the Arduino Nano 33 BLE Sense, is a multifunctional optical sensor used for gesture recognition, ambient light detection, color sensing, and proximity measurement. The sensor uses an I²C interface for communication and comes equipped with an additional infrared LED. [Ava15]

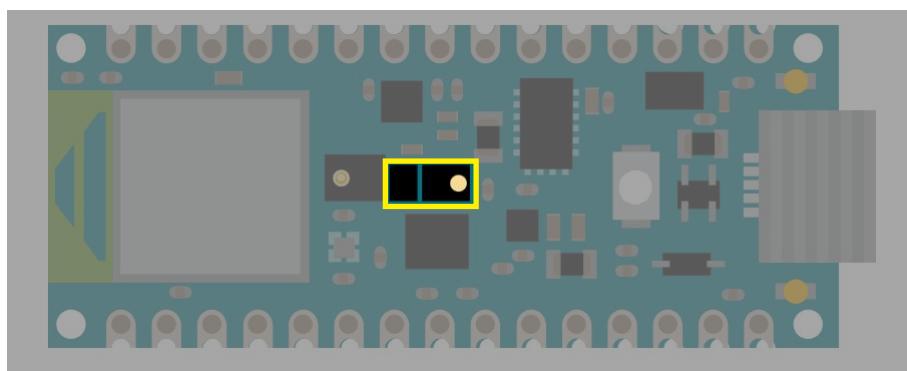


Figure 13.1.: Arduino Nano 33 BLE Sense's APDS-9960

The sensor is located centrally on the Arduino Nano 33 BLE Sense, as indicated in the Figure 13.1.

WS:Figure 6.1 must be converted into a TikZ image to maintain consistency

13.1. Functionality of the APDS-9960

The APDS-9960 is a versatile optical sensor that offers multiple functionalities, including color detection, proximity sensing, ambient light measurement, and gesture recognition. Each of these features plays a crucial role in various applications, ranging from smart lighting adjustments to touchless control systems. In the following sections, each function of the APDS-9960 will be explored in detail, explaining its working principle, the type of data it provides, and its potential use cases.

13.2. Key Technical Specifications

Power Supply

- Supply Voltage: 2.4 V to 3.6 V
- Maximum Voltage: 3.8 V

Power Consumption

- Active ALS Mode (Ambient Light Sensor): 200-250 μA
- Proximity and Gesture Modes: 790 μA (without LED)
- Sleep Mode: 1-10 μA

Temperature

- Operating Temperature Range: -30°C to +85°C
- Storage Temperature Range: -40°C to +85°C

Optical Properties

- LED Wavelength (max.): 950 nm
- LED Drive Current:
 - 100 mA (Standard), 50 mA, 25 mA, 12.5 mA
 - LED Boost Option: 100%, 150%, 200%, 300% (adjustable current boost)
- Max Detection Distance for Color, Proximity and Gesture Sensor 0mm to 100mm

Proximity Sensor

- Pulse Width for Proximity Measurement: 4 μs to 32 μs

Gesture Sensor

- LED Pulse Count: 1 to 64 pulses

Connections

- I²C Communication:
 - Data Rates: Up to 400 kHz
 - Pins:
 - * SDA (I²C Data)
 - * SCL (I²C Clock)
 - * INT (Interrupt, Open Drain, Active Low)
 - * LDR (LED Driver Input)

Dimensions

- Length: 3.94mm
- Width: 2.36mm
- Height: 1.35mm

13.3. Library [Arduino_APDS9960](#)

In the case of the APDS-9960 sensor, Arduino provides a library [Arduino_APDS9960](#) for calling functions related to color detection, distance measurement, or gesture recognition.

The library for the sensor is [Arduino_APDS9960](#). This allows measuring gestures, colors, light intensity, and distances with the sensor. Communication between the Arduino Nano 33 BLE Sense's chip and the APDS-9960 module occurs via an internal I²C interface [Ava15].

The library is included by using the command `#include <Arduino_APDS9960.h>`.

13.3.1. Installation of APDS-9960 Library

The library is installed as follows:

1. Open the Arduino IDE and navigate to **Tools** **Manage Libraries...**.
2. In the Library Manager, use the search bar to look for "Arduino_APDS9960".
3. Several libraries will be displayed. Select the library titled "Arduino_APDS9960" by Arduino and click *Install*.
4. Once installed, the IDE will show a message in the console confirming the installation. The "Install" button will also change to "Remove", as seen in Figure 13.2, indicating the library is ready for use.

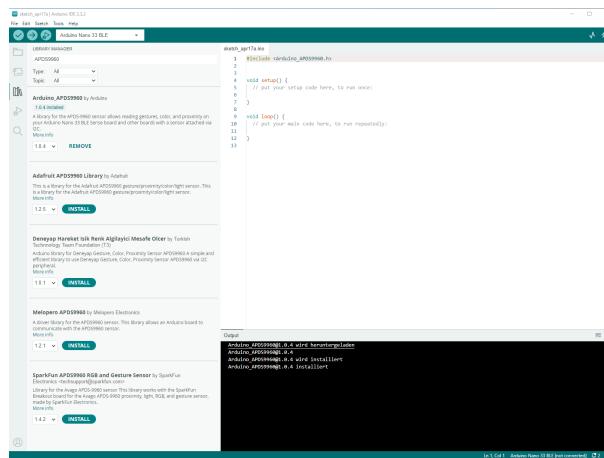


Figure 13.2.: APDS-9960 Library Installation

13.3.2. Functions

After the theoretical overview of the APDS-9960 sensor's functions, the following section will explain the practical implementations. It will demonstrate how the various functions of the sensor, such as color detection, gesture recognition, and proximity sensing, can be applied in practice to activate and evaluate the sensor's capabilities.

Here are the codes to utilize the various functions of the APDS-9960 sensor, such as color detection, gesture recognition, and proximity sensing, see [Ard24]:

- **`begin()`**: The `begin()` method activates and initializes the APDS9960 sensor. It is typically called during the `setup{}` phase.
 - Return value `TRUE`: Initialization was successful.

- Return value `FALSE`: Initialization failed.
- `end()`: The `end()` method deactivates the APDS9960 sensor.
- `colorAvailable()`: This method checks if color data is available to be retrieved.
 - Return value `TRUE`: Color data is available.
 - Return value `FALSE`: No color data is available.
- `readColor(...)`: This method retrieves the color values from the sensor.
It can either read just the color values or the color values along with the ambient light intensity.

To read only the color values:

```
int r, g, b;  
APDS.readColor(r, g, b);
```

The variables `r`, `g`, and `b` will contain the updated color values. The value range is $0, 1, 2, \dots, 255$.

To read both the color values and ambient light intensity:

```
int r, g, b, a;  
APDS.readColor(r, g, b, a);
```

The variables `r`, `g`, and `b` will contain the updated color values, while `a` will contain the ambient light intensity. The value range is $0, 1, 2, \dots, 255$.

- `proximityAvailable()`: This method checks if proximity data is available.
 - Return value `TRUE`: Proximity data is available.
 - Return value `FALSE`: No proximity data is available.
- `readProximity()`: This method reads the proximity value.
 - Return value `int proximity = APDS.readProximity()`: Returns the proximity value.
 - Return value `-1`: Proximity could not be determined.
- `gestureAvailable()`: This method checks if a gesture has been detected.
 - Return value `TRUE`: A gesture has been detected.
 - Return value `FALSE`: No gesture detected.
- `setGestureSensitivity()`: Gesture detection is influenced by lighting conditions, speed, and distance of movement. This function allows you to adjust the sensitivity of gesture recognition. A higher value detects more gestures, but increases the chance of false positives. A lower value reduces the likelihood of detecting gestures.
The valid range is `1` to `100`.
The default value is `80`.
- `readGesture()`: If a gesture has been detected, it can be retrieved using the `readGesture()` method. The possible return values are:

- `GESTURE_UP`: An upward movement has been detected.
 - `GESTURE_DOWN`: A downward movement has been detected.
 - `GESTURE_LEFT`: A leftward movement has been detected.
 - `GESTURE_RIGHT`: A rightward movement has been detected.
 - `GESTURE_NONE`: No gesture has been detected.
- `setInterruptPin()`: This method sets the pin used to trigger a measurement. The pin is usually detected automatically, but can be set manually using this function.
 - If `-1` is passed, no pin is connected.
 - If `0` or a higher value is passed, that pin will be used.
 - The default value depends on the board.
 - `setLEDBlack(...)`: The sensor includes an infrared LED that can be temporarily boosted to provide higher brightness. It can be set to provide up to 300% of its normal power. This can be configured using this method.

Usage:

- `0`: Boost to 100% (default power).
- `1`: Boost to 150%.
- `2`: Boost to 200%.
- `3`: Boost to 300%.

Return values:

- `0`: Failure.
- `1`: Success.

13.4. Simple Function Tests

The following code examples can be used to test and try out all functions of the APDS-9960.

13.4.1. Example Color Detection

The library `Arduino_APDS9960` includes an example code demonstrating how to perform color detection with the sensor APDS-9960 on an Arduino Nano 33 BLE Sense. This example also tests the ambient light sensor, as both functionalities rely on the same underlying technology. The measured color channel intensities (Red, Green, and Blue) are displayed in the Serial Monitor for real-time observation.

The sensor APDS-9960 enables accurate RGB color detection by measuring the intensity of red, green, and blue light reflected from an object. These values can be visualized in the Serial Plotter, allowing users to track changes in color intensity over time through a graphical representation. This feature is particularly useful for applications that require real-time color monitoring or dynamic light adjustments.

13.4.2. Example Color Detection - Manual

WS:to do **13.4.3. Setting Up the Color Sensor Measurement Program**

To begin the process of creating the color sensor measurement program, open the Arduino Cloud Editor and navigate to the "Libraries" tab. Search for the "Arduino-APDS9960" library and download it. Afterward, click on "More info" to access the GitHub repository, where several examples are provided.

Next, connect the Arduino Nano 33 BLE Sense to the computer, ensuring that the Cloud Editor detects both the board and the corresponding port. If the board is not automatically recognized, follow the instructions to install the necessary plugin to enable the editor to detect the board. Confirm that the correct port is selected. Finally, upload the program to the Arduino, and by opening the serial monitor, the measured values will be recorded in the Arduino IDE.

13.4.4. Example

Color Detection - Code

First, serial communication is started with `Serial.begin(9600)` to send data to the computer. The command `APDS.begin()` initializes the sensor and in the event of a potential error, an error message is output via the serial interface.

The code operates within the function `loop()` of a sketch to continuously read and display color values from the APDS-9960 sensor. Initially, it checks if a color reading is available by using the method `APDS.colorAvailable()`. If no reading is available, the program briefly pauses for 5 milliseconds to avoid excessive CPU usage. Once a reading is ready, the method `APDS.readColor(r, g, b)` retrieves the color data and stores the red, green, and blue values in the respective variables. These values are then printed to the Serial Monitor, labeled clearly as "r", "g", and "b". A one-second delay is added before the program repeats the loop, allowing for clear intervals between consecutive readings.

13.4.5. Example Color Detection - File

The program can be found at

Listing 13.1.: Simple sketch using the sensor APDS9960 for colors

```

1000  /*
1001   APDS-9960 - Color Sensor
1002
1003   This example reads color data from the on-board APDS-9960 sensor of
1004   the
1005   Nano 33 BLE Sense and prints the color RGB (red, green, blue) values
1006   to the Serial Monitor once a second.
1007
1008   The circuit:
1009   - Arduino Nano 33 BLE Sense
1010
1011   This example code is in the public domain.
1012 */
1013
1014 #include <Arduino_APDS9960.h>
1015
1016 void setup() {
1017     Serial.begin(9600);
1018     while (!Serial);
1019
1020     if (!APDS.begin()) {

```

```

1020     Serial.println("Error initializing APDS-9960 sensor.");
1022 }
1024 void loop() {
1025     // check if a color reading is available
1026     while (!APDS.colorAvailable()) {
1027         delay(5);
1028     }
1029     int r, g, b;
1030
1031     // read the color
1032     APDS.readColor(r, g, b);
1033
1034     // print the values
1035     Serial.print("r = ");
1036     Serial.println(r);
1037     Serial.print("g = ");
1038     Serial.println(g);
1039     Serial.print("b = ");
1040     Serial.println(b);
1041     Serial.println();
1042
1043     // wait a bit before reading again
1044     delay(1000);
1045 }
```

..../Code/Nano33BLESense/APDS9960/TestAPDS9960Color/TestAPDS9960Color.ino

13.4.6. Proximity

A simple code for testing the proximity sensor is also provided by the library. The proximity sensor is designed for measuring distances of up to 100mm.

13.4.7. Proximity - Manual

Open the Arduino Cloud Editor and navigate to the Libraries tab. In the search bar, search for the library [Arduino_APDS9960](#). Once located, open the Examples section within the library and select the sketch “ProximitySensor”. Connect the Arduino Nano 33 BLE Sense to the computer. Check that the Cloud Editor recognizes the board by verifying if both the board and port are listed in the dropdown menu.

13.4.8. Example Proximity - Code

First, serial communication is started with `Serial.begin(9600)` to send data to the computer. The method `APDS.begin()` initializes the sensor and in the event of a potential error, an error message is output via the serial interface.

The code operates within the function `loop()` of a sketch to continuously read and display proximity values from the sensor APDS-9960. It begins by checking if a proximity reading is available using the method `APDS.proximityAvailable()`. If a reading is ready, the method `APDS.readProximity()` retrieves the proximity value, where `0` indicates a close object, `255` indicates a distant object, and `-1` signals an error in reading. This proximity value is then printed to the Serial Monitor. A delay of 100 milliseconds is added before the program repeats the loop, ensuring a brief interval between consecutive readings.

13.4.9. Example Proximity - File

The sketch can be found at

Listing 13.2.: Simple sketch using the sensor APDS9960 for measuring the proximity

```

1000  /*
1001   APDS-9960 — Proximity Sensor
1002
1003   This example reads proximity data from the on-board APDS-9960 sensor
1004   of the
1005   Nano 33 BLE Sense and prints the proximity value to the Serial Monitor
1006   every 100 ms.
1007
1008   The circuit:
1009   - Arduino Nano 33 BLE Sense
1010
1011   This example code is in the public domain.
1012 */
1013
1014 #include <Arduino_APDS9960.h>
1015
1016 void setup() {
1017     Serial.begin(9600);
1018     while (!Serial);
1019
1020     if (!APDS.begin()) {
1021         Serial.println("Error initializing APDS-9960 sensor!");
1022     }
1023
1024 void loop() {
1025     // check if a proximity reading is available
1026     if (APDS.proximityAvailable()) {
1027         // read the proximity
1028         // - 0 => close
1029         // - 255 => far
1030         // - -1 => error
1031         int proximity = APDS.readProximity();
1032
1033         // print value to the Serial Monitor
1034         Serial.println(proximity);
1035     }
1036
1037     // wait a bit before reading again
1038     delay(100);
1039 }
```

..../Code/Nano33BLESense/APDS9960/TestAPDS9960Proximity/TestAPDS9960Proximity.ino

13.4.10. Gesture Detection

13.4.11. Example Gesture Detection

The library [Arduino_APDS9960](#) also provides an example sketch demonstrating gesture detection with the sensor APDS-9960 on an Arduino Nano 33 BLE Sense. Some LED feedback signals have been programmed to provide quick visual feedback.

13.4.12. Example Gesture Detection - Manual

WS:to do

13.4.13. Example Gesture Detection - Code

In this example sketch, the library [Arduino_APDS9960](#) is integrated first and then the setup is created. In addition, the LED pins are configured so that they behave as outputs.

After that, serial communication is initialized, and the program verifies whether the sensor is correctly initialized using `APDS.begin()`. If initialization fails, an error message is printed. The gesture sensitivity can be adjusted via `APDS.setGestureSensitivity(value)`, where values range from 1 to 100, affecting detection accuracy and sensitivity. The sketch sets a default sensitivity and signals the start of gesture detection by turning off the RGB LEDs with `digitalWrite(LED_R, HIGH)`, `digitalWrite(LED_G, HIGH)`, and `digitalWrite(LED_B, HIGH)`.

In the function `loop()`, the code continuously checks for available gestures using `APDS.gestureAvailable()`. When a gesture is detected, it is read using `APDS.readGesture()` and interpreted within a `switch` statement. Each gesture corresponds to specific LED behavior:

- `GESTURE_UP`: The red LED is briefly turned on, then off after a 1-second delay.
- `GESTURE_DOWN`: The green LED is briefly turned on, then off after a 1-second delay.
- `GESTURE_LEFT`: The blue LED is briefly turned on, then off after a 1-second delay.
- `GESTURE_RIGHT`: All LEDs are turned on simultaneously, then off after a 1-second delay.

The `default` case ensures no action if an undefined gesture is detected.

13.4.14. Example Gesture Detection - File

The sketch can be found at

Listing 13.3.: Simple sketch using the sensor APDS9960 for gesture detection

```

1000 /*
1001  APDS9960 — Gesture Sensor
1002  This example reads gesture data from the on-board APDS9960 sensor of
1003  the
1004  Nano 33 BLE Sense and prints any detected gestures to the Serial
1005  Monitor.
1006  Gesture directions are as follows:
1007  — UP:    from USB connector towards antenna
1008  — DOWN:  from antenna towards USB connector
1009  — LEFT:  from analog pins side towards digital pins side
1010  — RIGHT: from digital pins side towards analog pins side
1011  The circuit:
1012  — Arduino Nano 33 BLE Sense
1013  This example code is in the public domain.
1014 */
1015 #include <Arduino_APDS9960.h>
1016
1017 void setup() {
1018     Serial.begin(9600);
1019     //Red
1020     pinMode(LED_R, OUTPUT);
1021     //Green
1022     pinMode(LED_G, OUTPUT);
1023     //Blue
1024     pinMode(LED_B, OUTPUT);
1025
1026     while (!Serial);
1027     if (!APDS.begin()) {
1028         Serial.println("Error initializing APDS9960 sensor!");
1029     }

```

```

1030 // for setGestureSensitivity(..) a value between 1 and 100 is required
1031 .
1032 // Higher values makes the gesture recognition more sensible but less
1033 // accurate
1034 // (a wrong gesture may be detected). Lower values makes the gesture
1035 // recognition
1036 // more accurate but less sensible (some gestures may be missed).
1037 // Default is 80
1038 //APDS.setGestureSensitivity(80);
1039 Serial.println("Detecting gestures ...");
1040 // Turning OFF the RGB LEDs
1041 digitalWrite(LED_R, HIGH);
1042 digitalWrite(LED_G, HIGH);
1043 digitalWrite(LED_B, HIGH);
1044 }
1045 void loop() {
1046   if (APDS.gestureAvailable()) {
1047     // a gesture was detected, read and print to serial monitor
1048     int gesture = APDS.readGesture();
1049     switch (gesture) {
1050       case GESTURE_UP:
1051         Serial.println("Detected UP gesture");
1052         digitalWrite(LED_R, LOW);
1053         delay(1000);
1054         digitalWrite(LED_R, HIGH);
1055         break;
1056       case GESTURE_DOWN:
1057         Serial.println("Detected DOWN gesture");
1058         digitalWrite(LED_G, LOW);
1059         delay(1000);
1060         digitalWrite(LED_G, HIGH);
1061         break;
1062       case GESTURE_LEFT:
1063         Serial.println("Detected LEFT gesture");
1064         digitalWrite(LED_B, LOW);
1065         delay(1000);
1066         digitalWrite(LED_B, HIGH);
1067         break;
1068       case GESTURE_RIGHT:
1069         Serial.println("Detected RIGHT gesture");
1070         digitalWrite(LED_B, LOW);
1071         digitalWrite(LED_R, LOW);
1072         digitalWrite(LED_G, LOW);
1073         delay(1000);
1074         digitalWrite(LED_B, HIGH);
1075         digitalWrite(LED_G, HIGH);
1076         digitalWrite(LED_R, HIGH);
1077         break;
1078     }
1079   }
1080 }
1081 }
```

..../Code/Nano33BLESense/APDS9960/TestAPDS9960Gesture/TestAPDS9960Gesture.ino

13.4.15. Troubleshoot

Errors in color detection can be caused by insufficient lighting in the room. Please ensure that the environment is bright enough for the color detection function.

13.5. Calibration Color Detection

13.6. Calibration Color Detection

Due to variations in environmental light conditions and the sensitivity of the sensor, calibration is required to ensure accurate color detection. This section outlines the calibration process using a standard color chart under constant lighting conditions. To carry out the calibration, the Arduino, a connection cable (USB A to USB Micro), and a laptop or computer with the appropriate Arduino development environment are required.

The proximity and gesture feature is pre-configured and factory-calibrated to detect proximity and gesture at a distance of 100mm, eliminating the need for customer calibration. [Bro24]

13.6.1. Calibration Setup

A standardized color chart is used, which provides reference values for perfect red, green, and blue under constant lighting conditions. The RGB values from the sensor are read as raw, uncalibrated data, which must be normalized to a standard range (0–255) for accurate color detection.

WS:citation, picture

13.6.2. Step 1: Measuring Reference Values

The first step in the calibration process is to measure the raw sensor values for each primary color (red, green, and blue) using the standardized color chart. By placing the color chart in front of the sensor and reading the raw RGB values, we can establish the maximum possible sensor readings for each color channel.

$$\text{RawRed} = \max(R_{\text{measured}})$$

$$\text{RawGreen} = \max(G_{\text{measured}})$$

$$\text{RawBlue} = \max(B_{\text{measured}})$$

These maximum values are obtained by positioning the chart at a fixed distance of one centimeter and ensuring constant light intensity.

13.6.3. Step 2: Normalizing the Sensor Values

Once the maximum sensor readings for red, green, and blue are obtained, the raw sensor data is normalized to a 0–255 scale. This ensures that the sensor's readings correspond to standard RGB values. The following formula is used for normalization:

$$R_{\text{calibrated}} = \frac{R_{\text{raw}}}{\text{RawRed}} \times 255$$

$$G_{\text{calibrated}} = \frac{G_{\text{raw}}}{\text{RawGreen}} \times 255$$

$$B_{\text{calibrated}} = \frac{B_{\text{raw}}}{\text{RawBlue}} \times 255$$

Where:

- $R_{\text{raw}}, G_{\text{raw}}, B_{\text{raw}}$ are the raw sensor values for each channel.
- RawRed, RawGreen, RawBlue are the maximum measured values from the standard color chart.

13.6.4. Step 3: Implementing the Calibration in Code

The normalization process is implemented directly in the Arduino code to adjust the sensor's readings. To begin calibration, open the Serial Monitor, enter "OK" in the command line, and press Enter to confirm. The following steps will then be explained through text output.

After the user initiates the calibration by typing "OK" into the Serial Monitor, the sketch starts. The sketch guides the user through the calibration of red, green, and blue colors, with each phase confirmed by entering "OK". The pins for the RGB LEDs are defined, and variables store the maximum calibration values for each color. Status variables control the sequence, ensuring each phase is completed before moving to the next. These maximum values are later used for accurate color detection in the Application code.

The following code section begins by initializing serial communication at a baud rate of 9600, enabling interaction through the serial monitor. Next, it sets the RGB LED pins as outputs and turns on all LEDs to create white light, which helps capture accurate color measurements. The code then initializes the sensor APDS-9960, checking for successful setup. If the sensor fails to initialize, an error message is printed, and the program halts. If successful, a message confirms initialization and prompts the user to type "OK" in the serial monitor to proceed with the calibration process explanation.

The next code segment in the loop first checks if the user has entered "OK" to start the calibration process. After confirmation, it displays an explanation and instructions. After showing the explanation, it waits for the user to confirm by typing "OK" again, which then initiates the red color calibration by prompting the user to place the red chart in front of the sensor. The loop pauses at each step until the user confirms.

WS:Screen shot?

Each color calibration starts after the user types "OK" in the Serial Monitor. The sketch measures the highest detected value for each color over 10 seconds and sets it as the calibration maximum. After all colors are calibrated, it displays the final maximum values for red, green, and blue in the Serial Monitor, completing the process. The `maxRed`, `maxGreen`, and `maxBlue` values can later be entered into the application sketch to apply the calibration to the sketch.

13.6.5. Calibration File

The sketch can be found at

Listing 13.4.: APDS-9960: Example Calibration

```

1000 // Code to determine the calibration factors for the application 'ApplicationAPDS9960.ino'.
1001 // To start, open the serial monitor and type "OK" into the command line
1002 // , then press Enter.
1003 // After reading the information , always confirm with "OK".
1004 // file: APDS9960Calibration.ino
1005
1006 #include <Arduino_APDS9960.h>
1007
1008 // RGB LED Pins for the Arduino Nano 33 BLE Sense
1009 const int redPin = LEDR;
1010 const int greenPin = LEDG;
1011 const int bluePin = LEDB;
1012
1013 // Maximum RGB values based on calibration measurements
1014 int maxRed = 0;

```

```
1016 int maxGreen = 0;
1017 int maxBlue = 0;

1018 // Calibration state variables
1019 bool initialOKConfirmed = false;           // Confirmation to start
1020   explanation
1021 bool explanationShown = false;             // Flag to show calibration
1022   explanation
1023 bool explanationConfirmed = false;         // Confirmation that explanation
1024   was read
1025 bool redCalibrated = false;                // Flag for completed red
1026   calibration
1027 bool greenCalibrated = false;              // Flag for completed green
1028   calibration
1029 bool blueCalibrated = false;               // Flag for completed blue
1029   calibration
1030 bool readyForRed = false;                  // Flag to begin red calibration
1031 bool readyForGreen = false;                 // Flag to begin green calibration
1032 bool readyForBlue = false;                  // Flag to begin blue calibration

1033 void setup() {
1034   Serial.begin(9600);

1035   // Set up RGB LED pins as outputs
1036   pinMode(redPin, OUTPUT);
1037   pinMode(greenPin, OUTPUT);
1038   pinMode(bluePin, OUTPUT);

1039   // Turn on all LEDs (for white light) to capture accurate color
1040   digitalWrite(redPin, LOW);
1041   digitalWrite(greenPin, LOW);
1042   digitalWrite(bluePin, LOW);

1043   // Initialize the APDS-9960 sensor and check for success
1044   if (!APDS.begin()) {
1045     Serial.println("Error initializing the APDS-9960 sensor!");
1046     while (1); // Stop program if sensor fails to initialize
1047   }

1048   Serial.println("Sensor successfully initialized.");
1049   Serial.println("Type 'OK' to proceed to the explanation of the
1050   calibration process.");
1051 }

1052 void loop() {
1053   int r = 0, g = 0, b = 0, a = 0; // Variables to hold RGB and ambient
1054   values

1055   // Initial confirmation to proceed to calibration explanation
1056   if (!initialOKConfirmed) {
1057     if (Serial.available() > 0) {
1058       String input = Serial.readStringUntil('\n');
1059       input.trim(); // Remove whitespace
1060       if (input.equalsIgnoreCase("OK")) {
1061         initialOKConfirmed = true;
1062         explanationShown = true;
1063         Serial.println("Explanation:");
1064         Serial.println("Welcome to the calibration process.");
1065         Serial.println("You will be prompted to calibrate with the red,
1066         green, and blue color charts.");
1067         Serial.println("Follow the instructions and confirm each step by
1068         typing 'OK'.");
1069         Serial.println("Type 'OK' to begin the calibration process.");
1070       }
1071     }
1072   }
1073   return; // Exit loop until 'OK' is entered initially
1074 }

1075 // Wait for user confirmation after showing explanation
```

```

1074 if (explanationShown && !explanationConfirmed) {
1075     if (Serial.available() > 0) {
1076         String input = Serial.readStringUntil('\n');
1077         input.trim(); // Remove whitespace
1078         if (input.equalsIgnoreCase("OK")) {
1079             explanationConfirmed = true;
1080             readyForRed = true;
1081             Serial.println("Explanation confirmed.");
1082             Serial.println("Place the red color chart in front of the sensor");
1083             and type 'OK' when ready.");
1084         }
1085     }
1086     return; // Exit loop until explanation is confirmed
1087 }

1088 // Red calibration step
1089 if (readyForRed && !redCalibrated) {
1090     if (Serial.available() > 0) {
1091         String input = Serial.readStringUntil('\n');
1092         input.trim(); // Remove whitespace
1093         if (input.equalsIgnoreCase("OK")) {
1094             readyForRed = false;
1095             Serial.println("Measuring red value for 10 seconds... ");
1096
1097             unsigned long startTime = millis(); // Start time for 10-second
1098             calibration
1099             int maxMeasuredRed = 0;
1100
1101             // Measure red value for 10 seconds
1102             while (millis() - startTime < 10000) {
1103                 if (APDS.colorAvailable()) {
1104                     APDS.readColor(r, g, b, a);
1105                     if (r > maxMeasuredRed) {
1106                         maxMeasuredRed = r; // Update max red value if current is
1107                         higher
1108                     }
1109                 }
1110                 delay(100); // Avoid sensor read flooding
1111             }
1112             maxRed = maxMeasuredRed; // Set max red after calibration
1113             redCalibrated = true;
1114             readyForGreen = true;
1115             Serial.println("Red calibration completed.");
1116             Serial.println("Place the green color chart in front of the");
1117             sensor and type 'OK' when ready.");
1118         }
1119     }
1120     return; // Exit loop until red calibration is confirmed
1121 }

1122 // Green calibration step
1123 if (readyForGreen && !greenCalibrated) {
1124     if (Serial.available() > 0) {
1125         String input = Serial.readStringUntil('\n');
1126         input.trim(); // Remove whitespace
1127         if (input.equalsIgnoreCase("OK")) {
1128             readyForGreen = false;
1129             Serial.println("Measuring green value for 10 seconds... ");
1130
1131             unsigned long startTime = millis(); // Start time for 10-second
1132             calibration
1133             int maxMeasuredGreen = 0;
1134
1135             // Measure green value for 10 seconds
1136             while (millis() - startTime < 10000) {
1137                 if (APDS.colorAvailable()) {
1138                     APDS.readColor(r, g, b, a);
1139                     if (g > maxMeasuredGreen) {

```

```

1138         maxMeasuredGreen = g; // Update max green if current is
1139         higher
1140     }
1141     delay(100); // Avoid sensor read flooding
1142 }

1144     maxGreen = maxMeasuredGreen; // Set max green after calibration
1145     greenCalibrated = true;
1146     readyForBlue = true;
1147     Serial.println("Green calibration completed.");
1148     Serial.println("Place the blue color chart in front of the
1149     sensor and type 'OK' when ready.");
1150   }
1151   return; // Exit loop until green calibration is confirmed
1152 }

// Blue calibration step
1154 if (readyForBlue && !blueCalibrated) {
1155   if (Serial.available() > 0) {
1156     String input = Serial.readStringUntil('\n');
1157     input.trim(); // Remove whitespace
1158     if (input.equalsIgnoreCase("OK")) {
1159       readyForBlue = false;
1160       Serial.println("Measuring blue value for 10 seconds... ");
1161
1162       unsigned long startTime = millis(); // Start time for 10-second
1163       calibration
1164       int maxMeasuredBlue = 0;
1165
1166       // Measure blue value for 10 seconds
1167       while (millis() - startTime < 10000) {
1168         if (APDS.colorAvailable()) {
1169           APDS.readColor(r, g, b, a);
1170           if (b > maxMeasuredBlue) {
1171             maxMeasuredBlue = b; // Update max blue if current is
1172             higher
1173           }
1174         }
1175       }
1176       delay(100); // Avoid sensor read flooding
1177     }
1178     maxBlue = maxMeasuredBlue; // Set max blue after calibration
1179     blueCalibrated = true;
1180     Serial.println("Blue calibration completed.");
1181     Serial.println("Calibration complete.");
1182
1183     // Display final calibration results
1184     Serial.println("Calibration results:");
1185     Serial.print("maxRed = ");
1186     Serial.println(maxRed);
1187     Serial.print("maxGreen = ");
1188     Serial.println(maxGreen);
1189     Serial.print("maxBlue = ");
1190     Serial.println(maxBlue);
1191   }
1192 }
1193 return; // Exit loop until blue calibration is confirmed
1194 }
```

..../Code/Nano33BLESense/APDS9960/APDS9960Calibration/APDS9960Calibration.ino

13.6.6. Step 4: Validating the Calibration

To ensure the calibration is successful, the sensor readings should be tested again using the standard color chart. The calibrated values for red, green, and blue should be close to 255 for the respective pure colors on the chart. If the readings deviate significantly, further adjustment of the maximum reference values may be necessary.

The calibration of the sensor APDS-9960 is essential for accurate RGB detection. By using a standard color chart and constant lighting conditions, the sensor's raw values can be normalized to provide consistent and reliable color readings. This process can be further refined by adjusting for specific environmental factors or sensor placement.

13.7. Tests

13.7.1. Simple Function Test

13.7.2. Test all Functions

13.8. Simple Application

Similarly, by following all the steps for uploading and compiling the sketch we can see the results of sensor APDS-9960 on Serial Monitor, too. For seeing the different output, we can change the input for the sensor too, e.g: for color detection we can switch the colors, for gesture detection we can also switch the gestures, and for proximity also do the same. The resulted output as shown in the figure. 13.3

WS:rewrite and extend

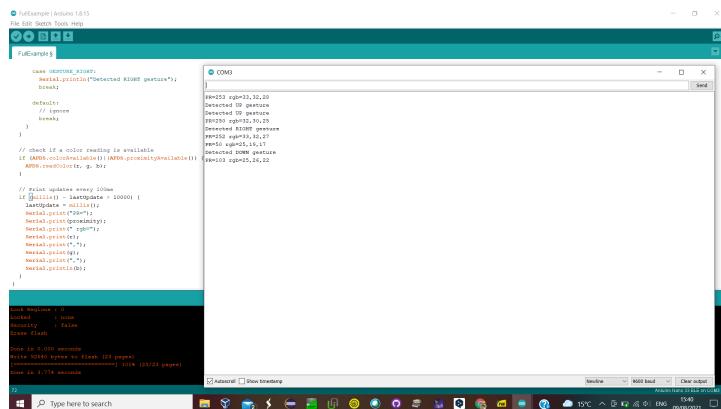


Figure 13.3.: Gesture, Proximity, Color Sensor Output Window

We can also run the single functionality of this sensor too, e.g; if we just need to capture the color of product, we can also run the color detection program. It depends upon the application and we can implement our application and modify the code as per our desire results.

Listing 13.5.: Simple sketch using the sensor APDS9960

```

1000 /**
1001 * @file TestAPDS9960.ino
1002 *
1003 * @brief Arduino sketch for color sensor APDS9960 and SSD1306 OLED
1004 *        display
1005 *
1006 * @author Elmar Wings
1007 * @version 1.0
1008 */

```

```
1008 // Library for sensor APDS9960
1010 #include <Arduino_APDS9960.h>
1012 // Library SSD1306 for text-only output and connection
1013 // with cable via the I2C protocol
1014 #include <SSD1306AsciiWire.h>
1016
1017 #define I2C_ADDRESS 0x3C /*< Enter the address of the OLED */
1018
1019 SSD1306AsciiWire oled; /*< Set the name of the OLED */
1020
1021 // Initialization of the measurement button
1022 const int TASTER_PIN = 11; /*< Pin number for measurement button */
1023
1024 // Initialization of the LED pin
1025 const int LED_PIN = 12; /*< Pin number for LED */
1026
1027 /**
1028 * @brief Setup function for Arduino
1029 */
1030 void setup() {
1031     Serial.begin(9600); /*< Initialize serial communication at 9600 baud
1032     */
1033
1034     // Define pin as INPUT and activate internal pull-up resistor
1035     pinMode(TASTER_PIN, INPUT);
1036     // Define pin as OUTPUT
1037     pinMode(LED_PIN, OUTPUT);
1038
1039 /**
1040 * @brief Initialize APDS9960 sensor
1041 */
1042 if (!APDS.begin()) {
1043     Serial.println("Error initializing APDS9960 sensor.");
1044 }
1045
1046 // Arduino is hereby registered in the I2C bus,
1047 // as it is to be registered as a master,
1048 // the address SEARCH SOURCE!!!! is omitted.
1049 Wire.begin();
1050 // Clock frequency in Hertz,
1051 // Standard: 100,000
1052 // Fast mode: 400,000
1053 Wire.setClock(400000L);
1054 // first screen size, then reference to I2C address
1055 oled.begin(&Adafruit128x64, I2C_ADDRESS);
1056 oled.setFont(System5x7);
1057 oled.setCursor(0, 40);
1058 oled.println("INITIALISIERUNG!\n");
1059
1060 /**
1061 * @brief Flash LED three times during initialization
1062 */
1063 for (int zaehler = 1; zaehler < 4; zaehler = zaehler + 1) {
1064     delay(2000);
1065     digitalWrite(LED_PIN, HIGH);
1066     delay(200);
1067     digitalWrite(LED_PIN, LOW);
1068     delay(200);
1069 }
1070 oled.clear();
1071 }
1072
1073 /**
1074 * @brief Main loop function for Arduino
1075 */
```

```

1076 void loop() {
1077     oled.println("READY!");
1078
1080     /**
1081      * @brief Check whether color detection is available
1082      */
1083     while (!APDS.colorAvailable()) {
1084         delay(5);
1085     }
1086
1087     /**
1088      * @brief Check whether distance measurement is available
1089      */
1090     while (!APDS.proximityAvailable()) {
1091         delay(5);
1092     }
1093
1094     /**
1095      * @brief Create integer variables to save
1096      *        the measured values for red, green and blue
1097      */
1098     int r, g, b, tasterCount{ 0 };
1099
1100    /**
1101     * @brief Status query of the measurement button
1102     */
1103    bool zustandTaster = digitalRead(TASTER_PIN);
1104
1105    if (zustandTaster == 0) {
1106        tasterCount = 1;
1107    }
1108
1109    /**
1110     * @brief If clause switch to measuring mode when the button is
1111     *        pressed
1112     */
1113    if (tasterCount == 1) {
1114        int proximity = APDS.readProximity();
1115
1116        if (proximity != -1) {
1117            Serial.print("Abstand: ");
1118            Serial.println(proximity);
1119
1120            if (proximity < 20) {
1121                oled.clear();
1122                digitalWrite(LED_PIN, HIGH);
1123
1124                /**
1125                 * @brief 3x Color scan and save in the 3 variables r, g and b
1126                 */
1127                for (int i = 0; i < 3; i++) {
1128                    APDS.readColor(r, g, b);
1129                    delay(1500);
1130                }
1131
1132                /**
1133                 * @brief Display largest value via integrated RGB LED,
1134                 *        option for troubleshooting
1135                 */
1136                if (r > g & r > b) {
1137                    digitalWrite(LED_R, LOW);
1138                    digitalWrite(LED_G, HIGH);
1139                    digitalWrite(LED_B, HIGH);
1140                } else if (g > r & g > b) {
1141                    digitalWrite(LED_G, LOW);
1142                    digitalWrite(LED_R, HIGH);
1143                    digitalWrite(LED_B, HIGH);
1144                } else if (b > g & b > r) {
1145                    digitalWrite(LED_B, LOW);
1146                }
1147            }
1148        }
1149    }
1150}
```

```

1144         digitalWrite(LED_R, HIGH);
1145         digitalWrite(LED_G, HIGH);
1146     } else {
1147         digitalWrite(LED_R, HIGH);
1148         digitalWrite(LED_G, HIGH);
1149         digitalWrite(LED_B, HIGH);
1150     }
1151
1152     /**
1153      * @brief Output values in the serial monitor
1154      *        for testing and error analysis
1155      */
1156     Serial.print("Red light = ");
1157     Serial.println(r);
1158     Serial.print("Green light = ");
1159     Serial.println(g);
1160     Serial.print("Blue light = ");
1161     Serial.println(b);
1162     Serial.println();
1163
1164     /**
1165      * @brief Display color on OLED display
1166      */
1167     if (r > g & r > b) {
1168         oled.println("Das Objekt ist rot!");
1169     } else if (g > r & g > b) {
1170         oled.println("Das Objekt ist gruen!");
1171     } else if (b > g & b > r) {
1172         oled.println("Das Objekt ist blau!");
1173     }
1174
1175     delay(4000);
1176     oled.clear();
1177 } else {
1178     oled.clear();
1179     oled.println("Kein Objekt \nvorhanden!\n");
1180     oled.println("Halten Sie ein\nObjekt vor den \nSensor \noder
1181 veraendern Sie\ndie Position!");
1182     digitalWrite(LED_R, HIGH);
1183     digitalWrite(LED_G, HIGH);
1184     digitalWrite(LED_B, HIGH);
1185     delay(4000);
1186     oled.clear();
1187 } else {
1188     oled.clear();
1189     oled.println("Abstandsmessung fehlgeschlagen!");
1190 } else {
1191     // Switch off LED if not pressed
1192     digitalWrite(LED_PIN, LOW);
1193
1194     oled.setFont(System5x7);
1195     oled.setCursor(0, 40);
1196     oled.println("READY!");
1197
1198     digitalWrite(LED_R, HIGH);
1199     digitalWrite(LED_G, HIGH);
1200     digitalWrite(LED_B, HIGH);
1201 }
1202
1203 // Set button to 0
1204 zustandTaster = 0;
1205

```

..../Code/Nano33BLESense/APDS9960/TestAPDS9960/TestAPDS9960.ino

13.9. Further Readings

- SparkFun Electronics, “APDS-9960 RGB and Gesture Sensor Hookup Guide”. A comprehensive guide that explains the functionality of the APDS-9960 and provides step-by-step instructions for implementation. Available at: <https://learn.sparkfun.com/tutorials/apds-9960-rgb-and-gesture-sensor-hookup-guide/all>, [Hym24]
- Maker Guides, “APDS-9960 Gesture and Color Sensor with Arduino”. This tutorial provides a hands-on introduction to using the APDS-9960 sensor with Arduino, including gesture and color recognition. Available at: <https://www.makerguides.com/apds-9960-gesture-and-color-sensor-with-arduino/>, [Mae24]

14. Sensor APDS 9960 Gesture

Introduction

WS:cite books,
applications, board

14.0.1. Sensormodul APDS 9960 for Gesture

Function of the Gesture Detection Sensor

The gesture recognition of the APDS-9960 is based on four additional angled photodiodes

- Up – Hand movement upwards.
- Down – Hand movement downwards.
- Left – Hand movement to the left.
- Right – Hand movement to the right.

which detect reflected infrared light (IR) from the integrated IR-LED. By analyzing the reflection, which measures direction-dependent changes using the photodiodes, the sensor can recognize movements and gestures such as swiping motions in different directions (e.g., up, down, left, right). The sensor converts this information into digital data, such as speed, direction, and distance of the movement. The 32-record FIFO register allows the sensor to temporarily store motion data, ensuring continuous processing. [Ava15]

14.1. General

General description
cite books

14.2. Specific Sensor

cite board

14.3. Specification

- cite data sheet
- Circuit Diagram

14.4. Bibliothek

- 14.4.1. Description**
- 14.4.2. Installation**
- 14.4.3. Functions**
- 14.4.4. Example - Manual**
- 14.4.5. Example**
- 14.4.6. Example - Code**
- 14.4.7. Example - Files**

14.5. Calibration

cite method

14.6. Simple Code**14.7. Simple Application****14.8. Tests**

- 14.8.1. Simple Function Test**
- 14.8.2. Test all Functions**

14.9. Simple Application**14.10. Further Readings**

15. Sensor APDS 9960 Color

The general functionality of a color sensor is based on the reflection of white light from the object to be measured. White light contains wavelengths that cover the entire visible spectrum (p. 189 [Ryb13]). Due to the molecular structure of the object or chemical processes, such as photosynthesis, certain wavelengths are absorbed while others are reflected. The reflected light rays that reach the color sensor are spectrally selected by color filters or a diffraction grating [HS23].

15.1. General

Color filters: By using parallel red, green, and blue color filters, the intensities of the RGB components are evaluated by three photodiodes behind the filters [HS23].

Diffraction grating: As light passes through a diffraction grating, the physical diffraction effect occurs. This effect, due to the varying wavelengths of light, produces an interference pattern that enables the spectral decomposition of the light. The spectrum of the incoming light is spatially dispersed, similar to the refraction in a prism (p. 208 ff. [Ryb13]). The intensity of the color components can be measured by an array of spatially arranged photodiodes. Diffraction gratings offer higher resolution than color filters [HS23].

Photodiodes are semiconductors that generate electrical energy through the internal photoelectric effect (p. 220 [Ryb13]). The signal from the photodiodes is amplified and stored in a data register through analog-to-digital conversion. The measurement is taken over a specific period. By accumulating the RGB data, the measured color can finally be determined [Ava15].

Color sensors are typically used in image processing and quality control applications [Ava15].

As shown in Figure 15.1, the incoming light first passes through UV and IR filters. It is then separated into its red, green, blue, and unfiltered components by color filters. These filtered components are subsequently converted from analog to digital units, as described below.

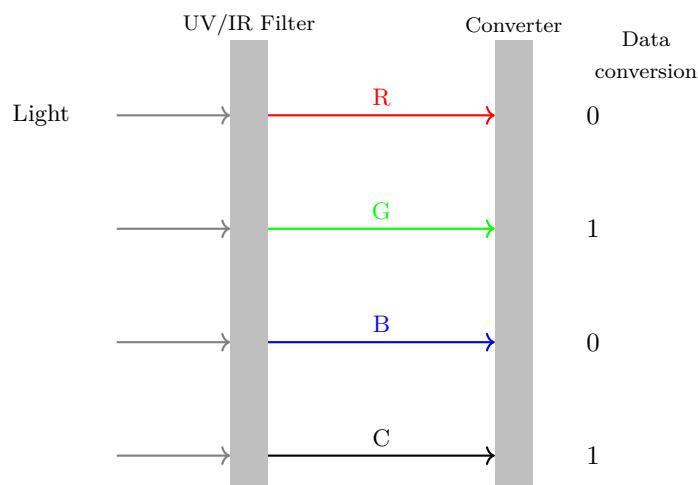


Figure 15.1.: Schematic of the main function of a color sensor

The block diagram 15.2 shows the processing flow of the APDS-9660 from light detection to communication via the I²C protocol with the Arduino. The individual steps are explained as follows:

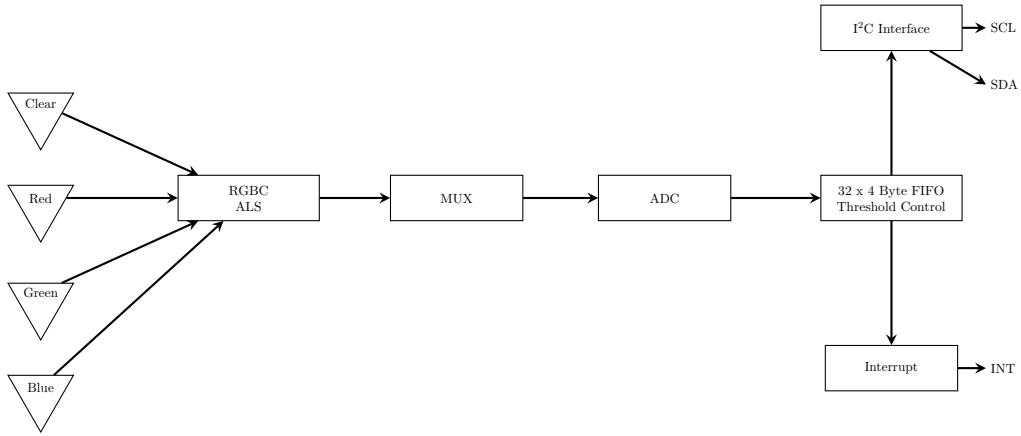


Figure 15.2.: Functional Block Diagram

WS:The graphic is not optimal. Description of the individual blocks is not clear. What does RGBC and ALS mean?

The photodiodes convert the intensities of the color components of the filtered light and the total light into electrical signals. These RGBC (Red, Green, Blue, Clear) and ALS (Ambient Light Sensor) signals are forwarded to the next processing stage using the multiplexer (MUX).

The MUX allows for the sequential processing of the color signals, as the ADC (Analog-to-Digital Converter) can only process one signal at a time. The ADC converts the analog signal selected by the MUX into a digital value. This conversion is necessary to make the signal suitable for digital processing and transmission.

The digital value is stored in a FIFO (First In, First Out) data register, which can store up to 32 data values (4 bytes each). A threshold controller monitors the data and triggers an interrupt when defined limits are reached. The detailed process is explained in Chapter Data Processing 15.1.

The stored data is transmitted to the Arduino via an I²C bus. The I²C bus uses two lines: the SCL (Serial Clock Line) for the clock and the SDA (Serial Data Line) for data transmission. The detailed functioning of the I²C bus is explained in Chapter ?? Protokoll I²C. [Ava15].

Data Processing

The color detection begins at the photodiodes with RGBC detection and ends with 16-bit values stored in the RGBC data registers. The signal from the photodiode array is accumulated over a period defined by the value in the ATIME register (ALS ADC Integration Time) before the results are transferred to the RGBCDATA registers. The gain factor can be set from 1x to 64x, which is controlled via the CONTROL<AGAIN> bit (ALS Gain Control). Performance parameters such as accuracy, resolution, conversion speed, and energy consumption can be adjusted to meet the application requirements.

Between measurements, a customizable, energy-efficient waiting period is maintained, the duration of which is determined by the control bits WEN (Wait Enable), WTIME (Wait Time), and WLONG (Wait Long Enable). The waiting time can range from 0 seconds to a maximum of 8.54 seconds.

An interrupt is triggered when the clear channel values exceed the thresholds defined in the AILTL/AIHTL (ALS low threshold, lower byte/ALS high threshold, lower byte) or AILTH/AIHTH (ALS low threshold, upper byte/ALS high threshold, upper

byte) threshold registers. To avoid unwanted or false interrupts, a persistence filter is integrated, ensuring that an interrupt is triggered only when a defined number of consecutive values fall outside the thresholds. This threshold is set in the APERS register (ALS Interrupt Persistence). If a value is within the thresholds, the persistence counter is reset. If the analog circuit is saturated, the ASAT bit is set, indicating potentially inaccurate RGBCDATA results. The AINT (ALS Interrupt) and CPSAT (Clear Diode Saturation) bits can be queried at any time via I²C. However, for AINT to trigger a hardware interrupt on the INT pin, the AIEN bit (ALS Interrupt Enable) must be set. Saturation of the analog-to-digital converter can be detected via the CPSAT bit; to enable this function, the CPSIEN bit (Clear Diode Saturation Interrupt Enable) must be set. The AVALID bit (ALS Valid) is reset by reading the RGBCDATA. ASAT and AINT can be reset via the CICLEAR (Clear Channel Interrupt Clear) or ACLEAR (All Non-Gesture Interrupt Clear) bits.

The RGBC results can be used to determine the color temperature in Kelvin or the ambient light intensity in Lux.

[Ava15]

15.2. Specific Sensor

cite board

15.3. Specification

- cite data sheet
- Circuit Diagram

15.4. Bibliothek

15.4.1. Description

15.4.2. Installation

15.4.3. Functions

15.4.4. Example - Manual

15.4.5. Example

15.4.6. Example - Code

15.4.7. Example - Files

15.5. Calibration

cite method

15.6. Simple Code**15.7. Simple Application****15.8. Tests****15.8.1. Simple Function Test****15.8.2. Test all Functions****15.9. Simple Application****15.10. Further Readings**

16. Sensor APDS 9960 Proximity

The proximity sensor is part of the gesture detection sensor and detects top-bottom gestures based on the temporal change in the measured distance. The proximity detection measures the distance between the sensor and an object by capturing the reflected IR light. The integrated IR LED emits light, which is measured by the same photodiodes as in gesture detection. The stronger the reflection, the closer the object is. The proximity sensor can also detect events such as the approach or withdrawal of an object and trigger corresponding interrupts when predefined thresholds are exceeded. To ensure reliable measurements, the sensor compensates for unwanted reflections by adjusting offset values and suppresses interference from ambient light. [Ava15]

16.1. General

General description
cite books

16.2. Specific Sensor

cite board

16.3. Specification

- cite data sheet
- Circuit Diagram

16.4. Bibliothek

16.4.1. Description

16.4.2. Installation

16.4.3. Functions

16.4.4. Example - Manual

16.4.5. Example

16.4.6. Example - Code

16.4.7. Example - Files

16.5. Calibration

cite method

16.6. Simple Code

16.7. Simple Application

16.8. Tests

16.8.1. Simple Function Test

16.8.2. Test all Functions

16.9. Simple Application

16.10. Further Readings

17. Sensor Ambient Light

The ambient light sensor (ALS) of the APDS-9960 works with the same photodiode array and measures the intensity of the ambient light. For this purpose, the UV and IR light filters are installed in front of the photodiodes to ensure accurate measurements of visible light. The sensor converts the measured light intensity into 16-bit data, which enables high resolution and accuracy. A function of programmable gain and customizable integration time ensures that the sensor can operate accurately in different lighting conditions, such as dim or very bright light. Another important feature of the sensor is its ability to correctly detect the intensity of ambient light even behind dark glass, which is particularly useful for devices with tinted covers. [Ava15]

17.1. General

General description
cite books

17.2. Specific Sensor

cite board

17.3. Specification

- cite data sheet
- Circuit Diagram

17.4. Bibliothek

17.4.1. Description

17.4.2. Installation

17.4.3. Functions

17.4.4. Example - Manual

17.4.5. Example

17.4.6. Example - Code

17.4.7. Example - Files

17.5. Calibration

cite method

17.6. Simple Code

17.7. Simple Application

17.8. Tests

17.8.1. Simple Function Test

17.8.2. Test all Functions

17.9. Simple Application

17.10. Further Readings

18. Application of the Color Sensor with OLED Display

The color detection machine utilizes an Arduino Nano 33 BLE Sense with an APDS-9960 sensor for reliable recognition of red, green, and blue colors, making it suitable for various industrial automation processes. Every 750 ms, the machine reads the color of an object and displays the result on an OLED screen. Before each color reading, the proximity sensor checks if an object is close to the sensor, if so, the white LED activates to improve sensor accuracy in low-light conditions.

18.1. Manual

The color detection machine consists of the Arduino Nano 33 BLE Sense and a 1.30" IIC OLED display, which need to be connected according to the circuit diagram in Figure 18.1.

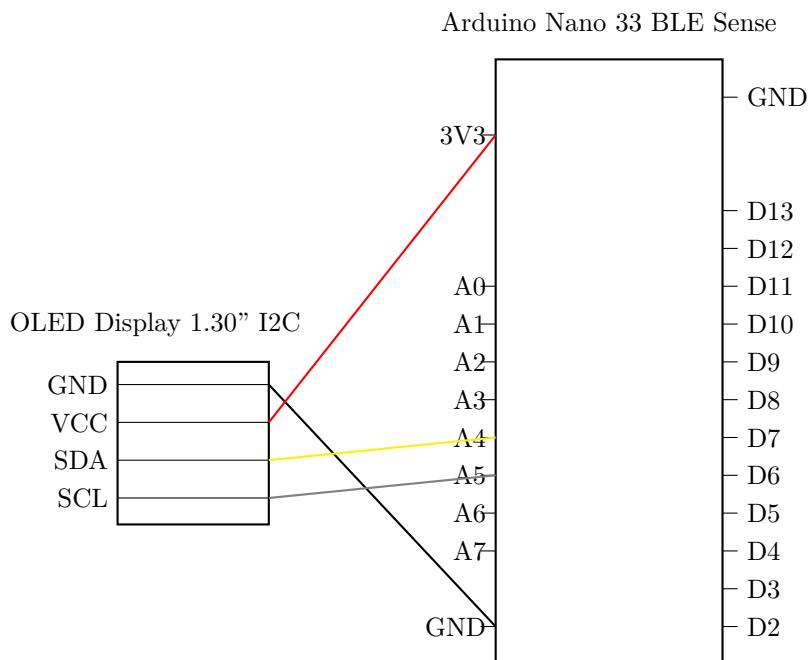


Figure 18.1.: Circuit diagram for connecting the Arduino Nano 33 BLE Sense with a 1.30" IIC OLED display.

Use a laptop or computer to open the ApplicationAPDS9960 code in the Arduino IDE. Before uploading the code, enter the maxRed, maxGreen, and maxBlue calibration values to ensure accurate color detection by the machine.

Once the Arduino is programmed, the color detection machine can start. Place an object approximately one centimeter in front of the APDS-9960 sensor on the Arduino. The OLED display will show important instructions and display detected colors. Each time an object is removed and replaced, the sensor will read the color in its usual

interval. Ensure proper lighting, and calibrate the sensor under the same conditions in which it will operate.

18.2. Code Explanation

The following section provides an explanation of the application code:

The first part of the code initializes the Arduino Nano 33 BLE Sense with the APDS-9960 sensor and the 1.30" IIC OLED display. It includes pre-determined calibration values for red, green, and blue (maxRed, maxGreen, maxBlue) as integer values. In the `setup()` function, it starts serial communication, sets the LED pin as an output, and initializes the APDS-9960 sensor, printing an error message if the initialization fails. It also sets up the OLED display to show an "INITIALIZING!" message. Afterward, the LED blinks three times to indicate that the setup is complete, and the display is cleared.

Listing 18.1.: Application of the sensor APDS9960: Setup

```

1000 // Code for using the APDS-9960 sensor as a color recognition machine.
1002 // An Arduino Nano 33 BLE Sense and a 1.30" IIC OLED display are
     required.
1003 // In addition, the calibration factors maxRed, maxGreen, and maxBlue
     should
1004 // be determined in advance using APDS9960Calibration.ino.
1006 // file: ApplicationAPDS9960.ino

1008 #include <Arduino.h>
1009 #include <U8g2lib.h>
1010 #include <Wire.h>
1011 #include <Arduino_APDS9960.h>

1012 #define I2C_ADDRESS 0x3C // I2C address for the OLED display
1013 const int LedPin = 12; // LED pin number

1016 // Calibration values obtained from the calibration program
1017 const int maxRed = 31; // Maximum calibrated red value
1018 const int maxGreen = 15; // Maximum calibrated green value
1019 const int maxBlue = 13; // Maximum calibrated blue value
1020
1021 // Initialize display object for a 128x64 OLED with I2C interface
1022 U8G2_SH1106_128X64_NONAME_F_HW_I2C oled(U8G2_R0, /* reset==*/
     U8X8_PIN_NONE);

1024 void setup()
{
1025     Serial.begin(9600); // Start serial communication
1026     pinMode(LedPin, OUTPUT); // Set the LED pin as an output
1027
1028     // Initialize APDS9960 sensor, log error if initialization fails
1029     if (!APDS.begin()) {
1030         Serial.println("Error initializing APDS9960 sensor.");
1031     }
1032
1033     // Initialize OLED display and display an "INITIALIZING" message
1034     oled.begin();
1035     oled.clearBuffer();
1036     oled.setFont(u8g2_font_ncenB08_tr); // Set display font
1037     oled.drawStr(10, 10, "INITIALIZING!");
1038     oled.sendBuffer(); // Send the buffer to display
1039
1040     // Blink LED three times as a visual indication that setup is complete
1041     for (int counter = 0; counter < 3; counter++) {
1042         digitalWrite(LedPin, HIGH);

```

```

1044     delay(200);
1045     digitalWrite(LedPin, LOW);
1046     delay(200);
1047 }
1048 oled.clearBuffer(); // Clear the display buffer after setup
1049 oled.sendBuffer();
1050 }

..../Code/Nano33BLESense/APDS9960/ApplicationAPDS9960/ApplicationAPDS9960.ino

```

In the following `loop()` function, the code displays a "READY" message on the OLED display and then sets a timeout of 1000 ms to wait for color data to become available, decrementing the timeout in steps of 5 ms. If color data is not available within the timeout, a "Color measurement timeout" message is printed in the serial monitor, and the function returns. The code then resets the timeout to 1000 ms and similarly waits for proximity data. If proximity data does not become available within the timeout, a "Proximity measurement timeout" message is printed, and the function returns.

Listing 18.2.: Application of the sensor APDS9960: Loop

```

1000 void loop()
1001 {
1002     // Display "READY" message
1003     oled.clearBuffer();
1004     oled.setCursor(10, 20);
1005     oled.print("READY");
1006     oled.sendBuffer();

1007     int timeout = 1000; // Timeout value for waiting for color data
1008     // Wait until color data is available or timeout occurs
1009     while (!APDS.colorAvailable() && timeout > 0) {
1010         delay(5);
1011         timeout -= 5;
1012     }
1013     if (timeout <= 0) {
1014         Serial.println("Color measurement timeout");
1015         return;
1016     }

1017     timeout = 1000; // Reset timeout for proximity data
1018     // Wait until proximity data is available or timeout occurs
1019     while (!APDS.proximityAvailable() && timeout > 0) {
1020         delay(5);
1021         timeout -= 5;
1022     }
1023     if (timeout <= 0) {
1024         Serial.println("Proximity measurement timeout");
1025         return;
1026     }
1027 }

..../Code/Nano33BLESense/APDS9960/ApplicationAPDS9960/ApplicationAPDS9960.ino

```

In the next section of the `loop()` function, the code reads the proximity value using `APDS.readProximity()`. If a valid proximity value is detected, it prints the proximity value to the serial monitor. If the proximity value is below a specified threshold (indicating an object is close), the OLED display is cleared, and the LED is turned on as a visual indication. To capture accurate color data, all LEDs are turned on to emit white light, and multiple color readings are averaged.

The red, green, and blue color values are then mapped to a 0-255 range using predefined calibration values (`maxRed`, `maxGreen`, `maxBlue`) and are constrained within the 0-255 range. The dominant color is determined based on the highest value among red, green, and blue, and the result is displayed on the OLED. If no object is detected within the proximity threshold, a message prompts the user to hold an object near the sensor or change its position.

If the proximity reading fails, an error message is displayed on the OLED. After processing, the code turns off the LED and resets the RGB LEDs to an off state.

Listing 18.3.: Application of the sensor APDS9960: Main function

```

1000 int r, g, b;
1001 int proximity = APDS.readProximity(); // Read proximity value
1002
1003 if (proximity != -1) {
1004     Serial.print("Proximity: ");
1005     Serial.println(proximity);
1006
1007     if (proximity < 20) { // If proximity is below threshold
1008         oled.clearBuffer();
1009         digitalWrite(LedPin, HIGH); // Turn on LED as visual indication
1010
1011         // Turn on all LEDs (for white light) to capture accurate color
1012         digitalWrite(LEDR, LOW);
1013         digitalWrite(LEDG, LOW);
1014         digitalWrite(LEDB, LOW);
1015
1016         // Average multiple color readings for accuracy
1017         int r_total = 0, g_total = 0, b_total = 0;
1018         for (int i = 0; i < 5; i++) {
1019             APDS.readColor(r, g, b);
1020             r_total += r;
1021             g_total += g;
1022             b_total += b;
1023             delay(750); // Delay between readings
1024         }
1025         r = r_total / 5;
1026         g = g_total / 5;
1027         b = b_total / 5;
1028
1029         // Map color values to a 0-255 range using calibration values
1030         r = map(r, 0, maxRed, 0, 255);
1031         g = map(g, 0, maxGreen, 0, 255);
1032         b = map(b, 0, maxBlue, 0, 255);
1033
1034         // Constrain values to the 0-255 range
1035         r = constrain(r, 0, 255);
1036         g = constrain(g, 0, 255);
1037         b = constrain(b, 0, 255);
1038
1039         // Print color values for debugging
1040         Serial.print("Red light = ");
1041         Serial.println(r);
1042         Serial.print("Green light = ");
1043         Serial.println(g);
1044         Serial.print("Blue light = ");
1045         Serial.println(b);
1046         Serial.println();
1047
1048         // Determine the dominant color and display result on OLED
1049         oled.setCursor(10, 20);
1050         if (r > g && r > b) {
1051             oled.print("The object is red.");
1052         } else if (g > r && g > b) {
1053             oled.print("The object is green.");
1054         } else if (b > g && b > r) {
1055             oled.print("The object is blue.");
1056         }
1057
1058         oled.sendBuffer(); // Display the color information
1059         delay(4000);
1060         oled.clearBuffer();
1061         oled.sendBuffer();
1062     } else {
1063         // Display message if no object is found within proximity

```

```

1064     threshold
1065     oled.clearBuffer();
1066     oled.setCursor(10, 20);
1067     oled.print("No object found.");
1068     oled.setCursor(10, 30);
1069     oled.print("Hold an object");
1070     oled.setCursor(10, 40);
1071     oled.print("in front of sensor");
1072     oled.setCursor(10, 50);
1073     oled.print("or change position.");
1074     oled.sendBuffer();
1075     delay(4000);
1076     oled.clearBuffer();
1077     oled.sendBuffer();
1078   }
1079 } else {
1080   // Display error if proximity reading fails
1081   oled.clearBuffer();
1082   oled.setCursor(10, 20);
1083   oled.print("Proximity failed!");
1084   oled.sendBuffer();
1085 }
1086 // Turn off LED and set RGB LEDs to off state after processing
1087 digitalWrite(LedPin, LOW);
1088 digitalWrite(LED_R, HIGH);
1089 digitalWrite(LED_G, HIGH);
1090 digitalWrite(LED_B, HIGH);
1091 }
```

..../Code/Nano33BLESense/APDS9960/ApplicationAPDS9960/ApplicationAPDS9960.ino

18.3. Application Test

The color recognition machine reliably identifies the colors red, green, and blue from the standardized color chart. After calibration, it can also recognize other colors like light blue, yellow-green, or orange as red, green, and blue. This flexibility ensures that objects do not need to be perfectly red, green, or blue to be detected. Activating the white LED when an object approaches proves to be very practical, as it serves as a feedback mechanism alongside the OLED display, indicating to the user that color recognition is now in progress.

18.4. Improvement Suggestions

A key improvement would be to integrate the color recognition machine into a stable, standalone setup, so it does not need to be handheld, allowing objects to be fixed in place in front of the sensor. This would eliminate potential issues like hand movement artifacts during detection.

Another enhancement would be to equip the device with a battery, enabling portable, laptop-free operation. To conserve battery life, the white LED is deliberately programmed to illuminate only during color measurement.

In an industrial application, adequate lighting should be ensured, potentially by adding external lighting that could be integrated into the code or kept continuously on. For use in dusty or humid environments, a thin protective cover over the sensor could prevent damage or contamination, allowing for easy cleaning or replacement as needed.

19. Mikrophone MP34DT05-A

- Allgemeine Information über Mikrophone
- Spezielle Betrachtung des Mikrofons
- Kalibrierung und Test des Mikrofons
- Algorithmen zur Auswertung eines Mikrofons
- Auswertung des speziellen Mikrofons
- Programmierung des Mikrofons

Some Arduino boards have a microphone on board which can be used for the application. Das Mikrofon des Arduino Nano 33 BLE Sense ist ein ultrakompaktes Modul, das den MP34DT05-Sensor verwendet. Es wurde speziell für den Arduino Nano 33 BLE Sense entwickelt. Dieser Sensor nutzt die Puls-Dichtemodulation (PDM), um ein analoges Signal in ein binäres Signal umzuwandeln. Durch PDM wird ein analoges Signal in binäre Werte umgewandelt. [STM21]

WS:cite books,
applications, board

19.0.1. Ton und Frequenzen

Ton, auch bekannt als Schall, ist genauer eine einzige Schallfrequenz, die eine mechanische Deformation in dem Medium Luft verursacht, welche sich als Longitudinalwelle ausbreitet.

Die Frequenz f , gemessen in Hertz (Hz), stellt die Schwingungen über Luftdruckschwankung pro Sekunde in einer Schallwelle dar. Beispielsweise kann der Mensch einen Frequenzbereich von etwa 20 Hz bis 20 000 Hz wahrnehmen. Es gelten folgende Formeln für die Beschreibung von Schall:

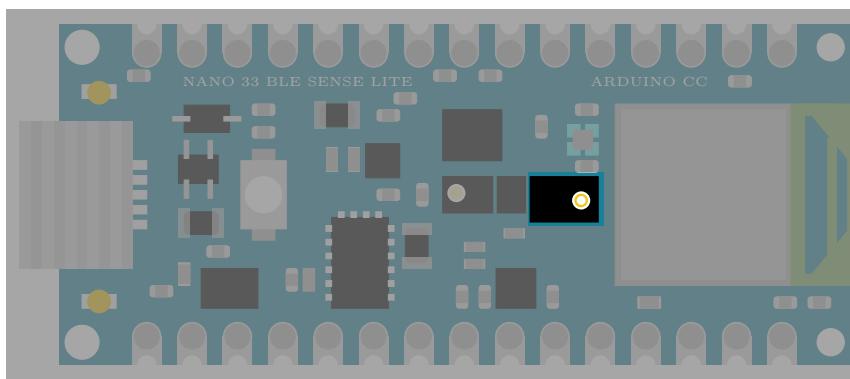


Figure 19.1.: Arduino Nano 33 BLE Sense's RGB LED with Pin 22, Pin 23, and Pin 24

- Schalldruck p :

$$p(t) = \hat{p} \cdot \cos(\omega t - \phi) \quad (19.1)$$

mit \hat{p} : Amplitude des Schalldrucks,

$\omega = 2 \cdot \pi \cdot f$: Kreisfrequenz,

ϕ : Phasenverschiebung

- Schallschnelle ν :

$$\nu(t) = \hat{\nu} \cdot \cos(\omega \cdot t) \quad (19.2)$$

mit $\hat{\nu}$: Amplitude des Schalldrucks

- Schallgeschwindigkeit c (für Gase):

$$c = \sqrt{(\kappa\rho)/p} \quad (19.3)$$

Für Luft beträgt die Schallgeschwindigkeit $c_{Luft} = 331,2 \text{ m/s}$.

Je schneller die Abfolge der Schwingungen, desto höher ist die Frequenz bzw. der Ton. Je größer die Amplitude ist, desto lauter ist der Ton.

Die Lautstärke wird durch die Amplitude beschrieben, die durch den Schalldruck erzeugt wird. Die Einheit für die Lautstärke wird in Dezibel (dB) angegeben. Hierfür wird der Abstand der Auslenkung zum Nullpunkt gemessen. Als Nullpunkt wurde, mit $P_{ref} = 2 \cdot 10^{-5} \text{ Pa}$ die menschliche Hörschwelle festgelegt. Ein negativer dB-Wert ist somit nicht mehr für den Menschen hörbar. ([Poh17])

In der Tonverarbeitung werden diese Schwingungen mechanisch, meist über Mikrofone aufgenommen.

19.0.2. Unterschied Ton, Klang und Geräusch

Ein Ton ist beschreibbar durch eine harmonische Funktion mit unveränderlicher Frequenz. Ein Klang ist hingegen ein komplexeres, aber periodisches Schallereignis. Wesentlicher Bestandteil der Charakterisierung eines Klanges ist die Überlagerung eines Grundtons mit seinen Oberschwingungen. Ein Geräusch ist generell nicht periodisch und zeigt veränderliche Strukturen.

WS:cite

19.0.3. Mikrofone

Die prinzipiellen Funktionsweisen verschiedener Mikrofonarten sind sehr ähnlich zueinander. Die Schallwellen versetzen eine Membran in Schwingung, diese wird durch verschiedene Methoden in elektrische Signale umgewandelt (siehe Abbildung 19.2).

Da Computerprozessoren mit diesen analogen elektrischen Signalen nicht rechnen können, werden sie durch elektronische Wandler in digitale Signale umgewandelt. Zunächst werden einige Methoden, die geeignet sind, Schwingungen der Membran in elektrische Signale umzuwandeln, betrachtet. Dies kann durch physikalische Prinzipien wie elektrodynamische Induktion, Kapazitätsänderungen, Piezoelektrische-Elemente, durch die Änderung des elektrischen Widerstands (Kohlemikrofone) usw. umgesetzt

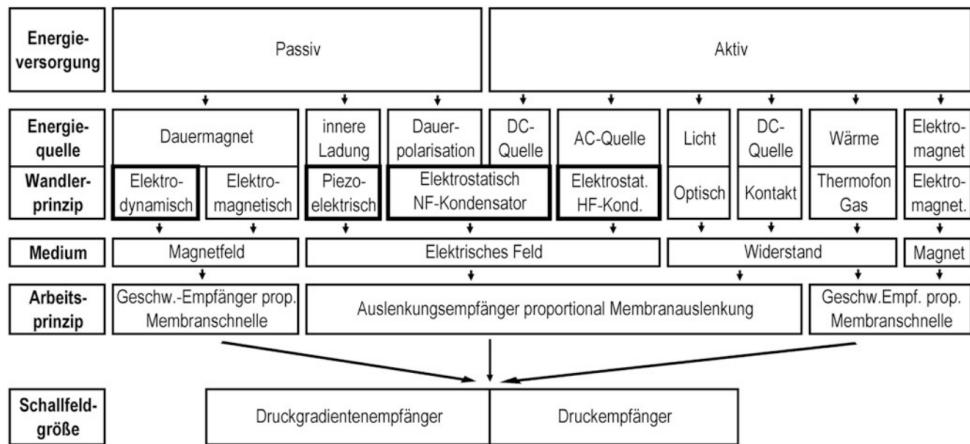


Figure 19.2.: Liste verschiedener Mikrofonarten mit Funktionsprinzip

werden. Neben diesen heute üblichen Methoden der Schallwandlungen gibt es auch noch eine Reihe anderer Methoden.

In diesem Fall wird das Mikrofon Thomann t.bone SC 420 benutzt. Dem Datenblatt des Mikrofons 19.3 sind folgende Angaben zu entnehmen:

- Beim Mikrofon handelt es sich um ein Kondensatormikrofon.
- Der nutzbare Frequenzbereich liegt bei 80 Hz - 18000 Hz.
- Das digitale Signal hat eine Auflösung von 16 bit (mono) bei einer Abtastfrequenz von 48 kHz.
- Es hat eine supernierenförmige Richtcharakteristik.
- Die Arbeitstemperatur liegt zwischen 0°C und 40°C, die erlaubte relative Luftfeuchtigkeit zwischen 20% und 80%.

Technical specifications

■ Connection/power supply	USB
■ Transducer principle	Condenser
■ Polar pattern	Supercardioid
■ Frequency range (-10 dB)	80 – 18,000 Hz
■ Sampling rate	16 Bit/48 kHz
■ Current consumption (mA)	150
■ Dimensions (Ø x D), without shock mount	51 mm x 187 mm
■ Thread	5/8-inch
■ Weight	285 g
■ Ambient conditions	Temperature range 0 °C...40 °C Relative humidity 20%...80% (non-condensing)

Figure 19.3.: Datenblatt des betrachteten Mikrofons

Kondensatormikrofon

Beim Kondensatormikrofon (siehe Abbildung 19.4) bildet die Membran eine Elektrode eines Kondensators. Durch die Bewegung der Membran ändert sich der Abstand zu der Gegenelektrode und damit die Kapazität des Kondensators. Der Kondensator wird über einen hochohmigen Widerstand geladen. Die Kapazitätsänderung führt dann zu einer Änderungen der am Kondensator anliegenden Spannung. Diese Spannungsänderung kann mit einem geeigneten Spannungsmesser gemessen werden.

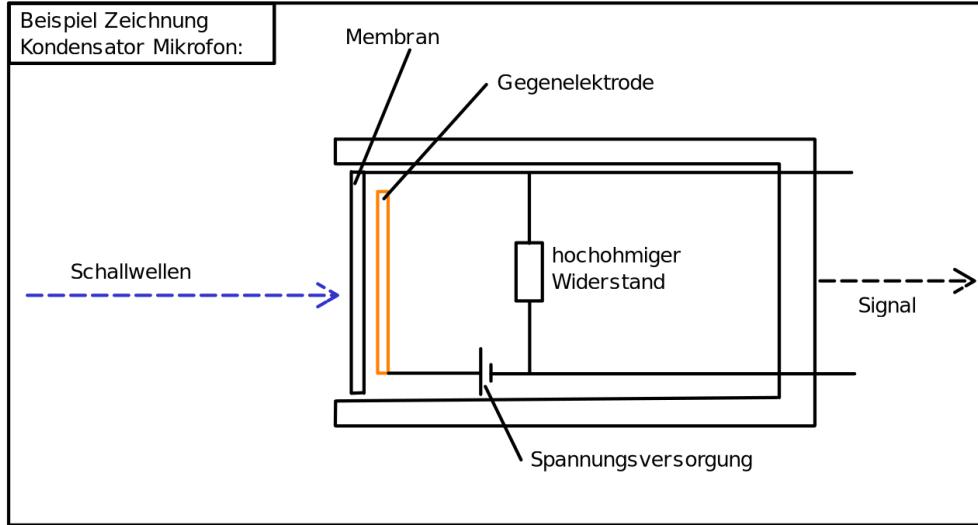


Figure 19.4.: Aufbau eines Kondensator Mikrofons

Anschließend muss dieses analoge, elektrische Signal noch in ein digitales Signal umgewandelt werden (siehe Abbildung 19.5). Dies ist notwendig, da Computerprozessoren nur mit digitalen Signalen arbeiten können. Dazu werden AD-Wandler (Analog zu Digital Wandler) genutzt. Die AD-Wandler können entweder im Mikrofon selbst eingebaut oder vom Mikrofon getrennt sein. AD-Wandler sind in heutigen Computern selbst eingebaut. Im professionellen Bereich werden aber meist hochwertige externe AD-Wandler eingesetzt. In diesem Projekt wird ein Mikrofon benutzt, bei dem der AD-Wandler direkt im Mikrofon verbaut ist. Das digitale Signal wird als binäres Signal über eine USB Schnittstelle an den Computer übermittelt.

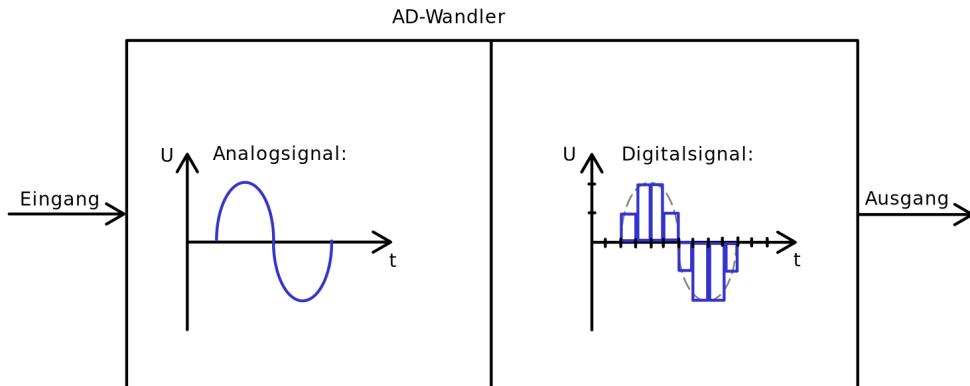


Figure 19.5.: Funktionsprinzip eines Analog zu Digital Wandlers

([Wei20])

19.0.4. Wavdatei

Unkomprimierte Audiodaten werden üblicherweise im Wave-Dateiformat gespeichert (siehe Abbildung 19.7). Dieses Dateiformat ist ein von Microsoft und IBM, im Jahr 1991 veröffentlichtes Format zur digitalen Speicherung von Audiodaten. Es setzt auf dem RIFF (Resource Interchange File Format) Dateiformat auf. Das besondere beim RIFF-Datenformat ist, dass in diesen Dateien Informationen über das Format des

Dateiinhalts am Anfang der Datei gespeichert sind. Bei WAV-Audiodateien handelt es sich um eine RIFF Datei mit einem WAVE Container. Normalerweise sind die Audiodateien als PCM (Pulse-Code-Modulation)-Rohdaten enthalten, wobei PCM das Pulsmodulationsverfahren ist, mit dem die analogen Signale in digitale Signale umgesetzt werden. Meist enthält eine RIFF Audiodatei einen WAVE Container, welcher zwei Untercontainer enthält: Den fmt (Format) Container, der das Format der Audiodaten und deren Speicherung beschreibt und einen data Container, welcher die eigentlichen Audiodaten in dem Format enthält, das im fmt Container beschrieben wird.

Eine Wave-Datei enthält in den ersten 4 Byte das Wort RIFF im ASCII-Format. Die folgenden vier Byte enthalten Größen der noch folgenden Daten im little-endian-Format. Der letzte Teil der Beschreibung sind noch einmal 4 Byte, die den Dateityp angeben. In unserem Fall also immer das Wort WAVE im ASCII-Format.

ASCII steht für „American Standard Code for Information Interchange“ und ist einer 7-Bit Zeichenkodierung, die der ISO-646 entspricht.

Die Zeichenkodierung besteht aus 128 Zeichen wovon 95 druckbar und 33 nicht druckbar sind. Einige hiervon sind:

- Buchstaben: A, B, C, ..., Z, a, b, c, ..., z
- Ziffern: 0, 1, 2, ..., 9
- Interpunktionszeichen: !, ", #,\$, %, &,

Eine ausführliche Darstellung ist in Abbildung 19.6.

Wichtig anzumerken ist, dass jedes Zeichen einem definiertem Zahlenwert entspricht.

	0 0x00	1 0x01	2 0x02	3 0x03	4 0x04	5 0x05	6 0x06	7 0x07	8 0x08	9 0x09	10 0x0A	11 0x0B	12 0x0C	13 0x0D	14 0x0E	15 0x0F
0 0x00	N_{V_L} 0	s_{Q_H} 1	s_{T_K} 2	E_{T_K} 3	E_{O_Y} 4	E_{S_Q} 5	A_{C_K} 6	B_{E_L} 7	B_S 8	H_T 9	L_P 10	V_T 11	F_P 12	C_R 13	s_S 14	s_I 15
16 0x10	D_{E_E} 16	D_{C_1} 17	D_{C_2} 18	D_{C_3} 19	D_{C_4} 20	N_{h_K} 21	s_{T_N} 22	E_{T_B} 23	C_{h_N} 24	E_M 25	s_{U_B} 26	E_{S_C} 27	F_S 28	G_S 29	R_S 30	U_S 31
32 0x20	s_p 32	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
48 0x30	0 48	1 49	2 50	3 51	4 52	5 53	6 54	7 55	8 56	9 57	:	;	<	=	>	?
64 0x40	@ 64	A 65	B 66	C 67	D 68	E 69	F 70	G 71	H 72	I 73	J 74	K 75	L 76	M 77	N 78	O 79
80 0x50	P 80	Q 81	R 82	S 83	T 84	U 85	V 86	W 87	X 88	Y 89	Z 90	[91	\ 92] 93	^ 94	- 95
96 0x60	` 96	a 97	b 98	c 99	d 100	e 101	f 102	g 103	h 104	i 105	j 106	k 107	l 108	m 109	n 110	o 111
112 0x70	p 112	q 113	r 114	s 115	t 116	u 117	v 118	w 119	x 120	y 121	z 122	{ 123	 124	 125	~ 126	D_{E_L} 127

Figure 19.6.: Symbole und numerische Werte des ASCII Formats

Die Zusatzwerte little endian(LE)/big endian(BE) geben an, von welcher Reihenfolge die Zahlenwerte der Bytes betrachtet werden.

- BE, auch als big-endian bekannt, speichert das höchstwertige Byte zuerst. Wenn mehrere Bytes gelesen werden, ist das erste Byte (oder die niedrigste Speicheradresse) das größte.
- LE, auch als little-endian bekannt, speichert das niederwertigste Byte zuerst. Wenn mehrere Bytes gelesen werden, ist das erste Byte (oder die niedrigste Speicheradresse) das kleinste.

In dem Container „fmt“ (Format) sind sämtliche Angaben zum Aufbau der nachfolgenden Audiodatei spezifiziert.

- Die ersten 4 Bytes leiten das Format Container mit dem Wort „fmt“ ein. Also die Buchstaben f, m, t und ein Leerzeichen. (ASCII) (big endian).
- (Subchunk1Size) Die nächsten 4 Bytes geben an, wie groß der restliche fmt Container noch ist. (little endian).
- (AudioFormat) Die nächsten zwei Bytes geben die Art an, wie die Audiodaten gespeichert sind. (üblicherweise 1 = PCM unkomprimiert) (little endian).
- (NumChannels) In den nächsten zwei Bytes steht, wie viele Kanäle in der Audiodatei enthalten sind (mono = 1, stereo = 2, ...).
- (SampleRate) Die nächsten 4 Bytes geben die Abtastrate des Signals in Hertz an.
- (ByteRate) In den nächsten 4 Bytes ist die Bandbreite des Audiosignals angegeben. Diese wird in Bytes pro Sekunde (Bytes/s) angegeben.
- (BlockAlign) In den nächsten zwei Bytes wird angegeben, wie viele Byte ein Block enthält. Ein Block entspricht einem Sample mal die Anzahl der Kanäle. (Bits pro Audiowert * Anzahl der Kanäle / 8).
- (BitsPerSample) In den hier letzten beiden Bytes des Beispiels wird angegeben, wie viele Bit ein Audiowert enthält. 8 Bit entsprechen einem Byte.

Würde bei den Audiodaten etwas Anderes als PCM verwendet werden, würde nach dem jetzigen Abschnitt noch ein Abschnitt folgen, der dieses Format spezifiziert. Als letztes sind im data-Container die wirklichen Audiodaten gespeichert, wie oben spezifiziert wurde. Eingeleitet wird dieser Container mit dem Wort „data“ in den ersten 4 Bytes (ASCII). In den darauf folgenden 4 Bytes wird die Größe der tatsächlichen Audiodaten in Byte (little Endian) angegeben.

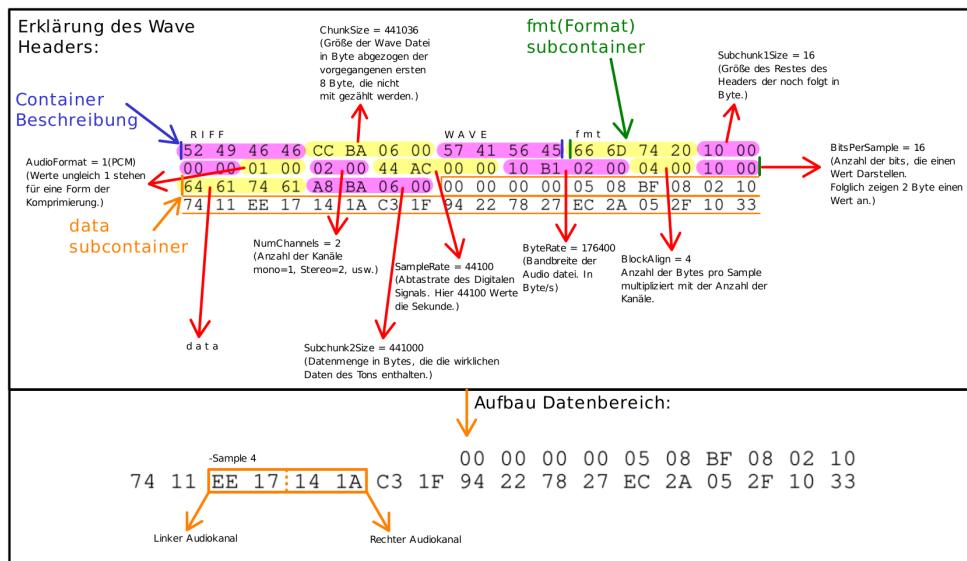


Figure 19.7.: Aufbau des Wave Dateiformats mit Erklärung [Wil03]

19.0.5. Package PyAudio Version 0.2.14

Die Bibliothek PyAudio ermöglicht den Zugriff auf ein Mikrofon. In diesem Fall wird das im Computer verbaute Mikrofon verwendet.

Zur Erläuterung der Einbindung der Bibliothek in einen Code wird ein Sample zum Abspielen einer Wave-Datei gezeigt.

```
import wave
import sys

import pyaudio

CHUNK = 1024
FORMAT = pyaudio.paInt16
CHANNELS = 1 if sys.platform == 'darwin' else 2
RATE = 44100
RECORD_SECONDS = 5

with wave.open('output.wav', 'wb') as wf:
    p = pyaudio.PyAudio()
    wf.setchannels(CHANNELS)
    wf.setsampwidth(p.get_sample_size(FORMAT))
    wf.setframerate(RATE)

    stream = p.open(format=FORMAT, channels=CHANNELS,
                     rate=RATE, input=True)

    print('Recording...')
    for _ in range(0, RATE // CHUNK * RECORD_SECONDS):
        wf.writeframes(stream.read(CHUNK))
    print('Done')

    stream.close()
    p.terminate()
```

Es ist zu sehen, dass zunächst durch den Befehl „`import pyaudio`“ die Bibliothek importiert wird. Mit dem Befehl `pyaudio.PyAudio()` kann nun auf die Bibliothek zugegriffen werden. `pyaudio.PyAudio.open()` öffnet als Nächstes einen Stream, das bedeutet, dass ein kontinuierlich bestehender Datenfluss ermöglicht wird, aus dem Audiodaten gelesen werden können mit `pyaudio.PyAudio.Stream.read()`. Um nun den Stream, also den kontinuierlichen Audiodatenfluss, zu beenden, wird `stream.close()` verwendet. Um den Zugriff auf das Mikrofon abzubrechen, verwendet man `p.terminate()`. (vgl. PyAudio Documentation — PyAudio 0.2.14 documentation (mit.edu))

Die Bibliothek PyAudio besitzt jedoch noch weitere Funktionen. Die für dieses Projekt wichtigste Funktion ist jedoch der Zugriff auf Lautsprecher und Mikrofon, sowie der Aufbau eines Streams für einen kontinuierlichen Audiodatenfluss. [Pha06]

19.0.6. Package Wave Version 3.12

Die Bibliothek Wave ist bereits in Python integriert und ermöglicht das Lesen von Wave-Dateien.

Zur Erläuterung ist nachfolgend ein Sample zu sehen.

```
import numpy as np
import wave
```

```
FRAMES_PER_SECOND = 44100

def sound_wave(frequency, num_seconds):
    time = np.arange(0, num_seconds, 1 / FRAMES_PER_SECOND)
    amplitude = np.sin(2 * np.pi * frequency * time)
    return np.clip(
        np.round(amplitude * 32768),
        -32768,
        32767,
    ).astype("<h")

left_channel = sound_wave(440, 2.5)
right_channel = sound_wave(480, 2.5)
stereo_frames = np.dstack((left_channel, right_channel)).flatten()

with wave.open("output.wav", mode="wb") as wav_file:
    wav_file.setnchannels(2)
    wav_file.setsampwidth(2)
    wav_file.setframerate(FRAMES_PER_SECOND)
    wav_file.writeframes(stereo_frames)
```

Das oben gezeigte Sample erstellt eine Audiodatei mit einer Frequenz von 440 Hz, die über den linken Lautsprecher ausgegeben wird und einer Frequenz von 480 Hz, die über den rechten Lautsprecher ausgegeben wird. Beide Frequenzen werden zeitgleich für eine Dauer von 2,5 Sekunden ausgegeben.

Die Nutzung der Bibliothek `wave` beginnt mit dem Import mittels `import wave`. Danach wird mithilfe von `sound_wave` eine Sinuswelle erzeugt. Nun wird die Frequenz definiert, die über den linken und den rechten Lautsprecher ausgegeben wird, mit der Funktion `frequency`. Mit `num_seconds` wird jetzt noch die Dauer in Sekunden für den linken und rechten Lautsprecher festgelegt. In Zeile 17 werden nun die Töne, die durch `sound_wave(frequency, num_seconds)` definiert wurden, zusammengeführt, sodass sie durch `with wave.open("output.wav", mode="wb") as wav_file:` in eine neu erstellte Wave-Datei mit Namen `output.wav` geschrieben werden können. [Fou24b]

19.0.7. Package NumPy Version 1.26

NumPy ist eine sehr große und umfassende Bibliothek. Die Funktion, die für die erläuterte Aufgabenstellung von Interesse ist, ist die schnelle Verarbeitung von Daten, daher wird auch nur diese Funktion beschrieben.

```
import sys
import numpy as np
import pyaudio

CHUNK = 1024
FORMAT = pyaudio.paInt16
CHANNELS = 1 if sys.platform == 'darwin' else 2
RATE = 44100
RECORD_SECONDS = 5

p = pyaudio.PyAudio()

stream = p.open(format=FORMAT, channels=CHANNELS,
```

```

rate=RATE, input=TRUE)

data = np.array([])

print('Recording...')
for _ in range(0, RATE // CHUNK * RECORD_SECONDS):
    in_data = stream.read(CHUNK)
    in_data = np.frombuffer(in_data,np.int16)
    data = np.concatenate((data,in_data))
    print('Done')

MAX_LENGTH = 1024
data = data[0::2]
data = data[-data.size//1024*1024:]

data = data.reshape(-1,data.size//1024)
data = np.mean(data, axis=1)

print(data)

stream.close()
p.terminate()

```

Das hier gezeigte Sample wurde inspiriert durch das Sample, das bereits zur Erklärung der Bibliothek [PyAudio](#) verwendet wurde. Dieses wurde um ein paar Zeilen zur Bibliothek [NumPy](#) ergänzt, um die hier relevante Funktion der Bibliothek erklären zu können.

Gestartet wird mit dem Befehl `import numpy as np`. Zudem wird in dieser Zeile der Bibliothek der Alias `np` zugeordnet, wodurch im Folgendem Code die Funktionen von [NumPy](#) durch die Verwendung des Alias aufgerufen werden können.

Die Bibliothek [PyAudio](#) stellt die Audiodaten in binärer Form dar. Da die Daten in dieser Form noch nicht ausgewertet werden können, ist es notwendig diese durch den Befehl `np.frombuffer(in_data,np.int16)` in einem Array darzustellen. Mit dem Befehl `np.concatenate((data,in_data))` können nun die gerade erstellten Daten und die ursprünglichen Daten zusammengefügt werden. Von Zeile 24 bis 26 werden alle Daten, die von der rechten Seite ausgegeben werden, herausgefiltert, sodass nur die Daten der linken Seite übrigbleiben. In Zeile 26 wird nun noch die Länge des Arrays auf eine konstante Ziellänge gebracht. Dabei muss die Ziellänge durch 1024 teilbar sein, was durch `MAX_LENGTH` definiert wird. Nun wird in der Zeile 28 das Array in ein zweidimensionales Array mit der Ziellänge 1024 umgeformt. Zeile 29 fasst die Arrays der zweiten Dimension in ihr jeweiliges arithmetisches Mittel zusammen. Nun muss das Ganze nur noch mit dem Befehl `print(data)` ausgegeben, und mit `stream.close()` geschlossen werden. [The22]

19.0.8. Package [Guizero](#) Version 1.5.0

[Guizero](#) ist eine Bibliothek, die auf der in Python enthaltenden Bibliothek Tkinter aufbaut. Die Bibliothek bietet die Möglichkeit, ein Dashboard zu erstellen, und bietet dazu einige Widges und Funktionen, die die Gestaltung des Dashboards erleichtern. Im Vergleich zu [Tkinter](#) ermöglicht [Guizero](#) eine einfachere Bedienung.

```

from guizero import App, MenuBar
def file_function():
    print("File option")

```

```
def edit_function():
    print("Edit option")

    app = App()
    menubar = MenuBar(app,
                      toplevel=["File", "Edit"],
                      options=[[
                        ["File option 1", file_function], ["File option 2",
                        file_function]],
                        [[["Edit option 1", edit_function], ["Edit option 2",
                        edit_function]]]
                      ])
    app.display()
```

Das Sample zeigt die Verwendung der Funktion zur Erstellung einer `MenuBar` mithilfe der Bibliothek `Guizero`. Zunächst erfolgt wieder das Importieren der gewünschten Klassen `App` und `MenuBar` des Moduls `Guizero` mit dem Befehl `from guizero import App, MenuBar`. Als allererstes erstellt die Funktion `App` ein Fenster. `MenuBar` kann daraufhin eine Menüleiste erstellen mit den Oberpunkten `File` und `Edit` und den Unterpunkten `File option 1`, `File option 2`, `Edit option 1` und `Edit option 2`. Dabei werden den Unterpunkten auch noch Funktionen zugewiesen, hier `file_function` und `edit_function`. [SO20]

19.0.9. Package `Threading` (Version 3.7)

`Threading` ist eine Bibliothek, die eine Parallelisierung arbeitsintensiver Aufgaben ermöglicht.

```
import logging
import threading
import time

def thread_function(name):
    logging.info("Thread %s: starting", name)

    if __name__ == "__main__":
        format = "%(asctime)s: %(message)s"
        logging.basicConfig(format=format, level=logging.INFO,
                            datefmt="%H:%M:%S")

    threads = list()
    for index in range(3):
        logging.info("Main : create and start thread %d.", index)
        x = threading.Thread(target=thread_function, args=(index,))
        threads.append(x)
        x.start()

    for index, thread in enumerate(threads):
        logging.info("Main : before joining thread %d.", index)
        thread.join()
        logging.info("Main : thread %d done", index)
```

Das Sample dient zur Erläuterung der Bibliothek, bzw. der hier benötigten Funktion. Es beginnt wieder mit dem Importieren der Bibliothek mit `import threading`.

Der Code erstellt insgesamt drei parallel laufende Programme. In jedem Programm oder auch Thread genannt wird die Funktion `thread_function(name)` ausgeführt, dabei wird der Thread zwei Sekunden pausiert durch `time.sleep(2)`. Als Letztes wird nun noch eine weitere for-Schleife gestartet, die auf die Beendigung der Threads wartet, woraufhin die Nachricht `Main : thread %d done` protokolliert wird. Das `%d` wird hierbei durch den jeweiligen Namen des Threads ersetzt. [Fou24a]

19.0.10. Package Librosa Version 0.10.2

[Librosa](#) ist eine Bibliothek, die vielzahlige Audiobearbeitungsfunktionen anbietet. Für die erläuterte Aufgabenstellung ist lediglich der Teilbereich `librosa.effects` von Interesse.

```
#Compress to be twice as fast
>>> y, sr = librosa.load(librosa.ex('choice'))
>>> y_fast = librosa.effects.time_stretch(y, rate=0.2)
#Or half the original speed
>>> y_slow = librosa.effects.time_stretch(y, rate=0.5)
```

Das obige Sample zeigt, wie eine Audiodatei über Librosa geöffnet wird, was in diesem speziellen Fall nicht benötigt wird, da die Audiodatei über einen anderen Weg gewonnen wird, und anschließend die Geschwindigkeit verdoppelt und in einem zweiten Beispiel halbiert wird über die Funktion `time_stretch`.

```
#Shift up a major third (four steps if bins_per_octave is 12)
>>> y, sr = librosa.load(librosa.ex('choice'))
>>> y_third = librosa.effects.time_shift(y, sr=sr, n_steps=4)
#Shift down by tritone (six steps if bins_per_octave ist 12)
>>> y_tritone = librosa.effects.pitch_shift(y, sr=sr, n_steps=-
6)
Shift up by 3 quarter-tones
>>> y_three_qt = librosa.effects.pitch_shift(y, sr=sr, n_steps=3,
...                                         bins_per_octave=24)
```

Das obige Sample zeigt nun eine weitere Funktion des Teilbereichs `librosa.effects`. Im ersten Beispiel wird die Tonhöhe der Tonspur um vier Halbtöne erhöht, im zweiten Beispiel um sechs Halbtöne verringert und im dritten Beispiel um drei Vierteltöne erhöht. [McF+23]

19.1. Funktionen

19.1.1. Laden

```
import wave
dateipfad = "beispiel.wav" #Dateipfad zur Wave-Datei angeben

#Wavedatei öffnen
with wave.open(dateipfad, 'r') as wave_datei:
    #Informationen über die Wavedatei ausgeben
    print("Anzahl der Kanäle:", wave_datei.getnchannels())
    print("Abtastrate:", wave_datei.getframerate())
    print("Anzahl der Frames:", wave_datei.getnframes())
    print("Samplebreite in Bytes:", wave_datei.getsampwidth())
```

Dieses Sample zeigt, wie eine Wave-Datei geladen werden kann, dabei werden Kennzahlen, wie Anzahl der Kanäle, Abtastrate, Anzahl der Frames und Samplebreite ausgegeben. Voraussetzung für das erfolgreiche Laden ist, dass der Dateipfad entsprechend des Speicherortes der Beispieldatei festgelegt wird. Dieser Pfad muss im Befehl wave.open eingefügt werden.

19.1.2. Abspielen

```
import wave
import pyaudio

wf = wave.open('audio.wav', 'rb')# öffnen der Wave-Datei

p = pyaudio.PyAudio()# Initialisierung des PyAudio-Objekts

# öffnen des Audio-Streams
stream = p.open(format=p.get_format_from_width(wf.getsampwidth()),
                 channels=wf.getnchannels(),
                 rate=wf.getframerate(),
                 output=True)

# Lesen der Daten aus der Wavedatei
data = wf.readframes(1024)

# Abspielen der Wavedatei
while data:
    stream.write(data)
    data = wf.readframes(1024)

# Beenden des Audio-Streams und des PyAudio-Objekts
stream.stop_stream()
stream.close()
p.terminate()
```

Es gelten für das Sample „Abspielen“ dieselben Voraussetzungen, wie für das Laden einer Wave-Datei, nämlich dass der Dateipfad für die abzuspielende Datei im Programm definiert werden muss. Das Sample öffnet die Datei audio.wav im Lesemodus und initialisiert den Audio-Stream mit den entsprechenden Parametern aus der Wave-Datei. Dafür wird der Befehl „stream“ aus der Bibliothek PyAudio verwendet. Es wird so lange „Abspielen“ ausgeführt, bis alle Daten aus der Wave-Datei abgespielt wurden. Zuletzt wird der Audio-Stream beendet mit dem Befehl „stream.stop_stream“.

19.1.3. Speichern

Voraussetzung des „Speicher“-Vorgangs ist immer, dass eine Wave-Datei bereits geöffnet ist. Dieser Schritt ist in den folgenden Samples mit enthalten. Beim erstmaligen Speichern öffnet sich automatisch die „Speichern unter“-Funktion. Hier müssen das Dateiformat und der Zielordner definiert werden. Für das Dateiformat bietet sich der Wave-Typ an, da dieser mit der Anwendung weiter bearbeitet werden kann und es sich um den Standard für nicht komprimierte Audio-Dateien handelt. Im weiteren Verlauf der Bearbeitung kann auch der Befehl „Speichern“ verwendet werden, unter der Bedingung, dass die zu bearbeitende Audio-Datei bereits einmal gespeichert wurde.

```
from guizero import App, FileSaveDialog
import wave
```

```

def save_wave_file():
    # Erstelle eine App-Instanz
    app = App(visible=False)  # App im unsichtbaren Modus starten

    # Oeffne den Dateiexplorer, um den Zielordner auszuwaehlen
    file_path = FileSaveDialog(
        app, title="Speichern unter", filetypes=[("Wave files", "*.wav")], initialfilename="*.wav").value

    if file_path:
        # oeffne die ausgewählte Datei im Schreibmodus
        with wave.open(file_path, 'w') as wf:
            wf.setnchannels(2)  # Zwei Kanäle (Stereo)
            wf.setsampwidth(2)  # 2 Bytes pro Sample
            wf.setframerate(44100)  # Abtastrate von 44100 Hz
            wf.setnframes(0)  # Anfangs keine Frames

        # Schreibe Daten in die Datei
        # (hier müssten die Audiodaten eingefügt werden)
        # Beispiel: wf.writeframes(b'audiodaten')

        print(
            "Die Datei wurde erfolgreich gespeichert unter:", file_path)
        else:
            print("Speichern abgebrochen.")

    # Schliesse die App
    app.destroy()

save_wave_file()

```

In diesem Sample wird die Funktion „Speichern“ unter definiert, die es ermöglicht, eine Wave-Datei zu speichern. Zuerst wird ein Dateiexplorer geöffnet, um den Zielordner auszuwählen und die Audio-Datei kann im festgelegten Dateiformat (Wave) gespeichert werden. Anschließend wird die ausgewählte Wave-Datei im Schreibmodus geöffnet und die Parameter der Wave-Datei (Anzahl der Kanäle, Abtastrate, etc.) festgelegt. Es wird eine Rückmeldung ausgegeben, ob die Datei erfolgreich gespeichert oder der Vorgang abgebrochen wird. Bei diesem Sample ist zu beachten, dass eine Wave-Datei ausgewählt werden muss, was in diesem Code nicht dargestellt wird.

```

import wave

input_file = 'input.wav'
wav_file = wave.open(input_file, 'r')

nchannels = wav_file.getnchannels()
sample_width = wav_file.getsampwidth()
framerate = wav_file.getframerate()
nframes = wav_file.getnframes()

audio_data = wav_file.readframes(nframes)

```

```
#erstellen der neuen Wavedatei im Schreibmodus
#setzen der Parameter basierend aus gelesenen Werten
output_file = 'output.wav'
wav_output = wave.open(output_file, 'w')
wav_output.setparams((nchannels, sample_width, framerate, nframes,
'NONE', 'not compressed'))

#schreiben von gelesenen Audiodaten in die neue Wavedatei
wav_output.writeframes(audio_data)

#schließen sowohl die Eingabe- als auch die Ausgabedatei
wav_file.close()
wav_output.close()
```

Im Sample Speichern wird eine vorhandene Wave-Datei überschrieben mit den vom Nutzer festgelegten Parametern. Zunächst wird die Ursprungsdatei eingelesen und anschließend werden neue Parameter gesetzt. Zuletzt wird die Ursprungsdatei mit überschrieben.

19.1.4. Anzeigen von zwei Audio-Dateien

Um zwei Audio-Dateien in einem Diagramm mit Matplotlib darzustellen, kann die Wave Bibliothek verwendet werden, um die Audio-Dateien zu laden und mit der Matplotlib geplottet zu werden.

Begonnen wird mit dem Importieren der genutzten Bibliotheken.

Danach werden die Audiodaten aus den beiden Audio-Dateien geladen und mithilfe von der Bibliothek NumPy in Arrays umgewandelt. Anschließend erstellt Matplotlib ein Diagramm und zeichnet die Audiodaten aus beiden Audio-Dateien in dasselbe Diagramm. Die Label-Parameter in den Plot-Funktionen werden verwendet, um die Legende zu erstellen, sodass sichtbar ist, welche Linie zu welcher Audio-Datei gehört. Wichtig ist, dass die richtigen Pfade zu den Audio-Dateien angeben sein müssen, wenn die Audio-Dateien nicht im selben Verzeichnis wie das Python-Skript liegen, muss zum Beispiel 'C:/Benutzer/IhrName/Dokumente/audio1.wav' anstelle von 'audio1.wav' verwendet werden.

Wichtig ist, dass die erforderlichen Bibliotheken installiert wurden, bevor dieser Code ausgeführt wird.

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

# Laden der Audiodateien
with wave.open('audiol.wav', 'rb') as wf1:
    y1 = wf1.readframes(-1)
    sr1 = wf1.getframerate()

    with wave.open('inputAudio2.wav', 'rb') as wf2:
        y2 = wf2.readframes(-1)
        sr2 = wf2.getframerate()

    # Umwandeln der Audio Dateien in numpy Arrays
    y1 = np.frombuffer(y1, dtype=np.int16)
    y2 = np.frombuffer(y2, dtype=np.int16)
```

```
# Achsenbestimmung des Diagramms
x1 = np.linspace(0, len(y1) / sr1, len(y1))
x2 = np.linspace(0, len(y2) / sr2, len(y2))

# Plotten der Daten
plt.figure(figsize=(10, 4))
plt.plot(x1, y1, label='Audio 1')
plt.plot(x2, y2, label='Audio 2')
plt.xlabel('Time (seconds)')
plt.ylabel('Amplitude')
plt.title('Audio Waveforms')
plt.legend()
plt.grid(True)
plt.show()
```

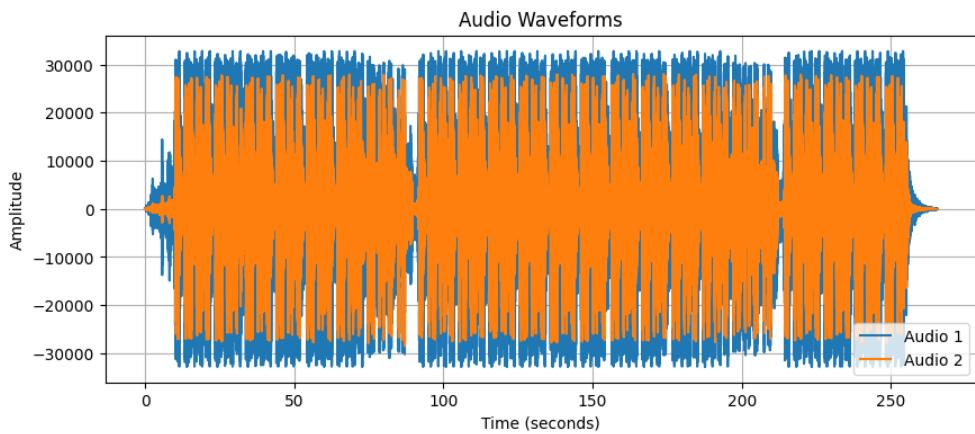


Figure 19.8.: Beispiel Anzeige von zwei Audio-Dateien

19.1.5. Glättung von Audio-Dateien

Zur Glättung der Audio-Datei wird die exponentielle Glättung verwendet. Die exponentielle Glättung ist einer der grundlegenden Algorithmen zur Glättung von Zeitreihen. Bei der exponentiellen Glättung wird die Wichtigkeit der vorhergegangenen Daten mit α gewichtet, je kleiner α , desto wichtiger sind die vorherigen Werte. Bei starkem Rauschen kann ein niedriger α gewählt werden, soll die Audio-Datei möglichst wenig verändert werden muss der α groß gewählt sein. Der mögliche nutzbare Zahlenbereich ist $1 \geq \alpha \geq 0$. Die genutzte Formel für die exponentielle Glättung sieht so aus.

$$S_i = \alpha * data_i + (1 - \alpha) * S_{i-1}$$

S_i = geglättet Daten zum Punkt i

α = alpha

$data_i$ = Daten zum Punkt i

S_{i-1} = geglättet Daten zum punkt i-1

Die benötigten Module sind `wave` und `numpy`. Für die Verwendung des Programms muss 'inputAudio.wav' durch den Pfad der zu glättenden Wave-Datei ersetzt werden. Der Ausgabebereich kann durch Ändern von `smoothedAudio.wav` definiert werden.

```
import wave
import numpy as np
# Definition von der Variable alpha
alpha = 0.01

# Definition der Glättungsfunktion
def exponential_smoothing(data):
    smoothed_data = np.zeros_like(data)
    smoothed_data[0] = data[0]
    for i in range(1, len(data)):
        smoothed_data[i] = alpha * data[i] + (1 - alpha) *
            smoothed_data[i - 1]
    return smoothed_data

def smooth_wave_file(input_file, output_file):
    with wave.open(input_file, 'rb') as wav_in:
        params = wav_in.getparams()
        data = np.frombuffer(wav_in.readframes(params.nframes),
                             dtype=np.int16)
        smoothed_data = exponential_smoothing(data)
        with wave.open(output_file, 'wb') as wav_out:
            wav_out.setparams(params)
            wav_out.writeframes(smoothed_data.tobytes())

    # Beispielaufruf:
    input_wave_file = 'input.wav'
    output_wave_file = 'smoothed_audio.wav'
    smooth_wave_file(input_wave_file, output_wave_file)
```

Der Unterschied einer geglätteten Audio-Datei ist in Abbildung 19.8 gut zu beobachten. Dort wird die Ursprungsaudiodatei (Audio 1) und die Glättung dieser (Audio 2) mit dem $\alpha = 0.01$ dargestellt.

19.1.6. Ausschnitt

Zum Abspielen eines Abschnittes einer Wave-Datei in Python können die Bibliotheken wave und pyaudio verwendet werden. Die wave Bibliothek wird benötigt, damit Python das genutzte wave audio Format überhaupt einlesen kann. Pyaudio wird genutzt um die geladenen Audio-Datei anschließend auch noch abspielen zu können. Um jetzt das Programm zu nutzen, muss erst die richtige Wave-Datei geöffnet werden, wozu der Pfad zur Audio-Datei angegeben werden muss. Danach kann die Startdauer in Sekunden und die Dauer des Ausschnitts ebenfalls in Sekunden, unter Setzen der gewünschten Werte, angegeben werden.

```
import pyaudio
import wave

# Setzen der gewünschten Werte
start = 4 # Startzeitpunkt in Sekunden
ende = 9 # Endzeitpunkt in Sekunden
dauer = ende - start # Dauer des Ausschnitts in Sekunden

# öffne die WAV-Datei
```

```
#Durch ändern des Dateipfades anpassen der genutzten Wave Datei
wave_datei = wave.open('asdf.wav', 'rb')
# Erstelle ein PyAudio-Objekt
p = pyaudio.PyAudio()

# oeffne den Stream basierend auf dem Wave-Objekt
stream = p.open(
    format=p.get_format_from_width(wave_datei.getsampwidth()),
    channels=wave_datei.getnchannels(),
    rate=wave_datei.getframerate(),
    output=True)

# Lese die Daten (basierend auf der Chunk-Groesse)
wave_datei.readframes(start * wave_datei.getframerate())
data = wave_datei.readframes(dauer * wave_datei.getframerate())

# Spielt den Stream (von Anfang bis zum Ende)
stream.write(data)

# Speichern in einer neuen Datei
#Dateipfad der Auschnitt datei
neue_wave_datei = wave.open('ausschnitt.wav', 'wb')
neue_wave_datei.setparams(wave_datei.getparams())
neue_wave_datei.writeframes(data)
neue_wave_datei.close()

# Aufräumen
wave_datei.close()
stream.close()
p.terminate()
```

19.1.7. Zoomen in einer Matplotlib

Um innerhalb der von Matplotlib dargestellten Wave-Datei zu zoomen, wird der Zeitbereich angepasst. Dies geschieht durch Änderung der Start- und Endzeit des darzustellenden Graphen in Matplotlib. In diesem Beispiel soll die Wave-Datei von Sekunde 0 bis Sekunde 2 dargestellt werden. Soll nun ein anderer Bereich dargestellt werden, muss der Wert von `start_time` bzw. `end_time` geändert werden.

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

# Pfad zur Wave-Datei
wave_file_path = 'asdf.wav'

# Wave-Datei oeffnen
wave_file = wave.open(wave_file_path, 'r')

# Abtastrate (Sampling Rate) der Wavedatei
fs = wave_file.getframerate()

# Anzahl der Frames in der Wave-Datei
```

```

data_size = wave_file.getnframes()

# Daten aus der Wave-Datei lesen
wave_data = wave_file.readframes(data_size)
signal = np.frombuffer(wave_data, dtype=np.int16)

# Einstellen des Zeitbereichs
start_time = 0 # Startzeitpunkt (in Sekunden)
end_time = 2 # Endzeitpunkt (in Sekunden)

# Indizes für den gewünschten Zeitbereich
start_index = int(start_time * fs)
end_index = int(end_time * fs)

# Zeitvektor zuschneiden
cropped_time = np.linspace(start_time, end_time,
end_index - start_index)
cropped_signal = signal[start_index:end_index]

# Plot erstellen
plt.figure(figsize=(10, 6))
plt.plot(cropped_time, cropped_signal, 'b')
plt.xlabel('Zeit (s)')
plt.ylabel('Amplitude')
plt.title('Wave-Audiovisualisierung (Ausschnitt)')
plt.grid(True)
plt.show()

# Wave-Datei schliessen
wave_file.close()

```

Mit `wave_file.close` kann der Plot anschließend geschlossen werden.

Neben dem Zoomen per Eingabe der Sekunden ist es auch möglich, die Eingabe per Mausrad umzusetzen. Das folgende Sample zeigt dies, anhand eines Sinussignals als Beispieldatenspur.

```

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.widgets import Slider

# Beispieldaten erzeugen (Sinuskurve als Tonspur)
sampling_rate = 44100 # Abtastrate
duration = 5 # Dauer in Sekunden
t = np.linspace(0, duration, int(sampling_rate * duration),
endpoint=False)
frequency = 440 # Frequenz in Hz (A4-Ton)
signal = 0.5 * np.sin(2 * np.pi * frequency * t)

# Plot initialisieren
fig, ax = plt.subplots()
plt.subplots_adjust(bottom=0.25)
l, = plt.plot(t, signal, lw=1)
ax.set_title('Tonspuranzeige')
ax.set_xlabel('Zeit [s]')

```

```

ax.set_ylabel('Amplitude')

# Achsen-Initialwerte
axcolor = 'lightgoldenrodyellow'
axpos = plt.axes([0.1, 0.1, 0.65, 0.03], facecolor=axcolor)
spos = Slider(axpos, 'Pos', 0.0, duration, valinit=0.0)

# Zoom- und Scroll-Funktion
def zoom(event):
    cur_xlim = ax.get_xlim()
    cur_ylim = ax.get_ylim()
    xdata = event.xdata # x-position of mouse
    ydata = event.ydata # y-position of mouse
    if event.button == 'up':
        # Zoom in
        scale_factor = 1 / 1.5
    elif event.button == 'down':
        # Zoom out
        scale_factor = 1.5
    else:
        # Nicht zoomen
        scale_factor = 1
    return

    new_width = (cur_xlim[1] - cur_xlim[0]) * scale_factor
    new_height = (cur_ylim[1] - cur_ylim[0]) * scale_factor
    relx = (cur_xlim[1] - xdata) / (cur_xlim[1] - cur_xlim[0])
    rely = (cur_ylim[1] - ydata) / (cur_ylim[1] - cur_ylim[0])

    ax.set_xlim([xdata - new_width * (1 - relx),
                xdata + new_width * (relx)])
    ax.set_ylim([ydata - new_height * (1 - rely),
                ydata + new_height * (rely)])
    fig.canvas.draw()

def update(val):
    pos = spos.val
    ax.set_xlim([pos, pos + (cur_xlim[1] - cur_xlim[0])])
    fig.canvas.draw_idle()

spos.on_changed(update)
fig.canvas.mpl_connect('scroll_event', zoom)
plt.show()

```

Als Plot-Oberfläche wird Matplotlib verwendet. Die Funktion „Zoom“ behandelt das Zoom-Ergebnis, welches durch das Scrollen mit der Maus ausgelöst wird. Außerdem ermöglicht der Codeabschnitt, der den Befehl „slider“ enthält, das Verschieben der Ansicht entlang der Zeitachse. Das interaktive Steuern des Zooms wird für diese Funktion durch den letzten Codeblock ermöglicht.

19.1.8. Änderung der Tonhöhe und Geschwindigkeit

Bei einer Tonhöhenänderung bzw. Frequenzskalierung wird das digitale Signal im Nachhinein korrigiert, ohne andere Aspekte der Tonfolge wie z.B. die Wiedergabegeschwindigkeit zu beeinflussen. Das Phänomen, bei dem sich durch die Frequen-

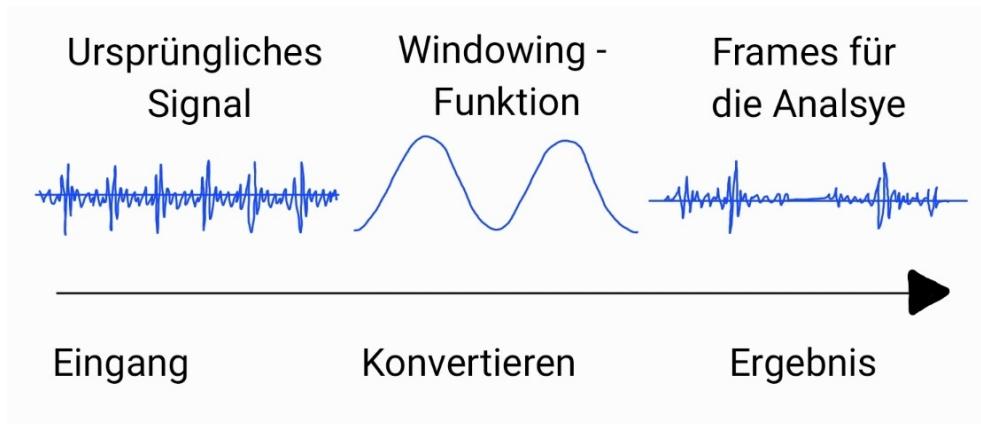


Figure 19.9.: Window-Funktion

zveränderung auch die Länge der Frequenz bzw. im übertragenen Sinne die Dauer der Audiodatei verändert, wird bei der Funktion umgangen (sog. Pitch-Shifting).

Eine ähnliche Situation tritt bei der Geschwindigkeitsveränderung auf. Das bloße Ändern der Wiedergabegeschwindigkeit führt zu Verzerrungen in den Tonhöhen. Das heißt, auch hier ist das Ziel, den Prozess der Komprimierung oder Streckung der Wiedergabedauer zu verändern, ohne den spektralen Inhalt zu beeinträchtigen (sog. Time-Stretching).

Folglich gilt für beide Funktionen: Um eine Änderung eines Parameters zu erreichen, muss die Abhängigkeit der Frequenz- und Zeitskalierung unterdrückt werden. Dafür wird „librosa.effects.pitch_shift“ bzw. „librosa.effects.time_stretch“ verwendet.

Das Verfahren basiert auf einem Phase Vocoder unter Anwendung der „Fast-Fourier Transformation“ (FFT) (siehe Kapitel 19.1.9). Im Folgenden wird der Algorithmus hinter dem Phase Vocoder verkürzt und vereinfacht erklärt.

Der erste Schritt bei einer Phase-Vocoder-Analyse ist die Windowing-Funktion. Dabei wird das digitale Samplesignal in zahlreiche Zeitblöcke unterteilt und diese Blöcke werden mit einer Hüllkurve multipliziert. Die dadurch entstehende Form der Window-Funktion hat Einfluss auf den bei diesem Prozess neu entstehenden Klang der Audio-Datei. Gängige Formen für einen Phase Vocoder sind meistens glockenförmig. Die Änderung des Signals wird in Abbildung 19.9 gezeigt.

Anschließend wird jedes Fenster einer Spektralanalyse unterzogen, die als STFT (Short-Time Fourier Transformation) bezeichnet wird. Der STFT-Algorithmus wendet nacheinander die diskrete Fourier-Transformation auf die gefensterten Abtastwerte an. Die STFT kann also als eine Folge von DFTs betrachtet werden. Eine DFT-Analyse gilt als zeitdiskret und diskret im Frequenzbereich, wodurch isolierte Spektrallinien dargestellt werden.

Jetzt muss die Anzahl der Frequenzbänder angegeben werden. Dies geschieht, indem für jedes positive Frequenzband ein gespiegeltes Frequenzband im negativen (imaginären) Bereich entsteht. Dabei sind beide Frequenzbänder Hälften der Amplitude des gemessenen realen Frequenzbands. Unter den verschiedenen Prozessen, die nach dieser Berechnung stattfinden, verwirft der Phasenvocoder die negativen Frequenzen und verdoppelt die Amplitude der positiven.

Das Ergebnis einer einzelnen FFT von fensterbehafteten Samples wird als Frame bezeichnet. Ein Frame besteht aus zwei Wertesätzen: den Magnituden (oder Amplituden) aller Frequenzbänder und den Anfangsphasen aller Frequenzbänder. Zusätzlich wird der Phasenunterschied zu dem vorherigen Frame bestimmt, was Hinweise darauf liefert, wo sich die Tonhöhe möglicherweise innerhalb des Frequenzbereichs dieses Bands befindet. Die Kombination aus Amplituden-/Phasen-unterschiedsdaten für jedes Band wird als Bin bezeichnet. Diese Paare von

Amplituden-/Phasendaten für jeden Bin werden als x-, y-Koordinaten ausgegeben. Zusätzlich muss noch der Analyseparameter Hop size oder Schrittweise bestimmt werden. Dieser Parameter gibt den Abstand zwischen den Anfangspunkten aufeinanderfolgender Fenster an. Mit ihm einher geht der Überlappungsfaktor, welcher die Menge an Informationen in der Analyse beeinflusst. Ein zu großer Überlappungsfaktor führt zu einem verschwommenen Klang.

Mit den aufgestellten Analysedaten wird jetzt die Resynthese des Klanges durchgeführt. Dabei wird durch eine inverse STFT eine laufende Aufzeichnung der Phasendifferenzen von Frame zu Frame in einem Prozess erstellt wird (Phase unwrapping).

Die Resynthese ist die Phase, in der sich das Vorgehen für Tonhöhenänderung und Geschwindigkeitsveränderung unterscheidet.

Für die Änderung der Tonhöhe werden in diesem Schritt die Frames in der Geschwindigkeit der Orginalanalysedatei wiedergegeben. Die Frequenzen des gesamten Spektrums können jedoch proportional nach oben oder unten verschoben werden, was zu einer Änderung der Tonhöhe, aber nicht der Geschwindigkeit führt.

Für die Geschwindigkeitsveränderung wird die Rate, mit der die Frames neu erstellt werden, erhöht oder verringert, je nachdem, wie schnell oder langsam der ursprüngliche Ton abgespielt werden soll. Der Phase-Vocoder fügt zwischen den einzelnen Bildern neue Datenpunkte ein, um den Übergang zwischen ihnen sanfter zu gestalten (interpoliert). Da der Frequenzinhalt der einzelnen Frames nicht verändert wird, führen Änderungen der Geschwindigkeit nicht zu Änderungen der Tonhöhe.

WS:cite

Sample Tonhöhe ändern:

```
import librosa
import matplotlib.pyplot as plt
import soundfile as sf

# Pfad zur WAV-Datei
wav_datei = "input.wav"
# Einlesen der Datei
signal, abtastrate = librosa.load(wav_datei, sr=None, mono=False)
# Ändern der Tonhöhe durch Veränderung des n_steps=(Wertes).
# Veränderung von 1 entspricht eines Halbtons.
höhen_änderung = librosa.effects.pitch_shift(signal,
n_steps=10,
sr=abtastrate,
bins_per_octave=12)

data = höhen_änderung

# Ausgabe der Wave-Datei

sf.write('Höhenänderung.wav', data.T,
abtastrate, subtype='PCM_24', format='WAV')
```

Sample Geschwindigkeit ändern:

```
import librosa
import matplotlib.pyplot as plt
import soundfile as sf

# Pfad zur WAV-Datei
```

```

wav_datei = "input.wav"
# Einlesen der Datei
signal, abtastrate = librosa.load(wav_datei,
sr=None,
mono=False)

#Beschleunigen der Wave-Datei
beschleunigtes_Signal = librosa.effects.time_stretch(
signal, rate=1.5)

data = beschleunigtes_Signal

#Ausgabe der Wave-Datei
sf.write('Geschwindigkeitaenderung.wav', data.T,
abtastrate, subtype='PCM_24', format='WAV')

```

19.1.9. Frequenzanalyse

Der Schall kann mit einem Mikrofon aufgezeichnet werden, wodurch die Spannungs-Zeit-Verläufe des Signals wiedergegeben werden. Allerdings ist es schwierig, die Signalkomponenten in dieser Form zu erkennen. Durch eine Frequenzanalyse können jedoch die Signaleigenschaften ermittelt und wichtige Muster erkannt werden. Eine Methode für die Frequenzanalyse ist die Spektralanalyse auf Basis einer Fast-Fourier-Transformation (FFT). FFT ist eine Spektralanalyse in der Audio- und Akustik-Messtechnik. Dabei wird ein zeitdiskretes Signal in seine Frequenzanteile zerlegt und dadurch analysiert. Eine Spektralanalyse ist ein grundlegendes Werkzeug für die Signalverarbeitung. Sie wird verwendet, um die Frequenzinhalte eines Signals zu untersuchen. Dabei wird ermittelt, welche Frequenzkomponenten im Originalsignal enthalten sind und wie sie verteilt sind.

WS:cite

FFT ist ein Algorithmus zur Berechnung der Fourier-Reihe, die auf diskrete Signale angewandt wird (DFT).

Für die diskrete Fourier Transformation gilt die Fourier Reihendarstellung (siehe Formel 19.1.9) mit den diskreten Fourier-Koeffizienten (siehe Formel 19.1.9).

$$f_T(t) = \sum_{k=-\infty}^{\infty} \gamma_k e^{ik\omega t} \quad \text{mit } \omega = \frac{2\pi}{\tau} \quad (19.4)$$

$$\gamma_k \equiv \gamma_k(f) = \frac{1}{T} \cdot \int_{T/2}^{-T/2} f_T(\tau) e^{-ik\omega\tau} d\tau \quad \text{für } k = 0, \pm 1, \pm 2, \dots \quad (19.5)$$

Wobei:

f_T = Zeitkontinuierliches, T – periodisches Signal

$e^{-ik\omega\tau}$ = Grundschwingung

$\omega = \frac{2\pi}{\tau}$ = Frequenz

γ_k = Verstärkungsfaktor der Grundschwingung

Die Fourier Reihenformel beschreibt im Allgemeinen die Darstellung periodischer Signale im Frequenzbereich als Summe von Sinus- und Kosinusschwingungen unterschiedlicher Frequenzen. Bei der DFT werden aus den kontinuierlichen, periodischen Signale nicht-periodische, diskrete Signale.

Dementsprechend muss, um eine solche Rechenoperation durchzuführen, noch folgende Spektralanalyseparameter definiert werden:

WS:cite

- Anzahl der abgetasteten Punkte (NFFT)
- Abtastfrequenz

Durch diese wird die Frequenzauflösung der Analyse bestimmt.

Für das Sample einer Frequenzanalyse werden die Bibliotheken `matplotlib.pyplot` und `numpy` verwendet. Das Signal wird in dem Sample erzeugt und bezieht sich auf keine Audio-Datei.

```
#verwendete Bibliotheken
import matplotlib.pyplot as plt
import numpy as np

# Reproduzierbarkeit der Zufallszahlen
np.random.seed(19680801)

#Erzeugung von Zeit und Signalsample
dt = 0.0005
t = np.arange(0.0, 20.5, dt)

#Generierung von zwei Sinussignalen
s1 = np.sin(2 * np.pi * 100 * t)
s2 = 2 * np.sin(2 * np.pi * 400 * t)

# "chirp"-Signal für Frequenzveränderung über die Zeit
s2[t <= 10] = s2[12 <= t] = 0

# Rauschen hinzufügen für realistische Szenarien
nse = 0.01 * np.random.random(size=len(t))

x = s1 + s2 + nse # Signal mit Rauschen
NFFT = 1024 # Anzahl der abgetasteten Punkte
Fs = 1/dt # the Abtastgeschwindigkeit

#Plotten des Signals und Spektrogramms
fig, (ax1, ax2) = plt.subplots(nrows=2, sharex=True)
ax1.plot(t, x)
ax1.set_ylabel('Signal')

#Berechnung der Spektrogramms
Pxx, freqs, bins, im = ax2.specgram(x, NFFT=NFFT, Fs=Fs)
# Mit:
# - Pxx: Spektralleistung
# - freqs: Frequenzvektor
# - im: Bild der Spektrogrammdaten

#Achsenbeschriftung
ax2.set_xlabel('Time (s)')
ax2.set_ylabel('Frequency (Hz)')
```

```

ax2.set_xlim(0, 20)

#Plot anzeigen
plt.show()

```

Die Ausgabe des Codes sind zwei Diagramme. Das obere Diagramm zeigt das Signal und das untere die Frequenzanalyse auf Basis der FFT. (siehe Abbildung 19.10)

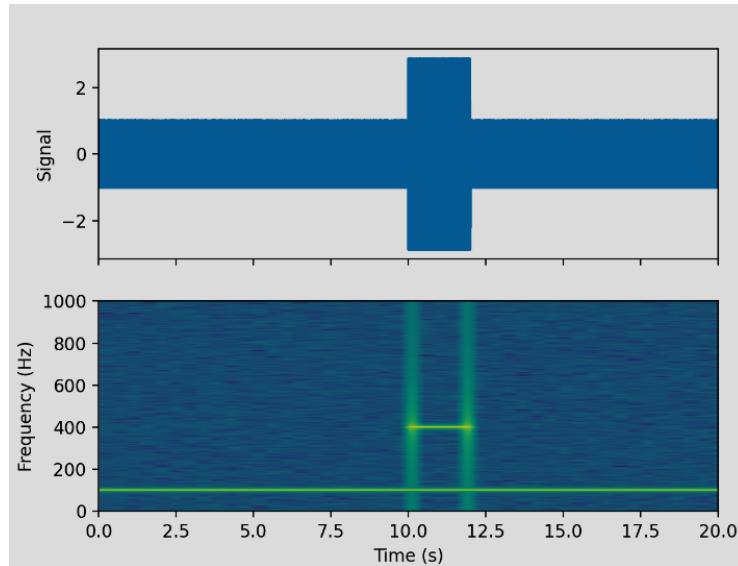


Figure 19.10.: Ausgabe Sample Frequenzanalyse

19.1.10. Overlay

Das folgende Sample beschränkt sich darauf, dass 3 Wave-Dateien gleichzeitig abgespielt werden.

```

import wave
import pyaudio
import threading

def play_sound(file):
    chunk = 1024
    wf = wave.open(file, 'rb')
    p = pyaudio.PyAudio()
    stream = p.open(format=p.get_format_from_width
                    (wf.getsampwidth()),
                    channels=wf.getnchannels(),
                    rate=wf.getframerate(),
                    output=True)
    data = wf.readframes(chunk)

    while data:
        stream.write(data)
        data = wf.readframes(chunk)

    stream.stop_stream()
    stream.close()

```

```

p.terminate()

files = ['sound1.wav', 'sound2.wav', 'sound3.wav']

threads = []
for file in files:
    thread = threading.Thread(target=play_sound, args=(file,))
    threads.append(thread)
    thread.start()

for thread in threads:
    thread.join()

```

Für das Sample müssen `sound1.wav`, `sound2.wav` und `sound3.wav` durch den gewünschten Dateipfad ersetzt werden. Die Gesamtlänge wird durch die Audio-Datei mit der längsten Dauer bestimmt, da der Code erst beendet wird, sobald alle Audio-Dateien komplett abgespielt wurden. Es werden in diesem Code zwei Schleifen verwendet, wobei die erste die Audio-Dateien in der Liste `files` durchläuft. Anschließend wird für jede Datei ein eigener Thread erstellt, welcher das synchrone Abspielen aller Dateien ermöglicht, indem die Funktion `playsound` mit entsprechenden Audio-Dateien als Argument aufgerufen werden. Parallel wartet die zweite Schleife darauf, dass alle Threads beendet sind, bevor das Programm zuletzt beendet wird.

19.2. General

General description

WS:cite books

19.2.1. Decibels

In electronics one often wants to represent the ratio of two signals on a logarithmic scale. For this we use the decibel, dB . If we have two powers P_1 and P_2

$$dB = 10 \log_{10} \left(\frac{P_2}{P_1} \right)$$

or equivalently – when comparing two signals with the same kind of waveform –

$$dB = 20 \log_{10} \left(\frac{V_2}{VP_1} \right).$$

Note dB is a relative unit of measurement, i.e. *This amplifier has a gain of 10 dB*. dB can be positive or negative. For example, $+10dB$ corresponds to P_2 greater than P_1 by a factor of 10, and $-3dB$ corresponds to P_2 less than P_1 by approximately a factor of 2.

For an absolute measure on a logarithmic scale there are a variety of other units. A common one is dBm .

$$dBm = 10 \log_{10}(P[mW])$$

Zero dBm corresponds to $1mW$ of power. And in a 50Ω system $0dBm$ corresponds to $220mVrms$.

WS:cite books

19.2.2. Puls-Dichtemodulation

Pulsdichtemodulation (PDM) ist eine faszinierende Technik, die im Arduino Nano 33 BLE Sense verwendet wird, um Audiosignale zu digitalisieren. PDM wandelt analoge Audiosignale in digitale Daten um, indem es die Dichte der Impulse misst. Statt kontinuierlicher Werte verwendet PDM nur zwei Zustände: 1 (Impuls vorhanden) oder 0 (kein Impuls). Um das Mikrofon zu verwenden, steht die Bibliothek `pdm.h` zur Verfügung. PDM arbeitet mit einer hohen Samplingrate, um genaue Audioinformationen zu erfassen. Die Impulse werden als Bitstream übertragen, der später in Audiodaten umgewandelt wird.

Die Puls-Dichtemodulation (PDM) ist eine interessante Technik, die sowohl Vor- als auch Nachteile aufweist.

Die Puls-Dichtemodulation (PDM) hat folgende Vorteile:

- Einfachheit: PDM ist weniger komplex als einige andere Modulationsverfahren. Die Implementierung von Sendern und Empfängern für PDM ist vergleichsweise unkompliziert.
- Robustheit gegenüber Störungen: PDM ist widerstandsfähig gegenüber Rauschen und Interferenzen.
- Hohe Samplingrate: PDM arbeitet mit einer hohen Abtastrate, um genaue Audioinformationen zu erfassen.
- Geringe Verluste: Im Vergleich zu analogen Steuerungen verursacht PDM weniger Verluste, da es die Versorgungsspannung schnell ein- und ausschaltet.

Die Nachteile der Puls-Dichtemodulation (PDM) sind:

- Große Lasten: Die Steuerung großer Verbraucher erfordert hohe Spannungen und Ströme. Standard-Anologschaltungen und DACs sind hierbei weniger effizient.
- Wärmeentwicklung: Leistungstransistoren, die in PDM-Schaltungen verwendet werden, erzeugen Wärme und Verluste.

Insgesamt ist PDM eine leistungsstarke Technik, die in verschiedenen Anwendungen wie Spracherkennung, Klanganalyse und Datenübertragung eingesetzt wird. Es ist wichtig, die Vor- und Nachteile je nach Kontext zu berücksichtigen

WS:citations

19.3. Specific Sensor

cite board

19.4. Specification

- cite data sheet
- Circuit Diagram

Der Sensor hat folgende Spezifikationen:

- Signal-Rausch-Verhältnis: 64 dB
- Empfindlichkeit: -26 dBFS ± 3 dB
- Temperaturbereich: -40 bis 85 1°C

Mikrofone werden in Mobilgeräten, Spracherkennungssystemen und sogar in Gaming- und Virtual-Reality-Eingabegeräten eingesetzt.

Mit dem eingebetteten MP34DT05-Sensor können Sie die Schallwerte Ihrer Umgebung messen und anzeigen.

WS:cite

19.5. Bibliothek pdm.h

19.5.1. Description

Die PDM-Bibliothek ermöglicht die einfache Verwendung des integrierten Mikrofons und ist auch über die Bibliothek [ArduinoSound](#) zugänglich.

19.5.2. Installation

19.5.3. Functions

Die PDM-Bibliothek für den Arduino Nano 33 BLE Sense ermöglicht die Verwendung von Puls-Dichtemodulation (PDM)-Mikrofonen, wie dem integrierten MP34DT05 auf dem Board. Hier sind die wichtigsten Funktionen, die diese Bibliothek bereitstellt:

begin() : Diese Funktion initialisiert die PDM-Schnittstelle. Sie nimmt zwei Parameter entgegen:

Parameter channels : Die Anzahl der Kanäle (1 für Mono, 2 für Stereo).

Parameter sampleRate : Die gewünschte Abtastrate in Hertz.

Beispiel:

```
1000 if (!PDM.begin(1, 16000)) {
1001     Serial.println("Fehler beim Starten der PDM!");
1002     while (1);
1003 }
```

end() : Mit dieser Funktion wird die PDM-Schnittstelle deinitialisiert:

```
1000 PDM.end();
```

available() : Diese Funktion gibt die Anzahl der verfügbaren Bytes im PDM-Empfangspuffer zurück. Das sind bereits eingetroffene Daten, die darauf warten, gelesen zu werden:

```
1000 int bytesAvailable = PDM.available();
```

read() : Mit dieser Funktion liest man Daten aus dem PDM in einen angegebenen Puffer:

```
1000 short sampleBuffer[256]; // Puffer fuer Samples (16-Bit)
1001 int bytesRead = PDM.read(sampleBuffer, bytesAvailable);
```

onReceive() : Diese Funktion setzt die Callback-Funktion, die aufgerufen wird, wenn neue PDM-Daten zum Lesen bereit sind:

```

1000 void onPDMdata() {
1002     int bytesAvailable = PDM.available();
1003     // Weitere Verarbeitung der Daten...
1004 }
1005 PDM.onReceive(onPDMdata);

```

19.5.4. Example - Manual

19.5.5. Example

Der Code verwendet die folgenden Bibliotheken:

- **PDM.h**: Diese Bibliothek ermöglicht die Kommunikation mit dem Mikrofon zur Aufnahme von PDM-Signalen [Arde]

19.5.6. Example - Code

Listing 19.1.: Simple sketch to control the built-in LED

```

1000 /**
1001 * @file TestLEDBuiltin.ino
1002 *
1003 * @brief Simple program for testing the built-in LED
1004 *
1005 *
1006 * Turns the built-in LED on for one second, then off for one second,
1007 * repeatedly.
1008 *
1009 * The LED is switched on for 1 second and switched off
1010 * for 1 second so that the LED flashes accordingly.
1011 *
1012 * On the Arduino Nano 33 BLE Sense, it is attached to digital pin 13
1013 *
1014 */
1015
1016 #include <../LEDs/BuiltinLED.h>
1017 #include <../LEDs/LED.h>
1018
1019 /**
1020 * @brief the setup function runs once when you press reset or power the
1021 * board
1022 *
1023 * Standard function of Arduino sketches
1024 *
1025 * Initialization of the pin LED_BUILTIN as output
1026 *
1027 * @param —
1028 *
1029 * @return void
1030 */
1031 void setup() {
1032     // Initialize the pin as an output
1033     BuiltinLEDinit();
1034 }
1035
1036 /**
1037 * @brief the loop function runs over and over again forever
1038 */

```

```

1038 * standard function of Arduino sketches
1039 *
1040 * swichting the led on / off for 1sec.
1041 *
1042 * @param —
1043 *
1044 * @return void
1045 */
1046 void loop() {
    // Turn the LED on
    BuiltInLED(SET_ON);
    // Wait for one second
    delay(1000);
    // Turn the LED off
    BuiltInLED(SET_OFF);
    // Wait for one second
    delay(1000);
}

```

..../Code/Nano33BLESense/Test/TestLEDBuiltin.ino

19.5.7. Example - Files

19.6. Calibration

cite method

19.7. Simple Code

19.8. Simple Application

19.8.1. Pin-Konfiguration

WS:Description

Die verwendeten Pins werden zu Beginn des Codes definiert:

- `microphoneDataPin`: Der Datenpin für das Mikrofon
- `microphoneClockPin`: Der Clock-Pin für das Mikrofon

19.8.2. Initialisierung und Setup

In der Funktion `setup()` werden die erforderlichen Initialisierungen durchgeführt:

- `PDM.onReceive(onPDMdata)`: Registriert den Empfang des Mikrofonsignals als PDM-Signal

19.8.3. Messungssteuerung

Die Funktion `onPDMdata()` wird aufgerufen, wenn Daten vom Mikrofon verfügbar sind. Dies liest die Daten in einen Array ein und zählt die Anzahl der gelesenen Samples.

Listing 19.2.: Einlesen der Daten

```

1000 const unsigned long averagingTime = 200; /*< Time [ms] to moving
1001   average the values */
1002 unsigned long averagingStartTime = 0;      /*< Start time [ms] to moving
1003   average the values */
1004 int valueSum = 0;                          /*< sum of the values to
1005   moving average */

```

```

int valueCount = 0; /*< number of the values to
moving average */

```

..../Code/Nano33BLESense/Test/TestMicrophone.ino

Die Hauptfunktion `loop()` enthält zwei Funktionen für die Messungssteuerung und die Benutzeroberfläche:

Listing 19.3.: Messungssteuerung

```

1000 int absoluteCount = 0; /*< absolute number of the values to moving
average */

```

1002 int redButtonResult = 0; /*< Button pressed? */

..../Code/Nano33BLESense/Test/TestMicrophone.ino

Die Funktion `HandleInput()` überwacht den Zustand der beiden Taster und steuert somit den Messungsablauf:

Listing 19.4.: Überwachung

```

1000 /**
1002 * @brief the setup function runs once when you press reset or power the
board
*
1004 * standard function of Arduino sketches
*
1006 * Initialization of
* — the display
* — the interrupt pin
* — the microphone (PDM)
1008 *
1010 *
1012 * @param —
1014 *
* @return void
*/
1016 void setup() {
display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
display.clearDisplay();
display.setTextColor(WHITE);
display.setTextSize(2);

1020 pinMode(buttonPin, INPUT_PULLUP);
Serial.begin(9600);

1022 PDM.onReceive(onPDMdata);
}

1024
1026 /**
1028 * @brief Interrupt service routine for the microphone
*
1030 *
* Attention: as short as possible
1032 *
* If an interrupt is on, th function checks if there are samples,
1034 * copies them into the buffer and sets a flag
*
1036 * @param —
*
* @return void

```

..../Code/Nano33BLESense/Test/TestMicrophone.ino

- Wenn der blaue Taster gedrückt wird, wird die Messung gestartet, indem `isMeasuring` auf `true` gesetzt wird und die Funktion `StartSampling()` aufgerufen wird.

- Wenn der rote Taster gedrückt wird, wird die Messung beendet, indem `isMeasuring` auf `false` gesetzt wird und die Funktion `StopSampling()` aufgerufen wird. Je nach vorherigem Zustand des roten Tasters wird entweder der maximale Wert SPL in dB angezeigt (`ShowMaxdBspl()`) oder der durchschnittliche dB SPL-Wert (`ShowAveragedBspl()`).
- `isDelayOver` wird auf `true` gesetzt, wenn die Startverzögerung (definiert durch `measureThresholdMilliseconds`) von 1200ms abgelaufen ist.
- Es gibt eine kurze Verzögerung (`delay(50)`) zwischen den Schleifendurchläufen, um Tastenprellungen zu vermeiden.

19.8.4. Mikrofondatenverarbeitung

Die Funktion `StartSampling()` initialisiert die Datenverarbeitung:

Listing 19.5.: Initialisierung der Datenverarbeitung

```

1000 * @brief the loop function runs over and over again forever
* 
1002 * standard function of Arduino sketches
*
1004 * The function reads the samples and sends the samples to the display
*
1006 * @param —
*
1008 * @return void
*/

```

..../Code/Nano33BLESense/Test/TestMicrophone.ino

`PDM.begin(1, 16000)`: Initialisiert eine Abtastrate von 16.000 Hz. Die Funktion `StopSampling()` beendet die Aufnahme von Audiodaten.

19.8.5. Anzeige der Ergebnisse

Die Funktionen `ShowResult()` und `ShowMicrophoneValues()` sind für die Anzeige der Messergebnisse auf dem OLED-Display verantwortlich. `ShowResult()` zeigt den aktuellen SPL-Wert in dB auf dem Display an und aktualisiert den maximalen Wert SPL in dB, wenn nach der Startverzögerung ein neuer Höchstwert erreicht wird. `ShowMicrophoneValues()` verarbeitet die vom Mikrofon empfangenen Signale. Der maximale Samplewert wird ermittelt und verwendet, um den Wert SPL in dB zu berechnen. Die Funktion berechnet auch einen Durchschnittswert über eine bestimmte Zeit, `averagingTime = 200ms`, und zeigt das Ergebnis auf dem Display an. Die Funktionen `ShowMaxdBspl()` und `ShowAveragedBspl()` zeigen den maximalen bzw. den durchschnittlichen Wert SPL in dB auf dem Display an.

19.8.6. Umrechnung auf den Wert dB SPL

Die Umrechnung des Mikrofonsignals auf den dB SPL-Wert erfolgt in der Funktion `getDbValueFromPMC()`:

Listing 19.6.: Umrechnung des Mikrofonsignals

```

1000 bool blueButtonIsPressed = analogRead(blueButtonPin) == LOW;
1001 bool redButtonIsPressed = analogRead(redButtonPin) == LOW;
1002 if (blueButtonIsPressed)
1003     isMeasuring = true;

```

..../Code/Nano33BLESense/Test/TestMicrophone.ino

`MaxSampleVoltage` wird berechnet, indem der maximale Samplewert `maxSampleValue` mit der Referenzspannung des Mikrocontrollers („Vref = 3,3 V“) multipliziert und durch den maximalen Wert eines 16-Bit-Signed-Integers (32767) dividiert wird. 32767 ist der maximale Wert eines 16-Bit-Signed-Integers, der der maximalen Amplitude des Audiosignals entspricht. Der dB SPL-Wert („dBspl“) wird mit der Formel „ $20 * \log_{10}(\text{Voltage} / \text{Vrms}) + \text{sensitivity}$ “ berechnet [Quelle der Formel: PDF Decibels Formula https://physicscourses.colorado.edu/phys3330/phys3330_sp19/resources/Decibels_Phys_3330.pdf University of Colorado System], wobei „Voltage“ der berechnete „maxSampleVoltage“ ist, und „sensitivity“ die Empfindlichkeit des Mikrofons in dBV (Dezibel-Volt) ist. „Vrms“ ist die RMS-Spannung (Root Mean Square), die einem Schalldruckpegel von 94 dB SPL entspricht. Dieser Wert wurde experimentell ermittelt.

19.8.7. Benutzeroberfläche

Die Funktion `HandleUI()` aktualisiert OLED-Display Anzeige:

Listing 19.7.: Aktualisierung des Bildschirms

```

1000 * @param —
1001 *
1002 * @return void
1003 */
1004 void StopSampling() {
1005     PDM.end();
1006 }
```

..../Code/Nano33BLESense/Test/TestMicrophone.ino

Wenn eine Messung aktiv ist (`isMeasuring == true`), werden die aktuellen Messwerte auf dem Display angezeigt. Andernfalls wird der Displayinhalt aktualisiert, ohne neue Werte anzuzeigen.

19.9. Tests

The embed on-board MP34DT05 sensor in Arduino Nano 33 BLE Sense has the functionality to sense audio voice from the environment. There is build in Arduino library for this particular sensor, which is PDM as shown in the figure. 19.11

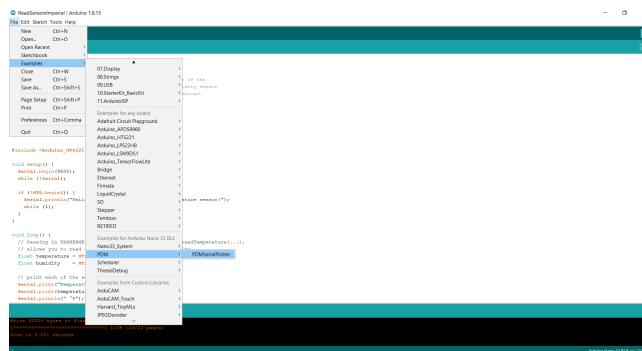


Figure 19.11.: MP34DT05, Digital Microphone

By following the same steps, for this sensor we can see the output the resulted frequency in the serial plotter instead of serial monitor as shown in figure. 19.12 It is more convenient to understand if we change the loudness of voice the plotter shows us a different frequency curve. MP34DT05 use to detect different voices or words too, with the help of these functionality we can easily make the valuable application with Arduino

Nano 33 BLE Sense by adding some external devices with GPIO pins with the help of 3.3V relay.

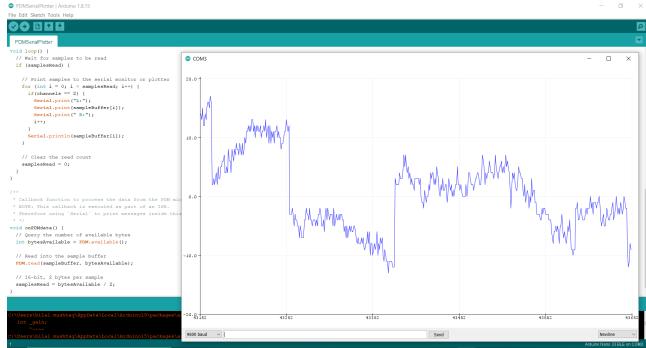


Figure 19.12.: MP34DT05, Serial Plotter

19.9.1. Simple Function Test

19.9.2. Test all Functions

The microphone sensor on the Arduino Nano 33 BLE Sense board is an integrated omnidirectional MEMS microphone. It is designed to capture sound waves and convert them into electrical signals, making it suitable for applications in sound detection, audio processing, and voice recognition.

19.10. Specific Sensor

The MP34DT05 is a compact, low-power omnidirectional digital MEMS microphone with an IC interface. It has a 64 dB signal-to-noise ratio, is capable of sensing acoustic waves and can operate in temperatures of -40 °C to +85 °C.

19.11. Specification

- **Type:** MEMS digital microphone
- **Output Format:** PDM
- **Sensitivity:** -26 dBFS
- **Signal-to-Noise Ratio (SNR):** 64 dB
- **Frequency Range:** 20 Hz to 20 kHz

19.12. PDM Library

To access the data from the MP34DT05, we need to use the PDM library that is included in the Arduino Mbed OS Nano Boards Package. If the Board Package is installed, you will find an example that works by browsing File > Examples > PDM > PDMSerialPlotter.

```
#include <PDM.h>
```

19.13. Calibration

The microphone typically does not require calibration for standard applications. However, in advanced audio analysis or machine learning applications, calibrating for specific environments or sound pressure levels may be necessary.

19.14. Simple Code

Below is an example of code to read data from the microphone:

Listing 19.8.: PDM Microphone Example Code

```

1000 /**
1001 * @file PDM_Microphone.ino
1002 * @brief Example to capture and process audio data using PDM microphone.
1003 *
1004 * This example demonstrates how to initialize the PDM microphone,
1005 * read audio samples into a buffer, and print them to the Serial Monitor
1006 .
1007 *
1008 * @note Ensure the PDM library is installed, and the board supports PDM
1009 * input.
1010 */
1011
1012 #include <PDM.h> //;< Include the library for PDM microphone.
1013
1014 void setup() {
1015     Serial.begin(9600); //;< Start Serial communication at 9600 baud rate.
1016
1017     // Initialize the PDM microphone with 1 channel and 16 kHz sample rate
1018
1019     if (!PDM.begin(1, 16000)) {
1020         Serial.println("Failed to start PDM!"); //;< Error message if
1021         initialization fails.
1022         while (1); //;< Halt the program if PDM initialization fails.
1023     }
1024 }
1025
1026 void loop() {
1027     // Check how many bytes are available in the buffer.
1028     int bytesAvailable = PDM.available();
1029     if (bytesAvailable > 0) {
1030         short buffer[bytesAvailable]; //;< Declare a buffer to hold audio
1031         samples.
1032
1033         // Read the audio samples into the buffer.
1034         PDM.read(buffer, bytesAvailable);
1035
1036         // Print each sample to the Serial Monitor.
1037         for (int i = 0; i < bytesAvailable / 2; i++) {
1038             Serial.println(buffer[i]);
1039         }
1040     }
1041 }
```

..../Code/Nano33BLESense/MikrophoneMP34DT05/MikrophoneMP34DT05/MikrophoneMP34DT05.ino

19.15. Simple Application

The microphone can be used in a sound level meter or voice-controlled interface. For instance:

- Detect claps for controlling lights.
- Capture audio for keyword spotting in machine learning applications.

19.16. Tests

- **Signal Test:** Verify that the microphone captures sound by observing output values on the serial monitor when exposed to different sound levels.
- **Frequency Response Test:** Use a frequency generator to check the microphone's response across its frequency range.
- **Noise Test:** Measure the baseline noise level in a quiet environment.

19.17. Further Readings

- Arduino Nano 33 BLE Sense Official Cheat Sheet: <https://docs.arduino.cc/tutorials/nano-33-ble-sense/cheat-sheet>
- PDM Library Documentation: <https://www.arduino.cc/reference/en/libraries/pdm/>
- MEMS Microphone Datasheet: <https://www.st.com/resource/en/datasheet/mp34dt05-a.pdf>

WS:citations

20. Inertial Measurement Unit

20.1. Introduction

An Inertial Measurement Unit (IMU) is an electronic device that measures and reports a body's specific force, angular rate, and sometimes the orientation of the body. It consists of a combination of three sensors accelerometers, gyroscopes, and magnetometer that work together to provide information about an object's acceleration, velocity, orientation, and gravitational forces.[Sab11]

Arduino's library [Arduino_LSM9DS1](#) of the LSM9DS1 sensor contains three examples that allow the user to use one of the sensors with little effort.

WS:missing citation

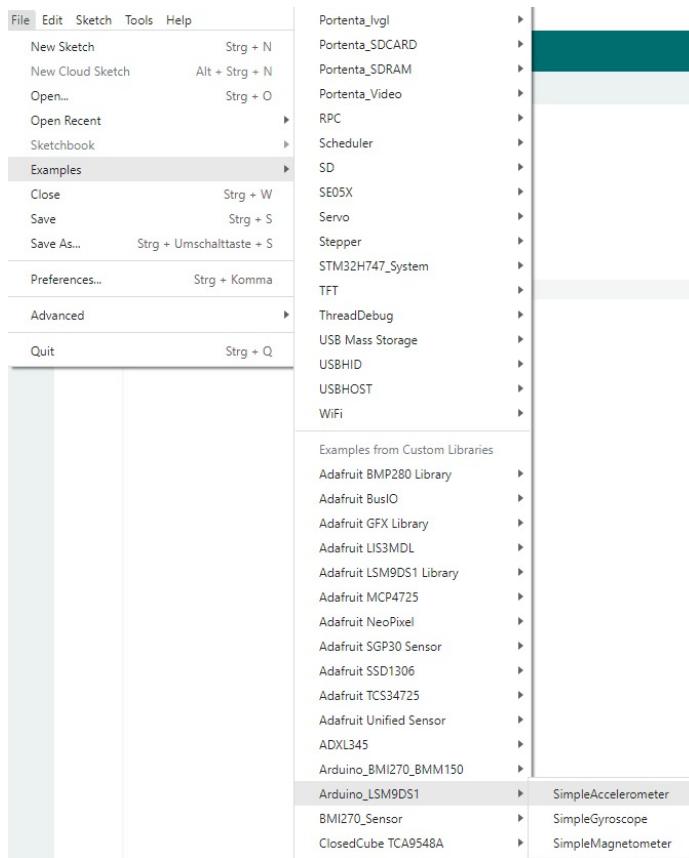


Figure 20.1.: Examples in the library [Arduino_LSM9DS1](#)

In addition to accelerometers and gyroscopes, IMUs may also include magnetometers, which measure the direction and strength of magnetic fields. These sensors can help provide additional information about the orientation and movement of an object in relation to Earth's magnetic field.[Wan+22]

Inertial Measurement Units (IMUs) are commonly used in various fields such as aerospace, robotics, and gaming. The device is made up of multiple sensors that can

measure the acceleration and angular rate of an object in three dimensions. The accelerometers measure linear motion in three axes (x, y, z), while the gyroscopes measure angular motion around these same axes.[Wah+11]

20.2. 6-Axis IMU LSM6DSOX

The LSM6DSOX is a 6-axis IMU developed by STMicroelectronics that features a high-performance 3-axis digital accelerometer and 3-axis digital gyroscope. The device is designed for accurate motion tracking and is commonly used in applications such as wearable devices and smart phones. It can measure linear and rotational motion simultaneously.[Stma] The device also includes a variety of other features such as a programmable digital signal processor (DSP), a configurable low-pass filter, and a built-in temperature sensor.

20.3. IMU LSM6DSOX Features

Features of the LSM6DSOX IMU, see [Stma], typically referring to the technical specifications and capabilities of the sensor.

Here are the features of LSM6DSOX IMU features:

- **Power consumption:**

Power consumption is an important factor when it comes to battery-powered devices like night vision and smart phones. These devices are designed to be portable and convenient, and they rely on batteries to power them. Therefore, the power consumption of each component is a critical consideration to extend battery life and to provide better user experience.

The LSM6DSOX has a power consumption of 0.55 mA in combo high-performance mode. This means that the sensor can operate at a low power consumption level while still delivering high-performance data. In this mode, the sensor can measure both acceleration and angular rate simultaneously, and provide accurate and reliable data with a high sampling rate.

The combo high-performance mode is an optimized mode that reduces power consumption without compromising on data accuracy and reliability. It achieves this by using advanced algorithms and signal processing techniques to filter out noise and unwanted signals, resulting in high-quality data with low power consumption.

- **Always-on:**

This means that the LSM6DSOX can be kept running all the time, even when a device is in sleep mode, without using up too much power. This is useful for applications that require continuous motion tracking, such as fitness tracking or navigation.

- **Smart FIFO:**

The Smart FIFO (First In First Out) is a buffer that can store up to 9 kilobytes of motion data. It is "smart" because it can be configured to store only the most important data, saving memory and reducing power consumption.[MAB22]

- **Android compatibility:**

The LSM6DSOX is compatible with the Android operating system, which is used in many smartphones and tablets.

- **Accelerometer full scale:**

The accelerometer in the LSM6DSOX can detect motion in up to four different full-scale ranges, from $\pm 2g$ to $\pm 16g$. full scale refers to the maximum range of acceleration that can be measured by the sensor without saturating or clipping the output signal.

For example, if an accelerometer has a full scale range of $\pm 2g$, [LAH22] it means that the sensor can accurately measure accelerations up to $2g$ in both the positive and negative directions. If the measured acceleration exceeds this range, the sensor output will saturate at the maximum value, which may cause distortion in the output signal.

- **Gyroscope full scale:**

The gyroscope in the LSM6DSOX can detect rotation in up to five different full-scale ranges, from ± 125 degrees per second (dps) to ± 2000 dps. Each range corresponds to a certain maximum angular velocity that the sensor can detect.

- **Analog supply voltage:**

1.71 V to 3.6 V: This is the voltage range that the LSM6DSOX can operate at.

- **Independent IO supply:**

The input/output connections on the LSM6DSOX can operate at a separate voltage of 1.62 V.

- **Compact footprint:**

2.5 mm x 3 mm x 0.83 mm: This is the size of the LSM6DSOX package, which is small enough to fit many types of electronic devices.

- **SPI / I²C & MIPI I3CSM serial interface with main processor data synchronization:**

These are three different types of communication interfaces that the LSM6DSOX supports. The main processor data synchronization means that the data collected by the sensor can be synchronized with other sensors or data sources.

- **Auxiliary SPI for OIS data output for gyroscope and accelerometer:**

This is an additional interface that can be used to output data from the gyroscope and accelerometer.

- **OIS configurable from Aux SPI, primary interface (SPI/I²C & MIPI I3CSM):**

The OIS (optical image stabilization) feature of the LSM6DSOX can be configured using either the auxiliary SPI or one of the other serial interfaces.

- **Advanced pedometer, step detector, and step counter:**

These features allow the LSM6DSOX to accurately track the number of steps taken by a person wearing a device that contains the sensor.

- **Significant Motion Detection, Tilt detection:**

The LSM6DSOX can detect significant motion events, such as a sudden acceleration or deceleration, as well as changes in orientation or tilt.

- **Standard interrupts:**

Free-fall, wakeup, 6D/4D orientation, click and double-click: These are pre-programmed events that can trigger.

20.4. IMU LSM6DSOX Data

The LSM6DSOX is a type of Inertial Measurement Unit (IMU) sensor that measures acceleration, angular speed and temperature. The data output from this sensor includes acceleration, gyroscope, and temperature measurements.

Accelerometer that measures the acceleration (A_x , A_y , A_z) the rate of change of acceleration over time. The acceleration in each axis is typically reported in units of meters per second squared (m/s^2). For example, if the LSM6DSOX measures an acceleration of $5\ m/s^2$ in the x-axis, it means that the object being measured is increasing in velocity by 5 meters per second every second in the x-direction. See figure 20.2

Gyroscope that measures the angular speed (Ω_x , Ω_y , Ω_z) is a measure of the rate of change of the orientation of the object being measured with respect to each axis. The angular velocity in each axis is typically reported in units of degrees per second ($^\circ/s$). For example, if the LSM6DSOX measures an angular velocity of $100^\circ/s$ in the z-axis, it means that the object being measured is rotating at a rate of 100 degrees per second around the z-axis. See figure 20.3

Temperature sensor that measures the ambient temperature (T) in degrees celsius ($^\circ C$). The temperature sensor is located on the same chip as the accelerometer and gyroscope, and it operates by detecting changes in the voltage output of a diode that is sensitive to temperature. The temperature sensor can be used in a variety of applications, such as monitoring the temperature of electronic devices, measuring the temperature of an environment in which the sensor is placed, or compensating for temperature changes in the output of the accelerometer and gyroscope.

20.4.1. Accelerometer:

[Chi+06]

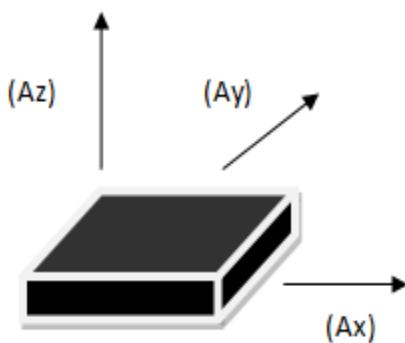


Figure 20.2.: Axis Acceleration of Accelerometer Sensor

WS:tikz!

- Acceleration in X-axis (A_x): meters per second squared (m/s^2)
- Acceleration in Y-axis (A_y): meters per second squared (m/s^2)
- Acceleration in Z-axis (A_z): meters per second squared (m/s^2)

20.4.2. Gyroscope

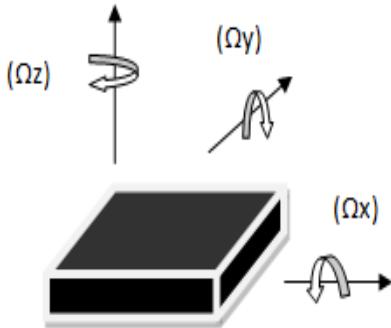


Figure 20.3.: Angular Speed of Gyroscope Sensor

- Angular speed in X-axis (Roll, Ω_x): degrees per second ($^{\circ}/s$)
- Angular speed in Y-axis (Pitch, Ω_y): degrees per second ($^{\circ}/s$)
- Angular speed in Z-axis (Yaw, Ω_z): degrees per second ($^{\circ}/s$)

20.5. Library setup in Arduino IDE

Install the LSM6DSOX Library:

- In the Arduino IDE, go to "Sketch" > "Include Library" > "Manage Libraries..." .
- The Library Manager will open, providing a search bar. Type "LSM6DSOX" into the search bar. Look for the library called "LSM6DSOX" in the search results.
- Click on the library and then click the "Install" button to install the library.

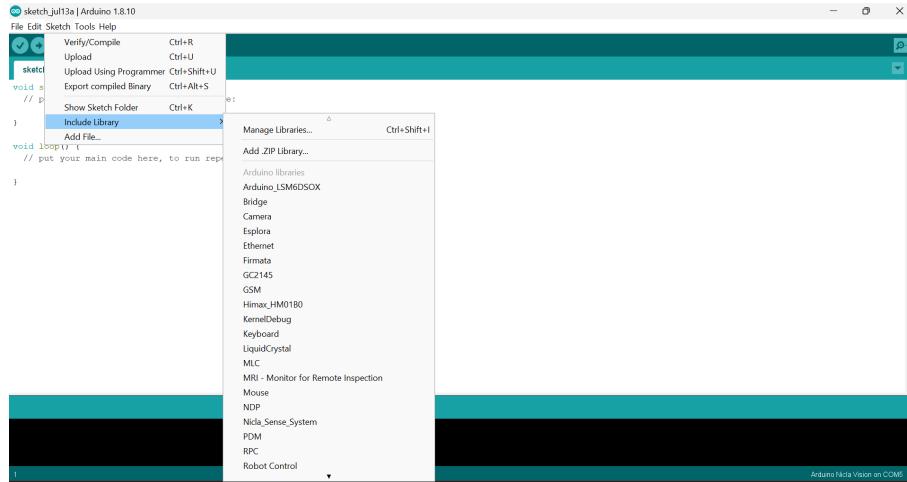


Figure 20.4.: Library setup in Arduino IDE

20.6. Applications

The LSM6DSOX 6-axis IMU (Inertial Measurement Unit) is a versatile sensor that combines a 3-axis accelerometer and a 3-axis gyroscope.

1. **Motion tracking and gesture detection:** The LSM6DSOX IMU can be used to track the motion of a device and detect gestures such as shaking, tilting, and rotating. This application is particularly useful in gaming, virtual reality, and augmented reality applications.
2. **Sensor hub:** The LSM6DSOX IMU can act as a sensor hub, collecting data from other sensors such as magnetometers, barometers, and GPS sensors. This enables the sensor to provide a more complete picture of the device's orientation and motion.[ŠDS17]
3. **Indoor navigation:** The LSM6DSOX IMU can be used for indoor navigation by tracking the device's motion and orientation. This application is useful in navigation and location-based services in indoor environments where GPS signals are not available.
4. **IoT and connected devices:** The LSM6DSOX IMU is an ideal sensor for IoT (Internet of Things) and connected devices. It can provide motion tracking data to a wide range of devices, such as smart homes, wearables, and industrial IoT devices.[Tri+22]
5. **Smart power saving for handheld devices:** The LSM6DSOX IMU can be used to optimize power consumption in handheld devices by detecting when the device is idle or in motion. This information can be used to adjust the device's power settings and conserve battery life.
6. **EIS and OIS for camera applications:** The LSM6DSOX IMU can be used to provide electronic image stabilization (EIS) and optical image stabilization (OIS) in camera applications. This allows for smoother video and reduces camera shake and blurring.
7. **Vibration monitoring and compensation:** The LSM6DSOX IMU can be used to monitor vibration levels in machinery and compensate for the effects of vibration. This is useful in industrial applications where vibration can cause damage or reduce the performance of machinery.

20.7. LSM9DS1(IMU)

20.8. Features

- 3 acceleration channels, 3 angular rate channels, 3 magnetic field channels.
- $\pm 2/\pm 4/\pm 8/\pm 16$ g linear acceleration full scale.
- $\pm 4/\pm 8/\pm 12/\pm 16$ gauss magnetic full scale.
- $\pm 245/\pm 500/\pm 2000$ dps angular rate full scale.
- 16-bit data output.

Applications

- Indoor navigation.
- Smart user interfaces.
- Advanced gesture recognition.
- Gaming and virtual reality input devices.
- Display/map orientation and browsing.

Description

The LSM9DS1 is a system-in-package featuring a 3D digital linear acceleration sensor, a 3D digital angular rate sensor, and a 3D digital magnetic sensor.

20.9. Pin description LSM9DS1

20.10. Pin connections LSM9DS1

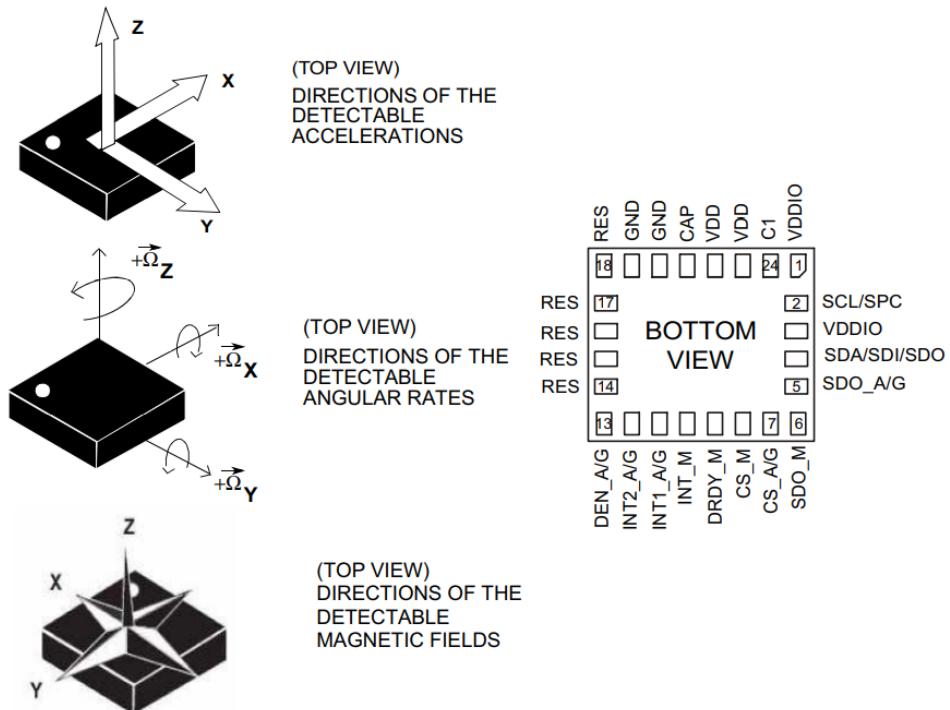


Figure 20.5.: Pin connections LSM9DS1
[Stmb]

Pin #	Name	Function
1	VDDIO ⁽¹⁾	Power supply for I/O pins
2	SCL/SPC	I ² C serial clock (SCL) / SPI serial port clock (SPC)
3	VDDIO ⁽²⁾	Power supply for I/O pins
4	SDA/SDI/SDO	I ² C serial data (SDA) SPI serial data input (SDI) 3-wire interface serial data output (SDO)
5	SDO_A/G	SPI serial data output (SDO) for the accelerometer and gyroscope I ² C least significant bit of the device address (SA0) for the accelerometer and gyroscope
6	SDO_M	SPI serial data output (SDO) for the magnetometer I ² C least significant bit of the device address (SA0) for the magnetometer
7	CS_A/G	SPI enable I ² C/SPI mode selection for the accelerometer and gyroscope (1: SPI idle mode / I ² C communication enabled; 0: SPI communication mode / I ² C disabled)
8	CS_M	SPI enable I ² C/SPI mode selection for the magnetometer (1: SPI idle mode / I ² C communication enabled; 0: SPI communication mode / I ² C disabled)
9	DRDY_M	Magnetic sensor data ready
10	INT_M	Magnetic sensor interrupt
11	INT1_A/G	Accelerometer and gyroscope interrupt 1
12	INT2_A/G	Accelerometer and gyroscope interrupt 2
13	DEN_A/G	Accelerometer and gyroscope data enable
14	RES	Reserved. Connected to GND.
15	RES	Reserved. Connected to GND.
16	RES	Reserved. Connected to GND.
17	RES	Reserved. Connected to GND.
18	RES	Reserved. Connected to GND.
19	GND	0 V supply
20	GND	0 V supply
21	CAP	Connected to GND with ceramic capacitor ⁽³⁾
22	VDD ⁽⁴⁾	Power supply
23	VDD ⁽⁵⁾	Power supply
24	C1	Capacitor connection (C1 = 100 nF)

Figure 20.6.: Pin description LSM9DS1
[Stmb]

Symbol	Parameter	Test conditions	Min.	Typ. ⁽¹⁾	Max.	Unit
LA_FS	Linear acceleration measurement range			±2		g
				±4		
				±8		
				±16		
M_FS	Magnetic measurement range			±4		gauss
				±8		
				±12		
				±16		
G_FS	Angular rate measurement range			±245		dps
				±500		
				±2000		
LA_So	Linear acceleration sensitivity	Linear acceleration FS = ±2 g		0.061		mg/LSB
		Linear acceleration FS = ±4 g		0.122		
		Linear acceleration FS = ±8 g		0.244		
		Linear acceleration FS = ±16 g		0.732		
M_GN	Magnetic sensitivity	Magnetic FS = ±4 gauss		0.14		mgauss/ LSB
		Magnetic FS = ±8 gauss		0.29		
		Magnetic FS = ±12 gauss		0.43		
		Magnetic FS = ±16 gauss		0.58		
G_So	Angular rate sensitivity	Angular rate FS = ±245 dps		8.75		mdps/ LSB
		Angular rate FS = ±500 dps		17.50		
		Angular rate FS = ±2000 dps		70		
LA_TyOff	Linear acceleration typical zero-g level offset accuracy ⁽²⁾	FS = ±8 g		±90		mg
M_TyOff	Zero-gauss level ⁽³⁾	FS = ±4 gauss		±1		gauss
G_TyOff	Angular rate typical zero-rate level ⁽⁴⁾	FS = ±2000 dps		±30		dps
M_DF	Magnetic disturbance field	Zero-gauss offset starts to degrade			50	gauss
Top	Operating temperature range		-40		+85	°C

Figure 20.7.: Sensor Characteristics
[Stmb]

Pin description LSM9DS1

20.10.1. Module specifications

Sensor characteristics

Temperature Sensor characteristics

Symbol	Parameter	Test condition	Min.	Typ. ⁽¹⁾	Max.	Unit
TODR	Temperature refresh rate	Gyro OFF ⁽²⁾		50		Hz
		Gyro ON		59.5		
TSen	Temperature sensitivity ⁽³⁾			16		LSB/°C
Top	Operating temperature range		-40		+85	°C

Figure 20.8.: Temperature Sensor Characteristics
[Stmb]

Absolute Maximum Ratings

Stresses above those listed as “Absolute maximum ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the

device under these conditions is not implied. Exposure to maximum rating conditions for extended periods may affect device reliability

Symbol	Ratings	Maximum value	Unit
Vdd	Supply voltage	-0.3 to 4.8	V
Vdd_IO	I/O pins supply voltage	-0.3 to 4.8	V
Vin	Input voltage on any control pin (including CS_A/G, CS_M, SCL/SPC, SDA/SDI/SDO, SDO_A/G, SDO_M)	0.3 to Vdd_IO +0.3	V
A _{UNP}	Acceleration (any axis)	3,000 for 0.5 ms	g
		10,000 for 0.1 ms	g
M _{EF}	Maximum exposed field	1000	gauss
ESD	Electrostatic discharge protection (HBM)	2	kV
T _{STG}	Storage temperature range	-40 to +125	°C

Figure 20.9.: Absolute Maximum Temperature
[Stmb]

20.10.2. Block Diagram

Accelerometer and gyroscope digital block diagram

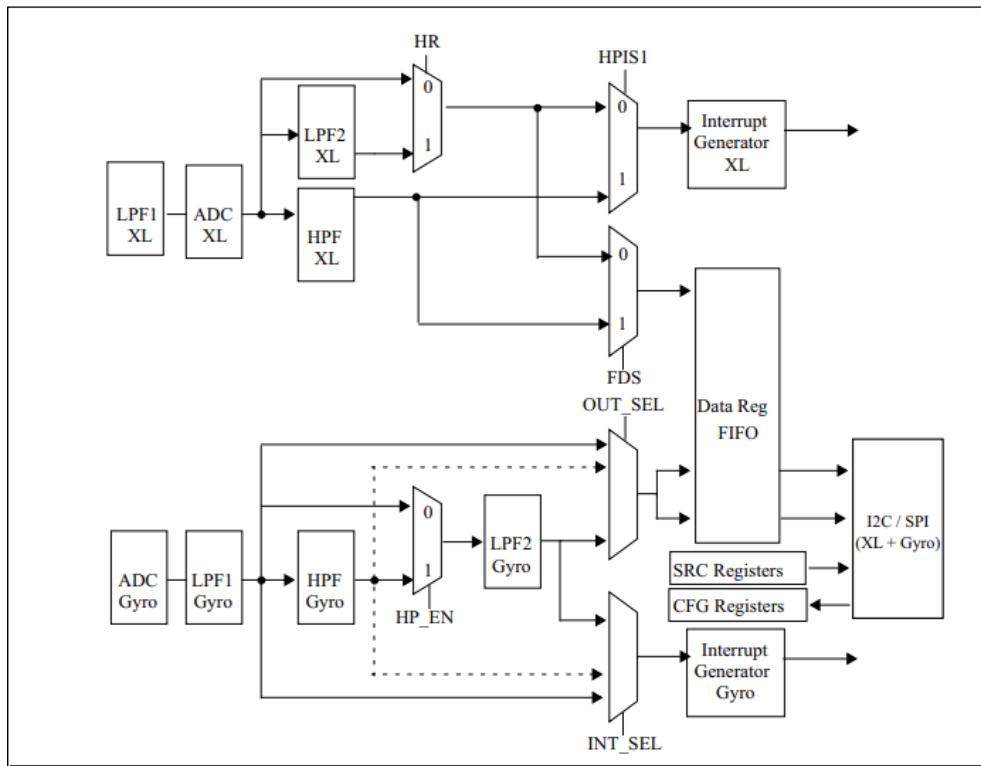


Figure 20.10.: Accelerometer and gyroscope block diagram
[Stmb]

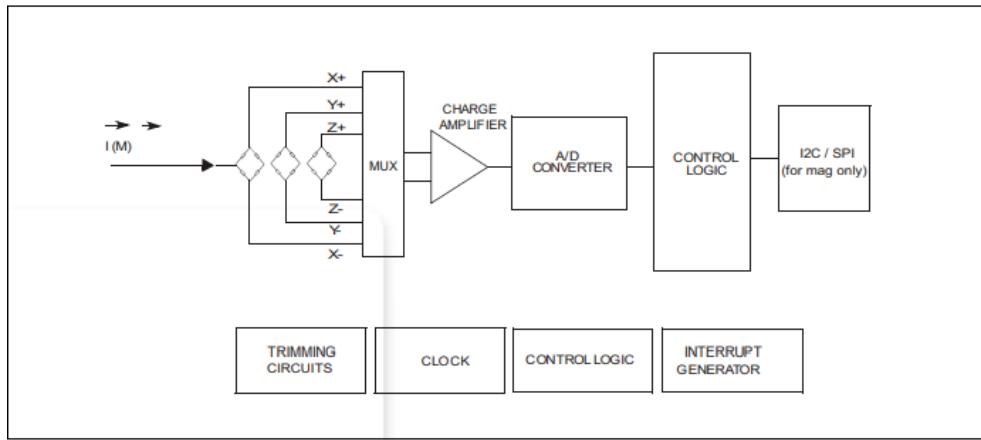
Magnetometer digital block diagram

Figure 20.11.: Magnetometer block diagram
[Stmb]

LSM9DS1 electrical connections

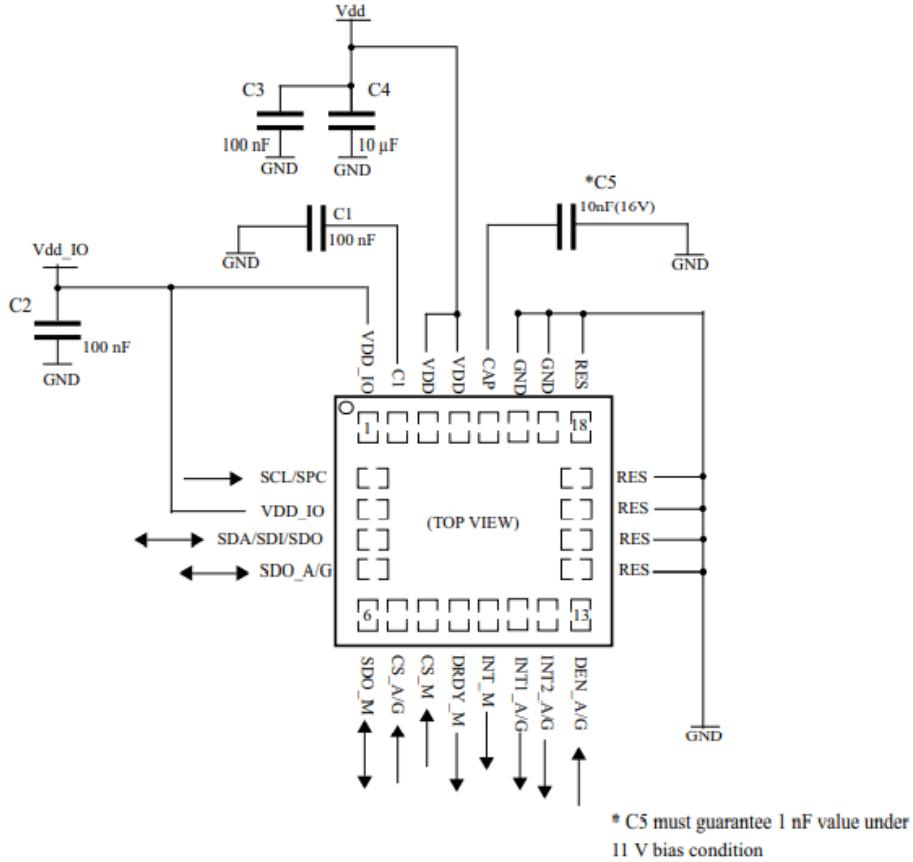


Figure 20.12.: LSM9DS1 electrical connections
[Stmb]

20.10.3. System Requirements

Operating System	Windows 7 or above, MacOS X or higher
CPU	Intel i3 or above
RAM	Min 2GB
Arduino IDE	V. 1.1819
Boards	Arduino mBED OS Nano
Libraries	LSM9DS1
Interface	USB port

20.10.4. Precautions to be taken

- This equipment complies with RF radiation exposure limits set forth for an uncontrolled environment.
- This Transmitter must not be co-located or operating in conjunction with any other antenna or transmitter.

- The operating temperature of the EUT can't exceed 85°C and shouldn't be lower than -40°C.
- The Frequency band should be in between 863-870Mhz.
- Arduino Nano 33 BLE only supports 3.3V I/Os and is NOT 5V tolerant so please make sure you are not directly connecting 5V signals to this board or it will be damaged.
- Keep away from water or fire.
- Hold it gently, do not harm the components and sensors present on the board.

20.11. Bibliotheken

Eine Bibliothek ist eine Sammlung von Quelltext und Funktionen, die es dem Anwender ermöglicht, zum Beispiel jegliche Sensoren bedienen zu können, ohne alle Rohdaten selbst zu verarbeiten. Für dieses Projekt wird die Bibliothek [Arduino_LSM9DS1](#) des angesprochenen LSM9DS1 benötigt. In dieser Bibliothek sind die Funktionen enthalten, um die Bewegungen zu erfassen und in Winkel umzurechnen. Zusätzlich wird die Bibliothek [SSD1306Ascii](#) zur Zeichendarstellung für das OLED-Display benötigt.

WS:citations for the lib

20.12. Beispiele auf dem Mikrocontroller

20.12.1. Testen des Sensors LSM9DS1

Das Beispiel [SimpleAccelerometer](#) wurde geladen und der Sensor gibt erfolgreich die Daten der Beschleunigung aus.



```

Output Serial Monitor ×
Message (Enter to send message to 'Arduino Nano 33 BLE' on 'COM4')
No Line Ending ▾ 2000000 baud ▾
Started
Accelerometer sample rate = 119.00 Hz

Acceleration in g's
X   Y   Z
0.02 -0.03 0.89
0.14 -0.08 1.41
0.01 -0.03 0.88
0.02 -0.03 0.97
0.02 -0.03 0.98
0.02 -0.03 0.98
0.02 -0.03 0.98
0.02 -0.03 0.98
0.02 -0.03 0.98
0.02 -0.03 0.97

```

Figure 20.13.: Output des Testprogramms

20.13. Programmierung

20.13.1. Programmablaufplan

20.13.2. Programmcode und Dokumentation

```

1000 #include <Arduino_LSM9DS1.h>
1001 #include <Wire.h>
1002 #include "SSD1306Ascii.h"
1003 #include "SSD1306AsciiWire.h"

```

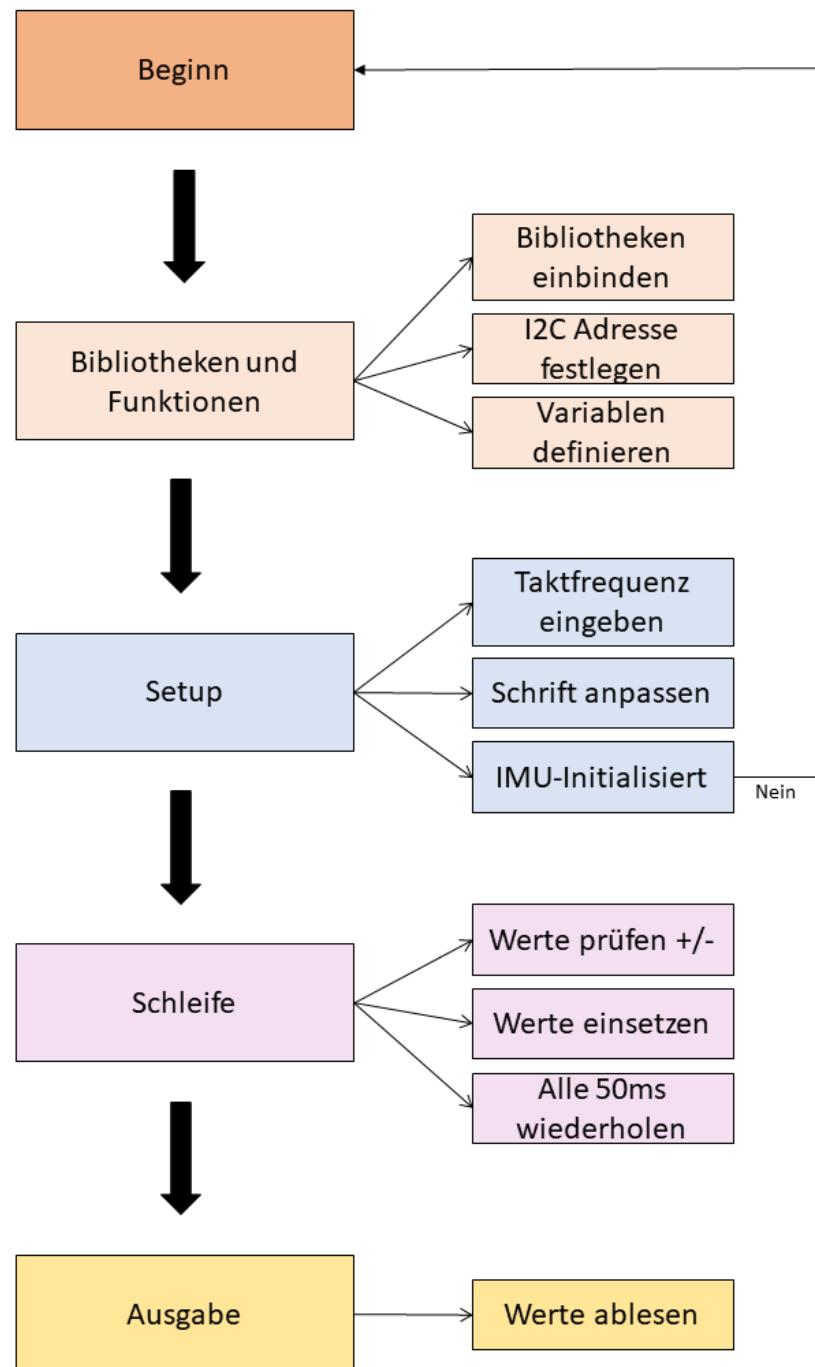


Figure 20.14.: Der Programmablaufplan

```

1004 SSD1306AsciiWire oled;
1006 #define I2C_ADDRESS 0x3C
1008 #define RST_PIN -1

```

Mit `#include` werden die Bibliotheken eingebunden. In diesem Fall werden die Bibliotheken des Sensors LSM9DS1 und die für das Display erforderliche `Wire.h` und `SSD1306Ascii.h` eingebunden. Über `#define` werden die Adressen festgelegt, damit eine Kommunikation stattfinden kann.

```

1000 float x, y, z;
1001 int degreesX = 0;
1002 int degreesY = 0;
1003 String yPre, xPre;

```

Mit `float` (Gleitkommazahl / 4 Bytes) und `Int` (Ganzzahl / 4 Bytes) werden numerische Variablen definiert. In diesem Fall haben wir `degreesX` und `degreesY`, in denen wir unsere jeweiligen X- und Y-Werte für die Ausgabe erhalten. Diese werden zu Beginn auf 0 gesetzt. Die Strings (String=Zeichenkette) `yPre` und `xPre` werden hier implementiert um später die Vorzeichen der X- und Y-Achse zwischen zu speichern.

Nun beginnt das Setup. Hier handelt es sich um eine Funktion, die nur ein einziges Mal aufgerufen wird, sobald der Arduino startet. Da keine Rückgabewerte aus der Methode benötigt werden, wird die Funktionstype `void` verwendet. Die Funktion `Wire.setClock` definiert die Takt Frequenz für die I2C-Kommunikation.

```

1000 void setup()
{
1001   Wire.begin();
1002   Wire.setClock(400000L);
1004
1005 #if RST_PIN >= 0
1006   oled.begin(&Adafruit128x64, I2C_ADDRESS, RST_PIN);
1007 #else // RST_PIN >= 0
1008   oled.begin(&Adafruit128x64, I2C_ADDRESS);
1009 #endif // RST_PIN >= 0
1010
1011   oled.setFont(TimesNewRoman16_bold);
1012   oled.clear();
1013   oled.println(" IMU Bereit ");
1014
1015   if (!IMU.begin())
1016   {
1017     oled.clear();
1018     oled.println(" Failed to initialize IMU! ");
1019     while (1);
1020   }
1021
1022   xPre = " ";
1023   yPre = " ";
1024 }

```

Die Funktion `oled.setFont` wird verwendet, um die Schriftart und Größe zu ändern. Bei der verwendeten Schriftart handelt es sich um `TimesNewRoman`. Um das Ablesen der Werte zu erleichtern ist die Schriftgröße 16 eingestellt. Der Befehl `oled.println` gibt nach dem Einschalten des Arduinos auf dem Display `"IMU Bereit"` aus. So soll dem Anwender mitgeteilt werden, dass die Messungen gestartet werden können.

Falls die IMU nicht einsatzbereit sein sollte, wird über die `if`-Schleife `"Failed to initialize IMU!"` ausgegeben. Mit `xPre` und `yPre` werden aus optischen Gründen drei Leerzeichen definiert. So befinden sich auf der Anzeige die Messwerte geordnet übereinander.

`void loop` bezeichnet eine Funktion, die jedes mal wenn sie am Ende ist wieder von vorne beginnt. In dieser Schleife werden die Werte, die der Sensor ausgibt, immer wieder aufgerufen und in die entsprechenden Ausgaben eingefügt. Dies sorgt für die Aktualisierung auf dem OLED-Display.

In der ersten `if`-Abfrage wird abgefragt, ob der X-Wert größer ist als `0.1`. Wenn dies der Fall ist, merkt sich das Programm mit `xPre` das positive Vorzeichen. Die nächsten beiden `if`-Abfragen überprüfen, ob die entsprechenden Werte kleiner gleich 10 Grad sind (siehe Kap. 4.6). Wenn der Y-Wert kleiner gleich 10 Grad ist, gibt das Display die Ausgabe `" X 0 Grad"` aus. Sobald der Wert größer als 10 Grad ist, beginnt `else` und gibt dem Display die Anweisungen `oled.print(" X ")`; für die Ausgabe des X-Wertes. Unmittelbar dahinter `oled.print(yPre)`; für das gemerkte Vorzeichen von Y. Nun wird der vom Sensor gemessene Y-Wert für X eingesetzt `oled.print(degreesY);`. Zum Schluss wird mit `oled.print(" Grad")`; Grad als Einheit hinter die Zeile gesetzt. Ausgaben wie `oled.print(" ")`; beinhalten lediglich eine Leerzeile zur richtigen Ausrichtung der Werte untereinander. Die Funktion `map()` beschäftigt sich mit der Ganzzahlmathematik, dadurch werden selbst Brüche als ganze Zahlen weitergegeben.

Info: Aufgrund des aufdruckten Koordinatensystems auf dem Gehäuse mussten die X- und Y-Werte getauscht werden, damit es für den Anwender bedienbar ist.

```

1000 void loop()
1001 {
1002     if (IMU.accelerationAvailable())
1003     {
1004         IMU.readAcceleration(x, y, z);
1005     }
1006     if (x > 0.1)
1007     {
1008         x = 100 * x;
1009         degreesX = map(x, 0, 97, 0, 90);
1010
1011         xPre = "+";
1012         oled.clear();
1013         oled.println();
1014
1015         if (degreesY <= 10)
1016         {
1017             oled.println(" X      0      Grad");
1018         }
1019         else
1020         {
1021             oled.print(" X ");
1022             oled.print(yPre);
1023             oled.print(" ");
1024             oled.print(degreesY);
1025             oled.print(" Grad");
1026             oled.println();
1027         }
1028         if (degreesX <= 10)
1029         {
1030             oled.println(" Y      0      Grad");
1031         }
1032         else
1033         {
1034             oled.print(" Y + ");
1035             oled.print(degreesX);
1036             oled.print(" Grad");
1037             oled.println();
1038         }
1039     }
1040 }
```

```
1038 } }
```

Ab der Zeile `if (degreesX <= 10)` erfolgt der gleiche Prozess wie oben beschrieben für den entsprechenden Y-Wert.

Im Folgenden Programmauszug wird die Schleife für die anderen Richtungen wiederholt. Für die X-Werte unter `-0,1` wird mit `xPre = " -";` das Vorzeichen gemerkt und die Ausgaben wiederholen sich mit den entsprechenden Vorzeichen und Werten.

```
1000 if (x < -0.1)
1001 {
1002     x = 100 * x;
1003     degreesX = map(x, 0, -100, 0, 90);
1004
1005     xPre = " -";
1006     oled.clear();
1007     oled.println();
1008
1009     if (degreesY <= 10)
1010     {
1011         oled.println(" X 0 Grad");
1012     }
1013     else
1014     {
1015         oled.print(" X ");
1016         oled.print(yPre);
1017         oled.print(" ");
1018         oled.print(degreesY);
1019         oled.print(" Grad");
1020         oled.println();
1021     }
1022
1023     if (degreesX <= 10)
1024     {
1025         oled.println(" Y 0 Grad");
1026     }
1027     else
1028     {
1029         oled.print(" Y - ");
1030         oled.print(degreesX);
1031         oled.print(" Grad");
1032         oled.println();
1033     }
1034 }
1035
1036 }
```

Hier beginnt die Abfrage für die Y-Werte `if (y > 0.1)`. Sobald das Y positiv ist, wird sich mit `yPre = "+";` das Positive Vorzeichen gemerkt und die Schleife läuft wie vorher schon bei den X-Werten beschrieben.

```
1000 if (y > 0.1)
1001 {
1002     y = 100 * y;
1003     degreesY = map(y, 0, 97, 0, 90);
1004
1005     yPre = "+";
1006     oled.clear();
1007     oled.println();
1008
1009     if (degreesY <= 10)
1010     {
1011         oled.println(" X 0 Grad");
1012     }
1013 }
```

```

1014     else
1015     {
1016         oled.print(" X + ");
1017         oled.print(degreesY);
1018         oled.print(" Grad");
1019         oled.println();
1020     }
1021
1022     if (degreesX <= 10)
1023     {
1024         oled.println(" Y 0 Grad");
1025     }
1026     else
1027     {
1028         oled.print(" Y ");
1029         oled.print(xPre);
1030         oled.print(" ");
1031         oled.print(degreesX);
1032         oled.print(" Grad");
1033         oled.println();
1034     }
1035     if (y < -0.1)
1036     {
1037         y = 100 * y;
1038         degreesY = map(y, 0, -100, 0, 90);
1039
1040         yPre = " -";
1041         oled.clear();
1042         oled.println();
1043
1044         if (degreesY <= 10)
1045         {
1046             oled.println(" X 0 Grad");
1047         }
1048         else
1049         {
1050             oled.print(" X - ");
1051             oled.print(degreesY);
1052             oled.print(" Grad");
1053             oled.println();
1054         }
1055
1056         if (degreesX <= 10)
1057         {
1058             oled.println(" Y 0 Grad");
1059         }
1060         else
1061         {
1062             oled.print(" Y ");
1063             oled.print(xPre);
1064             oled.print(" ");
1065             oled.print(degreesX);
1066             oled.print(" Grad");
1067             oled.println();
1068         }
1069     }
1070     delay(50);
1071 }

```

Hier endet die Schleife. Mit `delay(50)` wird sie alle 50 Millisekunden wiederholt. So wird die Aktualisierung des Displays realisiert. Der Winkelmesser funktioniert also in Echtzeit.

VS:Der Begriff Echtzeit ist nicht bekannt. Hier muss gemessen werden!

20.13.3. Definition Echtzeit

Echtzeit bedeutet, dass ein System auf ein Ereignis innerhalb eines vorgegebenen Zeitrahmens zuverlässig reagieren kann. Das System ist in der Lage, alle Daten innerhalb einer Zykluszeit einzulesen und ausgeben. [Sch05]

Sobald das Programm hochgeladen wurde, gibt es zwei wesentliche Ausgaben auf dem OLED-Display, die der Anwender nun sehen sollte.



Figure 20.15.: Anzeige des Arduino OLED Displays sobald der Arduino eingeschaltet ist

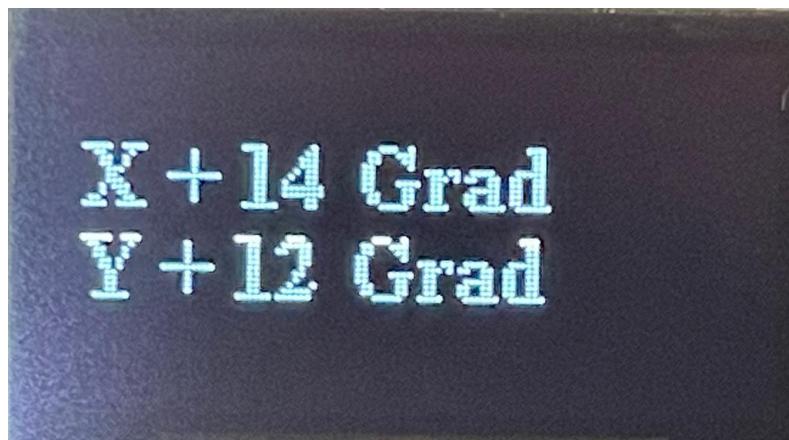


Figure 20.16.: Ausgabe von Messdaten

X+ 14 Grad, Y+ 12 Grad ist eine Ausgabe von einer Beispielbewegung des Arduinos und zeigt dem Benutzer eine Mischung aus einem positiven X- und Y-Wert.

20.14. Kalibrierung

Es war weder möglich eine fertige Kalibrierungssoftware von GitHub oder einen von einer KI erstellten Quelltext zu nutzen, um das Gerät zu kalibrieren. Der Arduino selbst zeigt präzise Winkel bis 90 Grad an, war aber nicht in der Lage Winkel kleiner als 10 Grad auszugeben. Dementsprechend wurde die if-Abfrage eingebaut, um Winkel die kleiner gleich 10 Grad sind auf dem Display als 0 Grad ausgegeben werden.

20.15. Probleme

Aufgrund mehrerer Fehler mit der seriellen Schnittstelle wurde zu Beginn angenommen, dass der Arduino einen Defekt hat. Nach weiteren Nachforschungen stellten wir allerdings fest, dass der Arduino sich ganz einfach zurücksetzen lässt. Mithilfe des kleinen Knopfs auf dem Arduino-Board, startet dieser in den Bootloader und lässt sich dann mit einer manuellen Änderung des COM-Ports wieder bespielen.

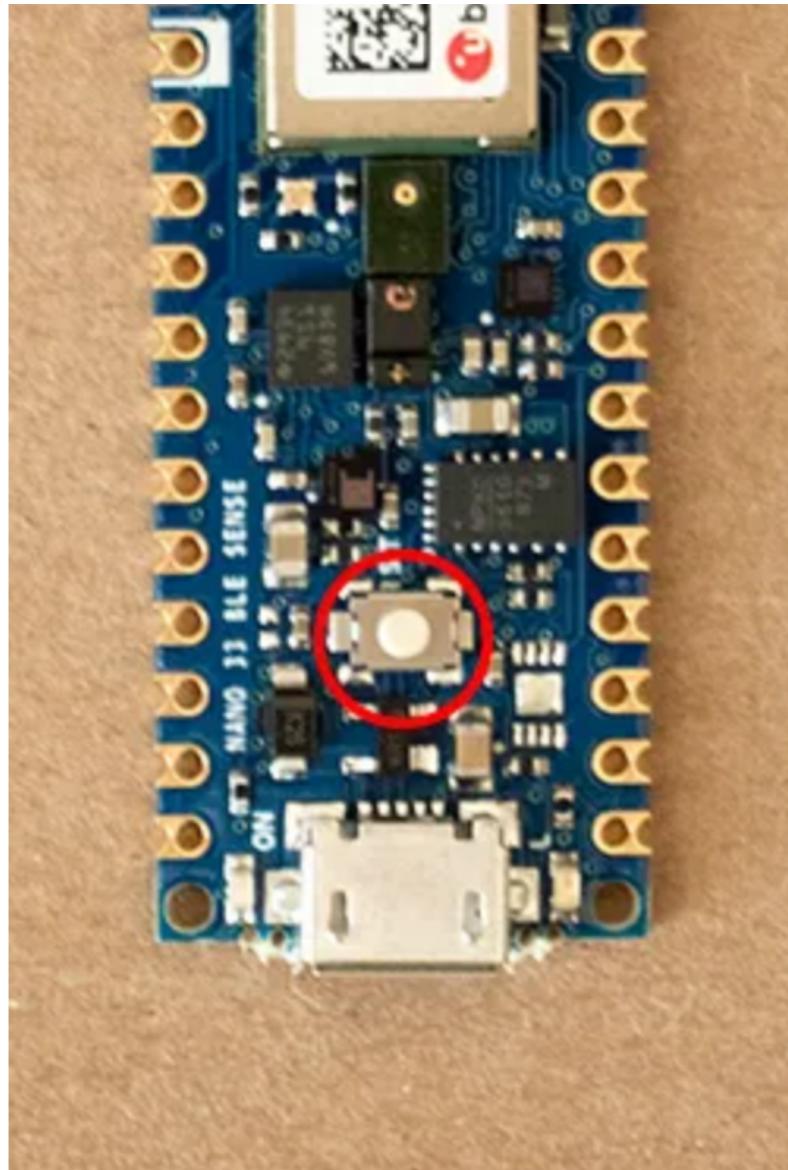


Figure 20.17.: Der Reset Button des Arduino

21. Calibration

21.1. Introduction

Calibration of an IMU (Inertial Measurement Unit) is the process of determining the bias, drift, and noise values of the sensors within the IMU. [Wah+11] This is done by measuring the output of the IMU in multiple positions and orientations and then using algorithms to determine the bias, drift, and noise values. The process is important for ensuring the accuracy of the IMU's measurements.[Vec]

21.2. Standard Operating Procedure

The standard operation procedure for calibrating LSM6DSOX IMU involves the following steps:

- Scope and Measurand(s) of Calibrations
- Description of the Item to be Calibrated
- Measurement Parameters, Quantities, and Ranges to be Determined
- Environmental Conditions and Stabilization Periods
- Procedure Include:
 - Handling, transporting, storing and preparation of items
 - Checks to be made before the work is started
 - Step by step process
- Handling, transporting, storing and preparation of items
- Checks to be made before the work is started
- Step by step process

Scope and Measurand(s) of Calibrations:

- The scope of the calibration process refers to the range of measurements or properties that are being assessed. In this case, the scope of calibration means determining the bias, drift, and noise values of the accelerometer and gyroscope readings. Bias refers to the systematic error in the sensor readings, while drift refers to the change in measurement over time due to factors such as temperature or humidity. Noise refers to the random fluctuations in the sensor readings.
- The measurands to be calibrated are the acceleration and angular velocity values of the sensor. Acceleration refers to the rate of change of velocity, and is typically measured in meters per second squared m/s^2 . Angular velocity refers to the rate of change of angular displacement, and is typically measured in degree per second ($^{\circ}/s$). Both acceleration and angular velocity are important measurements for applications such as robotics, navigation, and motion tracking.

- By calibrating the sensor, the accuracy of the acceleration and angular velocity measurements can be improved, leading to more reliable and precise data. This is especially important in applications where small errors in measurement can have significant consequences, such as in mobile robot.

Description of the Item to be Calibrated:

- The LSM6DSOX IMU sensor is a device that is designed to measure acceleration and angular velocity in three different axes. It is commonly used in applications where the measurement of motion or orientation is required, such as in drones, robotics, and virtual reality systems.
- The sensor uses a combination of accelerometers and gyroscopes to measure motion and orientation. The accelerometers measure changes in acceleration, while the gyroscopes measure changes in angular velocity. Together, these measurements can be used to calculate the orientation and movement of the sensor in three dimensions.
- The LSM6DSOX IMU sensor is a compact device that can be integrated into various systems and devices. It typically includes a microcontroller and communication interface, such as I2C or SPI, for sending the measured data to other devices or systems.

Measurement Parameters, Quantities, and Ranges to be Determined:

- The bias, drift, and noise are parameters that affect the accuracy and stability of the sensor readings. Bias refers to any systematic error in the sensor output that is not related to the input signal. It can be thought of as an offset that needs to be subtracted from the raw sensor output to get a more accurate measurement. Drift refers to the change in the sensor output over time, even when there is no change in the input signal. It can be caused by factors such as temperature changes or aging of the sensor components. Noise refers to the random fluctuations in the sensor output that can be caused by various sources, such as electrical interference or mechanical vibration.
- The quantities to be measured are the acceleration and angular velocity values in each axis of the sensor. Acceleration is measured in units of meters per second squared m/s^2 and angular velocity is measured in units of radians per second (rad/s).
- The range of the measurements will depend on the specifications of the sensor and the conditions in which it is being used. For example, the range of acceleration measurements might be $\pm 2g$, $\pm 4g$, $\pm 8g$, $\pm 16g$ (where g is the acceleration due to gravity) and the range of angular velocity measurements might be : ± 125 , ± 250 , ± 500 , ± 1000 , ± 2000 (degrees per second). The actual range of measurements will be determined by the sensitivity and resolution of the sensor and the specific application in which it is being used.

Environmental Conditions and Stabilization Periods:

- Environmental conditions play a critical role in ensuring accurate calibration of the sensor. Any significant vibration or movement in the environment can cause erroneous readings and introduce error into the calibration process. Therefore, it is essential to ensure that the environment in which the calibration is performed is stable and free from any such disturbances.
- The sensor also requires sufficient stabilization time to ensure that it is at a steady state before calibration. This stabilization period allows the sensor to adjust to

its environment and ensures that any initial drift is stabilized. The length of this stabilization period can vary depending on the sensor's specifications and the specific conditions in which the calibration is being performed. It is crucial to allow sufficient time for the sensor to stabilize before any measurements are taken to ensure accurate and reliable calibration results.

Procedure:**• Handling, transporting, storing, and preparing the items:**

The sensor should be handled carefully to avoid any damage or contamination. It should be transported and stored in a clean and dry environment, away from any sources of electromagnetic interference. Before starting the calibration, the sensor should be mounted securely on a stable surface and connected to the calibration equipment.

• Checks to be made before the work is started:

Before starting the calibration, several checks should be made to ensure that the setup and environment are suitable for calibration. These checks may include verifying equipment is a flat surface, additional checks may include ensuring that the surface is level and stable, and checking that the sensor is securely mounted and positioned correctly on the surface, checking that the sensor is connected and communicating properly, and verifying that the environment is stable and free from any significant vibration or movement.

• Step by step process:**1. Reading the accelerometer and gyroscope values from each axis:**

The sensor should be placed on a stable and flat surface to ensure accurate readings. The readings can be obtained using software designed to communicate with the sensor or a microcontroller. The readings are usually in the form of digital signals, which are then converted into acceleration and angular speed values.

2. Updating the Kalman filter with these values:

Once the readings from each axis are obtained, they are used to update the Kalman filter. The Kalman filter is a mathematical algorithm that estimates the bias, drift, and noise of each axis based on the readings obtained. The filter takes into account previous readings and estimates to provide a more accurate estimate of the current values. The Kalman filter is an essential part of the calibration process as it helps to correct for errors in the sensor readings.

3. Storing these values:

The estimated bias, drift, and noise values for each axis are stored in the sensor's memory or in a separate data storage device. These values can be used to correct for errors in subsequent measurements taken by the sensor. The storage location and format of the values depend on the sensor and the application. Some sensors have built-in memory, while others may require an external storage device. The values may be stored in a binary or text format depending on the application's requirements.

• Measurement Assurance:

– Measurement assurance is a critical aspect of any measurement process, which involves verifying the accuracy, precision, and reliability of the measurements. There are several ways to ensure measurement assurance, including comparing the measured values to known reference values, repeating the calibration process multiple times, and performing statistical analysis of the measurement data.

- Our approach is ensuring measurement assurance is to compare the estimated values to known reference values. This can involve using a reference standard or a calibration artefact with a known value, such as a calibrated weight or a temperature sensor with a known output. By comparing the measured values to the reference values, it is possible to determine the accuracy and precision of the measurement system and make any necessary adjustments to improve the measurement quality. For our LSM6DSOX IMU sensor measurement assurance is to compare all the accelerometer and gyroscope reading to zero when the IMU is in stationery over a flat surface and the temperature reading should be compared with external temperature sensor.

21.2.1. Low and high limit method

The low and high limit method involves recording minimum and maximum values on all three axes using a simple scratch to determine their absolute values. The sensor undergoes circular rotations along each axis multiple times[RS17b]. The centre point is then identified between these extremes. Increasing the number of rotations enhances the likelihood of capturing the absolute peak. The center point will be close to zero if the sensor exhibits no offset. However, slight variations may indicate a hard iron offset attributed to distortion caused by the Earth's magnetic field[RS17b]. This method assumes minimal soft iron distortion, evident from the rounded outlines in the graph. It is important to note that this method necessitates capturing values each time to prevent performance degradation due to component drift and aging sensors[RS17b]. For devices relying on primary batteries, calibration becomes essential after each battery change, as the battery inevitably serves as the main source of magnetic disturbance, and new batteries may behave differently from their predecessors[RS17b].

21.2.2. FreeIMU Calibration Application Magnetometer

In the FreeIMU Calibration Application Magnetometer method, raw magnetometer data undergoes pre-processing with axis-specific gain correction to convert the raw output into nanoTesla [RS17b]:

- $Xm\text{-nanoTesla} = \text{rawCompass.m.x} * (100000.0 / 1100.0);$
- Gain X [LSB/Gauss] for selected input field range;
- $Ym\text{-nanoTesla} = \text{rawCompass.m.y} * (100000.0 / 1100.0);$
- $Zm\text{-nanoTesla} = \text{rawCompass.m.z} * (100000.0 / 980.0);$

The converted data is saved in the Mag-raw.txt file, which can be opened with the Magneto program. To implement this method, the scaling factors(e.g. 100000.0/1100.0) must be replaced with values specific to your sensor to convert the output into nanoTesla. Magneto generates twelve calibration values that correct for various errors, including bias, hard iron, scale factor, soft iron, and misalignment [RS17b]. An additional benefit is that this method can be used to calibrate accelerometers by pre-processing raw accelerometer output, considering bit depth and G sensitivity, converting the data into milliGalileo. We can also enter a value of 1000 milliGalileo as the "norm" for the gravitational field [RS17b].

21.2.3. Example

```
1000 #include <ArduinoSound.h>
1002 void setup() {
1003     Serial.begin(9600);
1004     Sound.begin();
1005 }
1006 void loop() {
1007     int micValue = Sound.read();
1008     Serial.println(micValue);
1009     delay(1000);
1010 }
```

Listing 21.1.: Example Microphone

```
1000 #include <Arduino_LSM9DS1.h>
1002 void setup() {
1003     Serial.begin(9600);
1004     if (!IMU.begin()) {
1005         Serial.println("Failed to initialize IMU!");
1006         while (1);
1007     }
1008 }
1009 void loop() {
1010     float x, y, z;
1011     if (IMU.accelerationAvailable()) {
1012         IMU.readAcceleration(x, y, z);
1013         Serial.print("AccX: "); Serial.print(x);
1014         Serial.print(", AccY: "); Serial.print(y);
1015         Serial.print(", AccZ: "); Serial.println(z);
1016     }
1017     if (IMU.gyroscopeAvailable()) {
1018         IMU.readGyroscope(x, y, z);
1019         Serial.print("GyroX: "); Serial.print(x);
1020         Serial.print(", GyroY: "); Serial.print(y);
1021         Serial.print(", GyroZ: "); Serial.println(z);
1022     }
1023     delay(1000);
1024 }
1025 \end{lstlisting}
```

Listing 21.2.: Example IMU (Accelerometer and Gyroscope)

```
1000 #include <Wire.h>
1001 #include <SparkFun_APDS9960.h>
1002
1003 // Object declaration
1004 SparkFun_APDS9960 apds;
1005
1006 void setup() {
1007     Serial.begin(9600);
1008     // Initialize sensor
1009     if (!apds.init()) {
1010         Serial.println("Failed to initialize sensor!");
1011         while (1);
1012     }
1013     // Enable proximity and gesture sensing
1014     apds.enableProximitySensor(true);
1015     apds.enableGestureSensor(true);
1016     Serial.println("Sensor initialized");
1017 }
1018
1019 void loop() {
1020     // Read proximity value
1021     if (apds.proximityAvailable()) {
1022         uint8_t proximity = apds.readProximity();
1023         Serial.print("Proximity: ");
1024         Serial.println(proximity);
1025     }
1026
1027     // Read gesture
1028     if (apds.isGestureAvailable()) {
1029         uint8_t gesture = apds.readGesture();
1030         Serial.print("Gesture: ");
1031         Serial.println(gesture);
1032     }
1033
1034     delay(1000);
1035 }
```

Listing 21.3.: Example ADPS-9960

22. Errors

22.1. Introduction

IMU (Inertial Measurement Unit) is a sensor that measures and reports the linear and rotational motion of an object. The error in IMU refers to any deviation or inaccuracy in the measurements reported by the sensor from the true or expected values.

1. **Bias:** Bias is the constant offset in the readings of the IMU. It can be caused by a variety of factors, including manufacturing defects, aging of the sensors, or changes in temperature. These errors can lead to systematic errors that can persist over time and can be difficult to detect. To correct for the bias error, the IMU must be calibrated periodically. Calibration involves comparing the IMU's measurements with a known reference, and then estimating and subtracting the bias from the measurements to obtain more accurate results.[Sab11]
2. **Drift:** Drift refers to the gradual change in bias over time. It can be caused by aging of the sensors, temperature changes, or electronic interference. Unlike the bias error, drift can vary over time, and it can be challenging to detect and correct. As a result, it is essential to calibrate the IMU regularly to correct for drift. More advanced techniques such as Kalman filtering can also be used to estimate and compensate for drift over time.[TPM14]
3. **Noise:** Noise refers to the random variations in the sensor readings that can affect the accuracy of the measurement. This error is especially pronounced when the IMU is stationary. The noise can be caused by various factors such as electronic interference, vibrations, or environmental conditions. To reduce the noise error, various techniques such as filtering or averaging can be used. Filtering techniques can help remove random variations in the sensor readings and provide more accurate measurements. Averaging can also be used to reduce noise by averaging out random variations in the measurements over time.[FH14]

22.2. Affected Parameters

Bias, drift, and noise errors will affect all the accelerometers and gyroscopes parameters. Here's how they impact each parameter:

22.2.1. Accelerometer:

- Ax, Ay, and Az: The accelerometer measures linear acceleration in three directions, X, Y, and Z. If the accelerometer experiences bias, there will be a constant offset in the acceleration measurement in all three directions, even when there is no actual acceleration.
- Drift in an accelerometer can cause a slow, gradual change in the measured acceleration values over time. This means that even when the device is stationary, the accelerometer readings may drift away from the true acceleration values. For example, if the accelerometer is used to measure the tilt angle of a platform, drift can cause the tilt angle to slowly change even when the platform is not moving. Over time, this can result in significant errors in the measured tilt angle.
- Noise in an accelerometer can cause random fluctuations in the measured acceleration values. This means that even when the device is stationary, the accelerometer readings may vary randomly around the true acceleration values. For example, if the accelerometer is used to measure the vibration of a machine, noise can cause the measured vibration amplitude to fluctuate randomly around the true amplitude. This random fluctuation can make it difficult to accurately measure the machine's vibration characteristics.

22.2.2. Gyroscope:

- Ω_x , Ω_y , and Ω_z : The gyroscope measures the rate of change of angular velocity in three directions, Roll, Pitch, and Yaw. Bias in the gyroscope can cause a constant offset in the measured angular velocity values, even when there is no actual rotation. [NKG13]
- Drift in gyroscopes is caused by mechanical imperfections in the device, temperature changes, or other external factors that can lead to a gradual change in the measured angular velocity value over time, even when the device is not rotating. The drift error can accumulate over time and can significantly affect the accuracy of the gyroscope measurements. For example, a drone that uses a gyroscope for stabilization can experience drift error over time, causing it to drift off course and potentially crash.
- Noise in gyroscopes is caused by random fluctuations in the measured angular velocity values due to external disturbances such as vibrations or electromagnetic interference. The noise error can affect the precision of the gyroscope measurements and can lead to instability in the application. For example, in robotics, noise error can cause the robot to deviate from its intended path or make inaccurate movements.

To minimize these errors, calibration and filtering techniques can be used. Calibration can remove bias by using known reference values to adjust the sensor's measurements. Zero-g and zero-rate calibration techniques can be used to remove bias from accelerometers and gyroscopes, respectively. Filtering techniques can be used to remove noise and reduce drift. For example, a Kalman filter can be used to estimate the true acceleration or angular velocity values based on previous measurements and sensor models.[LBSK05]

23. IMU LSM6DSOX - Libraries and Functions

23.1. Libraries

A library refers to a collection of pre-written code that can be used by developers to perform specific tasks or functions without having to write the code from scratch. Libraries are designed to make the development process easier and more efficient by providing pre-built solutions to common programming challenges.

23.1.1. Wire.h

Wire.h is a library in Arduino that allows for communication between I2C devices. I2C stands for Inter-Integrated Circuit, which is a synchronous serial communication protocol used for connecting microcontrollers to peripheral devices. Wire.h provides functions for initializing the I2C bus, sending and receiving data over the bus, and managing multiple devices on the same bus. With this library, developers can easily connect multiple I2C devices, such as sensors or displays, to an Arduino board.[Ardb; Ari21]

23.1.2. Kalman.h

Kalman.h is a library that implements the Kalman filter algorithm. The Kalman filter is a mathematical technique used to estimate the state of a system based on incomplete measurements. It is commonly used in control systems, robotics, and navigation applications to improve the accuracy of sensor measurements and reduce errors. Kalman.h provides a simple interface for developers to implement the Kalman filter in their Arduino projects.[Ard19; Fet21]

23.1.3. Arduino_LSM6DSOX.h

Arduino_LSM6DSOX.h is a library that provides access to the LSM6DSOX accelerometer and gyroscope sensor on the Arduino Mbed OS Nicla Board. The LSM6DSOX sensor is a 6-axis sensor that can measure both linear acceleration and angular velocity. Arduino_LSM6DSOX.h provides functions for initializing the sensor, reading data from the sensor, and configuring the sensor parameters. With this library, developers can easily integrate the LSM6DSOX sensor into their Arduino projects and use the sensor data for various applications, such as gesture recognition or orientation detection.[Lib21]

23.1.4. LSM6DSOXSensor.h

[`LSM6DSOXSensor.h`](#) is a library that provides an interface for interacting with the LSM6DSOX sensor. The LSM6DSOX is a 6-axis inertial measurement unit (IMU) sensor that combines a 3-axis accelerometer and a 3-axis gyroscope in a single chip. It is commonly used in applications that require motion sensing and orientation tracking, such as robotics, drones, wearable devices, and Internet of Things (IoT) devices. The LSM6DSOXSensor.h library allows developers to easily interact with the LSM6DSOX sensor by providing functions and classes for configuring the sensor, reading raw sensor

data, and performing sensor fusion to obtain calibrated accelerometer and gyroscope data, as well as derived data such as orientation, linear acceleration, and angular velocity. The library abstracts the low-level communication with the sensor, providing a higher-level API that simplifies the process of working with the sensor.

23.2. Functions

A function is a block of code that performs a specific task or set of tasks. Functions are designed to be reusable and modular, meaning they can be called and executed multiple times throughout a program without having to rewrite the code every time. Functions can take input parameters, perform operations on them, and return output values.

23.2.1. **setup()**

The setup() function is a special function in the Arduino programming language that is called once at the beginning of the program. The purpose of the setup() function is to initialize variables, pin modes, and other settings that are necessary for the program to run correctly. This function is typically used to set up hardware components, such as sensors or displays, and configure any necessary settings, such as communication protocols or data rates.[Ardj]

23.2.2. **loop()**

The loop() function is another special function in the Arduino programming language that is called repeatedly after the setup() function. The purpose of the loop() function is to execute the main code of the program in a continuous loop until the program is terminated. This function is typically used to read sensor data, perform calculations, and control hardware components based on the input data.[Ardi]

23.2.3. **IMU.begin()**

IMU.begin() is a function provided by the Arduino_LSM6DSOX.h library, which is used to initialize the LSM6DSOX accelerometer and gyroscope sensor on the Arduino Mbed OS Nicla Board. The purpose of the IMU.begin() function is to set up the communication between the Arduino board and the LSM6DSOX sensor and to configure the sensor settings to match the requirements of the program. This function is typically called in the setup() function of the program to initialize the sensor before reading data from it in the loop() function.

23.2.4. **IMU.setAccelerometerRange()**

The IMU.setAccelerometerRange() function is used to set the range of the accelerometer on the LSM6DSOX sensor. The accelerometer range determines the maximum acceleration that can be measured by the sensor. This function takes an argument that specifies the range in Gs (gravitational force) and can be set to 2G, 4G, 8G, or 16G depending on the application requirements.

23.2.5. **IMU.setGyroscopeRange()**

The IMU.setGyroscopeRange() function is used to set the range of the gyroscope on the LSM6DSOX sensor. The gyroscope range determines the maximum angular velocity that can be measured by the sensor. This function takes an argument that specifies the range in degrees per second (dps) and can be set to 125dps, 250dps, 500dps, 1000dps, or 2000dps depending on the application requirements.

23.2.6. IMU.setAccelerometerDataRate()

The IMU.setAccelerometerDataRate() function is used to set the data rate of the accelerometer on the LSM6DSOX sensor. The data rate determines how often the sensor samples the acceleration data and can be set to 12.5Hz, 26Hz, 52Hz, 104Hz, 208Hz, 416Hz, 833Hz, or 1660Hz depending on the application requirements.

23.2.7. IMU.setGyroscopeDataRate()

The IMU.setGyroscopeDataRate() function is used to set the data rate of the gyroscope on the LSM6DSOX sensor. The data rate determines how often the sensor samples the angular velocity data and can be set to 12.5Hz, 26Hz, 52Hz, 104Hz, 208Hz, 416Hz, 833Hz, or 1660Hz depending on the application requirements.

23.2.8. IMU.accelerationSampleRate()

The IMU.accelerationSampleRate() function is used to read the current sample rate of the accelerometer on the LSM6DSOX sensor. This function returns the current data rate value in Hz.

23.2.9. IMU.gyroscopeSampleRate()

The IMU.gyroscopeSampleRate() function is used to read the current sample rate of the gyroscope on the LSM6DSOX sensor. This function returns the current data rate value in Hz.

23.2.10. IMU.temperatureAvailable()

The IMU.temperatureAvailable() function is provided by the Arduino_LSM6DSOX.h library for the LSM6DSOX accelerometer and gyroscope sensor on the Arduino Mbed OS Nicla Board. This function is used to check if the temperature data is available to be read from the sensor. It returns a boolean value of true if the temperature data is available, and false if it is not.

23.2.11. IMU.readTemperature()

The IMU.readTemperature() function is provided by the Arduino_LSM6DSOX.h library and is used to read the temperature data from the LSM6DSOX sensor on the Arduino Mbed OS Nicla Board. This function returns the temperature in degrees Celsius as a floating-point value. Before calling this function, you should use the temperatureAvailable() function to check if the temperature data is available.

23.2.12. IMU.accelerationAvailable()

The IMU.accelerationAvailable() function is provided by the Arduino_LSM6DSOX.h library and is used to check if the acceleration data is available to be read from the LSM6DSOX sensor on the Arduino Mbed OS Nicla Board. It returns a boolean value of true if the acceleration data is available, and false if it is not.

23.2.13. IMU.readAcceleration()

The IMU.readAcceleration() function is provided by the Arduino_LSM6DSOX.h library and is used to read the acceleration data from the LSM6DSOX sensor on the Arduino Mbed OS Nicla Board. This function returns a 3D vector that represents the acceleration in units of Gs (gravitational force) along the x, y, and z axes. Before

calling this function, you should use the accelerationAvailable() function to check if the acceleration data is available.

23.2.14. IMU.gyroscopeAvailable()

The IMU.gyroscopeAvailable() function is provided by the Arduino_LSM6DSOX.h library and is used to check if the gyroscope data is available to be read from the LSM6DSOX sensor on the Arduino Mbed OS Nicla Board. It returns a boolean value of true if the gyroscope data is available, and false if it is not.

23.2.15. IMU.readGyroscope()

The IMU.readGyroscope() function is provided by the Arduino_LSM6DSOX.h library and is used to read the gyroscope data from the LSM6DSOX sensor on the Arduino Mbed OS Nicla Board. This function returns a 3D vector that represents the angular velocity in units of degrees per second (dps) along the x, y, and z axes. Before calling this function, you should use the gyroscopeAvailable() function to check if the gyroscope data is available.

23.2.16. randomGaussian()

The randomGaussian() function is a built-in function in the Arduino programming language. Gaussian distribution is also known as normal distribution, and it is a probability distribution that describes how values are distributed around the mean. This function takes two arguments: the mean and the standard deviation of the distribution. It returns a random floating-point number with a mean value equal to the first argument and a standard deviation equal to the second argument. This function is commonly used in statistical simulations and signal-processing application.

23.2.17. kalmanX.setQ(), kalmanY.setQ(), kalmanZ.setQ()

These functions set the process noise covariance for the Kalman filter in the X, Y, and Z axes respectively. The process noise covariance controls how much we trust our prediction of the state of the system at the next time step. A higher value of Q indicates that the system is more likely to deviate from the predicted state and a lower value indicates that the system is more likely to follow the predicted state.

23.2.18. kalmanX.setR(), kalmanY.setR(), kalmanZ.setR()

These functions set the measurement noise covariance for the Kalman filter in the X, Y, and Z axes respectively. The measurement noise covariance controls how much we trust the sensor measurement of the system. A higher value of R indicates that the sensor measurement is less reliable and a lower value indicates that the sensor measurement is more reliable.

23.2.19. kalmanX.update(), kalmanY.update(), kalmanZ.update()

These functions update the Kalman filter in the X, Y, and Z axes respectively. They take a measurement of the system and use it to update the state estimate of the system. The function returns the estimated bias of the measurement. The estimated bias is subtracted from the measurement to get a corrected measurement, which is used to update the state estimate.

23.2.20. `kalmanX.getSigma()`, `kalmanY.getSigma()`, `kalmanZ.getSigma()`

These functions return the standard deviation of the Kalman filter output in the X, Y, and Z axes respectively. The standard deviation is a measure of the noise in the output of the Kalman filter.

23.2.21. `delay()`

This function causes the program to pause execution for a specified number of milliseconds. In this code, it is used to wait for a short time before reading from the IMU again. This allows time for the IMU to take a new measurement and for the Kalman filter to update its estimates.

23.2.22. `lsm6dsoxSensor.Set_X_FS()`

`lsm6dsoxSensor.Set_X_FS()` is a function provided by the `LSM6DSOXSensor.h` library that is used to set the full-scale range of the accelerometer in the LSM6DSOX sensor. The accelerometer measures acceleration along three axes (X, Y, Z) and the full-scale range determines the maximum range of acceleration that the sensor can measure without saturation.

The function takes an argument that specifies the desired full-scale range for the accelerometer. The available options for the resolution range may vary depending on the specific sensor model, but commonly include $\pm 2g$, $\pm 4g$, $\pm 8g$, or $\pm 16g$. The full-scale range is typically expressed in units of acceleration (e.g., g) and represents the maximum acceleration that the sensor can accurately measure along each axis.

When `lsm6dsoxSensor.Set_X_FS()` is called with the desired resolution range as an argument, the function sends the appropriate configuration commands to the LSM6DSOX sensor to set the accelerometer to the specified full-scale range. This ensures that the sensor is configured to accurately measure accelerations within the desired range and prevents saturation of the sensor's output, which can result in inaccurate readings.

23.2.23. `lsm6dsoxSensor.Set_G_FS()`

`lsm6dsoxSensor.Set_G_FS()` is a method or function call that likely belongs to a software library or codebase that interfaces with an LSM6DSOX sensor. The LSM6DSOX is a type of inertial measurement unit (IMU) sensor that combines a 3-axis accelerometer and a 3-axis gyroscope in a single chip.

The `Set_G_FS()` function is likely used to configure the full-scale (FS) range or sensitivity of the gyroscope component of the LSM6DSOX sensor. Gyroscopes measure angular velocity or rotational motion, and the full-scale range determines the maximum angular velocity that the gyroscope can accurately measure.

The full-scale range is typically specified in units of degrees per second (dps) or radians per second (rad/s) and represents the maximum rate of rotation that the gyroscope can detect without saturating its output. A higher full-scale range allows the gyroscope to measure faster rotations, but it may result in reduced resolution for slower rotations. The `Set_G_FS()` function likely takes an argument or parameter that specifies the desired full-scale range for the gyroscope, such as $+\/- 125$ dps, $+\/- 250$ dps, $+\/- 500$ dps, $+\/- 1000$ dps, or $+\/- 2000$ dps, depending on the capabilities of the LSM6DSOX sensor. The function would then configure the sensor to operate with the specified full-scale range for the gyroscope accordingly.

23.2.24. Initialize_IMU()

The Initialize_IMU() function is responsible for initializing and configuring the LSM6DSOX sensor for operation. It begins by establishing communication with the sensor using the Wire.begin() function, which is commonly used for I2C communication in Arduino-based projects. Next, it initializes the LSM6DSOX sensor using the lsm6dsoxSensor.begin() function, which sets up the sensor for operation.

The function then proceeds to set the full-scale range for the accelerometer and gyroscope using the lsm6dsoxSensor.Set_X_FS() and lsm6dsoxSensor.Set_G_FS() functions, respectively. In this case, the full-scale range is set to $\pm 4\text{g}$ for the accelerometer and $\pm 2000 \text{ deg/s}$ for the gyroscope, which determines the maximum range of motion that the sensor can measure.

The sample rates for both the accelerometer and gyroscope are then set to 104 Hz using the lsm6dsoxSensor.Set_X_ODR() and lsm6dsoxSensor.Se_G_ODR() functions, respectively. The sample rate determines how frequently the sensor measures and reports data, with a higher sample rate resulting in more frequent updates but potentially higher power consumption.

23.2.25. Factors_Calculation()

The Factors_Calculation() function performs several steps. First, it checks if temperature data is available from the IMU sensor using the IMU.temperatureAvailable() function. If temperature data is available, it reads the temperature readings from the IMU using the IMU.readTemperature() function and stores the value in the temperature variable. Then, it calculates the temperature factor by multiplying the temperature sensitivity, temperature, and temperature tolerance, and dividing the result by 100.0 to convert the tolerance from percentage to decimal. Next, it calculates the linear acceleration factor by multiplying the linear acceleration sensitivity and linear acceleration tolerance. Finally, it calculates the angular rate factor by multiplying the angular rate sensitivity and angular rate tolerance.

23.2.26. Factors_Calculation()

The Accelerometer_Gyroscope_Characteristics() function performs several steps. First, it checks if accelerometer data is available from the IMU sensor using the IMU.accelerationAvailable() function. If accelerometer data is available, it reads the accelerometer readings for the X, Y, and Z axes from the IMU using the IMU.readAcceleration() function and stores the values in the variables Ax, Ay, and Az. Then, it calculates the acceleration in m/s^2 for each axis by multiplying the raw accelerometer readings with the linear acceleration factor, temperature factor, and gravity. The calculated values are stored in the variables Ax_m_sec2, Ay_m_sec2, and Az_m_sec2. Next, it adds noise characteristics to the accelerometer readings by calculating the noise values for each axis based on the accelerometer noise density and sample rate, and then adding them to the corresponding calculated acceleration values. It also checks if gyroscope data is available from the IMU sensor using the IMU.gyroscopeAvailable() function. If gyroscope data is available, it reads the gyroscope readings for the X, Y, and Z axes from the IMU using the IMU.readGyroscope() function and stores the values in the variables Gx, Gy, and Gz. Finally, it converts the gyroscope readings from LSB to deg/s by multiplying with the angular rate factor and temperature factor, and stores the calculated values in the variables Gx_deg_sec, Gy_deg_sec, and Gz_deg_sec.

24. Sensor Inertial Mesure Unit

Inertial Measurement Units (IMU) are an component of many navigation and control systems. These devices integrate several sensors, including accelerometers and gyroscopes, to measure and follow an object's, linear and rotationnal, movements in three-dimensional space. IMUs are widely used in a variety of applications, such as autonomous vehicle navigation, virtual reality, image stabilization, and even in wearable devices for tracking physical activity. Their ability to provide precise data on linear and angular acceleration makes them indispensable tools for systems requiring precise control and orientation, and ongoing technological advances have reduced their size, power consumption and cost, widening their scope of application in many fields. [Stmb]

24.1. General Description

The Inertial Mesure Unit is used to mesured acceleration or rotation motion, this is an electronic devise who used 3 types of sensors : magnetometer, accelerometer and gyroscope. This IMU used sensor LSM9DS1 and they give velocity, orientation and gravitational force.

The magnetometer can mesured the strenght of magnetic fields, this mesure can give different information, if you pass a current through an electromagnet or in any other way.

24.1.1. Always On Mode

This feature implies that the LSM9DS1 can remain operational continuously, even when a device is in sleep mode, without excessively draining power. Such functionality proves invaluable for applications demanding uninterrupted motion tracking, like fitness monitoring or navigation systems.

The power consumption is an important factor when we have battery-powered device like smartphone or other portable application. This device is portable, so they need less battery than possible to do less weight than possible.

The LSM9DS1 boasts a power consumption of 0.55 mA when operating in combo high-performance mode. This signifies its ability to maintain a low power consumption level while delivering exceptional data quality. In this mode, the sensor excels in concurrently measuring acceleration and angular rate, ensuring precise and reliable data output even at a high sampling rate.

24.1.2. Tilt detection

The Tilt detection detect movement with accelerometer, if movement is doing so Tilt detect it. Tilt function with Ultra Low Power consumption.

Per example, if you have your smartphone in your front pant poket. Tilt detection can sayed if you standing to sitting or sitting to standing.

24.1.3. Significant Motion Detection

The Significant Motion Detection used in LSM9DS1 only accelerometer to detect big movement. It can be used for different usage :

- Waking up a device from sleep mode when the user starts moving.
- Tracking steps or activity changes in fitness applications.
- Triggering alarms or notifications when a significant movement is detected.

24.2. Specific Sensor

24.2.1. LSM9DS1

The sensor LSM9DS1, engineered by STMicroelectronics, is composed of a 3-axis Inertial Measurement Unit (IMU), which have a high-performance 3-axis digital accelerometer and 3-axis digital gyroscope. This device is meticulously crafted for precise motion follow up, finding widespread application in wearable devices and smartphones. Capable of concurrent measurement of both linear and rotational motion, it stands as a hallmark in motion-sensing technology.

The sensor LSM9DS1 was developed to detect movement, stationary/motion detection, tilt, pedometer functions, timestamping and to support the data and acquisition of an external magnetometer. This sensor gives hardware flexibility to connect the pins with different mode connections to external sensors to expand functionalities such as adding a sensor hub, auxiliary SPI.

24.2.2. Accelerometer

The STMicroelectronics LSM9DS1 accelerometer is a sensor designed to measure linear acceleration along the x, y, and z-axes. With a typical measurement range of $\pm 2g$, $\pm 4g$, $\pm 8g$, $\pm 16g$. It's give precision and reliability for a variety of applications. The accelerometer for LSM9DS1 has an acceleration sampling frequency that can be changed to the following values: 10 Hz, 50 Hz, 119 Hz, 238 Hz, 476 Hz, 952 Hz.

It's low power consumption makes it an great choice for battery-powered devices, with a typical operating current ranging from 0.55 mA to 1.25 mA. Additionally, it's wide operating voltage range, from 1.71 V to 3.6 V, allows it to easily adapt to different system configurations.

Whether it's tracking movements in wearable devices, stabilizing drones, or measuring vibrations in structures, the accelerometer LSM9DS1 delivers outstanding performance in a compact and robust package.

The STMicroelectronics accelerometer LSM9DS1 is a preferred choice for engineers and designers seeking a precise and reliable solution for their projects.

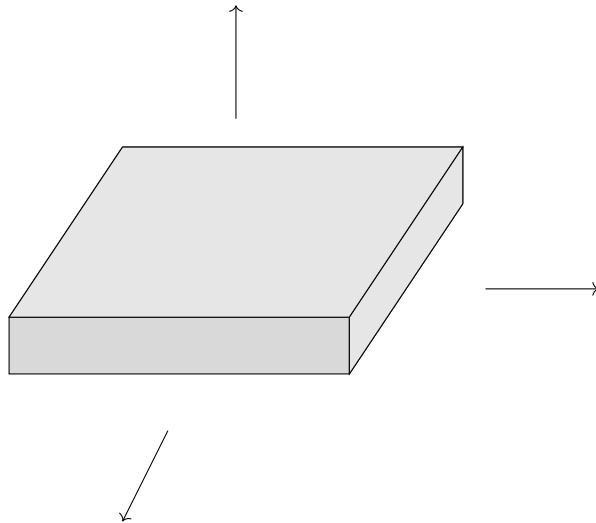


Figure 24.1.: LSM9DS1 axis on this card

24.2.3. Gyroscope

The STMicroelectronics LSM9DS1 gyroscope is a measuring instrument designed to assess rotation rates around the x, y, and z-axes. With a typical measurement range of ± 245 , ± 500 , ± 2000 dps, it offers adaptability to the specific needs of the application. The gyroscope of LSM9DS1 has an acceleration sampling rate that can be modified to take the following values: 14.9 Hz, 59.5 Hz, 119 Hz, 238 Hz, 476 Hz, 952 Hz.

This gyroscope operates with a voltage range of 1.71 V to 3.6 V, allowing it to easily integrate into different system configurations. Furthermore, its typical operating current ranges from 1 mA to 2 mA, ensuring optimal energy efficiency for battery-powered devices.

Whether it's for inertial navigation, drone stabilization, or other applications requiring precise motion measurement, the gyroscope of LSM9DS1 delivers reliable and accurate performance in a compact and robust form factor, meeting the highest requirements of engineers and designers.

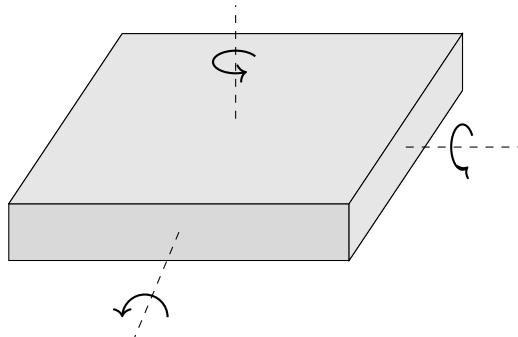


Figure 24.2.: LSM9DS1 axis on this card

24.3. Specification

24.3.1. Accelerometer Sensor

The accelerometer is composed by sensor type MEMS (Micro-Electro-Mechanical Systems) and they used microscopic structures like suspended beams or springs, which deform in response to acceleration. The deformation of these structures is measured using displacement sensors or changes in electrical resistance. MEMS accelerometers are commonly used due to their small size, low power consumption, and comparatively lower cost compared to other types of accelerometers.

The Accelerometer sensor function by :

- Acceleration Integration for Velocity Estimation: To estimate velocity, the acceleration measured by the accelerometer is integrated over time. Integrating acceleration over time yields velocity.
- Error Compensation: However, it's crucial to note that acceleration integration is susceptible to cumulative errors. Errors stemming from sensor noise, drift, and inaccuracies can accumulate over time, leading to inaccurate or biased velocity estimates.
- Utilization of Filters and Sensor Fusion Techniques: To compensate for these errors, advanced techniques such as Kalman filters, Complementary filters, or sensor fusion techniques can be employed. These techniques combine accelerometer data with other sensors such as gyroscopes to enhance velocity estimation accuracy.
- Calibration and Adjustment: It's also vital to calibrate and adjust the IMU to correct systematic errors and ensure precise measurements. This may involve steps such as compensating for Earth's gravity, correcting gyroscope drift, and reducing sensor noise.

24.3.2. PIN Connection

They are 4 ways to connect pin, ways depend of what they have after.

- Mode 1: You can utilize either the I²C / MIPI I3CSM slave interface or the SPI (3- and 4-wire) serial interface.
- Mode 2: This mode supports both the I²C / MIPI I3CSM slave interface or the SPI (3- and 4-wire) serial interface, along with an I²C interface master for external sensor connections.
- Mode 3: Here, you have the choice of employing either the I²C / MIPI I3CSM slave interface or the SPI (3- and 4-wire) serial interface for the application processor interface. Additionally, an auxiliary SPI (3- and 4-wire) serial interface is designated for external sensor connections, exclusively for the gyroscope.
- Mode 4: Similar to Mode 3, this mode offers the I²C / MIPI I3CSM slave interface or the SPI (3- and 4-wire) serial interface for the application processor interface. Additionally, it provides an auxiliary SPI (3- and 4-wire) serial interface for external sensor connections, catering to both the accelerometer and gyroscope.

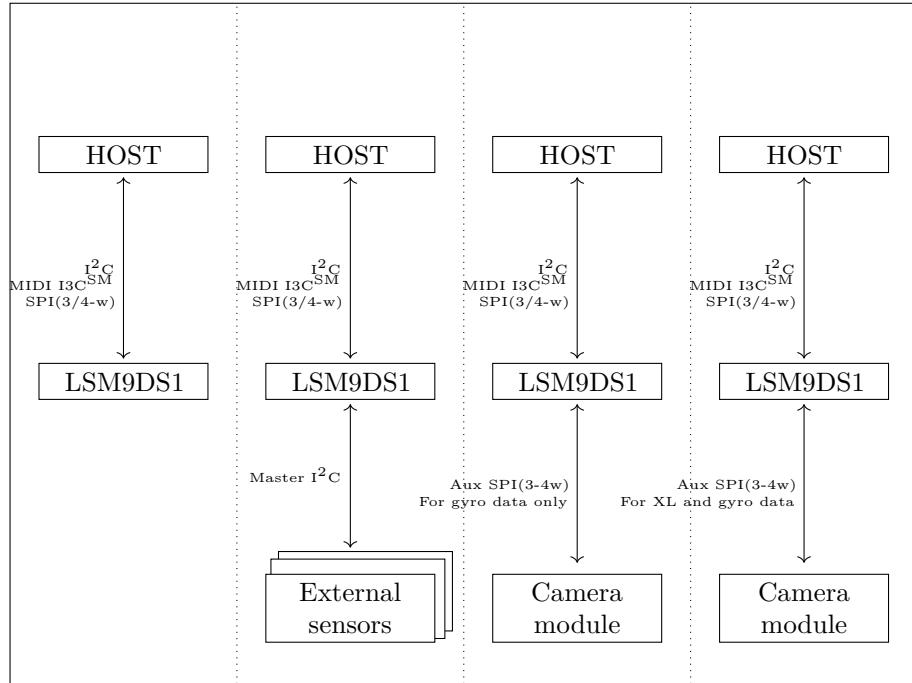


Figure 24.3.: Explication of pin mode

24.4. Library

24.4.1. Library Description

To support the project's requirements, we need 3 main libraries, each serving a specific need :

1. [Wire.h](#): This library is a standard Arduino library that facilitates communication between I²C (Inter-Integrated Circuit) devices. I²C is a synchronous serial communication protocol used to connect microcontrollers to external devices. [Wire.h](#) provides functions for initializing the I²C bus, sending and receiving data on the bus, and managing multiple devices on the same bus. With this library, developers can easily connect multiple I²C devices, such as sensors or displays, to an Arduino board.
2. [Kalman.h](#): This library implements the Kalman filter algorithm. The Kalman filter is a mathematical technique used to estimate the state of a system from incomplete measurements. It is commonly used in control systems, robotics and navigation applications to improve the accuracy of sensor measurements and reduce errors. [Kalman.h](#) provides a simple interface for developers to implement the Kalman filter in their Arduino projects.
3. [Arduino_LSM9DS1.h](#): The library [Arduino_LSM9DS1.h](#) is a software library designed to facilitate interaction with the sensor accelerometer LSM9DS1, developed by STMicroelectronics. This sensor combines an accelerometer, a gyroscope and a magnetometer in a single package. The library LSM9DS1 provides an interface for accessing the sensor's functionalities, such as reading acceleration, angular velocity and magnetic field data. It can be used to configure various sensor parameters, such as measurement scales, sampling rates and operating modes.

24.4.2. Installation

The library give some example of sketch, to use this sketch we need to download the library on ArduinoIDE. The library `Arduino_LSM9DS1.h` and `Kalman.h` can be download directly on Arduino. The library `wire.h` is used if you input : `#include <wire.h>`

```
1 #include <Wire.h>
2
```

Figure 24.4.: Wire include

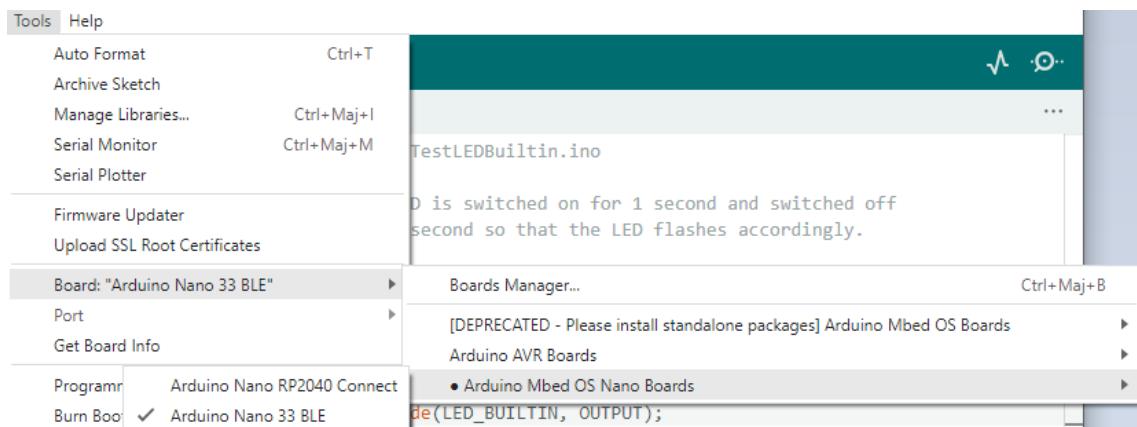


Figure 24.5.: Board card installation

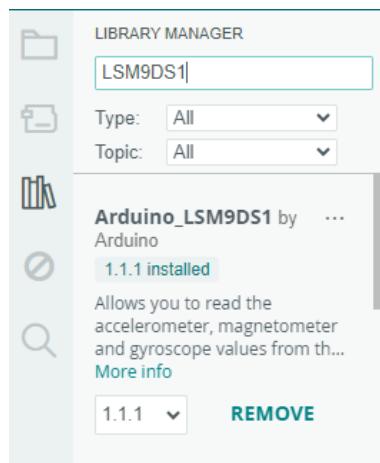


Figure 24.6.: Library choice

24.5. Function

The library LSM9DS1 on Arduino Nano 33 BLE Sense is composed of different function. The function is different if you was on I²C protocol or SPI protocol.

24.5.1. Library `Wire.h`

This library is used to develop communication between different components of Arduino board, it's used for I²C. The library `Wire.h` is composed by :

- `Wire.begin()`: Initializes the library `Wire.h` and prepares the Arduino to act as master or slave on the bus I²C.
- `Wire.beginTransmission(address)`: Starts a transmission to a slave device with the specified address.
- `Wire.write(data)` : Sends data on the I²C bus during transmission.
- `Wire.endTransmission()` : Ends the current transmission and releases the I²C bus.
- `Wire.requestFrom(address, quantity)` : Requests data from a slave device with the specified address.
- `Wire.available()`: Checks whether data is available to be read after a request.
- `Wire.read()`: Reads data received from a slave device.

We can test the function with an example of the library `Wire.h`. This is an example to test the I²C bus on the board.

Listing 24.1.: Simple sketch using the sensor LSM9DS1 detection

```

1000 #include <Wire.h>
1002
1003 // I2C Address of the LSM9DS1 magnetometer
1004 // module
1005 #define LSM9DS1_M_ADDR 0x1E
1006
1007 // I2C Address of the LSM9DS1 accelerometer
1008 // -gyroscope module
1009 #define LSM9DS1_AG_ADDR 0x6B
1010
1011 void setup() {
1012     Serial.begin(9600); // Initialize serial communication
1013     Wire.begin(); // Initialize Wire library
1014     // Initialize LSM9DS1
1015     initLSM9DS1();
1016 }
1017
1018 void loop() {
1019     // Read LSM9DS1 data
1020     readLSM9DS1();
1021     delay(1000); // Pause for one second between readings
1022 }
1023
1024 void initLSM9DS1() {
1025     // Initialize magnetometer module
1026     Wire.beginTransmission(LSM9DS1_M_ADDR);
1027     Wire.write(0x20); // Control register address for mode
1028     Wire.write(0x1C); // Activate continuous mode at 10 Hz
1029     Wire.endTransmission();
1030
1031     // Initialize accelerometer-gyroscope module
1032     Wire.beginTransmission(LSM9DS1_AG_ADDR);
1033     Wire.write(0x10); // Control register address for gyro mode
1034     Wire.write(0x38); // Activate continuous m
1035 }
```

.../Code/Nano33BLESense/IMU/Test/WireEx.ino

24.5.2. Function `IMU.begin()`

The function to initialized the IMU is `IMU.begin()`, they start the IMU initialization process. This step is important to configures the IMU's basic parameters, set the communications with other devices, and prepares the unit for proper operation. Accurate initialization is essential to ensure the accuracy and reliability of the data provided by the IMU.

An error during the initialization can lead to inaccurate measurements or unpredictable behavior. For example, incorrectly configured measurement parameters or sampling rates can compromise data quality, affecting overall system performance.

```
1000 if (!IMU.begin()) {
1001   Serial.println("Failed to initialize IMU!");
1002 }
```

24.5.3. Function `IMU.end()`

This function is composed without parameter. After do this function they return value of 1 if all is good and they return value of 0 if we had some problem.

This function is used to stop or deactivate the IMU once it is no longer required. It frees up the resources used by the IMU and ends its operation cleanly. When called, it ensures that all tasks associated with the IMU are properly finalized, avoiding memory leaks or other problems associated with incomplete de-initialization.

This step helps to ensure the stability and reliability of the system as a whole, by avoiding problems associated with incorrect IMU de-initialization.

The data was return at the end.

```
1000 if (!IMU.begin()) {
1001   Serial.println("Failed to initialize IMU!");
1002 }
```

24.5.4. Function `IMU.readAcceleration(x,y,z)`

The `IMU.readAcceleration()` is used to read the IMU accelerometer and obtain the data, this is expressed in gravity (g). This function provides information on the movements detected by the IMU's accelerometer. The resulting acceleration data can be used for a variety of applications, such as shock or motion detection, or inertial navigation. By regularly interrogating the accelerometer and analyzing variations in acceleration, it is possible to understand and react to changes in motion with precision, which is essential in many modern embedded systems and electronic devices.

The parameters x, y and z are floats giving information on the x, y or z-axis

```
1000 float x;
1001 float y;
1002 float z;
1004
1005 if (IMU.accelerationAvailable()) {
1006   IMU.readAcceleration(x, y, z);
1008
1009   Serial.print(x);
1010   Serial.print('\t');
1011   Serial.print(y);
1012   Serial.print('\t');
1013   Serial.println(z); }
```

24.5.5. Function `IMU.readGyroscope()`

Retrieves data from the IMU's gyroscope and returns angular velocity in degrees per second (dps). This function provides information on the rotational movement detected by the IMU's gyroscope. The angular velocity data retrieved can be used in various applications such as robotics, motion tracking or navigation systems. By regularly interrogating the gyroscope and analyzing angular velocity variations, it becomes possible to understand and respond precisely to rotational movements, which is crucial in applications requiring precise orientation control or stabilization.

The parameters x, y and z are floats giving information on the x, y or z axis

```

1000 float x, y, z;

1002 if (IMU.gyroscopeAvailable()) {
    IMU.readGyroscope(x, y, z);

1004 Serial.print(x);
1006 Serial.print('\t');
    Serial.print(y);
    Serial.print('\t');
    Serial.println(z); }
```

24.5.6. Function `IMU.accelerationAvailable()`

This function allows you to check the status of IMU acceleration data, indicating whether or not new data is ready to be retrieved.

In scenarios such as motion tracking, robotics or navigation systems, where the IMU continuously captures acceleration data, the availability of new data determines the system answers. Test the IMU at regular intervals is used for application and they can check for acceleration currently or non, ensuring that the system remains synchronized with object movements in real time.

When the function returns a value of 1, this means that new acceleration data is available, enabling the system to extract the data and process it further. A value of 0 indicates that there have been no recent acceleration updates.

```

1000 float x, y, z;

1002 if (IMU.accelerationAvailable()) {
    IMU.readAcceleration(x, y, z);

1004 Serial.print(x);
1006 Serial.print('\t');
    Serial.print(y);
    Serial.print('\t');
    Serial.println(z); }
```

24.5.7. Function `IMU.gyroscopeAvailable()`

This function ask if new gyro data are available for the IMU and they returns a value 0 if no new gyro data is available, and 1 if new gyro data is ready to be retrieved.

In applications requiring real-time monitoring of rotational motion, such as drone stabilization, virtual reality systems or inertial navigation, the system can determine whether to wait for updated gyroscope readings or proceed to process available data. A value of 1 indicates that new gyro data is available, enabling the system to retrieve and use this information for adapted the orientation tracking or motion control. Conversely, a feedback value of 0 indicates that there have been no recent updates to

the gyroscope measurements, prompting the system to wait for new data to become available before proceeding.

```

1000 float x, y, z;
1002 if (IMU.gyroscopeAvailable()) {
    IMU.readGyroscope(x, y, z);
1004
    Serial.print(x);
1006    Serial.print('t');
    Serial.print(y);
1008    Serial.print('t');
    Serial.println(z); }
```

24.5.8. Function `IMU.accelerationSampleRate()`

This function is designed to provide information on the sampling rate of the IMU's accelerometer. The function `IMU.accelerationSampleRate()` returns the accelerometer sampling rate, expressed in Hertz (Hz). This value represents the frequency at which the accelerometer takes measurements and provides acceleration data. It indicates how many times per second the accelerometer registers changes in acceleration along its axes.

In activities such as motion tracking, gesture recognition or vibration analysis, knowledge of the accelerometer's sampling rate ensures that the system can accurately capture and respond to changes in motion dynamics.

```

1000 Serial.print("Accelerometer sample rate = ");
1001 Serial.print(IMU.accelerationSampleRate());
1002 Serial.println(" Hz");
1003 Serial.println();
1004 Serial.println("Acceleration in g's");
1005 Serial.println("X \t Y \t Z");
```

24.5.9. Function `IMU.gyroscopeSampleRate()`

To recover and communicate the specific rate at which the gyroscope, integrated into the inertial measurement unit (IMU), collects data samples. This frequency is usually expressed in hertz (Hz), corresponding to the number of samples acquired per second. This is the frequency at which the gyroscope takes measurements, giving an idea of its operational efficiency and performance.

```

1000 Serial.print("Gyroscope sample rate = ");
1001 Serial.print(IMU.gyroscopeSampleRate());
1002 Serial.println(" Hz");
1003 Serial.println();
1004 Serial.println("Angular speed in degrees/second");
1005 Serial.println("X tY tZ");
```

25. Distance, Speed and Acceleration Detection Algorithms

25.1. Review of Distance Measurement Methods

Since Mitsubishi released the first cruise control with distance control in 1995, the vast majority of ACC functions have been based on Laser Radar or millimeter-wave radar (MWR).¹ But a few have opted to use a binocular camera as the basis for the technology, such as Subaru's EyeSight technology.²

Each of these different sets of technical solutions has its own advantages and disadvantages:

Technology	Advantages	Disadvantages
Laser Radar (LiDAR)	- High accuracy and resolution - Capable of creating detailed 3D maps - Effective for object detection and classification	- Expensive - Affected by adverse weather conditions - High power consumption
Millimeter-Wave Radar	- Less affected by weather conditions - Long-range detection capabilities - Lower cost compared to LiDAR	- Lower resolution than LiDAR - Limited in detecting small or non-metallic objects - Can be affected by interference
Binocular Camera Systems	- Lower cost compared to LiDAR and radar - High resolution for nearby objects - Uses passive sensing (no emissions)	- Computationally intensive - Accuracy decreases with distance - Affected by lighting conditions

Table 25.1.: Comparison of distance measurement technologies: LiDAR, Millimeter-Wave Radar, and Binocular Cameras

For cars using Laser Radar, LiDAR operates by emitting laser pulses towards objects in front of the vehicle. When these pulses hit an object, they are reflected back to the sensor, which measures the time it takes for the pulses to return. This time-of-flight measurement allows the system to calculate the precise distance to the object, as well as its relative speed and position. For cars using millimeter-wave radar, the functional implementation is similar.

For systems that use binocular cameras, this process can be relatively more complex; essentially distance recognition for binocular camera-based systems utilizes bionics: Binocular disparity. This method involves using a pair of cameras positioned at a certain distance apart to capture two images with different viewing angles. By comparing the disparity between the two images (i.e., the difference in pixel positions).³

$$\text{depth} = \frac{f \times \text{baseline}}{\text{disparity}} \quad (25.1)$$

where depth represents the distance of an object from the camera, f denotes the focal length of the camera, baseline is the distance between the two cameras, and disparity is the difference in image location of the object in the two camera views.

¹MI-PILOT, Mitsubishi Motors, <https://www.mitsubishi-motors.com/en/brand/technology/mipilot2/index.html>

²EyeSight technology, Subaru, <https://www.subaru.com/eyesight.html>

³Mansour, M., Davidson, P., Stepanov, O. and Piché, R., 2019. Relative Importance of Binocular Disparity and Motion Parallax for Depth Estimation: A Computer Vision Approach. *Remote Sensing*, 11(17), p.1990. <https://doi.org/10.3390/rs11171990>

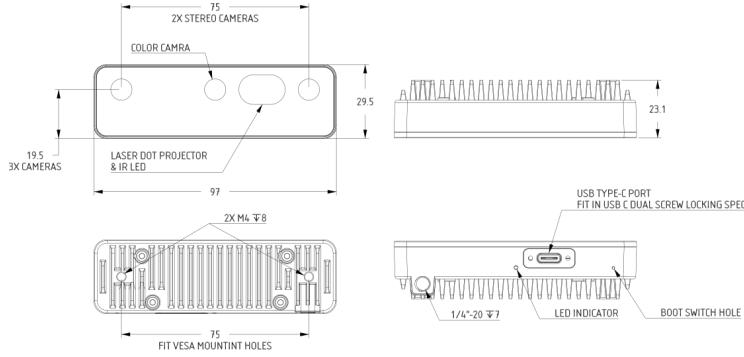


Figure 25.1.: Design of OAK-D Pro camera

For the OAK-D Pro camera, the baseline distance, which is the distance between the two monochrome cameras, is 7.5 cm.⁴

According to OAK's official documentation⁵, the OAK-D Pro can reach a theoretical maximum of 35m, but at this distance there is a very significant margin of error (about 33% of the theoretical error⁶) where the distance fluctuates very significantly. However, there are ways to improve the accuracy of distance detection, such as half-pixel mode to improve the accuracy of distance detection, and averaging distances over a period of time to calculate relative distances to improve the accuracy of relative speed calculations.

25.2. Review of Speed and Acceleration Detection Algorithms

For the measurement of relative velocity, the three schemes mentioned above will actually have two different principles and calculations.

Millimeter-wave radar primarily calculates the relative speed of objects using the Doppler effect. When the radar signal hits a moving object, the frequency of the reflected wave changes depending on the relative speed between the radar and the object. This frequency shift (Doppler shift) is directly proportional to the relative speed of the object.⁷

For LiDAR and Binocular Camera Systems, the relative speed of objects is calculated based on the change in distance over time. By continuously measuring the distance to an object and comparing it with previous measurements.

Each of these different sets of technical solutions has its own advantages and disadvantages:

⁴OAK-D Pro Camera Documentation, Luxonis, 2021

⁵OAK China official Website, OAK China, 2024

⁶Depth range enhancement, Luxonis Community, 2022

⁷Millimeter Wave Radar Sensors: Fundamentals, Texas Instruments, 2018

Technology	Advantages	Disadvantages
Laser Radar (LiDAR)	<ul style="list-style-type: none"> - High accuracy and resolution - Capable of creating detailed 3D maps - Effective for object detection and classification 	<ul style="list-style-type: none"> - Expensive - Affected by adverse weather conditions - High power consumption
Millimeter-Wave Radar	<ul style="list-style-type: none"> - Less affected by weather conditions - Long-range detection capabilities - Lower cost compared to LiDAR 	<ul style="list-style-type: none"> - Lower resolution than LiDAR - Limited in detecting small or non-metallic objects - Can be affected by interference
Binocular Camera Systems	<ul style="list-style-type: none"> - Lower cost compared to LiDAR and radar - High resolution for nearby objects - Uses passive sensing (no emissions) 	<ul style="list-style-type: none"> - Computationally intensive - Accuracy decreases with distance - Affected by lighting conditions

Table 25.2.: Comparison of distance measurement technologies: LiDAR, Millimeter-Wave Radar, and Binocular Cameras

Part III.

Arduino Nano 33 BLE Sense - External Sensors and Actors

26. Tiny Machine Learning Kit

The Tiny Machine Learning Kit will equip you with all the tools which are needed to start with machine learning. [Ardl]

The kit consists of

- an Arduino Nano 33 BLE Sense Lite,
- a camera module OV7675,
- USB A to USB Micro B cable, and
- a custom Arduino shield to make it easy to attach components.



Figure 26.1.: Das Tiny Machine Learning Kit [Ardl]

26.1. Das Arduino Tiny Machine Learning Shield

Das Tiny Machine Learning Shield wird von Arduino für das Tiny Machine Learning Kit hergestellt, um das Verbinden von Komponenten, wie beispielsweise der Kamera, zu vereinfachen.

Bei dem Tiny Machine Learning Shield handelt es sich um ein Breadboard, was speziell für den Arduino Nano 33 BLE Sense ausgelegt ist. Komponenten können entweder wie die Kamera direkt in passende Slots gesteckt oder mit Grove-Kabeln verbunden werden. Außerdem verfügt das Tiny Machine Learning Shield über eine Anschlussklemme, um externe Spannungsquellen anzuschließen und über einen Taster.

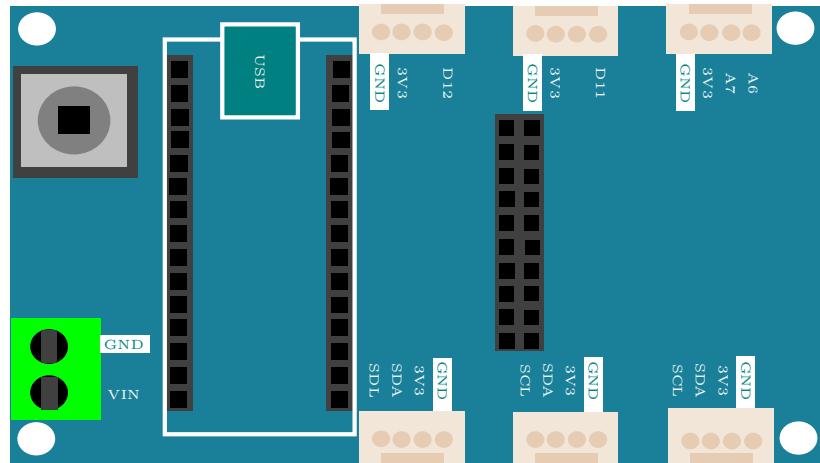


Figure 26.2.: Das Tiny Machine Learning Shield [Gua21]

Die Pinbelegung für die Grove-Schnittstellen sind auf dem Tiny Machine Learning Shield beschriftet. Die Belegung für den 10-poligen Steckplatz für die Kamera ist in der Grafik 26.3 dokumentiert.

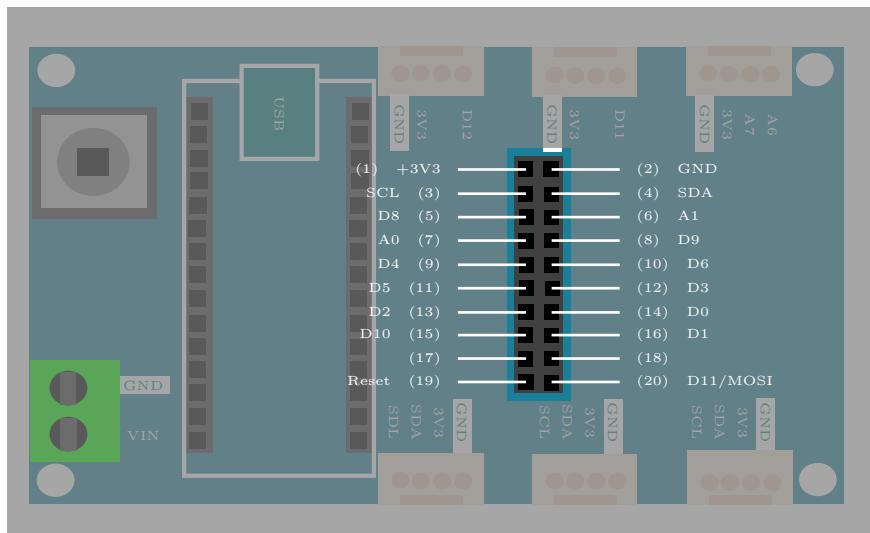


Figure 26.3.: Pin-Belegung des Tiny Machine Learning Shields [Gua21]

26.2. Die OV7675 Kamera

Da die auf dem Arduino fest verbauten Lichtsensoren nur Helligkeit und Farben messen können, ist in dem Kit eine Kamera enthalten. Diese kann verwendet werden, um Objekte zu erkennen oder einen Videostream auszugeben. In unserem Projekt findet die Kamera allerdings keine Verwendung.

Die Kamera kann direkt auf den mittleren Anschluss des Tiny Machine Learning Shields gesteckt werden. Es sind also keine zusätzlichen Kabel zum Verbinden notwendig.

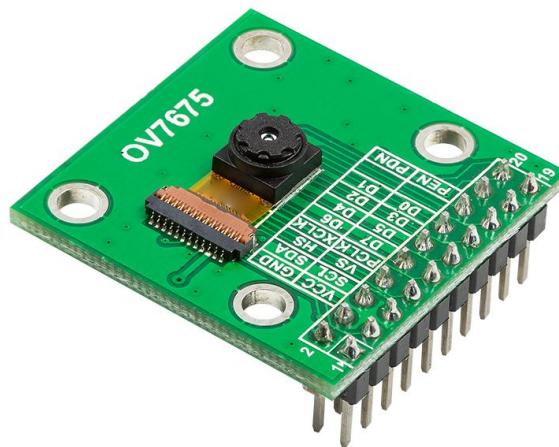


Figure 26.4.: Das OV7675 Kameramodul [Ardl]

26.3. USB-Micro-B- auf USB-A-Kabel

Zur Spannungsversorgung ist in der Box ein USB-Micro-B- auf USB-A-Kabel enthalten. Mit diesem können außerdem die Programme auf den Arduino aufgespielt sowie Daten vom Arduino auf das Programmiergerät übertragen werden.



Figure 26.5.: Ein USB-A auf Micro-USB Kabel

WS:Quelle fehlt

27. Powering the TinyML Shield

Now that you have your Arduino development board up and running, let's talk about how you could deploy it independent of your computer! While some embedded systems call on AC-DC converters (or "wall warts," colloquially) to provide low voltage power to their electronics (the Google home speaker, for example), others are battery powered. Both of these paradigms are applicable to real-world deployment of tinyML and both are achievable using your TinyML kit.

WS:Komplettes Kapitel muss überarbeitet werden
Sortierung, Quellen, Diagramm,...

27.0.1. USB Power Delivery

To this point, we have provided power over USB to our microcontroller via the microB port on the Nano 33 BLE Sense. The 5V that USB carries is then down regulated on the development board to 3.3V, the logical reference for the MCU. While there's nothing wrong with this in development, a prototype of your application ought not depend on drawing power from your computer. Instead, you could call on an AC-DC converter with USB output. This has the added benefit of likely raising the current capacity of the 5V power rail, from at most 500 mA via a computer to whatever the specification happens to be for a given wall-bound converter. This could be meaningful for driving certain power hungry actuators, like speakers.

27.1. Battery

While the above solution removes the need for the computer, you're still tethered to a wall. To go fully mobile you'll want to call on a battery. So what are our options? The idea of using a voltage regulator (specifically, the MPM3610) to cut down an input voltage to a nice, stable 3.3V level applies here as well. If you were to take a closer look at the linked datasheet, you'd find that the MPM3610 accepts input voltages from 4.5V to 21V. The 5V delivered over USB is within this range, and any compatible battery will need to be as well. This unfortunately eliminates the possibility of calling on single cell 3.7V lithium batteries, but makes the selection of a 9V alkaline battery fairly obvious.

You might be wondering how any battery might connect to the boards in front of you, but never fear, we've got you covered. At one corner of the Tiny Machine Learning Shield you'll find a green terminal block with silkscreen labels that read, "VIN" and "GND," where GND is our reference voltage and as such should be connected to the negative terminal of any compatible battery. This green terminal block is where you'd want to screw in wires carrying 4.5V to 21V, and we'll add that 9V clip, like this, that terminates in pre-stripped hook-up wire makes this quite easy!

Assembly Steps

1. Screw down a wire leading from the negative battery terminal (black) to GND (Most < 3mm flat-head screwdrivers will suffice here).
2. Repeat this process for the positive battery terminal (red) to VIN. And that's it you're all set to power your Arduino from a battery!

Important Notes

1. While there is clever circuitry on board to handle such an exception, it is generally good practice to avoid having competing power sources, so we'd recommend that you unplug the Nano 33 BLE Sense from USB power before connecting a battery
2. With about 550 mAh capacity, a 9V battery can source 15 mA for about 37 hours before you will need to get a new one.

/S:tikz-Bild des Shields fehlt.

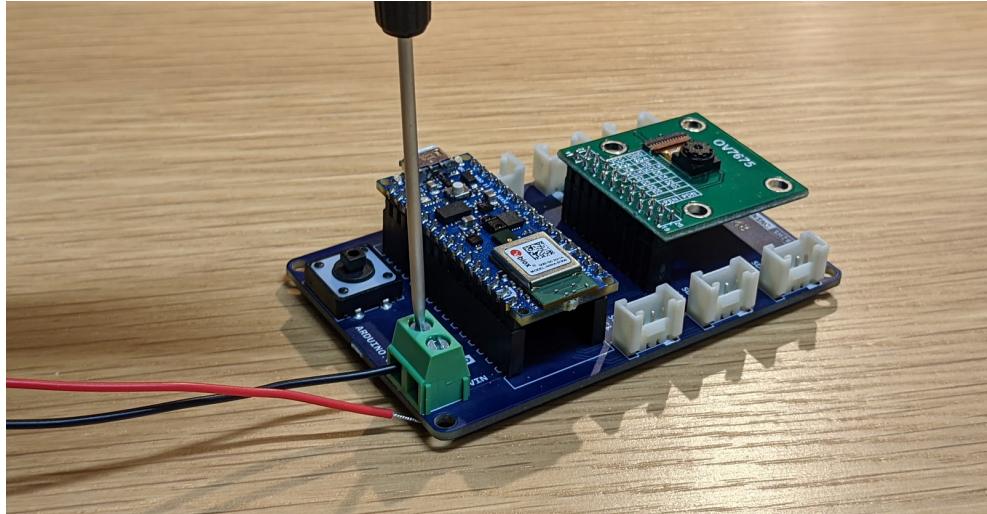


Figure 27.1.: Montage des Clips

An image of screwing down the black negative terminal to attach a wire.

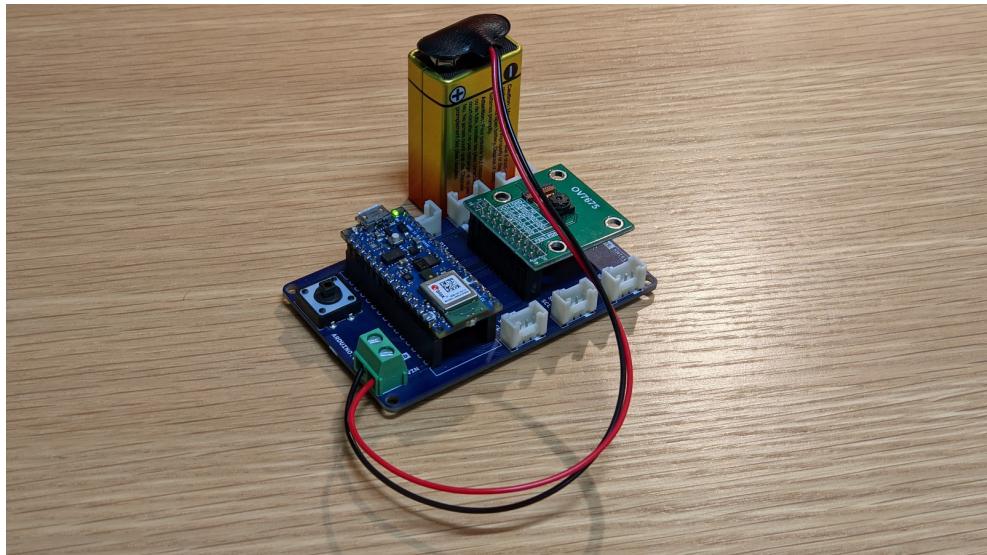


Figure 27.2.: Komplettes System mit Batterie

An image of the attached battery powering the device.

27.2. Checking the Battery Voltage

We use an analog input pin to read the voltage. As we are running from a 3.7V volt battery, we need to adjust the reference voltage used by the pin as otherwise it would

be comparing the voltage to itself. The statement `analogReference (INTERNAL)` sets the pin to compare the input voltage to a regulated 1.1V. We therefore need to reduce the voltage on the input pin to less than 1.1V for this to work. This is done by dividing the voltage using 2 resistors, 1m and 330k ohms. This divides the voltage by approximately 4 so when the battery is fully charged, which is 4.2V, the voltage at the pin input is $4.2/4 = 1.05V$.

```
// Battery Monitor
#define MONITOR_PIN A0           // Pin used to monitor supply voltage
const float voltageDivider = 4.0; // Used to calculate the actual voltage from
                                  // Using 1m and 330k ohm resistors divides the
voltage by approx 4             // You may want to substitute
actual values of resistors in an equation (R1 + R2)/R2
                                  // E.g. (1000 + 330)/330 = 4.03
                                  // Alternatively take the voltage reading across
                                  // the 2 resistors to ground and divide one

// Read the monitor pin and calculate the voltage
float BatteryVoltage()
{
    float reading = analogRead(MONITOR_PIN);
    // Calculate voltage - reference voltage is 1.1v
    return 1.1 * (reading/1023) * voltageDivider;
}
```

The function `BatteryVoltage()`, reads the analog pin, which will range from 0 for 0V to 1,023 for 1.1V and using this reading calculates the actual voltage coming from the battery.

The function `DrawScreenSave()` function calls this then selects the appropriate bitmap to display based on the following:

- If voltage is greater than 3.6V - full
- Voltage between 3.5 and 3.6V - 3/4
- Voltage between 3.4 and 3.5V - half
- Voltage between 3.3 and 3.4V - 1/4
- Voltage < 3.3V - empty

27.3. Batterieclip

Der Batterieclip in Abb. 51.1, der vom Hersteller *reichelt* ist, kann vertikal an einen 9-Volt-Block angeschlossen werden. Die dazugehörigen Anschlussdrähte haben eine Länge von 150 mm. Der Anschlussclip ist in der I-Form ausgeführt, weshalb er sich platzsparend ins Gehäuse einbinden lässt [Rei].

27.4. Spannungssensor

Der Spannungssensor in Abb. ?? von dem Hersteller *Shenzhen Global Technology Co., Ltd* kann bei der Versorgungsspannung von 3,3 V Spannungen in dem Bereich von 0 V bis 16,5 V messen. Dieser wird genutzt, um den Ladestand der Batterie zu überwachen. Die analoge Auflösung des Sensors liegt bei 10 Bit. Damit kann bei dem angegebenen Messbereich die Spannung mit der Auflösung von 0,016 V gemessen werden. Zur



Figure 27.3.: Batterieclip für 9-Volt-Block[Rei24b]

Eingangsschnittstelle gehört der Voltage at the Common Collector (VCC)-Anschluss und der Ground (GND)-Anschluss [She]. Die Bauteilmaße betragen 13 mm x 27 mm.



Figure 27.4.: Voltage Sensor 170640 von dem Hersteller *Shenzhen Global Technology Co., Ltd*[She]

Mit dem Sketch [TestBattery.ino](#), siehe ??TestBattery, soll der Spannungssensor getestet werden.

27.4.1. Durchführung

Für den Test werden die folgenden Hardware-Komponenten benötigt:

- Arduino Nano 33 BLE Sense Lite
- Tiny Machine Learning Shield
- USB-A auf USB-Mikro Verbindungskabel
- Grove Jumper zu Grove 4 Pin Kabel
- Spannungssensor
- Batterie
- Batterieclip

Listing 27.1.: Beispiel-Sketch zum Testen der Batterie

```
1000 /**
1001 * @file TestBattery.ino
1002 * @author Wings
1003 * @date 20.05.2024
1004 * @details The sketch is used to test the voltage sensor. For the test,
1005 *          the voltage is measured on a 9 V block battery and output on the
1006 *          serial monitor.
1007 */
1008 /**
1009 * Pin A7 is referenced as the voltage pin for the voltage sensor. */
1010 #define VoltagePin A7
1011 /**
1012 * The parameter SignalEingang is later assigned the value read in from
1013 *          the VoltagePin pin. */
1014 long SignalEingang;
1015 /**
1016 * The parameter Spannung stands for the voltage that is measured with
1017 *          the voltage sensor and has the unit volt. */
1018 double Voltage;
1019 /**
1020 * @brief Starting serial communication
1021 *
1022 * @details The function is executed when the programme is started. The
1023 *          serial interface to the computer is initialised at 9600 baud
1024 *          There is no return value.
1025 */
1026 void setup() {
1027     Serial.begin(9600);
1028 }
1029 /**
1030 * @brief Display of the measured voltage
1031 *
1032 * @details The function is executed continuously. The signal at the pin
1033 *          VoltagePin is read out and a voltage value is converted.
1034 *          The value read in at the voltage pin is between 0 and 1023. The
1035 *          voltage measurement range is between 0 and 16.5 V. The read-in value
1036 *          can be converted into a voltage using the function map().
1037 *          The voltage is displayed in volts on the serial monitor. There is a 5-
1038 *          second wait after each cycle to avoid flooding the serial monitor.
1039 *          There is no return value.
1040 */
1041 void loop() {
1042     SignalEingang = analogRead(VoltagePin);
1043     Voltage      = map(SignalEingang, 0, 1023, 0, 165); // Converting the
1044     input signal 165 for 16.5 V
1045     Voltage      = Voltage/10; // Converting the input signal 165 for
1046     16.5 V
1047     Serial.print(Voltage);
1048     Serial.println(" V");
1049     delay(5000);
1050 }
```

..../Code/Arduino/Battery/TestBattery.ino

Die Hardware-Komponenten werden zusammengebaut, aber die Batterie wird noch nicht angeschlossen. Dann wird der Arduino Nano 33 BLE Sense Lite mit einem Computer verbunden. Anschließend wird der Sketch `TestBattery.ino` auf den Arduino Nano 33 BLE Sense Lite geladen und der serielle Monitor in der Arduino IDE geöffnet. Die Batterie wird während des Tests an den Batterieclip angeschlossen und der gemessene Wert bei dem seriellen Monitor ausgelesen.

27.4.2. Ergebnisse

Zu Beginn des Tests zeigt der serielle Monitor die Spannung 0 V an. Nach dem Anschließen der Batterie an den Spannungssensor wird die Spannung 9,6 V angezeigt. Abbildung 51.2 zeigt den gemessenen Spannungsverlauf. Die angezeigte Spannung liegt in dem erwarteten Bereich einer 9 V Batterie.



The screenshot shows the Arduino Serial Monitor window. The title bar says "Output Serial Monitor X". Below it is a message input field with placeholder text "Message (Enter to send message to 'Arduino Nano 33 BLE' on 'COM3')". The main area displays a list of messages in a monospaced font:

```
21:21:57.265 -> 0.00 V
21:22:02.248 -> 0.00 V
21:22:07.289 -> 0.00 V
21:22:12.290 -> 9.60 V
21:22:17.247 -> 9.60 V
```

Figure 27.5.: Testoutput des Spannungssensors

28. TinyML Shield: Built-in Push Button

28.1. General

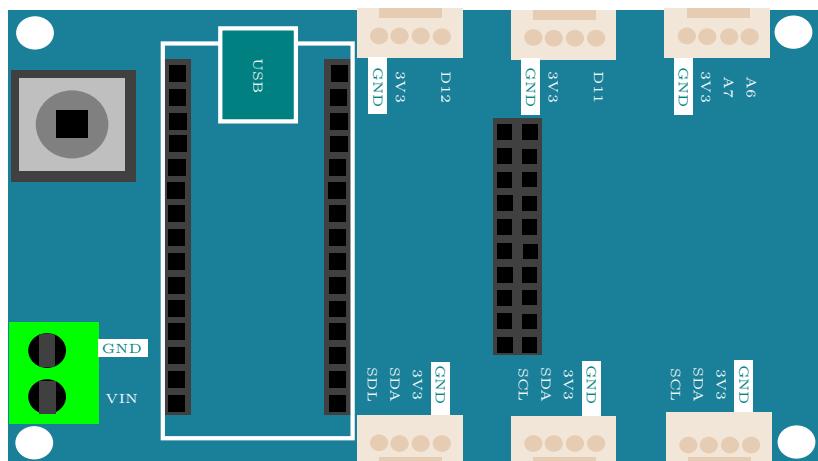
A push button is a simple switch mechanism used to control various devices and processes. It is typically made of hard materials like plastic or metal. The surface of a push button is designed to be easily depressed or pushed by the human finger or hand. When you press a push button, it either closes or opens an electrical circuit.

In industrial and commercial applications, push buttons can be linked together so that pressing one button releases another. Emergency stop buttons, often with large mushroom-shaped heads, enhance safety in machines and equipment. Pilot lights are sometimes added to push buttons to draw attention and provide feedback when the button is pressed. Color-coding is common to associate push buttons with their specific functions (e.g., red for stopping, green for starting). [Din]

28.2. Built-in Push Button

The Arduino Nano 33 BLE Sense features an onboard push button. This button is a simple electrical switch that can be activated by pressing it. When you press the button, it completes an electrical circuit. The push button is designed for user interaction and can be used for various purposes.

The built-in button `BUTTON_B` is connected with pin 11. Using the function `pinMode(BUTTON_PIN, INPUT_PULLUP)` the pin is declared as an input. As can be seen in the sketch, pressing the button can be used to trigger actions; typical actions include switching on an LED, changing modes, or initiating sensor readings. Overall, the push button provides a convenient way to interact with the Arduino Nano 33 BLE Sense and create responsive projects. [Ard23a; Ard23b; Arda]



WS:Push Button hier
nochmal mit tikz-code
hervorheben

Figure 28.1.: Tiny Machine Learning Shield [Gua21]

28.3. Specification

The built-in button is a small white button and connected to pin 11.

Built-in Button: `BUTTON_B = 11u`

If the pin is declared as input in the function `setup`, then it can be used.

The pin 11 must be defined as an input in the function `setup` by setting `pinMode(11, INPUT_PULLUP)`, otherwise the button cannot be read.

The pin 11 can also be used otherwise. Then the button is not in use. [Ard23a; Ard23b; Arda]

WS:

- cite data sheet
- Circuit Diagram

28.4. Simple Code

As soon as the button is connected, it can be used. It is not necessary to install a special library. Programming takes place in two steps:

1. In the first step, the pin is configured in the function `setup`:

Listing 28.1.: Defining the built-in button's pin as an input.

```
1000  pinMode(BUTTON_B, INPUT_PULLUP)
```

2. In the second step, the button can be used in the function `loop`. To read in the value, use the function `digitalRead`:

Listing 28.2.: Read the built-in button's state

```
1000  buttonState = digitalRead(BUTTON_B);
```

28.5. Tests

The simplest test is the flashing of an LED for 2 seconds, if the button is pressed, see sketch 28.3.

Listing 28.3.: Simple sketch to test the push button and the built-in LED

```
1000 #include <LEDGeneral.h>
1001 #include <LEDPower.h>
1002 #include <LEDRGB.h>
1003 #include <LEDSignsOfLife.h>
1004 #include <LEDbuiltin.h>
1005 #include <Arduino.h>
1006
1007 /**
1008 *  @file TestPushButton.ino
1009 *
1010 *  @brief Simple application reading in the built-in push button states.
1011 *
1012 *
1013 *  @details If the built-in push button is pressed, the the internal
1014 *          built-in LED is turn on for 2 seconds
```

```
1016 *
1018 */
1019
1020 #define BUTTON_B 11 /*< Button_B is here defined as pin 11 */
1021 #define BUTTON_PIN BUTTON_B /*< Use the onboard push button (BUTTON_B)
1022 */
1023
1024 int buttonState = 0; /*< state of the built-in push button */
1025
1026 /**
1027 * @brief the setup function runs once when you press reset or power the
1028 * board
1029 *
1030 * standard function of Arduino sketches
1031 *
1032 * Initialization of the built-in LED and the push button
1033 *
1034 * @param —
1035 *
1036 * @return void
1037 */
1038
1039 void setup() {
1040     // Initialize the pin as an output
1041     LEDBuiltinsetup();
1042     // Initialize the pin as an input
1043     pinMode(BUTTON_PIN, INPUT_PULLUP);
1044 }
1045
1046 /**
1047 * @brief the loop function runs over and over again forever
1048 *
1049 * standard function of Arduino sketches
1050 *
1051 * switching the built-in led on for 2 sec, if the push button is
1052 * pressed
1053 *
1054 * @param —
1055 *
1056 * @return void
1057 */
1058 void loop()
1059 {
1060     buttonState = digitalRead(BUTTON_PIN);
1061     if (buttonState == HIGH)
1062     {
1063         // Turn the LED on
1064         LEDBuiltin(SET_ON);
1065         // Wait for two second
1066         delay(2000);
1067         // Turn the LED off
1068         LEDBuiltin(SET_OFF);
1069         // Wait for one second
1070         delay(1000);
1071         buttonState = LOW;
1072     }
1073 }
```

..../Code/Nano33BLESense/PushButton/TestPushButton/TestPushButton.ino

28.6. Interrupt Function on the Arduino Nano 33 BLE Sense

An interrupt is a function that allows the microcontroller on the Arduino Nano 33 BLE Sense to immediately respond to an external event, such as pressing a button, without the need for the program to continuously check for that event. Normally, in a program, the microcontroller would repeatedly check if a button is pressed by running through a loop, which consumes processing power and can slow down other tasks. This is not efficient, especially when the program needs to perform other actions at the same time.

With an interrupt, the program can continue running other tasks, and only when the specified event (like a button press) occurs, the program is temporarily paused to execute a special function known as the **Interrupt Service Routine (ISR)**. The ISR is a small, efficient function designed to handle the event, such as changing a variable or triggering another action. After the ISR is executed, the program returns to where it left off, continuing its normal operation.

Using interrupts in this way helps make the program more efficient and responsive, as it doesn't waste time constantly checking for events and can focus on other tasks until something important happens. This is especially useful for real-time applications where quick responses to external events are necessary, such as in embedded systems, robotics, or sensor monitoring.

28.7. Simple Application

This code sketch 28.4 shows how to use an interrupt to control the built-in LED on the Arduino Nano 33 BLE Sense when the built-in push button is pressed.

The program sets up the built-in LED and push button. When the button is pressed, an interrupt runs a special function called an Interrupt Service Routine. The ISR is very short and only sets a flag variable `pushPressed` to `true`, indicating that the button has been pressed.

The main program `loop` checks this flag. If it is set to `true`, the program turns on the LED for 2 seconds, then turns it off and resets the flag to `false`. This ensures the system is ready to detect the next button press. Using the `true` and `false` values makes it easy to manage the button's state.

This method avoids constantly checking the button's state, making the program more efficient. Additionally, the Arduino Nano 33 BLE Sense allows all its pins to be used for interrupts. This makes it easy to attach interrupt functions to any pin, allowing quick and efficient responses to inputs like buttons or sensors. [Ardf]

WS:andere Überschrift
für simple Application

Listing 28.4.: Simple sketch connects the push button with an interrupt. Here, pushing the built-in button is handled by an interrupt. Then the built-in LED switch on for 2 sec.

```

1000 #include <LEDGeneral.h>
1001 #include <LEDPower.h>
1002 #include <LEDRGB.h>
1003 #include <LEDSignsOfLife.h>
1004 #include <LEDbuiltin.h>

1006 #include <LEDGeneral.h>
1007 #include <LEDPower.h>
1008 #include <LEDRGB.h>
1009 #include <LEDSignsOfLife.h>
1010 #include <LEDbuiltin.h>

1012 /**

```

```
* @file TestPushButtonInterrupt.ino
*
* @brief Simple application reading in the built-in push button states
*        using an interrupt
*
* @details If the builtin push button is pressed, the built-in LED is
*         switched on for 2 second and switched off again.
*         But in this example, an interrupt is used.
*/
#include <LEDGeneral.h>
#include <LEDBuiltin.h>

#define BUTTON_B 11 /*< Button_B is here defined as pin 11 */
#define BUTTON_PIN BUTTON_B

#define SET_ON true /*< Define flag for switching on */
#define SET_OFF false /*< Define flag for switching off */

// Initialize variables
// volatile bool pushPressed = false; /*< // Flag, whether the button is
// pressed. Declare as volatile for interrupt. safety */
int ledState = 0; /*< LED>Status zur Verarbeitng */

/** @brief the setup function runs once when you press reset or power the
*        board
*
* standard function of Arduino sketches
*
* Initialization of the built-in LED and the push button
*
* @param —
*
* @return void
*/
void setup() {
    // Initialize the pin as an output
    LEDBuiltinsetup();
    // Initialize the pin as an input
    pinMode(BUTTON_PIN, INPUT_PULLUP);
    // Initialize the interrupt function
    attachInterrupt(digitalPinToInterrupt(BUTTON_PIN), buttonPressed,
                    CHANGE);
}

/** @brief Interrupt service function
*
* Attention: as short as possible
*
* The function changes just one flag.
*/
void buttonPressed()
{
    if (pushPressed == false)
    {
```

```

1078     pushPressed = true;
1079 }
1080 /**
1082 * @brief the loop function runs over and over again forever
1084 *
1086 * standard function of Arduino sketches
1087 *
1088 * switching the built-in led on for 2 sec, if the push button is
1089 * pressed
1090 *
1091 * @param —
1092 *
1093 * @return void
1094 */
1095 void loop()
1096 {
1097     if (pushPressed)
1098     {
1099         // Turn the LED on
1100         LEDBuiltin(SET_ON);
1101         // Wait for one second
1102         delay(2000);
1103         // Turn the LED off
1104         LEDBuiltin(SET_OFF);
1105         pushPressed = false;
1106     }
1107     // ...
1108 }
```

..../Code/Nano33BLESense/Button/TestPushButtonInterrupt/TestPushButtonInterrupt.ino

This is just a simple example. The variable `BUTTON_B` is already defined, so the assignment is not necessary. The command `delay` should be avoided in an Arduino sketch. Instead, variables of the type `elapsedMillis` should be used.

28.8. Further Readings

- Boxall, John: *Arduino Workshop - A Hands-On Introduction with 65 Projects*. No Starch Press, 2021. [Box21]
- Voš, Andreas: *Volumio mit Drehgebern erweitern*. Make Magazin, 2024. [Voš24]
- Kühnel, Claus: *Arduino - Das umfassende Handbuch*. Rheinwerk Verlag GmbH, 2024. [Küh24]

29. Connectors of Type JST

29.1. Connector Series

JST connectors are electrical connectors manufactured to the design standards originally developed by J.S.T. Mfg. Co. (Japan Solderless Terminal). [Jsta] JST manufactures numerous series (families) and pitches (pin-to-pin distance) of connectors.

JST manufactures a large number of series (families) of connectors. The PCB (wire-to-board) connectors are available in top (vertical) or side (horizontal) entry, and through-hole or surface-mount. [Jstb]

This description refers exclusively to the mechanical structure and electrical behaviour. The pin assignment is not standardised.

Attention: The term “JST” is sometimes incorrectly used as a vernacular term meaning any small white electrical connector mounted on Printed Circuit Board (PCB)s.

29.2. JST VH

The technical data of the JST VH series is given in this section. They are taken from the data sheet for the series. [Jstd]

29.2.1. Product Profile

Series	VH connector
Category	Crimp Style Connectors (Wire-to-Board type)
Type	Crimp style, Compact type, With locking device Disconnectable type
Feature	With secure locking device

29.2.2. Specification

Pitch	3.96mm
Pin rows	1
Current rating	10A (AWG#16)
Voltage rating	250V
PC board mounting direction	Top entry, Side entry



Figure 29.1.: JST connector type VH <https://www.jst-mfg.com/product/index.php?series=262>

29.3. JST PH

The technical data of the JST PH series is given in this section. They are taken from the data sheet for the series. [Jstc]

29.3.1. Product Profile

Series	PH connector
Category	Crimp Style Connectors (Wire-to-Board type)
Type	Crimp style, Thin type Disconnectable type

29.3.2. Specification

Pitch	2.0mm
Current rating	2A (AWG#24)
Voltage rating	100V
PC board mounting direction	Top entry, Side entry

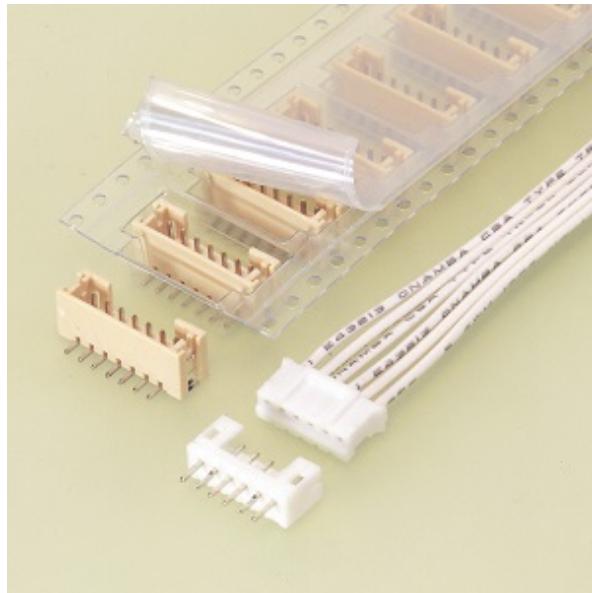


Figure 29.2.: JST connector type VH [Jstc]

29.3.3. JST PHR-2

One variant of the connector series is the JST PHR-2 type, which has 2 pins. Figure 29.3 shows a JST PHR-2 connector.



Figure 29.3.: JST connector type PHR-2 [Jstc]

30. Grove

Grove, ein Produkt von Seeed Studio, ist eine Hardware-Schnittstelle für Technikinteressierte. Es vereinfacht die Kommunikation zwischen Mikrocontrollern und Sensoren und reduziert das Löten. Diese Schnittstelle wurde im Jahr 2010 von Seeed Studio entwickelt, um die Zusammenarbeit zwischen verschiedenen Technologien zu vereinfachen und die Entwicklung neuer Projekte zu erleichtern.[See12; See13]

Mit der Verfügbarkeit eines Steckverbinder-Typs bietet diese Schnittstelle eine Vielzahl von Möglichkeiten für die Integration von Sensoren, Aktuatoren und Mikrocontrollern. Diese einfachen Kommunikationsverbindungen ermöglichen es den Nutzern, ihre Projekte ohne großen Zeitaufwand zu realisieren und zu testen. [See16] Mit jedem neuen Sensor, der mit der Grove-Familie auf dem Markt erscheint, wird die Flexibilität und kreative Möglichkeit für die Entwicklung von Projekten weiter gesteigert.

Die meisten Mikrocontroller verfügen nicht unmittelbar über die Grove-Schnittstelle. Damit Komponenten, die die Grove-Schnittstelle haben, werden häufiger Shields eingesetzt, die diese dann zur Verfügung stellt. Zum Beispiel verfügt der Arduino Nano 33 BLE Sense Lite über keine Grove-Anschlüsse. Das Tiny Machine Learning Shield ermöglicht die Verwendung von Grove-Komponenten für den Arduino Nano 33 BLE Sense, indem es als Schnittstelle zwischen dem Mikrocontroller und den Grove-Modulen fungiert. Der Arduino Nano 33 BLE Sense wird direkt auf das Shield gesteckt, welches über standardisierte Grove-Anschlüsse verfügt. Auf der Abbildung 30.1 ist dargestellt, wie der Mikrocontroller Arduino Nano 33 BLE Sense mit dem Shield verbunden ist.

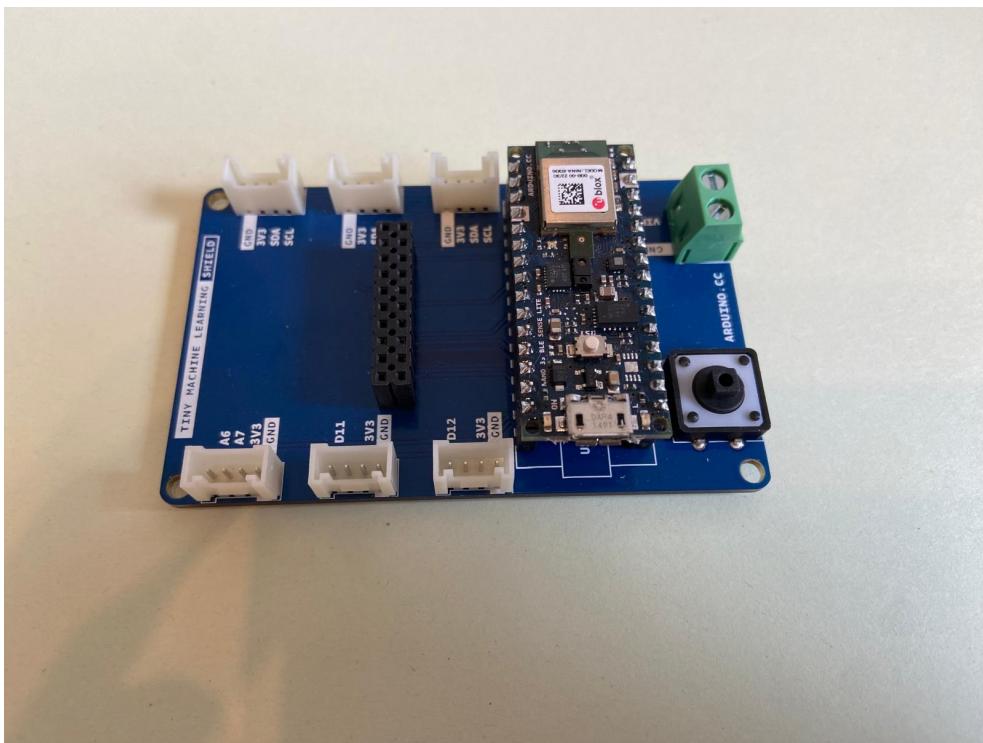


Figure 30.1.: Mikrocontroller gesteckt auf dem Shield

30.1. Grove-Universal-Kabel

Üblicherweise werden Sensoren und Mikrocontroller über ein Grove-Universal-Kabel verbunden. In der Abbildung 30.2, vom Hersteller *seeed studio* ermöglicht die Verbindung mit allen Modulen des Grove-Systems. Für ein Grove-System wird somit nur eine Kabelart benötigt. Die Schnittstellen des Kabels sind beidseitig Japan Solderless Terminal (JST)-VH-Stecker. In der Abbildung 30.2 beträgt beispielsweise die Kabellänge 50 mm [See16].



Figure 30.2.: Grove-Universal-Kabel[Rei24a]

30.2. Verbindung Grove-Schnittstelle zu 4-Pin-Lösungen

Da nicht alle Sensoren und Mikrocontroller über eine Grove-Schnittstelle verfügen, existieren Kabel, die einseitig die Grove-Schnittstelle haben. In der Abbildung 30.3 ist das Kabel dargestellt, das auf einer Seite die Grove-Schnittstelle zur Verfügung stellt und auf der anderen Seite 4 Pins. In diesem Beispiel hat das Kabel eine Länge von 30 cm [Rei24c].



Figure 30.3.: Grove Jumper zu Grove 4 Pin-Kabel[Rei24c]

30.3. Pinbelegung

Die Pinbelegung ist von *seeed studio* standardisiert. In der Tabelle 30.1 ist die Belegung erläutert. [See16]

Table 30.1.: Pinbelegung der Grove-Schnittstelle

Farbe	Pin	Funktion
Gelb	1	frei belegbar, D0 oder A0
Weiß	2	frei belegbar, D1 oder A1
Rot	3	VCC, 3,3V oder 5V
Schwarz	4	GND

Grove stellt über Pin 3 und 4 die Stromversorgung sicher. Der gelbe und der weiße Pin ist in Abhängigkeit vom Sensor belegt. Falls der Sensor nur einen Pin benötigt, so wird Pin 1 verwendet. Als Schnittstelle zum Mikrocontroller muss die Belegung für jede Anwendung definiert werden.

Für Protokolle, wie zum Beispiel I²C und Serial Peripheral Interface (SPI), sind die Belegung eindeutig festgelegt. Die Tabelle 30.2 listet die Pinbelegung der Grove-Schnittstelle für I²C-Grove-Stecker auf.

Table 30.2.: Pinbelegung der Grove-Schnittstelle für I²C-Grove-Stecker

Farbe	Pin	Funktion
Gelb	1	frei belegbar, zum Beispiel SCL auf I ² C-Grove-Steckern
Weiß	2	frei belegbar, zum Beispiel SDA auf I ² C-Grove-Steckern
Rot	3	VCC, 3,3V oder 5V
Schwarz	4	GND

Die Tabelle 30.3 enthält die Pinbelegung für die serielle Schnittstelle UART.

Table 30.3.: Pinbelegung der Grove-Schnittstelle für UART

Farbe	Pin	Funktion
Gelb	1	RX
Weiß	2	TX
Rot	3	VCC, 3,3V oder 5V
Schwarz	4	GND

31. Zwei Taster mit Grove-Anschluss von M5 Stack

31.0.1. Pin-Konfiguration

Die verwendeten Pins werden zu Beginn des Codes definiert:

- `buttonPin`: Der Pin, an dem der Taster angeschlossen ist, der zur Messungsteuerung verwendet wird
- `blueButtonPin`: Der Pin für den blauen Taster, der zur Startsteuerung der Messung verwendet wird
- `redButtonPin`: Der Pin für den roten Taster, der zur Beendigung der Messung und Anzeige der Ergebnisse verwendet wird

31.0.2. Initialisierung und Setup

In der Funktion `setup()` werden die erforderlichen Initialisierungen durchgeführt:

- `pinMode(buttonPin, INPUT_PULLUP)`: Konfiguriert den Taster-Pin als Eingang mit Pull-Up-Widerstand
- Zunächst wird der Zustand der beiden Taster überprüft und die Abfrage nach einem gedrückten Button ist aktiv (`blueButtonPressed`, `redButtonPressed`).
- Wenn der blaue Taster gedrückt wird, wird die Messung gestartet, indem `isMeasuring` auf `true` gesetzt wird und die Funktion `StartSampling()` aufgerufen wird.
- Wenn der rote Taster gedrückt wird, wird die Messung beendet, indem `isMeasuring` auf `false` gesetzt wird und die Funktion `StopSampling()` aufgerufen wird. Je nach vorherigem Zustand des roten Tasters wird entweder der maximale Wert SPL in dB angezeigt (`ShowMaxdBspl()`) oder der durchschnittliche dB SPL-Wert (`ShowAveragedBspl()`).
- `isDelayOver` wird auf `true` gesetzt, wenn die Startverzögerung (definiert durch `measureThresholdMilliseconds`) von 1200ms abgelaufen ist.
- Es gibt eine kurze Verzögerung (`delay(50)`) zwischen den Schleifendurchläufen, um Tastenprellungen zu vermeiden.

`PDM.begin(1, 16000)`: Initialisiert eine Abtastrate von 16.000 Hz. Die Funktion `StopSampling()` beendet die Aufnahme von Audiodaten.

31.1. 2.Beschreibung

Wenn per Taster ein Signal erzeugt werden soll, ist dies bei einem Schaltkreis, der nur aus Stromquelle und LED besteht, kein Problem. Bei gedrücktem Taster fließt Strom und die LED leuchtet. In unserem Anwendungsfall soll ein Tasterdruck digital

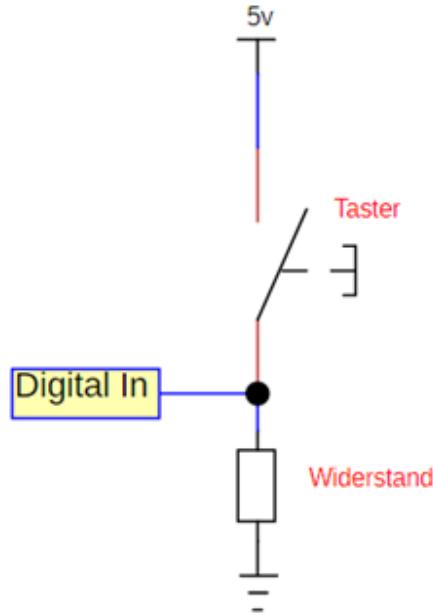


Figure 31.1.: Prinzipieller Aufbau Pull-down Schaltung (Eigene Darstellung)

ausgewertet werden, also soll der Zustand 0 oder 1 überprüft werden. In der Praxis entstehen beim Drücken sogenannte Floating Pins, der Arduino kann nicht erkennen welches Signal anliegt. Dieses Problem wird durch Pull-up- oder Pull-down-Widerstände gelöst. Der Dual-Button hat für jeden Taster eine Pull-down Schaltung. Dabei wird die Spannung der Signalleitung über einen hochohmigen Widerstand zu GND geführt und liefert ein klares Null-Signal. Bei Betätigung des Tasters fließt Strom zur Signalleitung und erzeugt ein Eins-Signal. Dieses kann nun vom Arduino deutlich ausgewertet und weiterverwendet werden.

Ein großer Vorteil der Dual Button Unit ist, dass keine Lötarbeiten anfallen. Per Grove Anschluss kann der Button direkt an das Shield angeschlossen werden. Außerdem sind die Taster farblich gekennzeichnet, was die Anwendung für den Endbenutzer stark vereinfacht. Erwähnenswert ist noch, dass das Gehäuse aus Klemmbausteinen gebaut werden könnte, da zwei passende Aufnahmen vorhanden sind. Dadurch kann ein Prototyp leicht gebaut werden, falls kein 3D Drucker zur Verfügung steht. Der blaue Taster ist mit dem weißen Kabel verbunden und der rote Taster mit dem gelben. [M5S21]



Figure 31.2.: Dual Button oben [M5S21]



Figure 31.3.: Dual Button unten [M5S21]

32. External Standard LED

32.1. General

LED stands Light-Emitting Diode. A diode is an electronic component that only allows current to pass in one direction. Most diodes are made of semiconductor materials; in the case of light-emitting diodes, it is usually silicon.

The way an LED works is known from the animal world. For example, the energetic firefly is a flying LED. The only difference is how the atoms inside are excited and thus made to glow. The fireflies achieve this through a chemical reaction. In the LED, this happens with the help of an electric current - in short: electroluminescence.

A light-emitting diode consists of an anode (positive pole) and a cathode (negative pole). A wire, the so-called bonding wire, ensures the flow of current between the two poles. The energy required for electroluminescence flows through this wire. The semiconductor crystal (also known as the LED chip) sits on the cathode. It is surrounded by a reflector trough that directs the light from the LED chip upwards. The entire structure is embedded in a protective plastic lens. It distributes the light in the room, thereby increasing both the efficiency and the luminous efficacy.

The chip is a semiconductor crystal made from two different materials that are doped differently. Doping means that there is an excess of positive or negative charge carriers. When current flows, the electrons react and energy is released in the form of light flashes (photons). The LED lights up.

The color of the light depends on the doping of the semiconductor layers. Depending on the energy level, photons with different wavelengths, i.e. light color, are released. For example, a high energy level produces short-wave blue light and a low energy level produces long-wave red light. The superposition of the three primary colors red, blue and green produces white light. The colors red and green produce yellow. Red and blue become magenta, green and blue produce cyan.

A multicolor light-emitting diode therefore contains three semiconductor crystals, each of which produces one of the three primary colors and expands the color palette in different compositions.

Depending on how the brightness of red, green and blue is controlled, white is produced in different shades between very bright cool white light (indicated on the packaging as "5000 Kelvin" and above) or dimmed warm feel-good light (around 3000 Kelvin). As this variant of the mixture is expensive, there is now a new manufacturing option: blue LED chips are coated with a yellowish phosphor layer.

The colors available for acleds are quite diverse, ranging from the basic red, green, and blue (RGB) to a much wider spectrum depending on the technology and application. [Qua] Here's a breakdown of the different types:

Red: This is the most common LED color, achieved by using materials like gallium arsenide phosphide (GaAsP). It has a dominant wavelength around 620-750 nm and appears vibrant red to the human eye.

Green: Another common LED color, typically made with indium gallium nitride (InGaN). Its dominant wavelength falls between 500-570 nm, producing a bright green color.

Blue: Blue LEDs are often made with gallium nitride (GaN) and emit light with a dominant wavelength of 450-480 nm. They appear as a deep, rich blue to our eyes.

Combinations and White Light:

White: While not a single color, white LEDs are crucial for various applications. They are typically achieved in two ways:

- **RGB combination:** Combining red, green, and blue LEDs in specific ratios can create white light. This method offers flexibility in color temperature control.
- **Phosphor conversion:** A blue or ultraviolet LED is coated with a phosphor that converts part of the emitted light to longer wavelengths, generating white light. This method is more efficient but offers less color control.

Beyond the Basics:

Amber: Often used in traffic signals and indicator lights, amber LEDs have a dominant wavelength around 585-605 nm and appear yellowish-orange.

Yellow: True yellow LEDs are less common but achievable with specific materials like aluminum gallium indium phosphide (AlGaInP). Their dominant wavelength is around 580-590 nm, appearing as a pure yellow color.

Other colors: Specialized LEDs can produce various other colors, including violet, pink, cyan, and even deep red and green for wider spectrum applications. These often involve more complex materials and manufacturing processes.

It's important to note that the specific color of an acled can vary slightly depending on factors like manufacturing tolerances, viewing angle, and operating conditions. Additionally, new acled technologies are emerging constantly, potentially expanding the color range further in the future. [Kai09; HEG21; HS23; Bai18]

32.2. Specification

A LED need to have a resistor placed in series with it. Otherwise, the unrestricted current will quickly burn out the LED. The resistor can be any value between 100 Ohms and about 10K Ohms. It is necessary to check the data sheet. Lower value resistors will allow more current to flow, which makes the LED brighter. Higher value resistors will restrict the current flow, which makes the LED dimmer. A typically value is 220 Ohm. Using the data sheet of the LED, the value of the resistor can be determined. [Qua; Hui] The maximal current to operate an LED is 20mA. It works also using 4mA until 15 mA. [Qua]

Most LEDs have polarity, which means that they need to be connected the right way around. Usually, the LED's shortest lead connects to the ground side, see figure 32.2

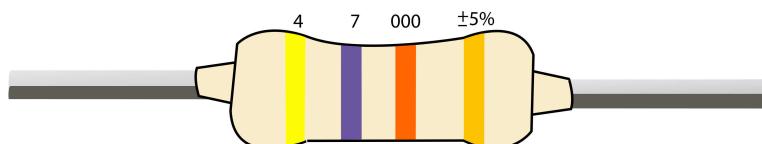


Figure 32.1.: Resistor with 220 Ohm for an external LED.

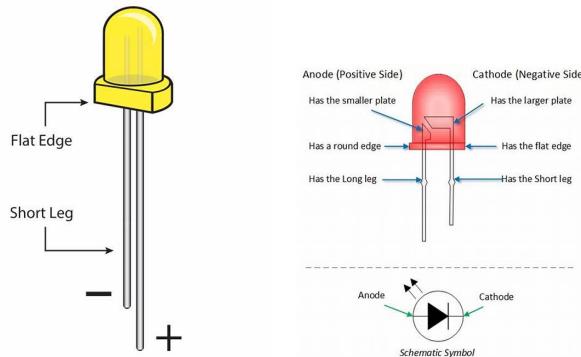


Figure 32.2.: Parts of a LED.

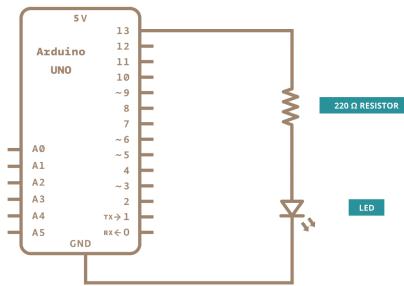


Figure 32.3.: Schematic Symbol for a LED.

The Arduino can control as many LEDs as long as there are enough pins available.

32.3. Using Standard External LED

32.3.1. Connecting a LED to the Arduino Board

32.3.2. Pins

As mentioned, the Arduino can control as many LEDs as long as there are enough pins available. There are two different types of output pins. The standard output pin can just switch between high and low. For some pins the PWM is usable. Using such a pin, the brightness of the LED can be controlled.

PWM A0 - A7 Welche Nummer? 16-23? 19 - 26

Digital D2-D13 Welche Nummer Pin 5 -16

WS:create a diagram that shows how to connect a common cathode RGB LED to the board

WS:Which pins

WS:Grafik!

32.4. Bibliothek

No special library is required to operate an external LED.

32.5. Simple Code

As soon as the acled is connected, it can be used. It is not necessary to install a special library. Programming takes place in three steps:

1. In the first step, the acled must be connected to a pin:

Listing 32.1.: Defining the pin for an external LED

```
1000 #define LED_EXT 14
```

2. In the second step, the pin is configured in the function `setup`:

Listing 32.2.: Defining the pin for an external LED

```
1000 pinMode(LED_EXT, OUTOUT)
```

3. In the third step, the acled can be used in the function `loop`. To turn OFF the LEDs, write a state `LOW` to the LED:

Listing 32.3.: Swichting On the LED

```
1000 // Swichting On the LED
1002 digitalWrite(LED, HIGH);
1003 //Waiting 1s
1004 delay(1000)
1005 // Swichting Off the LED
1006 digitalWrite(LED, LOW);
```

32.6. Tests

32.6.1. Simple Function Test

The simplest test is the flashing of the LED at 2 Hz, see sketch 32.4.

Listing 32.4.: Simple sketch to test a LED

```
1000 /**
1001 *
1002 * @file TestLED.ino
1003 *
1004 * @brief Simple program for testing an external LED
1005 *
1006 *
1007 * How to control an external LED with the Arduino Nano 33 BLE Sense
1008 *
1009 * The LED is switched on for 1 second and switched off
1010 * for 1 second so that the LED flashes accordingly.
1011 *
1012 *
1013 * Author: Elmar Wings
1014 * Created: 06.08.2024
1015 */
1016
```

```

1018 #include <../LEDs/LED.h>
1019 #include <../LEDs/SignsOfLife.h>
1020
1021 #define LED_EXT 14 /*< Define the pin number for the builtin-LED */
1022
1023 #define CycleTimeOn 1000 /*< Duty cycle [ms] */
1024
1025 #define CycleTimeOff 1000 /*< Switch-off time [ms] */
1026
1027
1028 /**
1029 * @brief the setup function runs once when you press reset or power the
1030 * board
1031 *
1032 * standard function of Arduino sketches
1033 *
1034 * Initialization of the pin LED_EXT as output
1035 *
1036 * @param —
1037 *
1038 */
1039 void setup() {
1040     // Initialize the function SignsOfLife
1041     SignsOfLifeInit(LED_EXT, CycleTimeOn, CycleTimeOff)
1042
1043     // Application
1044     // ...
1045 }
1046
1047 /**
1048 * @brief the loop function runs over and over again forever
1049 *
1050 * standard function of Arduino sketches
1051 *
1052 * swichting the led on / off for 1sec.
1053 *
1054 * @param —
1055 *
1056 * @return void
1057 */
1058 void loop() {
1059     // Switch builtin LED on/off
1060     SignsOfLife();
1061
1062     // Application
1063     // ...
1064 }

```

..../Code/Nano33BLESense/Test/TestLED.ino

32.6.2. Test all Functions

Standard pins can be used, but pins that support pulse width modulation can also be used. One such pin is used in the example 32.5. The brightness can then be varied.

Listing 32.5.: Simple sketch to test a LED with pulse width modulation

```

1000 /**
1001 * @file TestLEDBrightness.ino
1002 *
1003 * @brief Simple program for controlling the brightness of an external
1004 * LED of the Arduino Nano 33 BLE Sense
1005 *
1006 * Using PWM, the LED's brightness is increasing until full and reverse.
1007 */

```

```

1008 */
1010 #include <../LEDs/PowerLED.h>
1011 #include <../LEDs/LED.h>
1012
1013 #define LED_EXT 21 /*< Define the pin for the external LED */
1014 int ledBrightness = 0; /*< Define the initial brightness values (0-255)
1015 */
1016
1017 int ledStep = 5; /*< Define the increment/decrement value */
1018
1019
1020 /**
1021 * @brief the setup function runs once when you press reset or power the
1022 * board
1023 *
1024 * standard function of Arduino sketches
1025 *
1026 * Initialization of the pin LED_EXT as output
1027 *
1028 * @param —
1029 *
1030 * @return void
1031 */
1032 void setup() {
1033     // Initialize the pin as an output
1034     LEDinit(LED_EXT);
1035 }
1036
1037 /**
1038 * @brief the loop function runs over and over again forever
1039 *
1040 * standard function of Arduino sketches
1041 *
1042 * changing continuously the LED's brightness
1043 *
1044 * @param —
1045 *
1046 * @return void
1047 */
1048 void loop() {
1049     // Write the PWM values to the LED pin
1050     LEDBrightness(LED_EXT, redBrightness);

1051     // Update the brightness values
1052     Brightness += Step;

1053     // Check if the brightness values are out of range and reverse the
1054     // direction
1055     if (Brightness <= 0 || Brightness >= 255) {
1056         Step = -Step;
1057     }

1058     // Wait for 10 milliseconds
1059     delay(10);
1060 }
1061
1062 }
```

..../Code/Nano33BLESense/Test/TestLEDBrightness.ino

32.7. Simple Application

In the sketch 32.6, a variable is connected to pin 14. The pin is defined as an output in the function `setup`. An interrupt function is defined, changing the state of a flag.

If the flag has the value `true`, the led is switch on for 2 seconds. After the 2 seconds, the led is switch off and the value of the flag is set to `false` back.

Listing 32.6.: Simple sketch to control an external LED. Here, pushing the built-in button is handled by an interrupt. Then the LED switch on for 2 sec.

```

1000 /**
1001 *
1002 * @file TestPushButtonInterrupt.ino
1003 *
1004 * @brief Simple application reading in the built-in push button states
1005 * using an interrupt
1006 *
1007 * @details If the builtin push button is pressed, the built-in LED is
1008 * switched on for 2 second and switched off again.
1009 * But in this example, an interrupt is used.
1010 */
1011
1012 #include <LED.h>
1013 #include <BuiltinLED.h>
1014
1015 #define BUTTON_PIN BUTTON_B /*< Use the onboard push button (BUTTON_B)
1016 */
1017
1018
1019 // Initialize variables
1020 //
1021 volatile bool pushPressed = false; /*< // Flag, whether the button is
1022 * pressed. Declare as volatile for interrupt. safety */
1023
1024 int ledState = 0; /*< LED-Status zur Verarbeitng */
1025
1026 /**
1027 * @brief the setup function runs once when you press reset or power the
1028 * board
1029 *
1030 * standard function of Arduino sketches
1031 *
1032 * Initialization of the built-in LED and the push button
1033 *
1034 * @param —
1035 *
1036 * @return void
1037 */
1038 void setup() {
1039     // Initialize the pin as an output
1040     BuiltinLEDinit();
1041     // Initialize the pin as an input
1042     pinMode(BUTTON_PIN, INPUT_PULLUP);
1043     // Initialize the interrupt function
1044     attachInterrupt(digitalPinToInterrupt(BUTTON_PIN), buttonPressed,
1045                     FALLING);
1046 }
1047
1048 /**
1049 * @brief Interrupt service function
1050 *
1051 * Attention: as short as possible
1052 *
1053 * The function changes just one flag.
1054 */
1055 void buttonPressed()
1056 {

```

```
1056     if (pushPressed == false)
1058     {
1060         pushPressed = true;
1060     }
1062 /**
1063 * @brief the loop function runs over and over again forever
1064 *
1065 * standard function of Arduino sketches
1066 *
1067 * switching the built-in led on for 2 sec, if the push button is
1068 * pressed
1069 *
1070 * @param ——
1071 *
1072 */
1073 void loop()
1074 {
1075     if (pushPressed)
1076     {
1077         // Turn the LED on
1078         BuildinLED(SET_ON);
1079         // Wait for one second
1080         delay(2000);
1081         // Turn the LED off
1082         BuildinLED(SET_OFF);
1083         pushPressed = false;
1084     }
1085     // ...
1086 }
```

..../Code/Nano33BLESense/Test/TestPushButtonInterrupt.ino

This is just a simple example. The variable `BUTTON_B` is already defined, so the assignment is not necessary. The command `delay` should be avoided in an Arduino sketch. Instead, variables of the type `elapsedMillis` should be used.

32.8. Further Readings

WS:citations

33. External RGB-LED

33.1. Standard RGB-LED

An RGB-LED is a type of LED that can emit different colors of light. RGB stands for red, green, and blue, which are the three primary colors of light. An RGB-LED consists of three individual LEDs inside a single package, each with its own color and pin. By varying the brightness of each LED using PWM signals, an RGB-LED can produce a wide range of colors by mixing the primary colors. An RGB-LED is usually active-low, which means that setting the pin to LOW will turn the LED on, and setting the pin to HIGH will turn the LED off.

To connect a RGB-LED to an Arduino Nano 33 BLE Sense board, you need to use three resistors, one for each color pin. The value of the resistors depends on the type and specifications of the RGB LED, but a common value is 220 ohms. You also need to connect the common cathode or anode of the RGB-LED to the ground or 3.3V pin of the board, respectively. Here is a diagram that shows how to connect a common cathode RGB LED to the board:

WS:create a diagram that shows how to connect a common cathode RGB-LED to the board

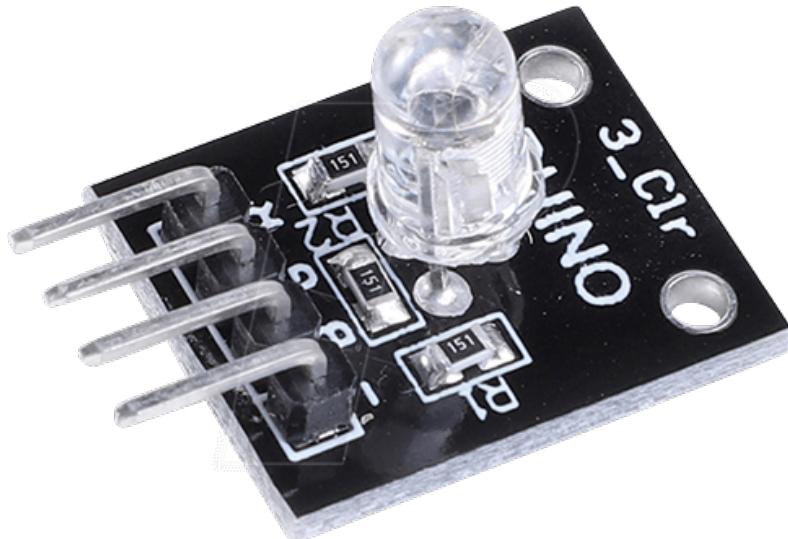


Figure 33.1.: External RGB-LED with Resistors [Simd]

External RGB-LED with Resistors [Simd; Kin]

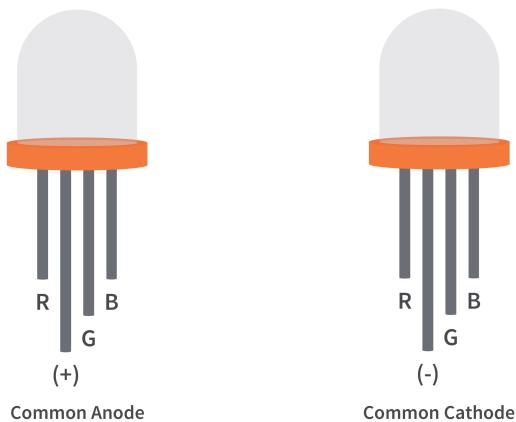
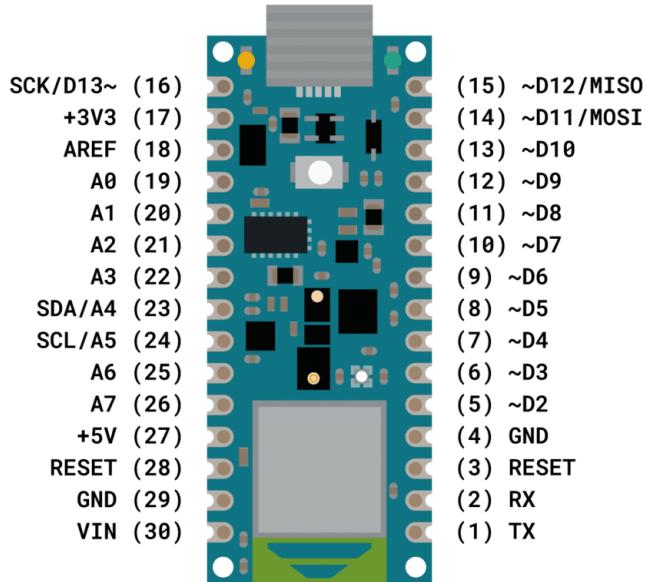


Figure 33.2.: Connectors of a RGB-LED.

33.2. Specific Sensor

WS:talk
WS:LED on
RGB-LED

WS:Which one

Red LED: LEDR = Pin 22

Green LED: LEDG = Pin 23

Blue LED: LEDB = Pin 24

33.3. Specification

cite data sheet

33.4. Calibration

cite method

33.5. Simple Code

33.6. Simple Application

33.7. Tests

33.7.1. Simple Function Test

To turn ON the LEDs, write a state `HIGH` to the LED:

```
1000 digitalWrite(LED_R, HIGH); //RED
1001 digitalWrite(LED_G, HIGH); //GREEN
1002 digitalWrite(LED_B, HIGH); //BLUE
```

Listing 33.1.: Swichting On the LEDs

To turn OFF the LEDs, write a state `LOW` to the LED:

```
1000 digitalWrite(LED_R, LOW); //RED
1001 digitalWrite(LED_G, LOW); //GREEN
1002 digitalWrite(LED_B, LOW); //BLUE
```

Listing 33.2.: Swichting Off the LEDs

33.7.2. Test all Functions

33.7.3. Brightness of the RGB-LED

In a short cut, using values between 255 - 0 to write to the RGB-LED is possible, too. Then the brightness is defined.

```
1000 analogWrite(LED_R, 72); //GREEN
1001 analogWrite(LED_G, 122); //BLUE
1002 analogWrite(LED_B, 234); //RED
```

Listing 33.3.: value between 255 - 0 to write to the RGB-LED

This sketch 33.4 will make the RGB-LED change colors smoothly by varying the brightness of each LED with different speeds. You can adjust the initial brightness values and the increment/decrement values to get different effects.

```

1000 // Define the pins for the RGB-LED
1002 #define RED_PIN 22
1003 #define GREEN_PIN 23
1004 #define BLUE_PIN 24
1005
1006 // Define the initial brightness values for each color (0–255)
1007 int redBrightness = 0;
1008 int greenBrightness = 0;
1009 int blueBrightness = 0;
1010
1011 // Define the increment/decrement value for each color
1012 int redStep = 5;
1013 int greenStep = 3;
1014 int blueStep = 7;
1015
1016 void setup() {
1017     // Set the LED pins as outputs
1018     pinMode(RED_PIN, OUTPUT);
1019     pinMode(GREEN_PIN, OUTPUT);
1020     pinMode(BLUE_PIN, OUTPUT);
1021 }
1022
1023 void loop() {
1024     // Write the PWM values to the LED pins
1025     analogWrite(RED_PIN, redBrightness);
1026     analogWrite(GREEN_PIN, greenBrightness);
1027     analogWrite(BLUE_PIN, blueBrightness);
1028
1029     // Update the brightness values for each color
1030     redBrightness += redStep;
1031     greenBrightness += greenStep;
1032     blueBrightness += blueStep;
1033
1034     // Check if the brightness values are out of range and reverse the
1035     // direction
1036     if (redBrightness <= 0 || redBrightness >= 255) {
1037         redStep = -redStep;
1038     }
1039     if (greenBrightness <= 0 || greenBrightness >= 255) {
1040         greenStep = -greenStep;
1041     }
1042     if (blueBrightness <= 0 || blueBrightness >= 255) {
1043         blueStep = -blueStep;
1044     }
1045
1046     // Wait for 10 milliseconds
1047     delay(10);
1048 }
```

Listing 33.4.: Different brightness levels for the RGB-LED colors

Colors

A RGB-LED is a device that can emit light of different colors by mixing the primary colors of red, green, and blue. The color of the light depends on the relative brightness of each LED, which can be controlled by PWM signals. By varying the brightness of each LED, the RGB-LED can produce a wide range of colors, such as yellow, cyan, magenta, white, and more. Some examples of the colors and their corresponding brightness values are:

Red: red = 255, green = 0, blue = 0

Green: red = 0, green = 255, blue = 0

Blue: red = 0, green = 0, blue = 255

Yellow: red = 255, green = 255, blue = 0

Cyan: red = 0, green = 255, blue = 255

Magenta: red = 255, green = 0, blue = 255

White: red = 255, green = 255, blue = 255

Black: red = 0, green = 0, blue = 0

The RGB-LED can also create intermediate colors by using different brightness values for each LED. For example, to create

- **orange**, one can use red = 255, green = 127, blue = 0.
- To create **pink**, one can use red = 255, green = 192, blue = 203.
- To create **purple**, one can use red = 128, green = 0, blue = 128.

The RGB-LED can also create gradients of colors by changing the brightness values gradually over time. This can create a smooth transition from one color to another, such as from red to green to blue and back to red.

33.8. Simple Application

33.9. Further Readings

34. Sensor BME280 für Temperatur, Luftfeuchtigkeit und den Luftdruck

34.1. Beschreibung der Hardware

Der BME280 ist ein Temperatursensor, der von Bosch Sensortec entwickelt wurde. Der Sensor bietet die Möglichkeit, die Temperatur, Luftfeuchtigkeit und den Luftdruck in der Umgebung zu messen, siehe Abbildung 34.1). [Fun]

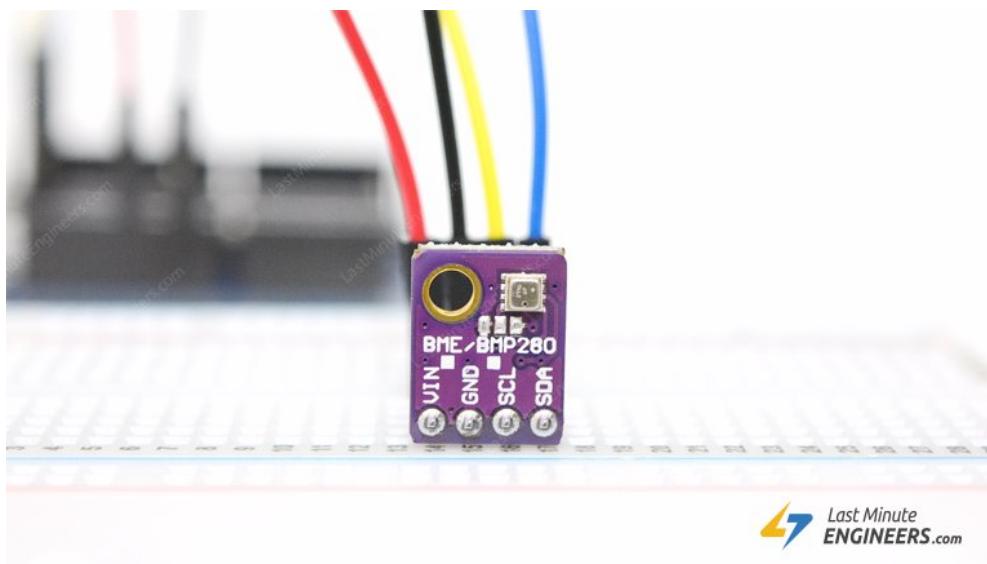


Figure 34.1.: Sensor BME280
[Las]

Bei dem BME280 handelt es sich um einen kombinierten Sensor für die Messung von Feuchtigkeit, Luftdruck und Temperatur. Dieser ist auf dem Entwicklerboard DEBO BME280 verbaut. Das Entwicklerboard hat die Maße 15,3 x 11,5 x 2,5 mm (LxBxH), wobei der Sensor BME280 die Maße 2,5 x 2,5 x 0,93 mm (LxBxH) aufweist. Die Schnittstellen I2C und SPI des Entwicklerboards ermöglichen die Kommunikation zwischen dem Arduino und dem Sensor. Die Versorgungsspannung bewegt sich zwischen 1,72 V und 3,6 V. Die gemessene Luftfeuchtigkeit erfolgt mit einer Genauigkeitstoleranz von ± 3 Prozent relativer Luftfeuchtigkeit und einer Reaktionszeit von einer Sekunde. Der Druckbereich für den Luftdruck beträgt 300 bis 1100 hPa mit einer relativen Genauigkeit von $\pm 0,12$ hPa und einer absoluten Genauigkeiten von ± 1 hPa. Der Temperatursensor hat einen Bereich von -40 bis +85 °C und besitzt eine voll Genauigkeit im Bereich von 0° bis 65° C. Der durchschnittliche Stromverbrauch des Sensors bei einer Frequenz von 1 Hz beträgt $1.8\mu A$ für die Messung von Feuchtigkeit und Temperatur, $2.8\mu A$ für die Messung von Luftdruck und Temperatur und $3.6\mu A$ für die Messung von Feuchtigkeit, Luftdruck und Temperatur. Der Sensor BME280 verfügt über mehrere Pins, die für verschiedene Zwecke verwendet werden. Hierbei ist es wichtig die richtige Pin-Belegung für den Sensor zu kennen. Im Allgemeinen sind die Pins wie folgt belegt: Der VCC-Pin wird mit der positiven Stromversorgung (+3.3 V oder +5

V) des Systems verbunden. Der GND-Pin muss mit dem Masseanschluss (GND) Ihres Systems verbunden werden. Der SDA-Pin ist der Datenpin für die Kommunikation I2C. Dieser wird mit dem entsprechenden SDA-Pin auf dem Mikrocontroller verbunden. Der SCL-Pin ist der Takt-/Clock-Pin für die Kommunikation I2C. Dieser muss mit dem entsprechenden SCL-Pin auf dem Mikrocontroller verbunden werden. SDI sorgt für die SPI-Kommunikation. [Bosa; Bosb]

34.2. Schaltplan des Aufbaus

Der folgende Schaltplan stellt die Komponenten des Arduino Nano BLE Sense Lite, des Sensors BME280 und des OLED-Displays dar. Als Spannungsquelle dient ein Computer, der den Arduino Nano 33 BLE Sense über ein USB-A auf Mikro-USB Kabel versorgt. Die Komponenten sind sinngemäß miteinander verbunden und in Abbildung 36.8 abgebildet.

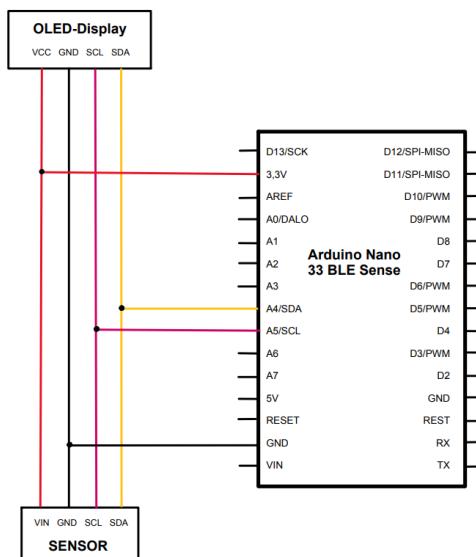


Figure 34.2.: Stationärer Aufbau der Wetterstation
[Arda]

34.2.1. Bibliothek [cactus_io_BME280](#) für den Sensor BME280

Die Bibliothek [cactus_io_BME280_I2C](#) ist eine spezielle Arduino Bibliothek zur Kommunikation mit dem Sensor BME280. Die Bibliothek erleichtert die Interaktion mit dem Sensor BME280 und bietet eine einfache Möglichkeit, Messwerte abzurufen und sie in Ihren Arduino-Projekten zu verwenden. Um die Bibliothek [cactus_io_BME280_I2C](#) zu verwenden, muss sie erst als ZIP-Datei heruntergeladen und anschließend von dem Arduino Library Manager installiert werden. Mit Hilfe von `#include<"cactus_io_BME280_I2C">` können diese in dem Arduino-Code eingebunden werden. [Iot]

34.2.2. Testen des OLED-Displays

Für die Programmierung des Displays gibt es viele Möglichkeiten. Da es viele Librarys gibt, muss zuerst einmal überlegt werden, welche verwendet werden sollen. Um einen ersten Eindruck über die Programmierung des Displays zu bekommen, wurde zunächst ein Beispielsketch getestet. Es handelt sich hier um den Sketch "Hello World",

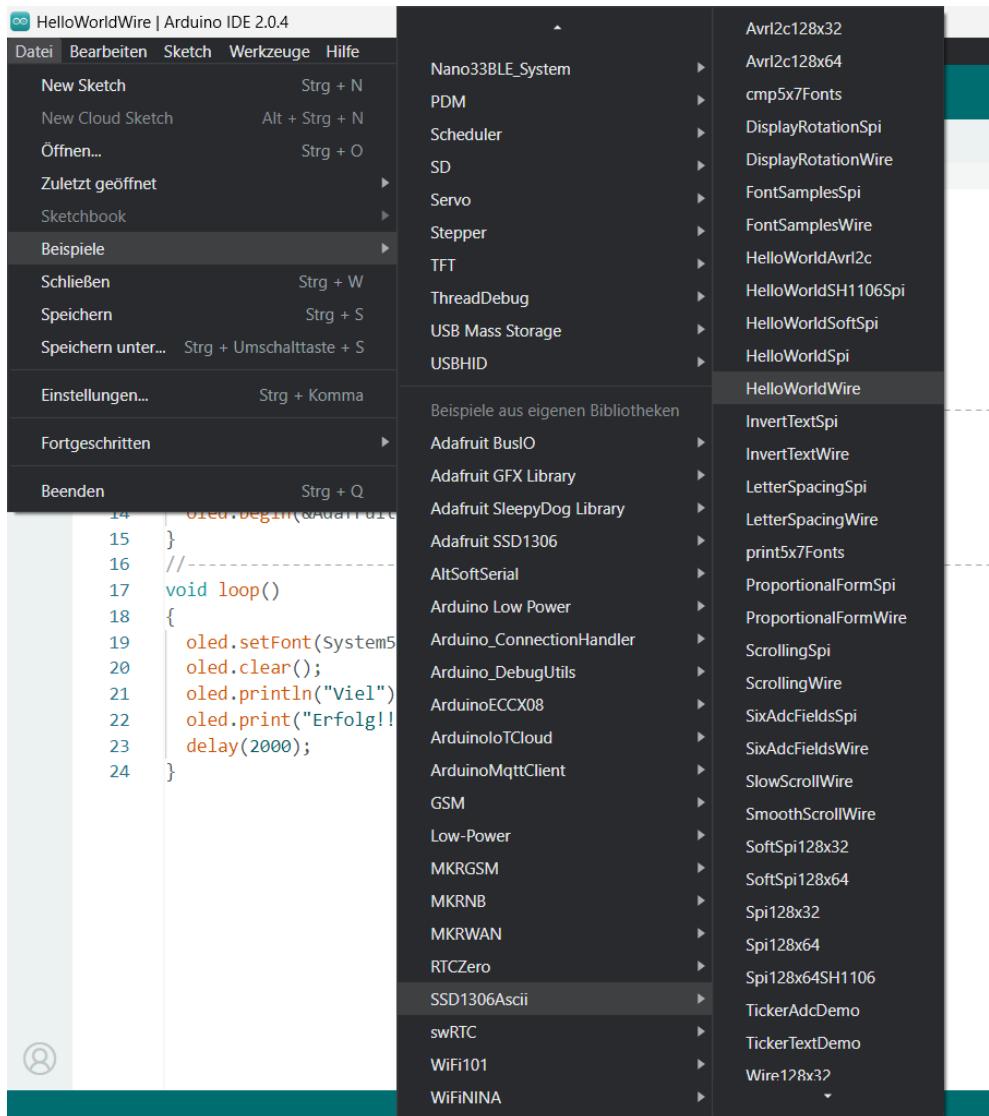


Figure 34.3.: Pfad des Testprogramms.

welcher unter `Datei -> SSD1206Ascii -> HelloWorldWire` zu finden ist (siehe Abbildung 36.9).

Das Testprogramm "Hello World" (siehe Seite 320) wird dafür benutzt, um den Text Hello World auf dem Display auszugeben (siehe Abbildung 36.10). Es wird hier verwendet, um sich mit der Software vertraut zu machen und um sicherzustellen, dass die Entwicklungsumgebung richtig eingerichtet ist. Außerdem wird getestet, ob das Programmieren funktioniert und alle Kabel richtig angesteckt sind.

Nachdem der Quellcode kompiliert und an den Arduino geschickt wurde, wurde auf dem Display der Text Viel Erfolg ausgegeben (siehe Abbildung 36.10). Hiermit ist sichergestellt worden, dass die Entwicklungsumgebung und der Compiler korrekt funktionieren.

[WS:citation](#)

```
1000 #include <Wire.h>
1001 #include <SSD1306Ascii.h>
1002 #include <SSD1306AsciiWire.h>
1003 #define I2C_ADDRESS 0x3c
1004
1005 SSD1306AsciiWire oled;
1006
1007 void setup()
1008 {
1009     Wire.begin();
1010     Wire.setClock(400000L);
1011     oled.begin(&Adafruit128x64, I2C_ADDRESS);
1012 }
1013
1014 void loop()
1015 {
1016     oled.setFont(System5x7);
1017     oled.clear();
1018     oled.println("Viel ");
1019     oled.print("Erfolg !!!");
1020     delay(2000);
1021 }
```

Listing 34.1.: Testprogramm für ein OLED-Display



Figure 34.4.: Erste Ausgabe Display

WS:citation

35. Servomotor JAMARA 033212

Elektromotoren besitzen bei geringen Drehzahlen eine geringe Kraftentfaltung. Um Platz zu sparen, werden deshalb Getriebe verwendet. Die hohe Motordrehzahl wird in ein langsames, aber hohes Drehmoment übersetzt. Dadurch bekommen Servomotoren ihre besonderen Eigenschaften: Sehr genaue Positions- und Geschwindigkeitsregelung. [Ber18]

WS:Ausführlicher!
Arduino-Programm?
Genauigkeit?

Unterteilt werden Servomotoren in zwei Kategorien: Open Loop und Closed Loop. Der Open Loop-Motor hat keine Begrenzung im Drehwinkel und kann sich um 360° frei drehen. Dieser wird auch als Schrittmotor bezeichnet. Die realisierte Anwendung zur Sonnennachführung benötigt jedoch keine volle Kreisbahn, weshalb der ausgewählte Motor in die Kategorie Closed Loop fällt. Die in der Industrie eingesetzten Servomotoren besitzen nicht zwingend ein Getriebe, sind daher deutlich größer als ein Modellbauservo. Basierend auf einem Brushless-Motor ist Magnet und Spule sehr groß ausgelegt, um die Kräfte aufzubauen.

WS:Quelle fehlt!

Bei dem verwendeten Servomotor ist das Chassis aus eingefärbtem, durchsichtigem Kunststoff gefertigt, weshalb ein Blick den Aufbau und die wesentlichen Bauteile zeigt.

Motor: Als Aktuator ist im JAMARA 033212 ein Gleichstrommotor verbaut. Da der Motor in der Baugruppe das schwerste Bauteil ist, muss je nach Anwendung aus Einsatzzweck und Realisierung durch Getriebe und Motor ein korrektes Verhältnis abgeleitet werden. In der Sonnennachführung verläuft die Hauptlast vertikal nach unten und wird dort gelagert. Der Servomotor muss also nur ein vergleichsweise geringes Drehmoment aufbringen.

Getriebe: Um bei kleinen Motoren das notwendige Drehmoment aufzubringen, wird ein mehrstufiges Getriebe mit hoher Untersetzung eingesetzt. Bei hochwertigen, teuren Servomotoren besteht das Getriebe aus Metallzahnradern und ist kugelgelagert, der vorhandene Servomotor besitzt ein kostensparendes Kunststoffgetriebe.

Positionssensor: Bei dem Servomotor ist an der Ausgangswelle ein Potentiometer angebracht. Ein Potentiometer ist ein verstellbarer Widerstand welcher als Spannungsteiler ausgelegt ist. Bei Drehung der Welle um einen bestimmten Winkel wird der Widerstand des Potentiometers verändert. Das mittlere Anschlussbein ist der Wischer, welcher am Schleifring entlang schleift.

Motorsteuerung: In der Motor Steuerung kommen Spannung, Position und externes Steuersignal zusammen. Geregelt wird abhängig von der Last die Spannung und von der externen Steuerung die Position. Unterschiede gibt es in Analog- und Digitaler Steuerung. Bei der digitalen Steuerung ist ein zusätzlicher Mikrocontroller verbaut welcher die Position exakter anfahren und halten kann. Dieser ist jedoch auch teurer, weshalb wir uns für die analoge Ausführung entschieden haben.

Die Spannung aus dem Spannungsteiler (Potentiometer) wird mit der Spannung aus dem Impulssteller verglichen. Aus dem Impulssteller kommt das veränderte Signal vom Arduino. Das Signal vom Arduino kommt über eine Signalleitung mit 50 Hz. Je nachdem wie breit das Signal ist, kann die Winkelstellung

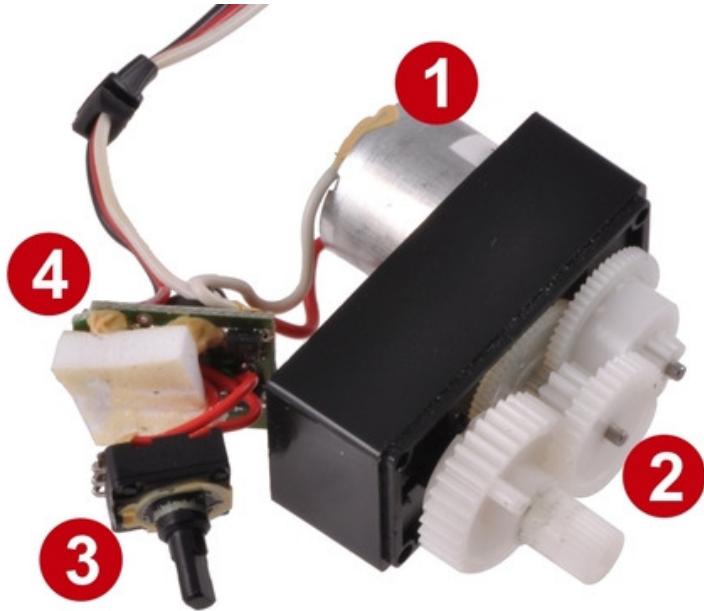


Figure 35.1.: Aufbau eines Servomotors Quelle

VS:Eigene Beschriftung
mit overpic

durch den Motor Treiber verändert werden. Die Änderung der Signalbreite wird auch als Puls Weiten Modulation bezeichnet, kurz PWM. [Dej18; Ibr18]

Dadurch ist es möglich den Servomotor mit nur einer Steuerleitung anzusteuern. Bei steigender Spannung erhöht sich das Impulssignal bei gleichbleibender Breite. Der Servomotor ändert mit höherer Spannung seine Position schneller und auch die Stellkraft steigt. Beim Halten der Position ist der Stromverbrauch sehr gering, weshalb Servomotoren auch als Verstelleinheit eingesetzt werden.

35.1. Datenblatt JAMARA 033212

Die Zusatzbezeichnung 9g bezieht sich auf das Eigengewicht, welches ungefähr 9 Gramm beträgt und die Bauklasse kennzeichnet. Zum Beispiel im Flugzeugmodellbau als Klappen- oder Fahrwerkssteuerung. Wichtig ist für die Anwendung vor allem die Stellkraft. Im Vorfeld muss die aufzubringende Kraft bekannt sein, um einen Passenden Servomotor auszuwählen. Am Datenblatt wird nochmal deutlich was bei steigender Spannung passiert. Die Kraft und die Geschwindigkeit nehmen zu. Den einfachen, günstigen Servomotor erkennt man auch am Kunststoffgetriebe und an fehlenden Kugellagern.

35.2. Schaltplan

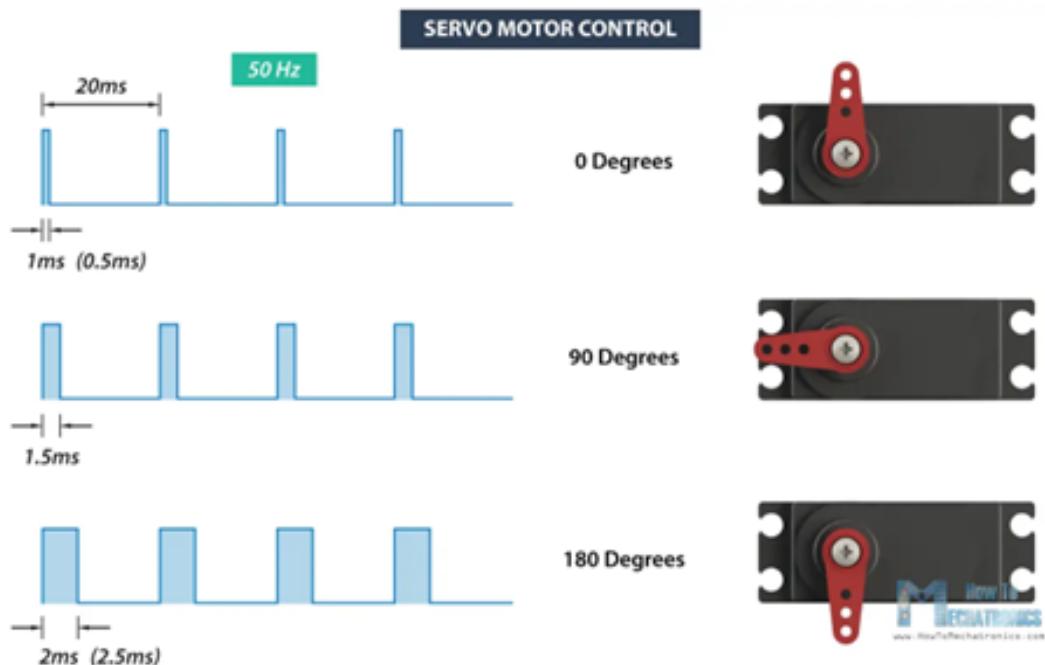


Figure 35.2.: PWM am Servomotor Quelle

WS: keine geeignete
Quelle, richtig zitieren;
eigene Zeichnung mit
tikz

Technische Daten:

Spannung	4,8V ~ 6,0 V
Stellkraft (kg/cm)	1,2 kg/cm (4,8 V) 1,4 kg/cm (6,0 V)
Stellzeit (Sek./60°)	0,11 Sek. (4,8 V) 0,09 Sek. (6,0 V)
Getriebe	Kunststoff
Kugellager	nein
Abmessungen (mm)	32 x 23 x 29,5 mm
Gewicht (g)	12 g

Figure 35.3.: Auszug aus Gebrauchsanleitung JAMARA 033212, Seite 1 [Jam]

WS: Informationen
herausziehen, eigene
Tabelle

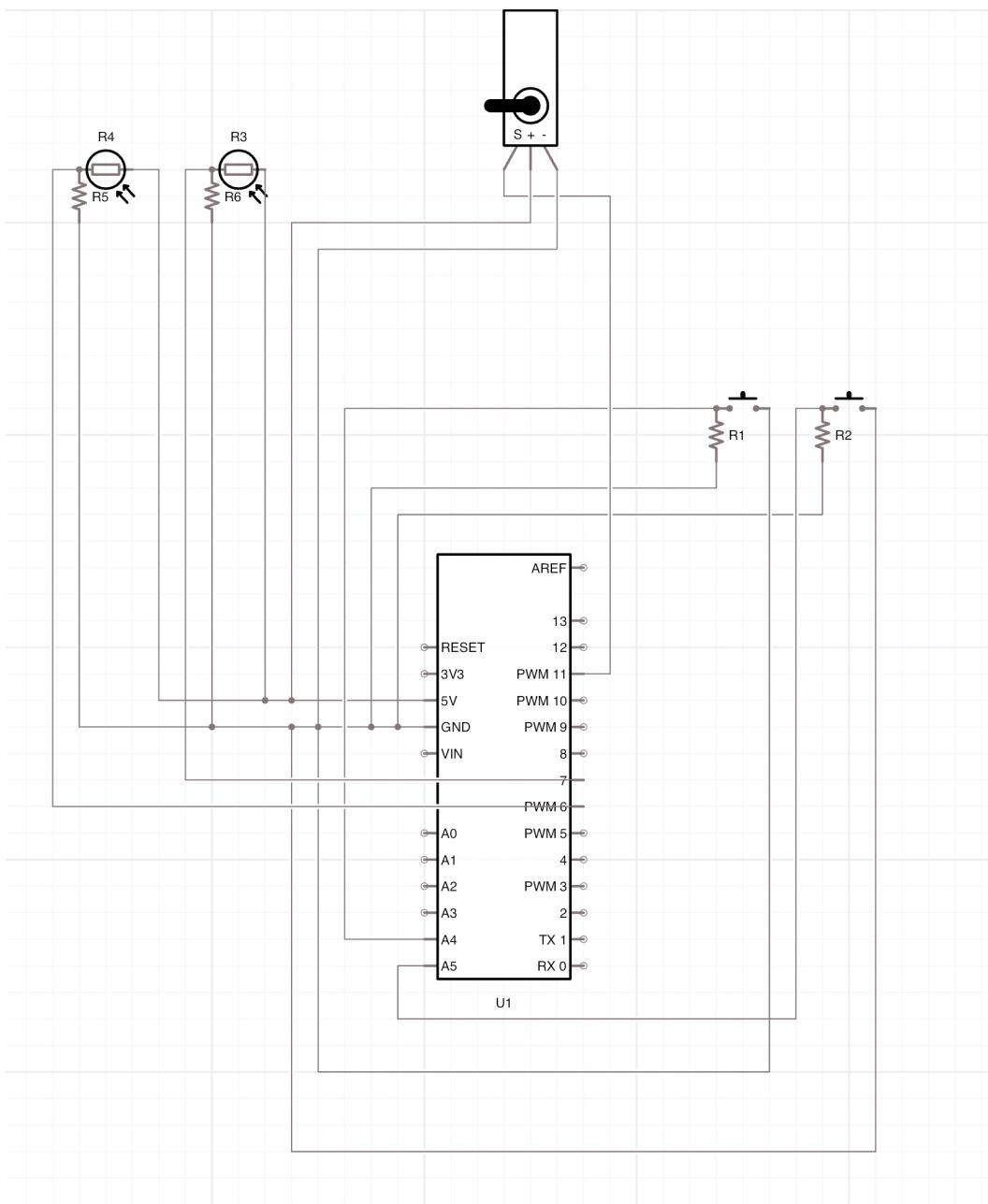


Figure 35.4.: Schaltplan zum Anschluss eines Servomotors JAMARA 033212

**WS:Wo ist das
Grove-Kabel?
Motherboard?**

36. OLED-Display

36.1. OLED

Im Folgenden wird das Display DEBO OLED2 0.96 beschrieben. Dieses kleine Display mit einem schwarzen Hintergrund und blauer Anzeigefarbe lässt sich mit I²C ansteuern. Die hohe Auflösung bietet ein scharfes Bild.[Sime]

Technische Daten:

- Auflösung: 128 × 64 Pixel
- Hintergrundfarbe: Schwarz
- Anzeigefarbe: blau
- Maße: 27 mm × 27 mm × 11 mm
- Anschluss: 4-polig Anschluss
- Interface: I²C
- SSD-Controller: SSD1306
- Spannungsversorgung: 3,3 - 5 V

WS:Wo sind die
Programmdateien?
Version der Bibliothek?
Welche Möglichkeiten?
Was ist OLED?
Funktionsbeschreibung
Grafiken? ...

36.2. Anschluss

In der Abbildung 36.1 ist einer Schaltplan zu sehen, in dem das Display verwendet wird. Die Masse des OLED-Displays ist in der Farbe schwarz gekennzeichnet und ist über das Arduino Shield an den Masse Port des Arduino verbunden. Die Spannungsversorgung mit 3,3V für das OLED-Display ist in rot gekennzeichnet. Die beiden Pins für die Datenübertragung und für das OLED-Display gehen in die vorgesehenen SDA und SCL Pins am Arduino.

WS:Schaltbild,
Anschlüsse?
Stromverbrauch?
Lebensdauer?...

WS:Farbe und Numme
WS:Nur Display
darstellen

36.3. Programmierung

36.3.1. [Wire.h](#)

Die [Wire.h](#) Bibliothek gehört nicht zu den spezielleren Bibliotheken. Trotzdem spielt sie eine wichtige funktionale Rolle, da durch sie die Verbindung zwischen dem Arduino und dem OLED-Display ermöglicht wird. [Wire.h](#) kommt immer dann zum Einsatz, wenn eine Kommunikation über I²C erfolgen soll. Die Datenübertragung mittels I²C geschieht über die zwei Anschlüsse SDA und SCL. SDA ist eine serielle Datenübertragungsleitung und SCL sendet die erforderlichen Taktimpulse. Diese beiden Leitungen bilden in Kombination mit der Spannungsversorgung, 3V3 und GND, alle benötigten Anschlüsse, um das Display mit dem Arduino zu verbinden.

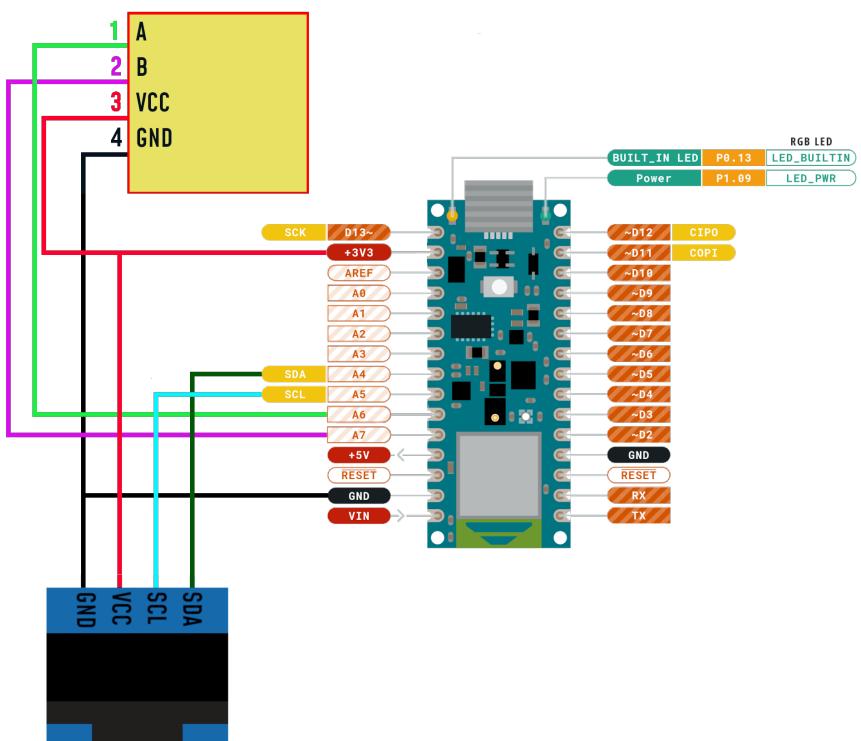


Figure 36.1.: Gesamter Schaltplan

36.3.2. OLED-Display

Das OLED-Display benötigt, wie zuvor erwähnt, die Bibliothek SSD1306Ascii. In ihr sind jegliche Funktionen implementiert, um das Display individuell einzurichten. Mithilfe der ebenfalls herunterzuladenden Beispiele, ermöglicht man dem Anwender so einen schnellen Funktionstest. So kann beispielsweise überprüft werden, ob das Display korrekt angeschlossen ist und ob die Ausgabe funktioniert.

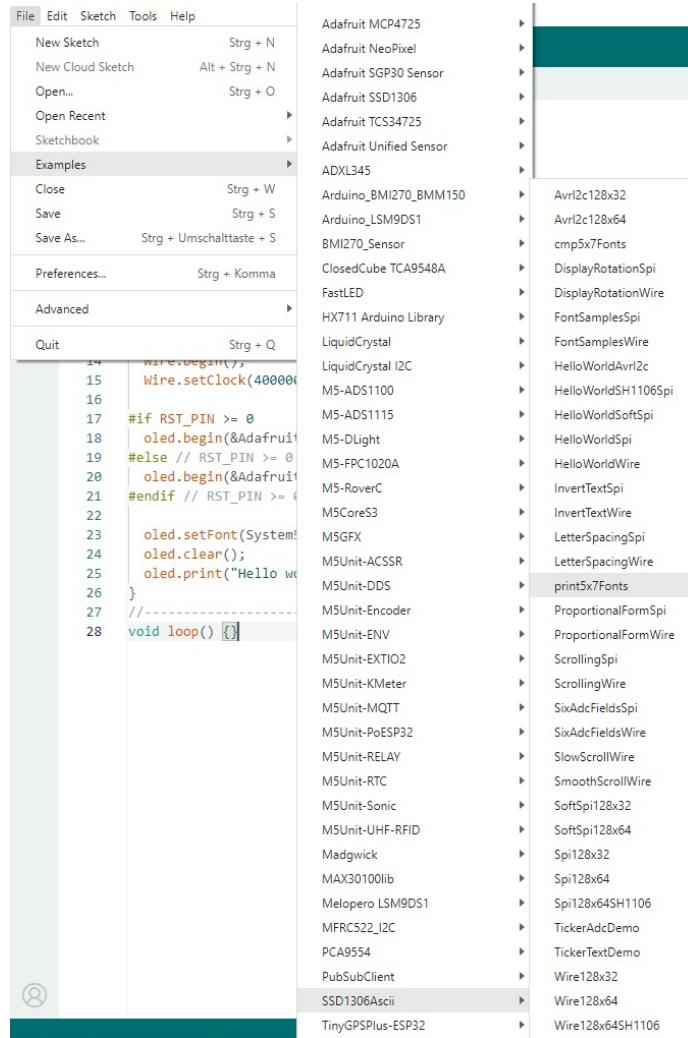


Figure 36.2.: Beispiele in der OLED Bibliothek

36.3.3. Testen des OLED-Displays

Auf dem OLED-Display wurde das Beispiel `HelloWorldWire` geladen, um die richtige Ausgabe des Displays zu gewährleisten. Wie in Abbildung 4.4 zu sehen ist, zeigt das Display die erwartete Ausgabe an.

36.4. Software

36.4.1. Verwendete Bibliotheken

Zur Ansteuerung des Displays werden folgende Bibliotheken verwendet:

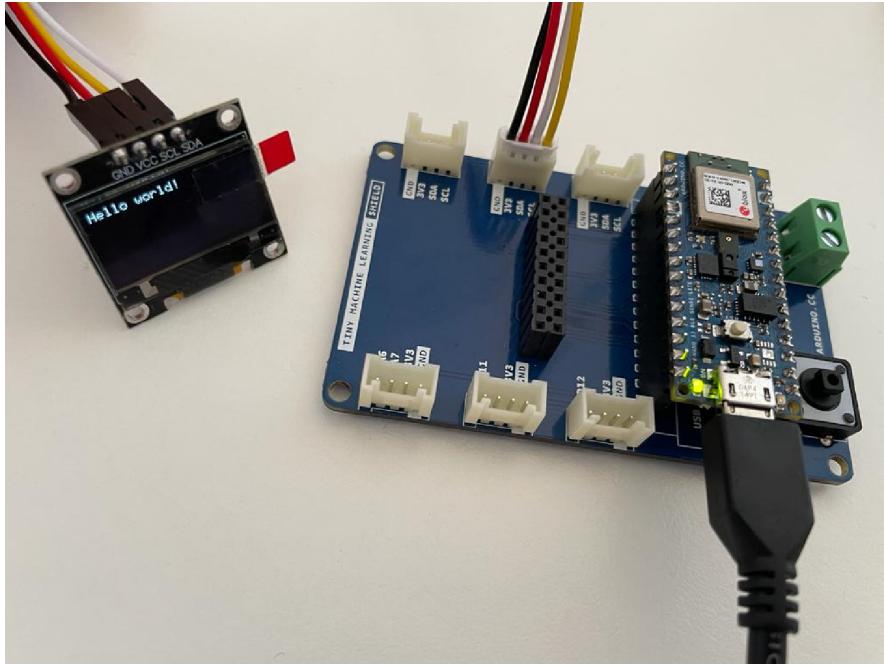


Figure 36.3.: Testausgabe des OLED Display

- [Wire.h](#): Diese Bibliothek ermöglicht die Kommunikation über den I2C-Bus, der für die Ansteuerung des OLED-Displays verwendet wird [Ardg]
- [Adafruit-GFX.h](#): Eine Grafikbibliothek, die von Adafruit entwickelt wurde und grundlegende Funktionen zur Darstellung von Text und Grafiken auf Displays bietet [Ada]
- [Adafruit-SSD1306.h](#): Eine Bibliothek für das OLED-Display SSD1306, die auf der Adafruit-GFX-Bibliothek basiert [Ardc]

VS:Ist dies kompatibel?
Was ist damit möglich?

36.4.2. OLED-Display

Ein Objekt der Klasse [Adafruit_SSD1306](#) wird erstellt, um das OLED-Display zu steuern:

```
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);
```

Das Display wird mit der Auflösung 128 Pixel × 64 Pixel initialisiert:

```
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
```

36.4.3. Initialisierung und Setup

In der `setup()`-Methode werden die erforderlichen Initialisierungen durchgeführt:

- [display.begin\(...\)](#): Initialisiert das OLED-Display
- [display.clearDisplay\(\)](#): Löscht den Displayinhalt
- [display.setTextColor\(WHITE\)](#): Setzt die Textfarbe auf Weiß
- [display.setTextSize\(2\)](#): Setzt die Textgröße auf 2

- `pinMode(buttonPin, INPUT_PULLUP)`: Konfiguriert den Taster-Pin als Eingang mit Pull-Up-Widerstand
- `Serial.begin(9600)`: Initialisiert die serielle Kommunikation mit einer Baudrate von 9600
- `PDM.onReceive(onPDMdata)`: Registriert den Empfang des Mikrofonsignals als PDM-Signal

36.4.4. Messungssteuerung

Die Funktion `onPDMdata()` wird aufgerufen, wenn Daten vom Mikrofon verfügbar sind. Dies liest die Daten in einen Array ein und zählt die Anzahl der gelesenen Samples:

```
61 void onPDMdata() {
62     int bytesAvailable = PDM.available();
63     PDM.read(sampleBuffer, bytesAvailable);
64     samplesRead = bytesAvailable / 2;
65 }
```

Figure 36.4.: Code Einlesen/Zählen

Die Hauptfunktion `loop()` enthält zwei Funktionen für die Messungssteuerung und die Benutzeroberfläche:

```
67 void loop() {
68     HandleInput();
69     HandleUI();
70 }
```

Figure 36.5.: Code Messungssteuerung

Die Funktion `HandleInput()` überwacht den Zustand der beiden Taster und steuert somit den Messungsablauf:

Listing 36.1.: Monitoring

- Zunächst wird der Zustand der beiden Taster überprüft und die Abfrage nach einem gedrückten Button ist aktiv (`blueButtonIsPressed`, `redButtonIsPressed`).
- Wenn der blaue Taster gedrückt wird, wird die Messung gestartet, indem `isMeasuring` auf `true` gesetzt wird und die Funktion `StartSampling()` aufgerufen wird.
- Wenn der rote Taster gedrückt wird, wird die Messung beendet, indem `isMeasuring` auf `false` gesetzt wird und die Funktion `StopSampling()` aufgerufen wird. Je nach vorherigem Zustand des roten Tasters wird entweder der maximale Wert SPL in dB angezeigt (`ShowMaxdBspl()`) oder der durchschnittliche dB SPL-Wert (`ShowAveragedBspl()`).
- `isDelayOver` wird auf `true` gesetzt, wenn die Startverzögerung (definiert durch `measureThresholdMilliseconds`) von 1200ms abgelaufen ist.
- Es gibt eine kurze Verzögerung (`delay(50)`) zwischen den Schleifendurchläufen, um Tastenstellungen zu vermeiden.

36.4.5. Mikrofondatenverarbeitung

Die Funktion `StartSampling()` initialisiert die Datenverarbeitung:

Listing 36.2.: Start and stop sampling

`PDM.begin(1, 16000)`: Initialisiert eine Abtastrate von 16.000 Hz. Die Funktion `StopSampling()` beendet die Aufnahme von Audiodaten.

36.4.6. Anzeige der Ergebnisse

Die Funktionen `ShowResult()` und `ShowMicrophoneValues()` sind für die Anzeige der Messergebnisse auf dem OLED-Display verantwortlich. `ShowResult()` zeigt den aktuellen SPL-Wert in dB auf dem Display an und aktualisiert den maximalen Wert SPL in dB, wenn nach der Startverzögerung ein neuer Höchstwert erreicht wird. `ShowMicrophoneValues()` verarbeitet die vom Mikrofon empfangenen Signale. Der maximale Samplewert wird ermittelt und verwendet, um den Wert SPL in dB zu berechnen. Die Funktion berechnet auch einen Durchschnittswert über eine bestimmte Zeit, `averagingTime = 200ms`, und zeigt das Ergebnis auf dem Display an. Die Funktionen `ShowMaxdBspl()` und `ShowAveragedBspl()` zeigen den maximalen bzw. den durchschnittlichen Wert SPL in dB auf dem Display an.

36.4.7. Umrechnung auf den Wert dB SPL

Die Umrechnung des Mikrofonsignals auf den dB SPL-Wert erfolgt in der Funktion `getDbValueFromPMC()`:

Listing 36.3.: Conversion of the microphone signal

`MaxSampleVoltage` wird berechnet, indem der maximale Samplewert `maxSampleValue` mit der Referenzspannung des Mikrocontrollers („Vref = 3,3 V“) multipliziert und durch den maximalen Wert eines 16-Bit-Signed-Integers (32767) dividiert wird. 32767 ist der maximale Wert eines 16-Bit-Signed-Integers, der der maximalen Amplitude des Audiosignals entspricht. Der dB SPL-Wert („dBspl“) wird mit der Formel „ $20 * \log_{10}(\text{Voltage} / \text{Vrms}) + \text{sensitivity}$ “ berechnet [Quelle der Fomel: PDF Decibels Formula https://physicscourses.colorado.edu/phys3330/phys3330_sp19/resources/Decibels_Phys_3330.pdf University of Colorado System], wobei „Voltage“ der berechnete „maxSampleVoltage“ ist, und „sensitivity“ die Empfindlichkeit des Mikrofons in dBV (Dezibel-Volt) ist. „Vrms“ ist die RMS-Spannung (Root Mean Square), die einem Schalldruckpegel von 94 dB SPL entspricht. Dieser Wert wurde experimentell ermittelt.

36.4.8. Benutzeroberfläche

Die Funktion `HandleUI()` aktualisiert OLED-Display Anzeige:

Listing 36.4.: OLED-Aktualisierung

Wenn eine Messung aktiv ist („`isMeasuring == true`“), werden die aktuellen Messwerte auf dem Display angezeigt. Andernfalls wird der Displayinhalt aktualisiert, ohne neue Werte anzuzeigen.

36.5. Das OLED-Display SSD1306

Das OLED-Display SSD1306 dient dazu, die Messwerte des Arduinos auszugeben. Es besitzt eine Maße von ca. 27 x 27 x 4,1 mm und ist durch seinen hohen Kontrast sehr gut lesbar. Das Display besteht aus 128x64 OLED Bildpunkten, die durch den Chip SSD1306 gesteuert werden können. Das Display benötigt eine Betriebsspannung von 3,3 V bis 5 V und hat einen Stromverbrauch von 0,04 W im normalen Betrieb. Die

```

216 void HandleUI() {
217   if (isMeasuring) {
218     ShowMicrophoneValues();
219   } else {
220     //display.clearDisplay();
221     display.display();
222   }
223 }
```

Figure 36.6.: Code

Betriebstemperatur von -30 °C bis +80 °C sollte dabei nicht überschritten werden. Angesteuert werden kann das Display über die Schnittstelle I2C mit den Pins VCC, GND, SCL und SDA. Mit Hilfe der beiden Bibliotheken Adafruit GFX und Adafruit SSD1306 kann das Display programmiert werden. Hierzu aber mehr in dem Kapitel

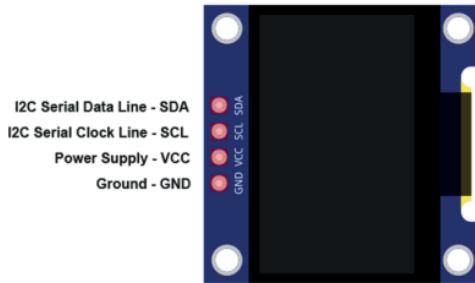


Figure 36.7.: Pins des OLED-Displays.

Das Display verfügt über vier Pins, welche in der Abbildung 36.7 zu sehen sind. Der VCC-Pin, der für die Spannungsversorgung des Geräts sorgt, wird mit dem 5V-Pin des Mikrocontrollers verbunden. Der GND-Pin des Geräts wird mit dem GND-Pin des Mikrocontrollers verbunden, um eine gemeinsame Masseverbindung herzustellen. Der SDA-Pin des Geräts muss entweder mit dem speziellen SDA-Pin oberhalb des Pin 13 oder mit dem analogen Pin A4 des Mikrocontrollers verbunden werden. Der SCL-Pin des Geräts wird entweder mit dem speziellen SCL-Pin oberhalb des Pin 13 oder alternativ mit dem analogen Pin A5 des Mikrocontrollers verbunden. [Az a; Funb]

36.6. Schaltplan des Aufbaus

Der folgende Schaltplan stellt die Komponenten des Arduino Nano BLE Sense Lite, des Sensors BME280 und des OLED-Displays dar. Als Spannungsquelle dient ein Computer, der den Arduino Nano 33 BLE Sense über ein USB-A auf Mikro-USB Kabel versorgt. Die Komponenten sind sinngemäß miteinander verbunden und in Abbildung 36.8 abgebildet.

36.7. Genutzte Bibliotheken

Bei Bibliotheken handelt es sich um Ansammlungen von dem Code und Funktionen für bestimmte Anwendungen oder Hardware. Diese werden oft genutzt um Aufgaben

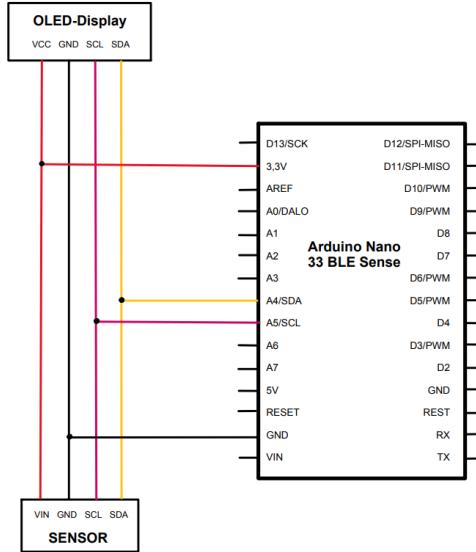


Figure 36.8.: Stationärer Aufbau der Wetterstation
[Arda]

leichter lösen zu können und ein neu schreiben von komplexen Funktionen zu vermeiden. In unserem Fall verwenden wir Bibliotheken für den Arduino, den Sensor BME280 und das OLED-Display.

36.7.1. Bibliothek Wire.h

Die Bibliothek Wire.h ist eine Standardbibliothek für Arduino-Plattformen, welche Funktionen und Methoden für die Kommunikation zur Verfügung stellt. Mit Hilfe der Schnittstelle I2C ermöglicht die Bibliothek die Kommunikation zwischen einem Arduino und einem anderen I2C-fähigen Gerät wie z.B. Sensoren oder Displays. I2C ist ein Kommunikationsbus, der im Vergleich zu seriellen Schnittstellen den Vorteil hat, dass er mit mehr als zwei Geräten kommunizieren kann. Ein I2C-Bus benötigt zwei Leitungen: SCL für ein Taktsignal und SDA für Daten. Um die Wire.h Bibliotek anwenden zu können, muss sie am Anfang des Sketchs eingebunden werden: `#include<...>` [W3c]

36.7.2. Bibliothek SSD1306Ascii.h für das Testprogramm

Bei der Bibliothek SSD1306Ascii.h handelt es sich um eine benutzerdefinierte Bibliothek, die ähnlich zu der Adafruit Bibliothek SSD1306 ist. Beide sind für die Ansteuerung von OLED-Displays notwendig und basieren auf den Controller-Chip SSD1306. Sie ermöglichen das einfache Schreiben von Text, Zeichen von Formen und Anzeigen von Bitmap-Bildern auf dem Display. Um die Bibliothek SSD1306Ascii.h zu verwenden, muss sie erst als ZIP-Datei heruntergeladen und anschließend von dem Arduino Library Manager installiert werden. Mit Hilfe von `#include<SSD1306Ascii.h>` kann man sie in dem Arduino-Code einbinden. [FunA]

36.7.3. Testen des OLED-Displays

Für die Programmierung des Displays gibt es viele Möglichkeiten. Da es viele Libraries gibt, muss zuerst einmal überlegt werden, welche verwendet werden sollen. Um einen ersten Eindruck über die Programmierung des Displays zu bekommen, wurde zunächst ein Beispieldsketch getestet. Es handelt sich hier um den Sketch [Hello World](#), welcher unter Datei -> SSD1206Ascii -> HelloWorldWire zu finden ist (siehe Abbildung36.9).

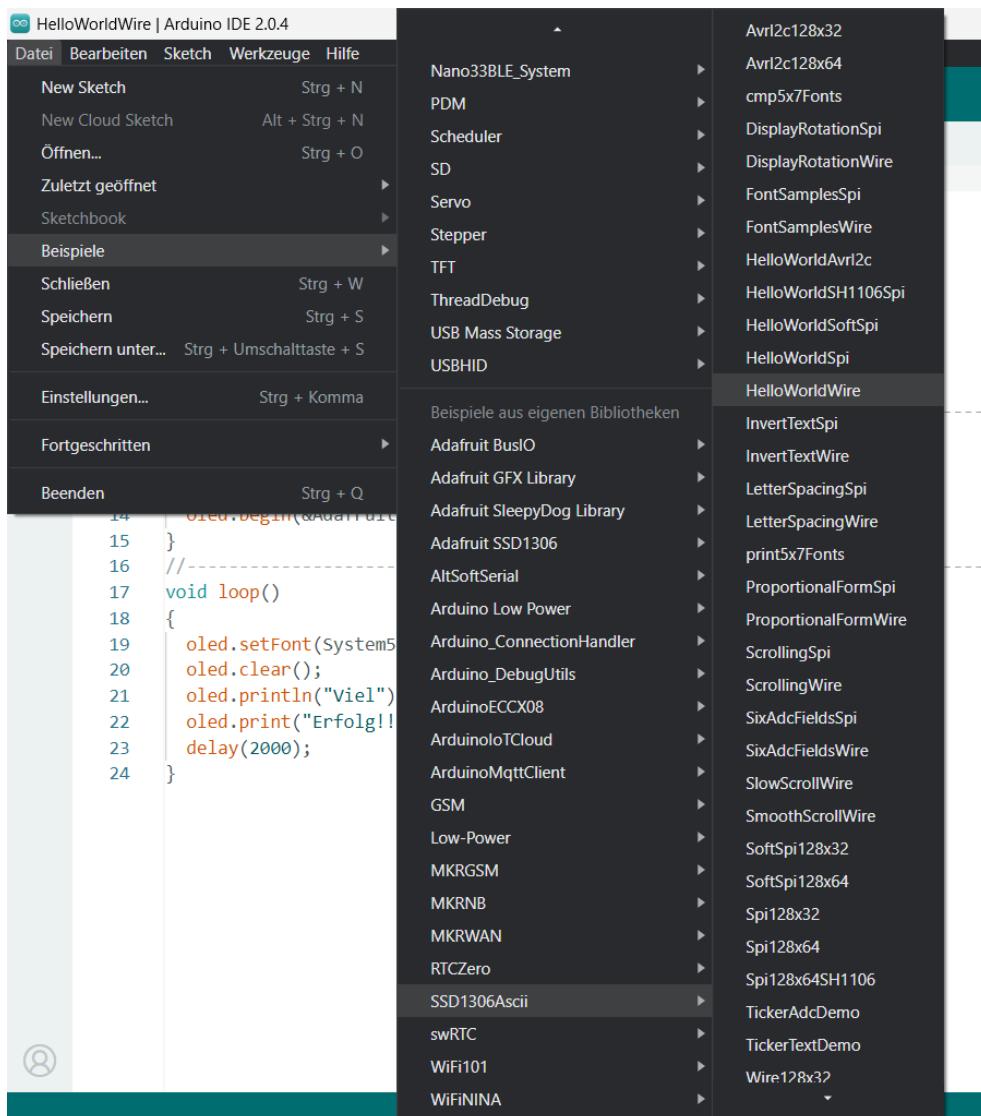


Figure 36.9.: Pfad des Testprogramms.



Figure 36.10.: Erste Ausgabe Display
[Funb]

Das Testprogramm [Hello World](#) (siehe Seite 320) wird dafür benutzt, um den Text Hello World auf dem Display auszugeben (siehe Abbildung 36.10). Es wird hier verwendet, um sich mit der Software vertraut zu machen und um sicherzustellen, dass die Entwicklungsumgebung richtig eingerichtet ist. Außerdem wird getestet, ob das Programmieren funktioniert und alle Kabel richtig angesteckt sind.

Listing 36.5.: Simple program for OLED displays

```
..../Code/Arduino/OLED/Grove -OLED Display SSD1308/TestSSD1306.ino
```

Nachdem der Quellcode kompiliert und an den Arduino geschickt wurde, wurde auf dem Display der Text Viel Erfolg ausgegeben (siehe Abbildung 36.10). Hiermit ist sichergestellt worden, dass die Entwicklungsumgebung und der Compiler korrekt funktionieren. [Funa]

36.8. Beispiel eines OLED-Displays

Zur Ausgabe unserer Messwerte verwenden wir ein 0,96 Zoll OLED Display, welches über den I²C Port mit dem Tiny Machine Learning Shield verbunden ist. Es verfügt über die Anschlüsse SDA und SCL zur Datenübertragung, über VCC zur Spannungsversorgung und GND für die Masse. Außerdem ist es mit einem internen Spannungsregler ausgestattet, der 3,3V- und 5V-Betrieb ermöglicht.

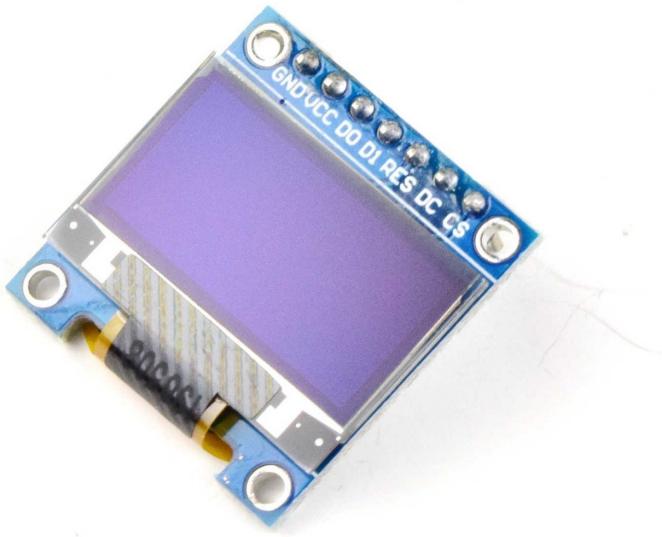


Figure 36.11.: Das Display zur Ausgabe der Messwerte

Quelle: [Arduino - Display 0,96", Grove OLED-Display, SSD](#)

WS:Quelle fehlt, Infos,

...

Allgemeines			
Typ	OLED		
Ausführung	Zubehör	Display	
Farbe	monochrom		
Display	bis 1,9 Zoll		
Diagonale	1,0 Zoll		
Auflösung, physik.	128 x 64 Pixel		
Diagonale	2,44 cm		
Ausführung	Modell	Arduino	Besonderheiten
Besonderheit	-		
Sonstiges	Spezifikation	SSD1308	
Herstellerangaben			
Hersteller		SEEED	
Artikelnummer des Herstellers		104030008	
Verpackungsgewicht		0.01 kg	

36.8.1. OLED

Als weiteren Punkt werden Klassenobjekte im Header initialisiert und Adressen von Peripheriegeräten festgelegt. In diesem Fall sind es das Objekt *oled* der Klasse *SSD1306AsciiWire* und die Definition der Adresse des OLED-Displays. Die Adresse wurde zuvor durch einen I²-Scanner bestätigt.

```
1000 #define I2C_ADDRESS 0x3C
      SSD1306AsciiWire oled;
```

Ein weiterer Punkt ist die Initialisierung und Deklarierung von globalen Variablen. Beim Farberkennungsautomaten sind es zwei Variablen. Über Pin D11, kurz 11, wird der Messtaster abgefragt. Pin D12, kurz 12, gibt bei Bedarf ein Signal an die externe RGB-LED. Durch das Voranstellen von *const* werden beide Variablen zu Konstanten

und könne außer über den Source-Code nicht geändert werden. Dies ergibt Sinn, da die Verkabelung sich nicht verändern wird und somit auch nicht die gewählten Pins.

```
1000 const int TasterPin = 11;
const int LedPin = 12;
```

void setup{}

In der *setup()*-Funktion wird zuerst die serielle Kommunikation mit einer Baudrate von 9600 bit/s gestartet. Baudrate oder auch Bitrate beschreibt die Übertragungsdauer eines Bits. Bei einer Baudrate von 9600 dauert es 104,17 µs um ein Bit zu übertragen. Je höher die Baudrate ist, desto schneller wird ein Bit übertragen. Der Empfänger tastet meist mehrmals pro gesendetem Bit ab und bildet dann nach dreimaligem Abtasten den Mittelwert, welcher dann als empfangenes Bit weiterverarbeitet wird [GW22]. Nach dem Starten der seriellen Kommunikation werden die Modi der zwei genutzten Pins definiert.

```
1000 Serial.begin(9600);
pinMode(buttonPin, INPUT);
1002 pinMode(ledPin, OUTPUT);
```

Manche Pins können als Input oder als Output genutzt werden. Im Fall des Arduino Nano 33 BLE Sense Lite sind die Pins D13, AREF, A0-A7, TX, RX und D2-D12 als General Purpose Input Output (GPIO) nutzbar. Über das genutzte Shield kann auf A6, A7, D11 und D12 zugegriffen werden. Genutzt werden D11 und D12. Der Tasterpin D11 wird als Input definiert um das Signal aufnehmen zu können, welches durch Drücken des Tasters ausgelöst wird. Pin D12 wird als Output definiert um die externe Rot-Grün-Blau (RGB)-LED einzuschalten. Der Befehl *pinMode()* beinhaltet zusätzlich noch die Möglichkeit einen internen Pull-Up-Widerstand einzuschalten [Ardk].

Anschließend wird das I²C-Protokoll mit dem Objekt *Wire* und der Funktion *begin()* gestartet. Dabei wird der Arduino als Master im I²C-Protokoll angemeldet [Ardg]. Anschließend wird die Taktfrequenz mit 400kHz festgelegt [Ardh].

```
Wire.begin();
Wire.setClock(400000L);
```

Im nächsten Schritt wird das OLED-Display gestartet. Hierfür wird mit dem Objekt *oled* die Funktion *begin()* aufgerufen. Dieser Befehl benötigt als Funktionsparameter eine Geräteinitialisierung und die Adresse des Displays. Für die Initialisierungsphase des Farberkennungsautomaten wird ein der Text *INITIALISIERUNG!* auf dem Display ausgegeben. Hierfür wird die Schriftart und die Startposition des Textes auf dem Display festgelegt [Ardc].

```
oled.begin(&Adafruit128x64, I2C_ADDRESS);
oled.setFont(System5x7);
oled.setCursor(0, 40);
oled.println("INITIALISIERUNG!\n");
```

Der letzte Schritt in der *setup()*-Funktion ist das dreimalige Blinken der externen RGB-LED. Hierfür wird eine for-Schleife genutzt. Innerhalb dieser Schleife wird die LED nach 2 s für 0,2 ms eingeschaltet und danach wieder ausgeschaltet. Anschließend wird der Bildschirm gelöscht.

```
for (int zaehler = 1; zaehler < 4; zaehler = zaehler + 1) {
    delay(2000);
```

```
    digitalWrite(LedPin, HIGH);
    delay(200);
    digitalWrite(LedPin, LOW);
    delay(200);
}
oled.clear();
```


37. Kamera-Modul ArduCAM OV2640

37.1. OV7675 Camera Module

- 0.3 MP CMOS image sensor
- active array size: 640×480
- output formats: YUV422, Raw RGB, ITU656, RGB565
- input clock frequency: $1.5 \approx 27$ MHz
- maximum image transfer rate: VGA 30fps, QVGA 60fps, QQVGA 240fps
- pixel size: $2.5 \mu m \times 2.5\mu m$
- image area: $1640 \mu m \times 1220\mu m$
- <https://www.arducam.com/products/camera-breakout-board/0-3mp-ov7675/>
- https://github.com/ArduCAM/ArduCAM_USB_Camera_Shield
- <https://github.com/ArduCAM/Arduino>

37.2. Indroduction

ArduCAM is Arduino based open source camera platform which is well mated to Arduino boards. It is a high definition 2MP SPI camera, which reduce the complexity of the camera control interface. It integrates 2MP CMOS image sensor OV2640, and provides miniature size, as well as the easy to use hardware interface and open source code library. The ArduCAM mini can be used in any platforms like Arduino, Raspberry Pi, Maple, Chipkit, Beaglebone black, as long as they have SPI and I2C interface and can be well mated with standard Arduino boards. The figure ?? shows the mini ArduCAM Camera. The mini ArduCAM OV2640 is well suited for tinyML application, it is easy to configure with Arduino boards. For making the ML applications, especially Images caputuring, object and gesture detection, it supports to take capture and send back to Arduino microcontroller for getting desire results. [Arducam]

<https://www.arducam.com/ov2640/>

<https://www.arducam.com/focal-length-calculator/>

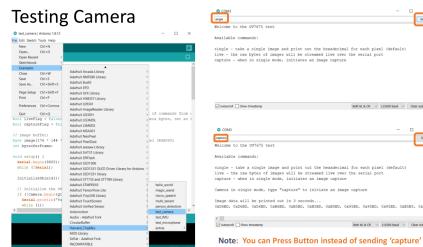


Figure 37.1.: Test program

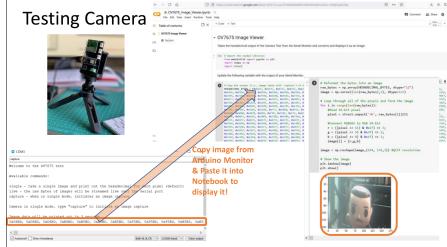


Figure 37.2.: Test program

37.3. Produktbeschreibung

Arducam-M-2MP ist eine optimierte Version von Arducam Shield Rev.C und ist eine hochauflösende 2MP-SPI-Kamera, die die Komplexität der Kamerasteuerung verringert. Sie verfügt über einen 2-MP-CMOS-Bildsensor OV2640 und hat eine Miniaturgröße sowie eine einfach zu bedienende Hardware-Schnittstelle und die Open-Source-Code-Bibliothek. Die Arducam Mini-Kamera kann auf allen Plattformen wie Arduino, Raspberry Pi, Maple, Chipkit, Beaglebone Black verwendet werden, solange sie über eine SPI- und I2C-Schnittstelle verfügen und mit Standard-Arduino Boards verbunden werden können. Die Arducam Mini-Kamera bietet nicht nur die Möglichkeit, eine Kamera-Schnittstelle hinzuzufügen, die in einigen Mikrocontrollern nicht vorhanden ist, sondern bietet auch die Möglichkeit, mehrere Kameras zu einem einzigen Mikrocontroller hinzuzufügen.

Anwendung:

- IoT-Kameras.
- Roboterkameras.
- Wildlife-Kameras.

Andere batteriebetriebene Produkte. Kann auf Plattformen wie MCU, Raspberry Pi, ARM, DSP, FPGA verwendet werden.

Eigenschaften:

- 2-Megapixel-Bildsensor OV2640.
- M12-Mount- oder CS-Mount-Objektivhalter mit wechselbaren Objektivoptionen.
- IR-empfindlich mit entsprechender Objektivkombination.
- I2C-Schnittstelle für die Sensorkonfiguration.
- SPI-Schnittstelle für Kamera-Befehle und Datenstrom.
- Alle E/A-Anschlüsse sind für 5 V/3,3 V geeignet.
- Unterstützt JPEG-Komprimierungsmodus, Einzel- und Mehrfachaufnahmemodus, einmaliges Erfassen mehrerer Lesevorgänge, Burst-Lese-Operation, Niedrige-Energie-Modus usw..
- Kann mit Standard-Arduino-Boards verbunden werden.
- Open-Source-Code-Bibliothek für Arduino, STM32, Chipkit, Raspberry Pi, BeagleBone Black.

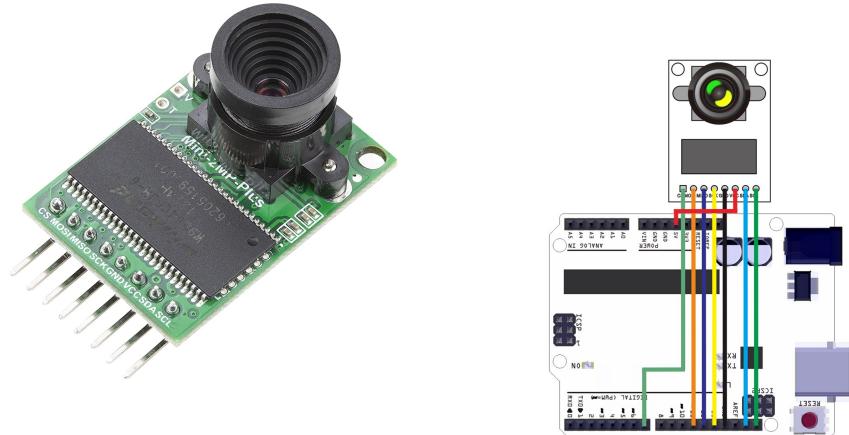


Figure 37.3.: Kamera IMX477 der Firma Arducam; [Ard21]

- Schlanke Form.

Lieferumfang:

1 x Arducam Mini-Modul Kameraschutz mit OV2640 2 MP, Objektiv, für Arduino UNO Mega2560 Board.

Hinweis: Arduino UNO ist nicht enthalten.

https://www.amazon.com/dp/B07D58GDDV/ref=sr_1_17_sspa?__mk_de_DE=%C3%85%C3%A5%C3%96%C3%84%C3%85&dchild=1&keywords=arducam&qid=1622358684&sr=8-17-spons&psc=1&spLa=ZW5jcnlwdGVkUXVhbGlmaWVyPUEyWUdNSVhJSFNUQUtLJmVuY3J5cHRlZE1kPUEwNjI4NT

37.3.1. Pin Configuration of Arducam OV2640 2MP Mini

Arducam Mini 2MP OV2640 is a small mini size camera, we can easily embed this camera with any kind of Arduino or other electronics boards, if they have the serial peripheral interface (SPI) and chip select (CS) . It has has 8 pins, the following figure 37.4 shows the functionality of each pin.

Pin No.	PIN NAME	TYPE	DESCRIPTION
1	CS	Input	SPI slave chip select input
2	MOSI	Input	SPI master output slave input
3	MISO	Output	SPI master input slave output
4	SCLK	Input	SPI serial clock
5	GND	Ground	Power ground
6	+5V	POWER	5V Power supply
7	SDA	Bi-directional	Two-Wire Serial Interface Data I/O
8	SCL	Input	Two-Wire Serial Interface Clock

Figure 37.4.: ArduCAM Pin Config

It offers to add a camera interface with microcontroller the one who dont have camera capability, also there is a option to add multiple cameras with microcontroller.

37.4. ArduCAM Interface with Arduino

ArduCAM OV2640 needs the SPI and I2C connection with the arduino boards. It will be connecting until these two connections are make sure. The figure 37.5 shows the ArduCAM connection with Arduino Mega 2560, the same connection will need with the other arduino boards too until the availability of SPI and I2C connection. These ArduCAM cameras are easy to configure with arduino and depends upon the application, it is possible to connect multiple camera with single board to make the edge computing application. Arducam Interface

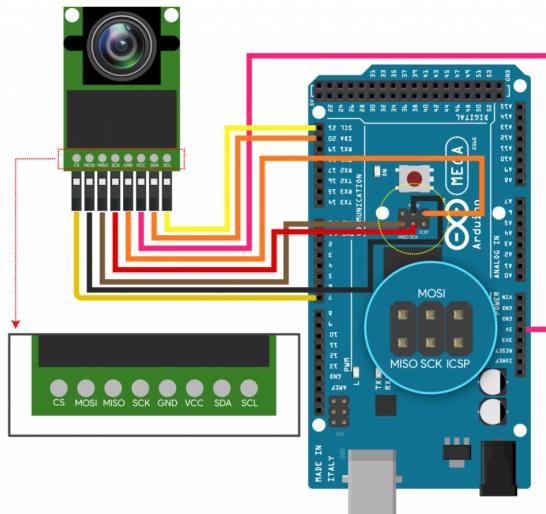


Figure 37.5.: ArduCAM Interface with Arduin Mega 2560

ArduCAM is very small in size, even it is possible to fix the camera on the Arduino board. There is no external battery require for ArduCam to operate, It needs 5V/70mA operating power supply, so it will get the power from the arduino board too. By having the innovative functionality with arduino boards, this can be used in the following applications.

37.5. ArduCAM Library Introduction

<https://github.com/ArduCAM/Arduino>

Dies ist eine Open-Source-Bibliothek für die Aufnahme von hochauflösenden Standbildern und kurzen Videoclips auf Arduino-basierten Plattformen unter Verwendung der Kameramodule von ArduCAM. Die Kamera-Breakout-Boards sollten vor dem Anschluss an die Arduino-Bretter mit dem ArduCAM-Shield funktionieren. ArduCAM-Kameramodule der Mini-Serie wie Mini-2MP, Mini-5MP(Plus) können direkt an Arduino-Brettern angeschlossen werden. Zusätzlich zu Arduino kann die Bibliothek auf beliebige Hardware-Plattformen portiert werden, solange sie über eine I2C- und SPI-Schnittstelle verfügen, die auf dieser ArduCAM-Bibliothek basiert.

Now Supported Cameras

- OV7660 0.3MP
- OV7670 0.3MP
- OV7675 0.3MP
- OV7725 0.3MP

- MT9V111 0.3MP
- MT9M112 1.3MP
- MT9M001 1.3MP
- MT9D111 2MP
- OV2640 2MP JPEG
- MT9T112 3MP
- OV3640 3MP
- OV5642 5MP JPEG
- OV5640 5MP JPEG

Supported MCU Platform

Theoretically support all Arduino families

- Arduino UNO R3 (Tested)
- Arduino MEGA2560 R3 (Tested)
- Arduino Leonardo R3 (Tested)
- Arduino Nano (Tested)
- Arduino DUE (Tested)
- Arduino Genuino 101 (Tested)
- Raspberry Pi (Tested)
- ESP8266-12 (Tested) (http://www.arducam.com/downloads/ESP8266_UNO/package_ArduCAM_index.json)
- Feather M0 (Tested with OV5642)

Note: ArduCAM library for ESP8266 is maintained in another repository `ESP8266` using a json board manager script.

37.6. Libraries Structure

Die Basisbibliotheken bestehen aus zwei Unterbibliotheken: [ArduCAM](#) und [UTFT4ArduCAM_SPI](#). Diese beiden Bibliotheken sollten direkt unter die Bibliotheken des Arduino-Verzeichnisses kopiert werden, damit sie von der Arduino-IDE erkannt werden.

Die ArduCAM-Bibliothek ist die Kernbibliothek für ArduCAM-Shields. Sie enthält unterstützte Bildsensortreiber und Benutzerland-API-Funktionen, die Befehle zum Erfassen oder Lesen von Bilddaten erteilen. Es gibt auch ein Beispielverzeichnis innerhalb der ArduCAM-Bibliothek, das die meisten Funktionen der ArduCAM-Shields illustriert. Die vorhandenen Beispiele sind Plug-and-Play, ohne dass eine einzige Zeile Code geschrieben werden muss.

Die Bibliothek [UTFT4ArduCAM_SPI](#) ist eine modifizierte Version von UTFT, die von Henning Karlsen geschrieben wurde. Wir haben sie portiert, um das ArduCAM-Shield mit LCD-Bildschirm zu unterstützen. Daher wird die Bibliothek [UTFT4ArduCAM_SPI](#) nur benötigt, wenn das ArduCAM-LF-Modell verwendet wird.

37.7. How to use

Die Bibliotheken sollten vor dem Ausführen von Beispielen konfiguriert werden, andernfalls erhalten Sie eine Fehlermeldung beim Kompilieren.

37.7.1. Edit `memoriesaver.h` file

Öffnen Sie die Datei `memoriesaver.h` im ArduCAM-Ordner und aktivieren Sie die Hardwareplattform und das Kameramodul, das zu Ihrer Hardware passt, indem Sie die Makrodefinition in der Datei auskommentieren oder auskommentieren. Wenn Sie zum Beispiel eine ArduCAM-Mini-2MP haben, sollten Sie die Zeile `#define OV2640_MINI_2MP` auskommentieren und alle anderen Zeilen auskommentieren. Und wenn Sie ein ArduCAM-Shield-V2 und ein OV5642-Kameramodul haben, sollten Sie die Zeile `#define ARDUCAM_SHIELD_V2` und die Zeile `#define OV5642_CAM` auskommentieren und dann alle anderen Zeilen.

37.7.2. Choose correct CS pin for your camera

Öffnen Sie eines der Beispiele und verdrahten Sie die SPI- und I2C-Schnittstelle, insbesondere die CS-Pins, entsprechend den Beispielen mit dem ArduCAM-Shield. Hardware und Software sollten konsistent sein, um die Beispiele korrekt auszuführen.

37.7.3. Upload the examples

Im Beispielordner befinden sich sieben Unterverzeichnisse für verschiedene ArduCAM-Modelle und die Host-Anwendung. Der Ordner Mini ist für die Module ArduCAM-Mini-2MP und ArduCAM-Mini-5MP.

1. Der Ordner `Mini_5MP_Plus` ist für ArduCAM-Mini-5MP-Plus (OV5640/OV5642) Module.
2. Der Ordner `RevC` ist für ArduCAM-Shield-RevC oder ArduCAM-Shield-RevC+ Shields.
3. Der Ordner `Shield_V2` ist für das ArduCAM-Shield-V2 Schild.
4. Der Ordner `host_app` ist die Host-Erfassungs- und Anzeigeanwendung für alle ArduCAM-Module.
5. Der Ordner `RaspberryPi` ist eine Beispielanwendung für die Raspberry Pi-Plattform, siehe weitere Anleitung.
6. Der Ordner `ESP8266` ist für ArduCAM-ESP8266-UNO-Board-Beispiele für Bibliothekskompatibilität. Bitte versuchen Sie stattdessen, ESP8266 mit dem Skript `josn board manager` zu repositoryen.

Selecting correct COM port and Arduino boards then upload the sketches.

Arducam MINI Kamera Demo Tutorial für Arduino
Arducam Kamera-Schild V2 Demo Tutorial für Arduino

37.7.4. How To Connect Bluetooth Module

Mit dieser Demo

https://github.com/ArduCAM/Arduino/blob/master/ArduCAM/examples/mini/ArduCAM_Mini_Video_Streaming_Bluetooth
So laden Sie den Host V2:

- For ArduCAM_Host_V2.0_Mac.app, please refer to this link:
www.arducam.com/downloads/app/ArduCAM_Host_V2.0_Mac.app.zip
- For ArduCAM_Mini_V2.0_Linux_x86_64bit, Please refer to this link:
www.arducam.com/downloads/app/ArduCAM_Mini_V2.0_Linux_x86_64bit.zip

38. Sensor ov7675

<https://www.arducam.com/?s=ov7675&id=98994/>
<https://www.arducam.com/focal-length-calculator/>
<https://docs.arduino.cc/tutorials/giga-rl-wifi/giga-camera/>
<https://blog.arduino.cc/2020/06/24/machine-vision-with-low-cost-camera-modules/>
<https://www.instructables.com/TinyML-Image-Recognition-With-Edge-Impulse-Nano-33/>
[Ard24a][Ard21a]

38.1. Code

- Arduino sketch

```
1000  /*
1001   OV767X - Camera Capture Raw Bytes
1002
1003   This sketch reads a frame from the OmniVision OV7670 camera
1004   and writes the bytes to the Serial port.
1005
1006   This sketch waits for the letter 'c' on the Serial Monitor,
1007   it then reads a frame from the OmniVision OV7670 camera and
1008   prints the data to the Serial Monitor as a series of bytes.
1009
1010  The website https://rawpixels.net - can be used to visualize the
1011      data:
1012      width: 176
1013      height: 144
1014      offset: 0
1015      Predefined Format: RGB565
1016      Pixel Format: RGBA
1017      Ignore Alpha checked
1018      Little Endian not checked
1019
1020  Circuit:
1021      - Arduino Nano 33 BLE board
1022      - OV7670 camera module:
1023          - 3.3 connected to 3.3
1024          - GND connected GND
1025          - SIOC connected to A5
1026          - SIOD connected to A4
1027          - VSYNC connected to 8
1028          - HREF connected to A1
1029          - PCLK connected to A0
1030          - XCLK connected to 9
1031          - D7 connected to 4
1032          - D6 connected to 6
1033          - D5 connected to 5
1034          - D4 connected to 3
1035          - D3 connected to 2
1036          - D2 connected to 0 / RX
1037          - D1 connected to 1 / TX
1038          - D0 connected to 10
1039
1040  This example code is in the public domain.
1041 */
1042 #include <Arduino_OV767X.h>
```

```

1044 long bytesPerFrame; // variable to hold total number of bytes in
1045   image
1046   long numPixels;
1047
1048 // Declare a byte array to hold the raw pixels received from the
1049 // OV7670
1050 // Array size is set for QCIF; if other format required, change size
1051 // QCIF: 176x144 X 2 bytes per pixel (RGB565)
1052 byte data[176 * 144 * 2];
1053
1054 void setup() {
1055   Serial.begin(115200);
1056   while (!Serial);
1057
1058   // Begin the OV7670 specifying resolution (QCIF, Pixel format
1059   // RGB565 and frames per second)
1060   if (!Camera.begin(QCIF, RGB565, 1)) {
1061     Serial.println("Failed to initialize camera!");
1062     while (1);
1063   }
1064
1065   bytesPerFrame = Camera.width() * Camera.height() * Camera.
1066   bytesPerPixel();
1067   numPixels = Camera.width() * Camera.height();
1068
1069   // Optionally, enable the test pattern for testing
1070   // If the next line is uncommented, the OV7670 will output a test
1071   // pattern
1072   // Camera.testPattern();
1073 }
1074
1075 void loop() {
1076   // Wait for a 'c' from Serial port before taking frame
1077   if (Serial.read() == 'c') {
1078
1079     // Read frame from OV7670 into byte array
1080     Camera.readFrame(data);
1081
1082     // Write out each byte of the array to the serial port
1083     // Probably a quicker way to do this
1084     for (int i = 0; i < bytesPerFrame; i++){
1085       Serial.write(data[i]);
1086     }
1087
1088     // Write out a FF byte to tell receiving program that data is
1089     // finished
1090     // Somewhat dangerous - but has worked so far
1091     delay(100);
1092     Serial.write(0xFF);
1093   }
1094 }
```

..../Code/Arduino/CAM/ov7675/Nano33BLEProgramToReadov7670Data.ino

- save as png

```

1000 from PIL import Image
1001 import os
1002
1003 # Delete the temp file 'imagefile' if it exists
1004 if os.path.exists("imagefile"):
1005   os.remove("imagefile")
1006   print("Deleted temp image file")
1007 else:
1008   print("Temp image file does not exist")
1009
1010 # Open file that contains the RGB565 raw data
1011 image = open('camera', mode='rb')
```

```

1012 # Open a temp file that will contain RGB888 raw data
1014 result = open('imagefile', mode = 'wb')

1016 # If using other picture dimensions, change here
1017 Width = 176
1018 Height = 144

1020 # Binary masks used in RGB565 to RGB888 conversion
1021 MASK5 = 0b00011111
1022 MASK6 = 0b00111111

1024 print("Starting conversion")

1026 # Loop through the raw file and convert to RGB888
1027 # Note that the raw pixel data needs to be converted to integer
1028 # in order for the bit shifting and masking to compute
1029 # Result has to converted back
1030 # This is done with commands .from_bytes and .to_bytes

1032 for x in range(Width * Height):
1033     p = image.read(2) #read two bytes
1034     im = int.from_bytes(p, byteorder='big')
1035     #print(p)

1036     #Conversion of RGB565 to RGB888
1037     red = ((im >> 11) & MASK5) << 3
1038     green = ((im >> 5) & MASK6) << 2
1039     blue = (im >> 0 & MASK5) << 3

1042     r = red.to_bytes(1, 'big')
1043     g = green.to_bytes(1, 'big')
1044     b = blue.to_bytes(1, 'big')

1046     #print(red, green, blue)
1047     result.write(r)
1048     result.write(g)
1049     result.write(b)

1050 result.close()
1051 image.close()

1054 rawData = open("imagefile", 'rb').read()

1056 # the image size
1057 imgSize = (Width, Height)

1058 # create the image file ready for saving
1059 img = Image.frombytes('RGB', imgSize, rawData)

1062 # can give any format you like .png, jpg etc.
1063 img.save("finalpic.png")
1064 print("Finished conversion")

```

..../Code/Arduino/CAM/ov7675/ConvertRawToRGB888AndSaveAspng.py

- save as png

```

1000 import serial
1001 import os

1002 # Delete the existing camera file if it already exists
1003 if os.path.exists("camera"):
1004     os.remove("camera")
1005     print("deleted file camera")
1006 else:
1007     print("camera does not exist")

```

```

1010 # Open a raw file and set it up to receive camera data
1011 image = open('camera', mode='wb')
1012 stopChar = bytes.fromhex('ff')

1014 # Open a serial port that is connected to an Arduino (below is
1015     Linux, Windows and Mac would be "COM4" or similar)
1016 # No timeout specified; program will wait until all serial data is
1017     received from Arduino
1018 # Port description will vary according to operating system. Linux
1019     will be in the form /dev/ttyXXXX
1020 # Windows and MAC will be COMX. Use Arduino IDE to find out name ,
1021     Tools -> Port'
1022 ser = serial.Serial('/dev/ttyACM0')
1023 ser.flushInput()

1024 # Write out a single character encoded in utf-8; this is defalt
1025     encoding for Arduino serial comms
1026 # This character tells the Arduino to start sending data
1027 ser.write(bytes('c', 'utf-8'))

1028 # Loop through and read data received from camera
1029 while True:
1030     #Read in data from Serial a byte at a time
1031     ser_byte = ser.read()
1032     #print(ser_byte)

1033     #If Arduino has sent a byte FF, exit loop
1034     if (ser_byte == stopChar):
1035         break

1036     #Write received data to file
1037     image.write(ser_byte)

1038 # Close port and image file to exit
1039 ser.close()
1040 image.close()
1041 print("image transfer complete")

```

..../Code/Arduino/CAM/ov7675/ReadCameraDataFromNano33BLEViaSerial.py

38.1.1. OV7670 Camera and Image Sensor with Nano 33 BLE

source: <https://www.hackster.io/umpheki/ov7670-camera-and-image-sensor-with-nano-33-b>

The OV7670 is an image sensor that can be used to capture pictures when controlled by a microprocessor such as the Nano 33 BLE. To quote the datasheet from OmniVision: “The OV7670/OV7171 CAMERACHIP image sensor is a low voltage CMOS device that provides the full functionality of a single-chip VGA camera and image processor in a small footprint package. The OV7670/OV7171 provides full-frame, sub-sampled or windowed 8-bit images in a wide range of formats, controlled through the Serial Camera Control Bus (SCCB) interface.”

While the OV7670 is a powerful and versatile IC, it produces only Raw pixel data. For most images to be useful, they need to be in jpeg, png, tiff or similar format. This project shows how you can capture a raw image from the camera and then convert this into a png file.

Here are the major components for this project

- Arduino Nano 33 BLE
- OV7670 camera module
- Breadboard
- Connectors

- Computer with Python ver 3 installed

OV7670 CMOS VGA Sensor

Datasheet for the OV7670 included with this project.

The block diagram from the datasheet:

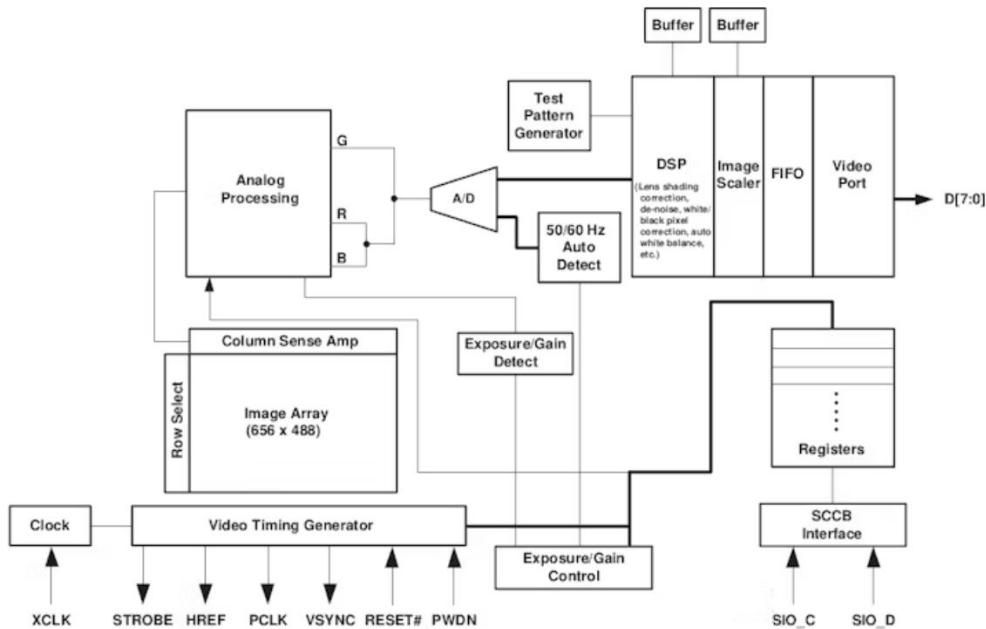


Figure 38.1.: Blockdiagram, der Kamera OV7675

The image array captures the arriving light and after processing the signal, the signal is converted to a digital signal in a A/D converter. This digital signal can then be read from the data bus (D0 – D7) controlled by SIO_C and SIO_D

The OV7670 is capable of capturing video in addition to still images. This project will focus on still images only.

The output formats available from the OV7670 are given on the first page of the datasheet. In this project, the format used is RGB565 (more explanation later)

The image array can take pictures up to 640 x 480 pixels definition. Because of memory limitations on the Nano 33 BLE, pictures up to 320 x 240 are possible (QVGA)

The specific module used in this project includes a simple lens that focuses light on the CMOS array, the focal length of which can be manually adjusted. Loosen the small set screw and twist the lens.

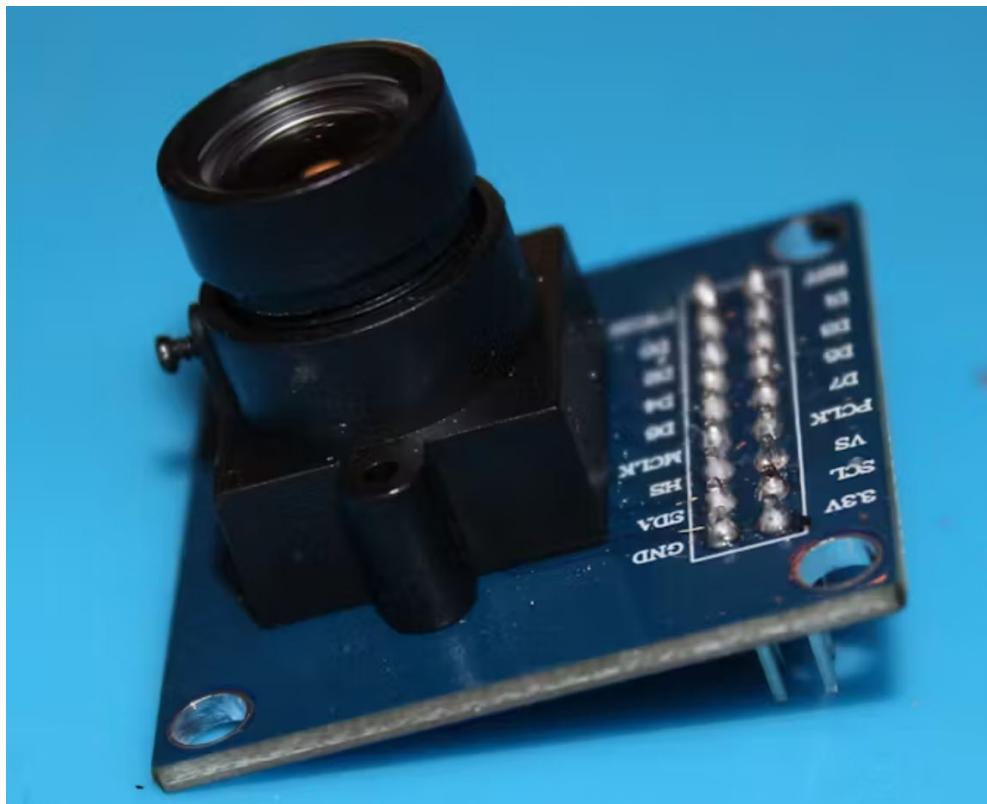


Figure 38.2.: Camera Module OV7675

38.1.2. Connection

Connection between the Nano and OV7670 as follows

OV7670	Nano 33 BLE Pin
3.3 V	3.3 V
GND	GND
SIO_C	A5
SIO_D	A4
VSYNC	8
HREF	A1
PCLK	A0
XCLK	9
D7	4
D6	6
D5	5
D4	3
D3	2
D2	0/RX
D1	1/TX
D0	10

Figure 38.3.: Connection Table of the Camera Module OV7675

Completing this connection is a little tricky and results in a tangle of wires. Picture included of the final connection

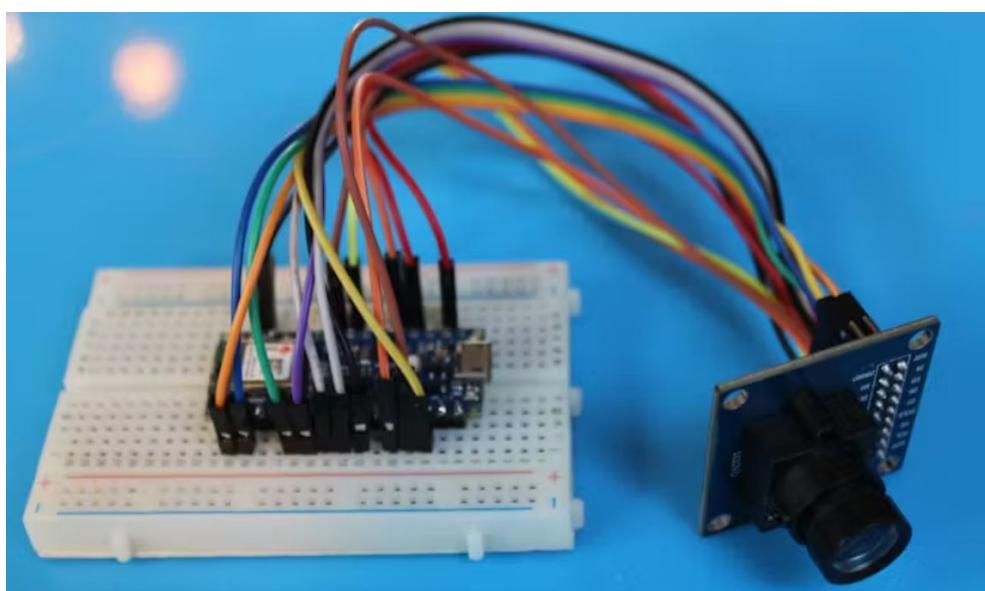


Figure 38.4.: Camera Module OV7675 connected to the Arduino Nano 33 BLE Sense

RGB Formats In this project, the RGB565 raw format will be used.
If you are interested in a complete explanation of all possible formats, consult the Wikipedia article:

https://en.wikipedia.org/wiki/List_of_monochrome_and_RGB_color_formats

Or read this article:

<https://support.touchgfx.com/docs/basic-concepts/color-formats>

RGB565 uses a total of 16 bits as shown

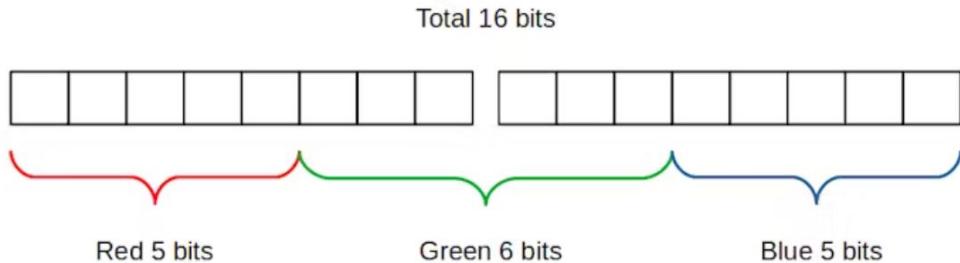


Figure 38.5.: RGB565

Green is allocated 6 bits because the human eye is more sensitive to graduations in green than red or blue.

This format was used in the early days of computing when processing and memory was at a premium. Most systems today use RGB888, where each color is allocated 8 bits for a total of 24 bits (so called true color).

This project requires the RGB565 format to be converted to RGB888 format. This is done by a combination of bit shifting and binary masking. The formula to achieve the conversion below:

p = original 16 bit raw 565 pixel information

red component of RGB888 = $r = (p \gg 11 \& 0b00011111) \ll 3$

green component of RGB888 = $g = (p \gg 5 \& 0b00111111) \ll 2$

blue component of RGB888 = $b = (p \ll 0 \& 0b00011111) \ll 3$

To show a specific example, assume a pixel in RGB565 representation which is only red:

$p = 11111000\ 00000000$

right shift 11 bits $p = 00000000\ 00011111$

bit mask with $0b00011111$ $p = 00000000\ 00011111$

left shift 3 and drop leading eight bits $r = 11111000$

Note that the result is not 100% accurate because low information content cannot be converted to high information content. (However, high information content can always be converted to low information content.)

38.1.3. Arduino OV7670 library

Before using the OV7670, the Arduino library must be installed.

For Arduino IDE 2.x, search for OV7670 and install the library

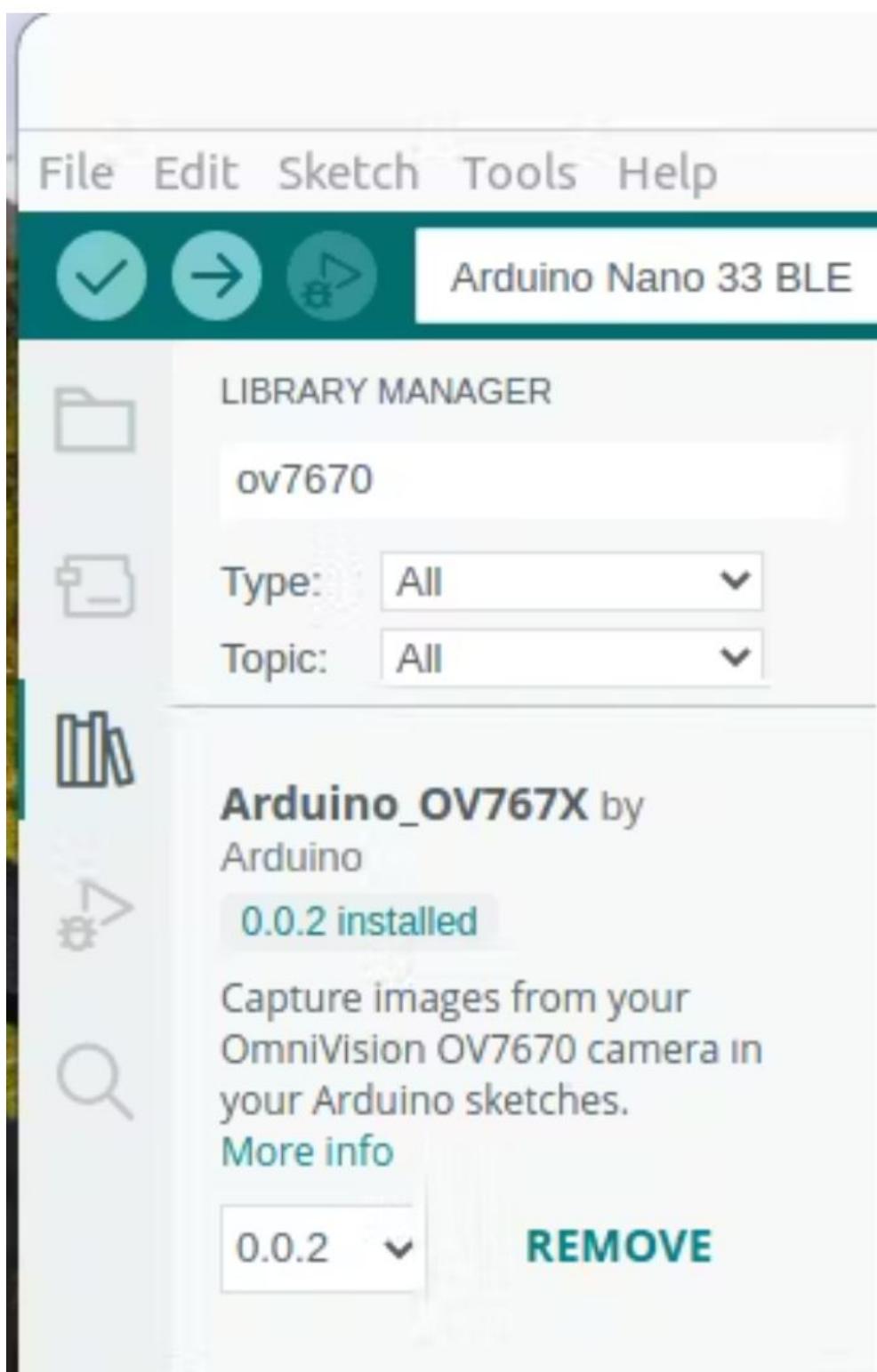


Figure 38.6.: Installation of the Arduino Library

38.1.4. Testing the OV7670

The python program used to read data from the Nano 33 BLE requires that the python library pySerial be installed. This can be installed with the command

`pip install pyserial`

(or sudo pip install pyserial, depending on how your computer is configured)

To check if the library is installed use

`pip show pyserial`

To test the OV7670, two programs are required. Here are the steps:

1. Upload the program “RawCameraCapture.ino” to the Nano 33 BLE
2. Close the Arduino IDE; the serial monitor in the IDE prevents the python program from communicating with the Nano
3. Place the camera to point at the subject. Some experimentation with lighting and distance from subject may be required
4. Navigate to the directory containing the Python program “ReadCamera2.py”
5. Run the the program with the command
`python3 ReadCamera2.py`
6. Once the process is finished, a message “image transfer complete” will print in the terminal
7. The program creates a file called “camera” which contains the raw pixel data
8. Navigate to the website <http://rawpixels.net> in a browser
9. Upload the file `camera`
10. Settings as follows:
 - width: 176
 - height: 144
 - offset: 0
 - Predefined Format: RGB565
 - Pixel Format: RGBA
 - Ignore Alpha checked
 - Little Endian not checked



Figure 38.7.: Dialogue of RawPixel

The picture should display in the raw pixels interface.

38.1.5. Final Step

Once the test is successful, the final step is to create the PNG image file. Really simple:

- Navigate to the directory containing the Python program `ConvertToPNG.py`
- Run the the program with the command
`python3 ConvertToPNG.py`
- This program creates a temporary RGB888 file called `imagefile`
- A file called `finalpic.png` will be created

38.2. Kameramodul OV7675

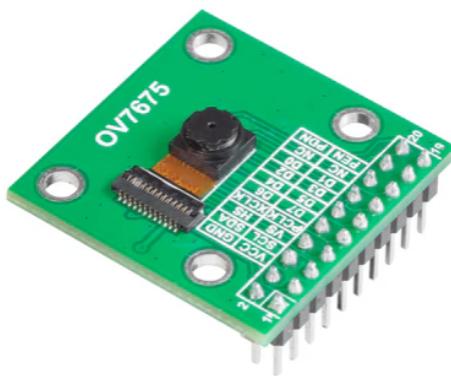


Figure 38.8.: Kameramodul OV7675

Um einen Prozess zu überwachen oder ein System zu automatisieren, bietet sich im Rahmen eines Arduino Prozesses das “Arducam 0,3 MP OV7675” Kameramodul 38.8 an, da dieses mit den Arduino Bibliotheken kompatibel ist. Hierbei ist OV7675 die Modellnummer. Der Senor des Moduls verfügt über eine Auflösung von 640 x 480 Pixeln. Die Pixelgröße, die vom Sensor eingefangen werden kann, beträgt 2,5 µm x 2,5 µm. Das Signal-Rausch-Verhältnis, ein Kennwert für die Klarheit des Bildes, beträgt 38 dB. Der Dynamikbereich, welcher angibt, wie Helligkeitsunterschiede erfasst werden können, hat einen Wert von 71 dB. Um Belichtungszeiten zu steuern, verfügt das Modul über elektrische Rollladen. Des Weiteren beinhaltet der Sensor RGB-Farbsfilter (Rot, Grün, Blau). Daten können über die Formate RAW/YUV/RGB ausgegeben werden. Dies geschieht über einen 20-poligen DVP (Digital Video Port). Verwendet werden darf und kann der Senor in einem Temperaturbereich von -30 °C bis +70 °C. Die Geometrie des Moduls spiegelt sich in der Brettgröße von 30,5 mm x 30,5 mm wider. [Ard24c]

38.3. Kamera Modul OV7675

Um das Kameramodul OV7675 nutzen zu können, muss die Funktion des Bauteils vor Gebrauch getestet werden. Hierzu wird der Sketch `TestCameraRawBites320x240x2` 38.4 ausgeführt. Folgende Headerdateien werden zum Durchführen des Tests benötigt: `TinyMLShield.h`; `ArduinoOV7675.h`. Im Test können unterschiedliche Funktionen der Kamera überprüft werden. Dazu gehört der „single“ Befehl. Die Kamera nimmt ein

```
/*
TestENC

Dieser Code implementiert einen Webserver auf einem Arduino
mit einem Ethernet-Shield, der es ermöglicht, über eine
Webschnittstelle eine LED zu steuern und Aktionen auszulösen.

*/
#include <SPI.h>
#include <EthernetENC.h>

#define ON HIGH
#define OFF LOW
#define RELAY A6

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED }; // MAC-Adresse des Ethernet-
IP Address ip(192, 168, 0, 141); // Statische IP-Adresse des Arduinos

EthernetServer server(80); // Ethernet-Server auf Port 80

String displayText = "Initialer Text"; // Variable für den anzuzeigenden Text
bool testExecuted = false; // Variable zum Speichern des Teststatus
```

Listing 38.1.: Der Sketch [TestENheader.ino](#) in Arduino für das Microcontroller Board

einzelnes Bild auf, wenn im Nachhinein der Befehl `capture` geschrieben wird. Dadurch gibt das Programm Hexadezimalwerte für jeden Pixel des Bildes aus. In diesem Fall muss der ausgegebene Text in Google Colab eingefügt werden, sodass aus der Sequenz im serieller Monitor ein Bild konvertiert werden kann. Die Funktion „`live`“ streamt die Bytes der aufgenommenen Bilder über die serielle Schnittstelle. Der Befehl „`capture`“ löst im single-Modus eine einzelne Bildschirmaufnahme aus.

Um live Übertragungen des Kamera-Moduls streamen zu können, was für unser Projekt essentiell ist, wird ein weiteres Programm „`Processing`“ benötigt. Dieses wird auf dem Endgerät installiert.

38.4. header

Der Header dieses Codes 38.1 richtet die Ethernet Verbindung mit den Bibliotheken `<SPI.h>` und `<EthernetENC.h>` ein. Außerdem werden die Makros `ON` und `OFF` deklariert. Die Ip Adresse wird zugewiesen und der Server, über den HTTP-Anfragen empfangen und Antworten gesendet werden, wird auf Port 80 gesetzt.

38.5. [void setup\(\)](#)

Im Void Setup 38.2 wird eine Sequenz für die LED festgelegt, um den Betriebsstatus sichtbar zu machen. Die Ethernet-Verbindung wird aufgebaut und mögliche Fehler werden im seriellen Monitor ausgegeben.

```

void setup() {
    pinMode(RELAY, OUTPUT);

    // LED Steuerung beim Start
    digitalWrite(RELAY, ON); // LED an fuer 1 Sekunde
    delay(1000);
    digitalWrite(RELAY, OFF); // LED aus fuer 0,5 Sekunden
    delay(500);
    digitalWrite(RELAY, ON); // LED an fuer 1 Sekunde
    delay(1000);
    digitalWrite(RELAY, OFF); // LED aus bleiben

    Serial.begin(9600); // Initialisierung der seriellen Kommunikation
    //while (!Serial) { ; } // Warte auf Verbindung zur seriellen Schnittstelle

    Ethernet.init(10); // Ethernet-Initialisierung mit CS-Pin 10
    if (Ethernet.begin(mac) == 0) { // Versuche DHCP-Konfiguration
        Serial.println("Failed to configure Ethernet using DHCP");
        Ethernet.begin(mac, ip); // Bei Fehler, verwende statische IP
    }

    server.begin(); // Start des Ethernet-Servers
    Serial.print("Server is at ");
    Serial.println(Ethernet.localIP()); // Ausgabe der IP-Adresse des Servers
}

```

Listing 38.2.: Der Sketch [TestENCVoidSetup.ino](#) in Arduino für das Microcontroller Board

38.6. **void loop()**

In der Funktion `loop` 38.3 befindet sich der Hauptteil des einfachen Webservers, der über HTTP kommuniziert und zur Funktionsprüfung eine LED ansteuert.

38.7. Test der Bilddaufnahme

Mit diesem Code 38.4 und der dazugehörigen Bibliothek <Arduino OV767X.h> wird die Kamera angesteuert. Bei jedem aufgenommenen Bild überträgt die Kamera 1.228.800 Bits kontinuierlich über die serielle Schnittstelle an den Computer. Außerdem wurde die grüne LED mit implementiert, um den Betriebsstatus anzuzeigen und eine mögliche Fehlersuche zu vereinfachen.

38.8. Bildverarbeitung

In der Processing Anwendung werden die seriell übertragenen Daten vom Processing Skript 38.5 verarbeitet und um 90 Grad nach links gedreht. Somit wird das live Bild in einem kleinen Fenster gestreamt 38.9 .

```

void loop() {
    EthernetClient client = server.available(); // Warte auf Verbindung vom Client
    if (client) {
        Serial.println("New client");
        bool currentLineIsBlank = true;
        String request = "";

        while (client.connected()) {
            if (client.available()) {
                char c = client.read();
                request += c;

                if (c == '\n' && currentLineIsBlank) {
                    Serial.println(request);

                    // HTML-Seite senden
                    client.println("HTTP/1.1 200 OK");
                    client.println("Content-Type: text/html");
                    client.println();
                    client.println("<html><body style='background-color: #ADD8E6;'><h2 style='color: black; font-size: 1.5em; margin-bottom: 10px;'>Bildaufnahme</h2><form style='width: 100%; border: 1px solid #ccc; padding: 10px; border-radius: 10px;'><input type='submit' value='Bildaufnahme' style='width: 100%; height: 40px; background-color: #ADD8E6; color: black; font-weight: bold; border: none; font-size: 1em;'><input type='submit' value='Test' style='width: 100px; height: 40px; background-color: #ADD8E6; color: black; font-weight: bold; border: none; font-size: 1em;'></form></body></html>");
                    client.println();
                }

                // Anzeigen des Textfelds
                client.println("<p>Hier wird ihr Bild in einem Hexadezimal-String ausgegeben</p>");
                client.println("<p>"); 
                client.println(displayText);
                client.println("</p>");

                // HTTP-Anfrage analysieren
                if (request.indexOf("/?CAPTURE") != -1) {
                    // Funktion fuer den Capture-Button: Funktion Capture aufrufen
                    captureText();
                    client.println("<p>Bild wurde erfasst</p>");
                } else if (request.indexOf("/?TEST") != -1) {
                    // Funktion fuer den Test-Button: LED schnell blinken lassen
                    blinkLED();
                    testExecuted = true; // Teststatus aktualisieren
                }

                if (testExecuted) {
                    client.println("<p>Test erfolgreich ausgefuehrt</p>");
                    testExecuted = false; // Teststatus zuruecksetzen
                } else {
                    // client.println("<p>Keine Aktion durchgefuehrt</p>");
                }

                client.println("</body></html>");
                break;
            }
            if (c == '\n') {
                currentLineIsBlank = true;
            } else if (c != '\r') {
                currentLineIsBlank = false;
            }
        }
    }
    delay(1); // Kurze Pause, bevor die Verbindung geschlossen wird
    client.stop(); // Schliesse Verbindung zum Client
    Serial.println("Client disconnected");
}

```

```

/*
TestCameraRawBites320x240x2

Dieser Sketch liest ein Bild von der OmniVision OV7670-Kamera
und schreibt die Bytes an den seriellen Port. Verwenden Sie den
Processing-Sketch "CameraVisualizerHochkant320x240x2"
um die Kameraausgabe zu visualisieren.

*/
#include <Arduino_OV767X.h>

int bytesPerFrame;
// Definiere die Pin-Nummer fuer die LED
const int ledPin = A6;

byte data[320 * 240 * 2]; // QVGA: 320x240 X 2 bytes per pixel (RGB565)

void setup() {
    Serial.begin(9600);
    while (!Serial);
    // Setze den Pin-Modus als Ausgang
    pinMode(ledPin, OUTPUT);

    if (!Camera.begin(QVGA, RGB565, 1)) {
        Serial.println("Failed to initialize camera!");
        while (1);

        digitalWrite(ledPin, LOW);
    }

    bytesPerFrame = Camera.width() * Camera.height() * Camera.bytesPerPixel();

    // Optionally, enable the test pattern for testing
    // Camera.testPattern();
}

void loop() {
    Camera.readFrame(data);

    Serial.write(data, bytesPerFrame);
    digitalWrite(ledPin, HIGH);
}

```

Listing 38.4.: Der Sketch [TestCameraRawBites320x240x2.ino](#) in Arduino für das Microcontroller Board

```

/*
 CameraVisualizerHochkant320x240x2

Dieser Code liest ein Bild von einer OV7670-Kamera ueber die
serielle Schnittstelle ein, konvertiert die empfangenen RGB565-Daten
RGB-Daten und zeigt das Bild um 90 Grad gedreht
auf dem Bildschirm an.

*/
import processing.serial.*;
import java.nio.ByteBuffer;
import java.nio.ByteOrder;

Serial myPort;

// must match resolution used in the sketch
final int cameraWidth = 320;
final int cameraHeight = 240;
final int cameraBytesPerPixel = 2;
final int bytesPerFrame = cameraWidth * cameraHeight * cameraBytesPerPixel;

PIImage myImage;
byte[] frameBuffer = new byte[bytesPerFrame];

void setup()
{
    size(240, 320);

    // Seriellen Port auswaehlen
    myPort = new Serial(this, Serial.list()[0], 9600);

    // if you know the serial port name
    //myPort = new Serial(this, "COM5", 9600);
    // Windows
    //myPort = new Serial(this, "/dev/ttyACM0", 9600);
    // Linux
    // myPort = new Serial(this, "/dev/cu.usbmodem14401", 9600);
    // Mac

    // wait for full frame of bytes
    myPort.buffer(bytesPerFrame);

    myImage = createImage(cameraWidth, cameraHeight, RGB);
}

void draw()
{
    // Rotate the image by 90 degrees clockwise
    pushMatrix();
    translate(width / 2, height / 2);
    rotate(HALF_PI);
    image(myImage, -myImage.width / 2, -myImage.height / 2);
    popMatrix();
}

void serialEvent(Serial myPort) {
    // read the saw bytes in
    myPort.readBytes(frameBuffer);

    // access raw bytes via byte buffer
    ByteBuffer bb = ByteBuffer.wrap(frameBuffer);
    bb.order(ByteOrder.BIG_ENDIAN);
}

```

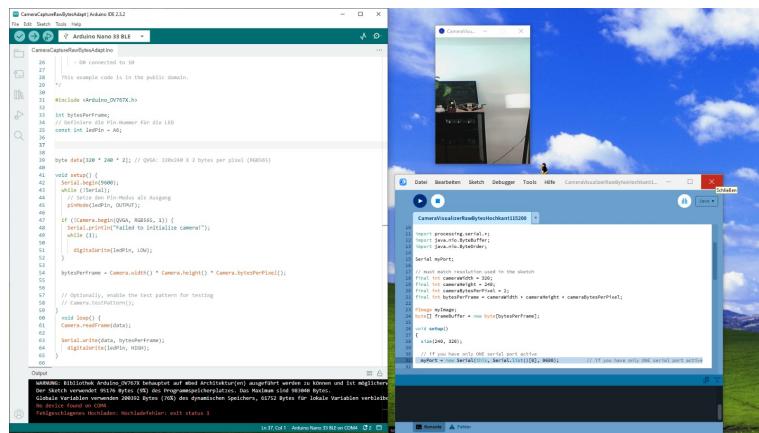


Figure 38.9.: Bildschirmaufnahme

38.9. Produktbeschreibung

Arducam-M-2MP ist eine optimierte Version von Arducam Shield Rev.C und ist eine hochauflösende 2MP-SPI-Kamera, die die Komplexität der Kamerasteuerung verringert. Sie verfügt über einen 2-MP-CMOS-Bildsensor OV2640 und hat eine Miniaturgröße sowie eine einfach zu bedienende Hardware-Schnittstelle und die Open-Source-Code-Bibliothek. Die Arducam Mini-Kamera kann auf allen Plattformen wie Arduino, Raspberry Pi, Maple, Chipkit, Beaglebone Black verwendet werden, solange sie über eine SPI- und I2C-Schnittstelle verfügen und mit Standard-Arduino Boards verbunden werden können. Die Arducam Mini-Kamera bietet nicht nur die Möglichkeit, eine Kamera-Schnittstelle hinzuzufügen, die in einigen Mikrocontrollern nicht vorhanden ist, sondern bietet auch die Möglichkeit, mehrere Kameras zu einem einzigen Mikrocontroller hinzuzufügen.

WS:Plagiat! cite!

Anwendung:

- IoT-Kameras.
 - Roboterkameras.
 - Wildlife-Kameras.

Andere batteriebetriebene Produkte.

Kann auf Plattformen wie MCU, Raspberry Pi, ARM, DSP, FPGA verwendet werden.

Eigenschaften:

- 2-Megapixel-Bildsensor OV2640.
 - M12-Mount- oder CS-Mount-Objektivhalter mit wechselbaren Objektivoptionen.
 - IR-empfindlich mit entsprechender Objektivkombination.
 - I2C-Schnittstelle für die Sensorkonfiguration.
 - SPI-Schnittstelle für Kamera-Befehle und Datenstrom.
 - Alle E/A-Anschlüsse sind für 5 V/3,3 V geeignet.
 - Unterstützt JPEG-Komprimierungsmodus, Einzel- und Mehrfachaufnahmemodus, einmaliges Erfassen mehrerer Lesevorgänge, Burst-Lese-Operation, Niedrige-Energie-Modus usw..

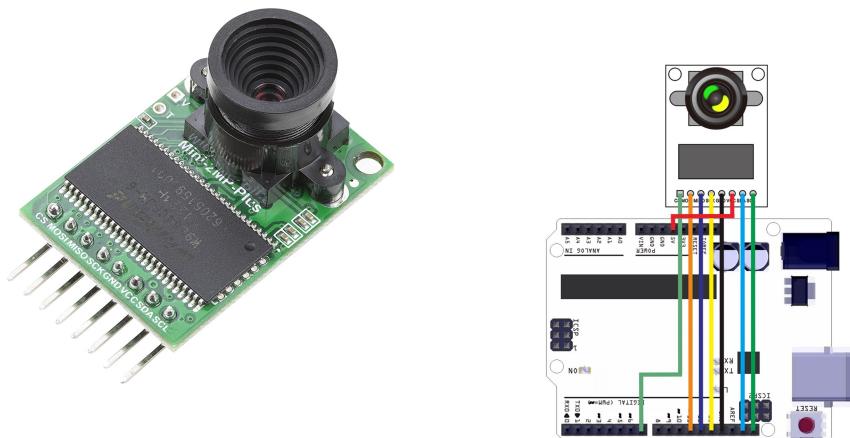


Figure 38.10.: Kamera IMX477 der Firma Arducam; [Ard21]

- Kann mit Standard-Arduino-Boards verbunden werden.
 - Open-Source-Code-Bibliothek für Arduino, STM32, Chipkit, Raspberry Pi, BeagleBone Black.
 - Schlanke Form.

Lieferumfang:

1 x Arducam Mini-Modul Kameraschutz mit OV2640 2 MP, Objektiv, für Arduino UNO Mega2560 Board.

Hinweis: Arduino UNO ist nicht enthalten.

<a href="https://www.amazon.com/dp/B07D58GDDV/ref=sr_1_17_sspa?__mk_de_DE=ÄÖËÅ;ÄTÄŚ&dchild=1&keywords=arducam&qid=1622358684&sr=8-17-spons&psc=1&spLa=ZW5jcnlwdGVkUXVhbGlmaWVyPUEyWUdNSVhJSFNUQUtLJmVuY3J5cHR1ZElkPUEwNjI4NTIwRaspberry Pi Kamerakabel, iUniker 15-poliges Flachbandkabel, Pi Kamera Flex Kabel, Flex CSI Kabel 50 cm/1 m/2 m für Raspberry Pi 3B+, 3B, 2B (nicht für Pi Zero)

38.10. ArduCAM Library Introduction

<https://github.com/ArduCAM/Arduino>

Dies ist eine Open-Source-Bibliothek für die Aufnahme von hochauflösenden Standbildern und kurzen Videoclips auf Arduino-basierten Plattformen unter Verwendung der Kameramodule von ArduCAM. Die Kamera-Breakout-Boards sollten vor dem Anschluss an die Arduino-Boards mit dem ArduCAM-Shield funktionieren. ArduCAM-Kameramodule der Mini-Serie wie Mini-2MP, Mini-5MP(Plus) können direkt an Arduino-Boards angeschlossen werden. Zusätzlich zu Arduino kann die Bibliothek auf beliebige Hardware-Plattformen portiert werden, solange sie über eine I2C- und SPI-Schnittstelle verfügen, die auf dieser ArduCAM-Bibliothek basiert.

Now Supported Cameras

- OV7660 0.3MP

- OV7670 0.3MP
- OV7675 0.3MP
- OV7725 0.3MP
- MT9V111 0.3MP
- MT9M112 1.3MP
- MT9M001 1.3MP
- MT9D111 2MP
- OV2640 2MP JPEG
- MT9T112 3MP
- OV3640 3MP
- OV5642 5MP JPEG
- OV5640 5MP JPEG

Supported MCU Platform

Theoretically support all Arduino families

- Arduino UNO R3 (Tested)
- Arduino MEGA2560 R3 (Tested)
- Arduino Leonardo R3 (Tested)
- Arduino Nano (Tested)
- Arduino DUE (Tested)
- Arduino Genuino 101 (Tested)
- Raspberry Pi (Tested)
- ESP8266-12 (Tested) (http://www.arducam.com/downloads/ESP8266_UNO/package_ArduCAM_index.json)
- Feather M0 (Tested with OV5642)

Note: ArduCAM library for ESP8266 is maintained in another repository ESP8266 using a json board manager script.

38.11. Libraries Structure

Die Basisbibliotheken bestehen aus zwei Unterbibliotheken: [ArduCAM](#) und [UTFT4ArduCAM_SPI](#). Diese beiden Bibliotheken sollten direkt unter die Bibliotheken des Arduino-Verzeichnisses kopiert werden, damit sie von der Arduino-IDE erkannt werden.

Die ArduCAM-Bibliothek ist die Kernbibliothek für ArduCAM-Shields. Sie enthält unterstützte Bildsensortreiber und Benutzerland-API-Funktionen, die Befehle zum Erfassen oder Lesen von Bilddaten erteilen. Es gibt auch ein Beispielverzeichnis innerhalb der ArduCAM-Bibliothek, das die meisten Funktionen der ArduCAM-Shields illustriert. Die vorhandenen Beispiele sind Plug-and-Play, ohne dass eine einzige Zeile Code geschrieben werden muss.

Die Bibliothek [UTFT4ArduCAM_SPI](#) ist eine modifizierte Version von UTFT, die von Henning Karlsen geschrieben wurde. Wir haben sie portiert, um das ArduCAM-Shield mit LCD-Bildschirm zu unterstützen. Daher wird die Bibliothek [UTFT4ArduCAM_SPI](#) nur benötigt, wenn das ArduCAM-LF-Modell verwendet wird.

38.12. How to use

Die Bibliotheken sollten vor dem Ausführen von Beispielen konfiguriert werden, andernfalls erhalten Sie eine Fehlermeldung beim Kompilieren.

38.12.1. 1. Edit `memoriesaver.h` file

Öffnen Sie die Datei `memoriesaver.h` im ArduCAM-Ordner und aktivieren Sie die Hardwareplattform und das Kameramodul, das zu Ihrer Hardware passt, indem Sie die Makrodefinition in der Datei auskommentieren oder auskommentieren. Wenn Sie zum Beispiel eine ArduCAM-Mini-2MP haben, sollten Sie die Zeile `#define OV2640_MINI_2MP` auskommentieren und alle anderen Zeilen auskommentieren. Und wenn Sie ein ArduCAM-Shield-V2 und ein OV5642-Kameramodul haben, sollten Sie die Zeile `#define ARDUCAM_SHIELD_V2` und die Zeile `#define OV5642_CAM` auskommentieren und dann alle anderen Zeilen.

38.12.2. 2. Choose correct CS pin for your camera

Öffnen Sie eines der Beispiele und verdrahten Sie die SPI- und I2C-Schnittstelle, insbesondere die CS-Pins, entsprechend den Beispielen mit dem ArduCAM-Shield. Hardware und Software sollten konsistent sein, um die Beispiele korrekt auszuführen.

38.12.3. 3. Upload the examples

Im Beispieldordner befinden sich sieben Unterverzeichnisse für verschiedene ArduCAM-Modelle und die Host-Anwendung. Der Ordner Mini ist für die Module ArduCAM-Mini-2MP und ArduCAM-Mini-5MP.

1. Der Ordner `Mini_5MP_Plus` ist für ArduCAM-Mini-5MP-Plus (OV5640/OV5642) Module.
2. Der Ordner `RevC` ist für ArduCAM-Shield-RevC oder ArduCAM-Shield-RevC+ Shields.
3. Der Ordner `Shield_V2` ist für das ArduCAM-Shield-V2 Schild.
4. Der Ordner `host_app` ist die Host-Erfassungs- und Anzeigeanwendung für alle ArduCAM-Module.
5. Der Ordner `RaspberryPi` ist eine Beispielanwendung für die Raspberry Pi-Plattform, siehe weitere Anleitung.
6. Der Ordner `ESP8266` ist für ArduCAM-ESP8266-UNO-Board-Beispiele für Bibliothekskompatibilität. Bitte versuchen Sie stattdessen, ESP8266 mit dem Skript `josn board manager` zu repositoryen.

Selecting correct COM port and Arduino boards then upload the sketches.

Arducam MINI Kamera Demo Tutorial für Arduino
Arducam Kamera-Schild V2 Demo Tutorial für Arduino

38.12.4. 4. How To Connect Bluetooth Module

Mit dieser Demo

https://github.com/ArduCAM/Arduino/blob/master/ArduCAM/examples/mini/ArduCAM_Mini_Video_Streaming_Bluetooth

So laden Sie den Host V2:

- For ArduCAM_Host_V2.0_Mac.app, please refer to this link:
www.arducam.com/downloads/app/ArduCAM_Host_V2.0_Mac.app.zip
- For ArduCAM_Mini_V2.0_Linux_x86_64bit, Please refer to this link:
www.arducam.com/downloads/app/ArduCAM_Mini_V2.0_Linux_x86_64bit.zip

39. Lens Calibration Tool

39.1. Circle Grid Camera Calibration Patterns

You can download a PDF version of the circle grid calibration pattern that our SceneScan stereo vision sensor uses for camera calibration. This pattern is compatible to the OpenCV asymmetric circle grid, and can thus also be used for other purposes. When printing the pattern, please ensure that your PDF viewer or printer driver do not scale the document. Otherwise you will not be able to receive accurate metric measurements through stereo vision. You can verify that your printed pattern has the correct scaling, by measuring the reference line at the top of each document.

39.2. Siemens Star

The siemens star enables the exact focusing of camera lenses. That is why we print it on the back of our calibration boards. In order to make the camera focusing easier for you as well, you can download our siemens star here.

39.3. Enjoyyourcamera Testkarte für Kameras und Objektive 30 × 45 cm

Testen Sie Ihre Objektive jetzt selbst! Entwickelt vom Tierfotografen Benny Rebel (Bekannt aus Film & Fernsehen), 400dpi Druck. Testbereiche für Schärfe, Auflösung, Kontrast, Chromatische Abberation, Fokus. Nahtlos erweiterbar für größere Test-Fläche.

Mehrere Testkarten zu einer großen Testkarte zusammenfügbar Eine Testkarte für Normal- und Teleobjektive, ab 4 Testkarten für Weitwinkelobjektive Festes Papier, hohe Druckqualität, extrem feine Detailelemente Einfache Bedienung, Lieferung im stabilen Karton Sehr viele Testkriterien in einer Karte vereint.

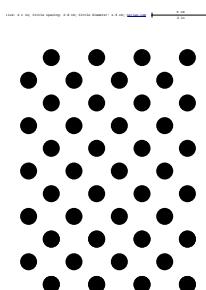


Figure 39.1.: Circle Grid Camera Calibration Patterns

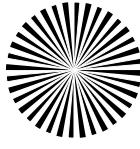


Figure 39.2.: Siemens Star



Figure 39.3.: Siemens Star

39.3.1. Testkarte zum Prüfen von Kameras und Objektiven in Bezug auf ihre Qualität und mögliche Defekte

Mithilfe dieser von dem Naturfotografen Benny Rebel entwickelten Testkarte lassen sich Kameras und Objektive relativ unkompliziert hinsichtlich ihrer Qualität und möglicher Defekte testen. Damit bietet die Testkarte eine preisgünstige Möglichkeit, die eigene Ausrüstung individuell zu prüfen, und das unabhängig von Testberichten, die wegen einer hohen Serienstreuung bei einigen Objektiv- und Kameramarken oft nicht repräsentativ sind.

39.3.2. Zahlreiche Testkriterien: z.B. Schärfe in der Mitte und an den Bildrändern, Randabschattungen u.v.m.

Die Karte deckt zahlreiche Testkriterien ab. Die Kriterien für Kameras sind zum Beispiel die Schärfe in der Bildmitte und an den Rändern, die Auflösung, die Farbwiedergabe, den Kontrastumfang, den Weißabgleich, den Moiré-Effekt, eventuelle Probleme beim automatischen Scharfstellen und das Bildrauschen bei unterschiedlichen ISO-Einstellungen testen. Die Testkriterien für Objektive sind unter anderem Schärfe, Auflösung, Kontrast, Brillanz, Farbwiedergabe, Randabschattungen, chromatische Aberration, Verzeichnung, Vignettierung, eventuelle Probleme beim automatischen Scharfstellen, eine mögliche Dezentrierung der Linsenelemente sowie die Beugung bei unterschiedlichen Blenden.

39.3.3. Mehrere Testkarten zu einer großen Testkarte zusammenfügbar - ideal für Weitwinkelobjektive

Die Testkarte ist so konzipiert, dass man vier oder neun identische Testkarten zu einer großen Testkarte zusammenfügen kann. Für diesen Zweck sind die Testelemente an den Rändern und in den Ecken der Testkarte derart abgeschnitten, dass sie auf einer benachbarten Karte nahtlos weiterverlaufen. Eine große Testfläche, wie sie durch das Zusammenfügen mehrerer Testkarten entsteht, ist wichtig zum Prüfen von weitwinkeligen Objektiven. Solche Objektive besitzen von Natur aus einen weiten Bildwinkel und demzufolge auch einen sehr großen Bildausschnitt. Mit einer großen Testkarte ist sichergestellt, dass die Testfläche den gesamten Bildausschnitt des jeweiligen Objektivs abdeckt, damit man selbst in den Bildecken die Schärfe und andere Kriterien prüfen kann.

39.3.4. Eine Testkarte für Normal- und Teleobjektive, vier Testkarten für Weitwinkelobjektive usw.

Für Normal- und Teleobjektive genügt eine einzelne Testkarte, für Objektive mit einer Brennweite zwischen 50mm und 24mm sollte man vier Testkarten verwenden, und für stark weitwinkelige Objektive mit einer Brennweite von weniger als 24mm empfiehlt sich die Benutzung von 9 Testkarten. Zum Testen von Superweitwinkelobjektiven mit einer Brennweite von weniger als 16mm ergibt es sogar Sinn, die Testkarten nicht direkt aneinanderzufügen. Stattdessen sollte man einen Abstand zwischen den Karten lassen. Wichtig: Die obengenannten Brennweiten beziehen sich auf Objektivtests an Vollformatkameras. Bei Objektivtests an Crop-Kameras verändern sich die Grenzen entsprechend dem Formatfaktor der Kamera. Während man zum Beispiel für ein 35mm-Objektiv an einer Vollformatkamera vier Testkarten benötigt, braucht man für das gleiche 35mm-Objektiv an einer Crop-Kamera mit einem Formatfaktor von 1,5 nur eine Testkarte, denn $35\text{mm} \times 1,5$ ist gleich 52,5mm. Prinzipiell gilt aber: Je größer die Testfläche, desto besser sind die Testergebnisse. Es ergibt also durchaus Sinn, auch zum Testen eines Normalobjektivs vier Testkarten zu verwenden.

39.3.5. Festes Papier, hohe Druckqualität, extrem feine Detailelemente für besonders genaues Testen

Die Testkarte ist auf ein sehr festes, stabiles Papier gedruckt. Außerdem ist die Karte mit einer Druckqualität von 400 dpi extrem detailreich. Die feinsten Detailelemente sind sogar nur einen Punkt breit. Das ermöglicht ein ganz besonders genaues Testen von Objektiv- und Kameraeigenschaften. Außerdem befinden sich die wesentlichen Testflächen sowohl in der Mitte der Karte als auch an den Rändern und in den Ecken. So kann man Eigenschaften wie zum Beispiel die Schärfequalität im gesamten Bildbereich testen. Das hilft dabei, selbst subtile Mängel wie einen Schärfeabfall in den Ecken aufzuspüren.

39.3.6. Handhabung

1. Bringen Sie die Testkarte(n) an einer senkrechten, geraden Fläche an. Dafür eignen sich zum Beispiel Holzplatten, gerade Wände oder Glasscheiben. Stellen Sie sicher, dass die Karte gleichmäßig beleuchtet wird, und dass auf der Karte keine Reflexionen zu sehen sind.
2. Stellen Sie sicher, dass sich die Objektivachse in einem 90 Grad Winkel zur Vorderseite der Testkarte(n) befindet. Richten Sie die Kamera so aus, dass der Bildausschnitt exakt die Testkarte(n) abdeckt. Das Zentrum der Objektivachse sollte genau auf den Mittelpunkt der Testkarte(n) gerichtet sein.
3. Um ein unverfälschtes Ergebnis zu bekommen, muss die Kamera absolut vibrationsfrei ausgelöst werden. Je nach Brennweite können schon geringe Abweichungen eine Randunschärfe auf dem Foto erzeugen. Benutzen Sie deshalb am besten ein Stativ und einen Fernauslöser. Aktivieren Sie außerdem, sofern vorhanden, die Spiegelvorauslösung. Und deaktivieren Sie die Bildstabilisierung.
4. Für einen Standardtest empfehlen sich folgende Einstellungen:Stellen Sie die Kamera auf Blendenvorwahl (Zeitautomatik) und die Lichtempfindlichkeit auf ISO 200. Wählen Sie idealerweise das Bildformat RAW. Arbeiten Sie außerdem mit dem manuellen Weißabgleich, nach dem Sie diesen auf das Umgebungslicht abgestimmt haben. Für weiterführende Tests, wie zum Beispiel das Bildrauschen bei verschiedenen ISO-Einstellungen, können Sie Einstellungen an der Kamera verändern.

5. endenreihen: Als erstes Bild bietet sich immer die Offenblende an. Danach kann man der Reihe nach alle Blendenschritte testen (z.B. 1.0 - 1.4 - 2.0 - 2.8 - 4.0 - 5.6, 8.0 - 11 - 16 - 22).
6. Brennweitenreihen bei Zoom-Objektiven: Auf jeden Fall sollten Sie die Anfangsbrennweite und die Endbrennweite testen. Bei einem 24-70mm Objektiv wären das die Brennweiten 24mm und 70mm. Danach können Sie die Brennweiten in den Schritten testen, wie Sie auch als Festbrennweiten häufig genutzt werden: Das hat den Vorteil, dass man dann die Abbildungsleistung des jeweiligen Objektivs auch mit der Qualität gängiger Festbrennweiten vergleichen kann.
7. Und ganz wichtig: Um beim Testen von Wechselobjektiven und Wechselobjektivkameras feststellen zu können, ob ein Defekt vom Objektiv oder von der Kamera ausgeht, sollten Sie Objektive an verschiedenen Kameras testen, und Kameras mit verschiedenen Objektiven. Defekte und Qualitätsmängel müssen nicht zwingend vom Objektiv ausgehen, auch ein schief in der Kamera verbauter Chip kann eine Ursache für eine mangelnde Bildqualität sein.

39.4. B.I.G. RES7 Testtafel für Kameras und Objektive

Die RES7 Testtafel, in der Größe 32x47cm mit matter Oberfläche, wurde für Fotografen entwickelt, um die Bildqualität zu testen und die Grenzen der Auflösung von Kamera und Objektiv beurteilen zu können.

Mit der Tafel kann die Messung der Bildqualität von Digitalkameras, die Messung der Auflösung von optischen Geräten (Vergrößerung 30-50x), die Kontrolle der Belichtung, die grobe Bestimmung gering auflösender Objektive (5-10x) sowie die Kontrolle der Fokussierung und der Schärfentiefe durchgeführt werden.

Die Testtafel eignet sich für Fotokameras bis zu 4000 aktiven Pixeln in der Bildhöhe, d.h. für alle Kompakt-, sowie DSLR- oder DSLM-Kameras mit einer maximalen Auflösung von 24 Megapixeln.

39.4.1. BIG RES7 Test Board für Kameras und Objektive

Das BIG RES7 Test Board für Kameras und Objektive in der Größe 32 x 47 cm mit einer matten Oberfläche wurde für Fotografen entwickelt, um die Bildqualität zu testen und die Grenzen der Kamera- und Objektivauflösung abzuschätzen.

Das Board kann zur Messung der Bildqualität von Digitalkameras, zur Messung der Auflösung optischer Geräte (30-50-fache Vergrößerung), zur Kontrolle der Belichtung, zur groben Bestimmung niedrig auflösender Objektive (5-10-fach) und zur Kontrolle von Fokus und Schärfentiefe verwendet werden.

39.4.2. Merkmale des BIG RES7 Test Boards für Kameras und Objektive

- Entwickelt für Fotografen zum Testen der Bildqualität
- Um die Grenzen der Kamera- und Objektivauflösung abzuschätzen
- Um die Bildqualität von Digitalkameras zu messen
- Um die Auflösung von optischen Geräten zu messen (Vergrößerung 30-50x)
- Matte Oberfläche
- Abmessungen: 32 x 47 cm

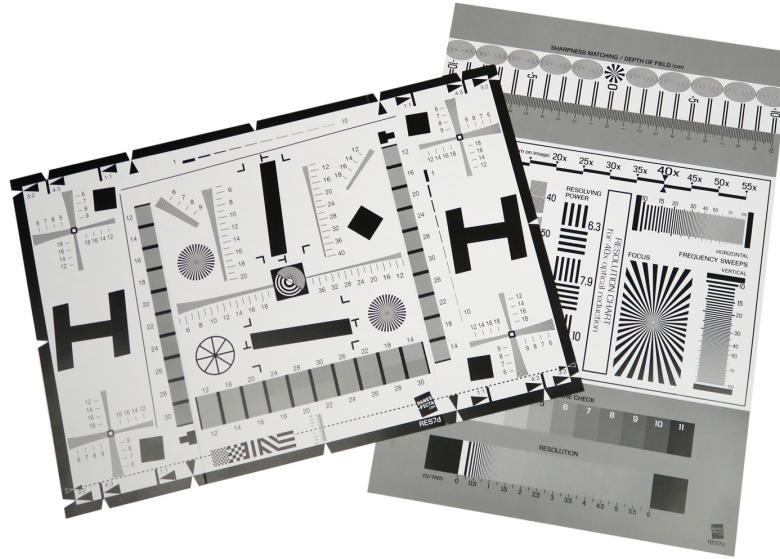


Figure 39.4.: Siemens Star

39.5. Pattern Generator

<https://calib.io/pages/camera-calibration-pattern-generator>

<https://markhelyjones.com/projects/calibration-checkerboard-collection>

Customize a pattern to match your application perfectly and download a printable PDF file. The pattern generator and its outputs can be used for internal development or educational use only. Any commercial purposes or re-sale is strictly prohibited without prior consent.

NOTE: When printing on a standard inkjet or laser printer, please make sure that your software or printer does not apply any scaling to the pattern. Also make sure that no rasterisation is performed in the printer driver. It is advisable to measure the final pattern after printing.

Once you have settled on a pattern and would like to order a professionally made, highly accurate calibration target, feel free to send us an email at sales@calib.io , and we will happily give you a quote!

39.6. Chess Board

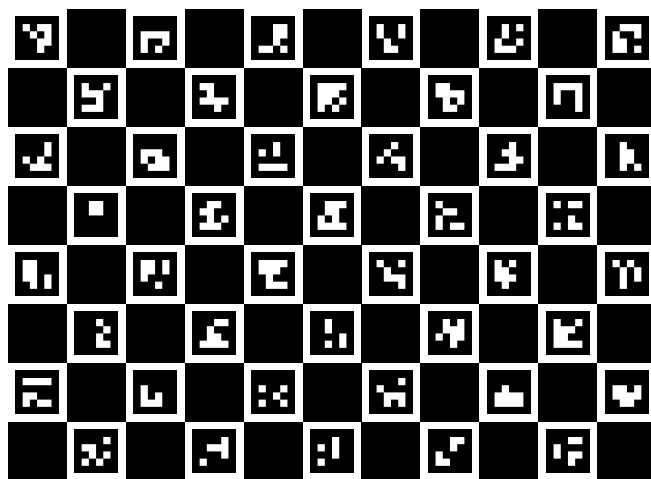
[NKK07] [Nto+07] [Nto+09] [Pro+12]

39.7. To Do

Arducam Lens Calibration Tool, Sichtfeld (Field of View, FoV) Test Chart Folding Card, Pack of 2

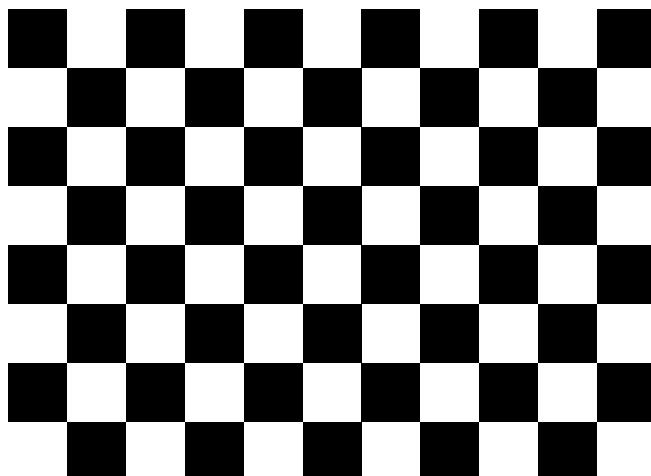
https://www.amazon.com/-/de/dp/B0872Q1RLD/ref=sr_1_40?__mk_de_DE=%C3%A4%C3%96%C3%A4%C3%A4%C3%A4&dchild=1&keywords=arducam&qid=1622358908&sr=8-40

Multifunktional für Objektiv: Objektivfokus kalibrieren, FoV messen und Schärfe abschätzen



www.calib.io | 8x11 | Checker Size: 15 mm | Marker Size: 11 mm | Dictionary: Aruco DICT_4X4.

Figure 39.5.: Siemens Star



www.calib.io | 8x11 | Checker Size: 15 mm.

Figure 39.6.: Siemens Star

7x9 checkerboard for camera calibration.
Squares are: 20x20 mm if printed to 1:1 scale on a A4 paper.

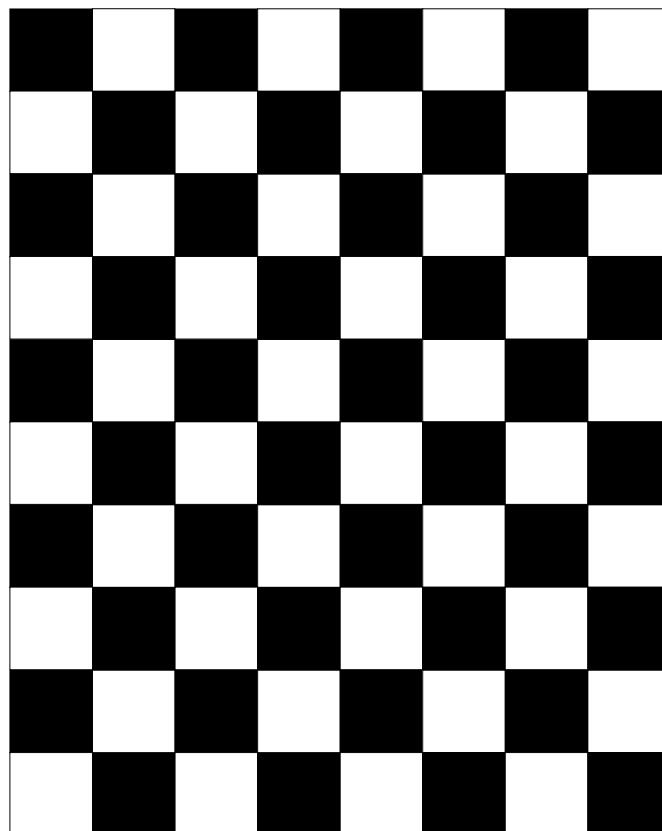


Figure 39.7.: Siemens Star

Einfach zu bedienen: Schnelle Einrichtung in Sekunden und Messung in Minuten mit Online-Videotutorial.

Ein handliches Werkzeug: Einfaches Ermitteln des Sichtfelds des Objektivs ohne Berechnung

Sauber und aufgeräumt: Faltbarer Kartenstil, mehrfach gefaltet und aufgerollt für schnelle Einstellung und bessere Lagerung

Anwendung: Fokuskalibrierung, Schärfeabschätzung und Bildfeld-Schnellmessung für M12, CS-Mount, C-Mount und DSLR-Objektive.

<https://www.arducam.com/product/arducam-lens-calibration-tool-field-of-view-fov/>

39.8. Übersicht

Sind Sie immer noch frustriert von der Berechnung des FOV Ihrer Objektive und den unscharfen Bildern? Arducam hat jetzt ein multifunktionales Tool für Objektive veröffentlicht, mit dem Sie das Sichtfeld des Objektivs ohne Berechnung erhalten und den Objektivfokus schnell und einfach kalibrieren können.

39.9. Applications

Focus calibration, sharpness estimation and field of view quick measuring for M12, CS-Mount, C-Mount, and DSLR lenses

39.10. Package Contents

2*Foldable Lens Calibration Card

39.10.1. Licht, Farben und Farbmodelle

Farbe ist eine Eigenschaft des Lichts, die durch die verschiedenen Wellenlängen des sichtbaren Lichtspektrums entsteht. Farben werden durch das menschliche Auge wahrgenommen, wenn Licht auf Objekte trifft und reflektiert wird. Diese reflektierten Lichtwellen werden von den Photorezeptoren (Zapfen) in der Netzhaut des Auges erfasst und vom Gehirn interpretiert. [HI16]

Es gibt in der Programmierung verschiedene Farbmodelle, die je nach Anwendungsbereich verwendet werden. Die wichtigsten Farbmodelle sind:

Das additive RGB-Farbmodell (Rot, Grün, Blau)

Die Farben bei dem Farbmodell RGB werden mittels der Farbmischung von Rot, Grün und Blau zusammengesetzt. Mittels dieser drei Grundfarben können verschiedene Kombinationen und damit unterschiedliche Farben erzeugen werden. Mischt man alle Grundfarben, erhält man weiß. Schaltet man alle Grundfarben ab, so erhält man Schwarz. Die Grundfarben können Werte zwischen 0 und 255 einnehmen, wodurch die Helligkeit dieser Grundfarbe eingestellt werden kann. [HI16]

- Beispiel: RGB (255, 0, 0) repräsentiert die Farbe Rot.

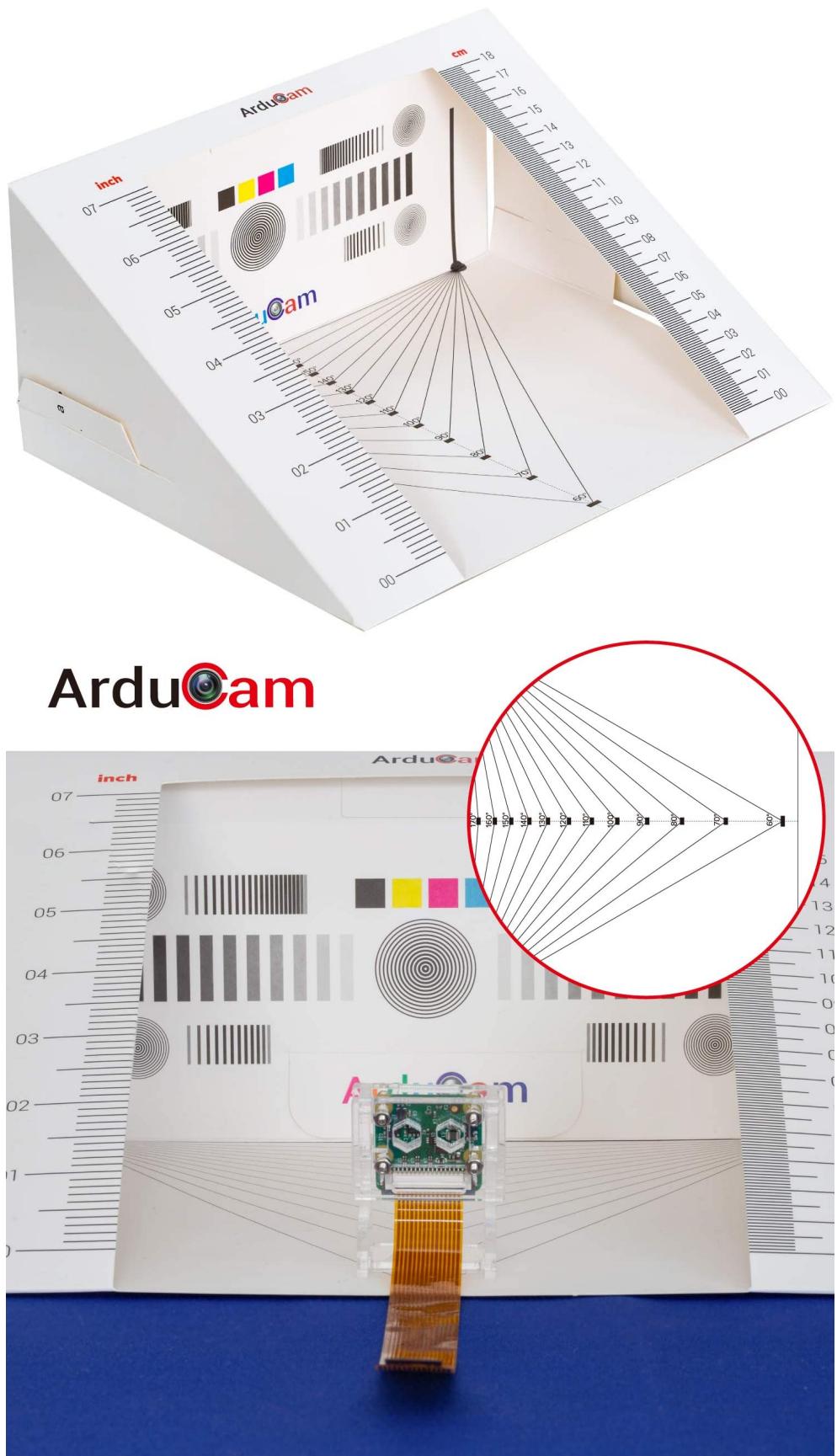


Figure 39.8.: Kalibrierungswerkzeug der Firma Arducam; [Ard21]

Das HSV Farbmodell

HSV (Hue, Saturation, Value): Ein Farbmodell, das Farben in Bezug auf Farbton, Sättigung und Helligkeit beschreibt. Dieses Modell ist besonders nützlich in der Bildbearbeitung und bei der Farbauswahl, da es eine intuitive Darstellung der Farbkomponenten bietet.

Hue (Farbton): Bestimmt die Grundfarbe und wird in Grad von 0 bis 360 angegeben.

Saturation (Sättigung): Bestimmt die Intensität der Farbe und wird als Prozentwert von 0% (Grau) bis 100% (voll gesättigte Farbe) dargestellt.

Value (Helligkeit): Bestimmt die Helligkeit der Farbe und wird ebenfalls als Prozentwert von 0% (schwarz) bis 100% (volle Helligkeit) dargestellt. [HI16]

Beispiel:

- Hue (Farbton): 0°
- Saturation (Sättigung): 50%
- Value (Helligkeit): 100%

Diese Werte beschreiben ein helles Rot im HSV-Farbmodell.

Graustufen

Graustufenerkennung bezieht sich auf die Verarbeitung von Bildern, die keine Farbinformationen enthalten, sondern nur Helligkeitswerte. Ein Graustufenbild ist ein Bild, bei dem jeder Pixel einen Helligkeitswert hat, der typischerweise in einem Bereich von 0 (schwarz) bis 255 (weiß) liegt.

Umwandlung von Farbbildern in Graustufenbilder.

Farbbilder können in Graustufenbilder umgewandelt werden, indem die Farbkanäle (Rot, Grün und Blau) kombiniert werden, um einen einzelnen Helligkeitswert zu erzeugen. Eine Möglichkeit ist das Umwandeln mit der Durchschnittsmethode, bei dem der Graustufenwert durch den Durchschnitt der RGB-Werte berechnet wird.

$$Gray = \frac{R + G + B}{3} \quad (39.1)$$

Um die Wahrnehmungsempfindlichkeit des menschlichen Auges auf Farben zu berücksichtigen, kann folgende Formel verwendet werden:

$$Gray = 0.33 \times R + 0.22 \times G + 0.17 \times B \quad (39.2)$$

Dieses Verfahren liefert in der Regel bessere Ergebnisse, da es die menschliche Wahrnehmung besser widerspiegelt. [HI16]

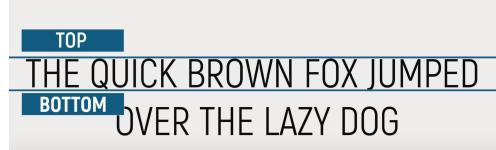


Figure 39.9.: Zeilenerkennung



Figure 39.10.: Buchstaben in binäre Matrix umwandeln

39.10.2. Texterkennung

Grundsätzlich funktioniert Texterkennung über das sogenannte “Optical character recognition” (OCR), hierfür müssen aber erstmal einige Voraussetzungen getroffen werden.

Voraussetzungen

Das Bild muss begradigt werden, dies kann hier aber vernachlässigt werden, wenn man davon ausgeht, dass die Kamera vorher in der Testumgebung kalibriert wurde. Zudem muss das Bild ggf. in ein binäres schwarz-weiß-Bild umgewandelt werden.

Identifizieren der Buchstabenmatrix

Es werden zuerst die einzelnen Zeilen identifiziert, indem nach zwei durchgehenden Reihen weißer Pixel in X-Richtung gesucht wird, zwischen welchen schwarze und weiße Pixel gemischt sind, wie in Abbildung 39.9 zu sehen ist.

Analog dazu werden auch die einzelnen Buchstaben identifiziert, indem nach zwei, durchgehend weißen Reihen in Y-Richtung gesucht wird, zwischen welchen schwarze und weiße Pixel gemischt sind.

Dann werden die einzelnen Buchstaben in eine binäre Matrix übertragen (siehe Abbildung 39.10), um sie im nächsten Schritt analysieren zu können.

Zuordnung der Buchstaben durch Sektionierung

Der Mittelpunkt der Buchstabenmatrix wird mithilfe folgender Formel errechnet.

$$d \equiv \sqrt{(Y_2 - Y_1)^2 + (X_2 - X_1)^2} \quad (39.3)$$

Mithilfe des Mittelpunkts wird die Buchstabenmatrix in mehrere kleine Sektionen geteilt, um sie besser analysieren zu können.

Eine mögliche Unterteilung in Sektionen ist in Abbildung 39.11 zu sehen.

Nun werden die einzelnen Sektionen mit der gleichen Sektion sämtlicher Buchstaben, aus sämtlichen Schriftarten verglichen und dem Buchstaben zugeordnet, der die größte statistische Übereinstimmung hat.

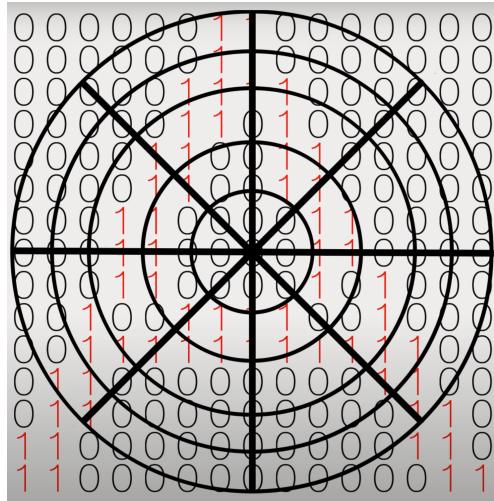


Figure 39.11.: Sektionierung der Buchstabenmatrix

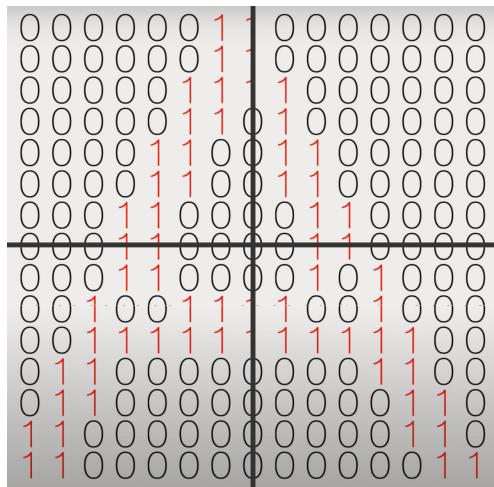


Figure 39.12.: Unterteilung der Buchstabenmatrix in Quadranten

Einfacherer Prozess der Zuordnung

Um den Prozess zu vereinfachen, lässt sich die Buchstabenmatrix auch in Quadranten teilen, anstatt der vielen Sektionen, wie in Abbildung 39.12 zu sehen.

Um dennoch relativ zuverlässige Ergebnisse zu erhalten, können folgende zwölf Merkmale genutzt werden:

F1 = Summe aller schwarzen Pixel im ersten Quadranten / Summe aller schwarzen Pixel der gesamten Buchstabenmatrix

F2 = Summe aller schwarzen Pixel im zweiten Quadranten / Summe aller schwarzen Pixel der gesamten Buchstabenmatrix

F3 = Summe aller schwarzen Pixel im dritten Quadranten / Summe aller schwarzen Pixel der gesamten Buchstabenmatrix

F4 = Summe aller schwarzen Pixel im vierten Quadranten / Summe aller schwarzen Pixel der gesamten Buchstabenmatrix

$$F5 = F1 + F2$$

$$F6 = F2 + F3$$

$$\begin{aligned}F7 &= F3 + F4 \\F8 &= F1 + F4 \\F9 &= F2 + F4 \\F10 &= F1 + F3\end{aligned}$$

F11 = Anzahl der Eckpunkte, mithilfe der Harris Corner method

F12 = Anzahl der Pixel, der konvexen Hülle um den Buchstaben / Anzahl der Pixel, der kompletten Buchstabenmatrix. [JCI14]

Dann ist das Verfahren analog zu dem der Sektionierung, es wird nach der höchsten Übereinstimmung aller Merkmale in jedem Quadranten, mit dem gleichen Quadranten sämtlicher Buchstaben, aus sämtlichen Schriftarten, gesucht.

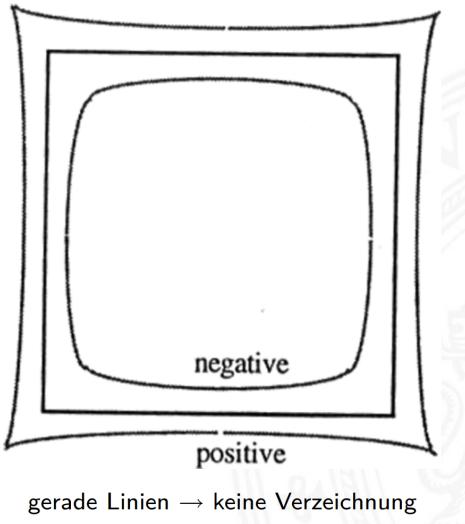


Figure 39.13.: Radiale Verzeichnung

39.10.3. Kamerakalibrierung mit Schachbrett

Die Kamerakalibrierung ist ein fundamentaler Prozess in der Computer Vision, der darauf abzielt, die Parameter einer Kamera zu bestimmen, um Verzeichnungen zu korrigieren und realitätsgerechte räumliche Informationen aus den aufgenommenen Bildern zu extrahieren. Dies ist essenziell für Anwendungen wie 3D-Rekonstruktion, Robotik, Augmented Reality und maschinelles Sehen.

Einführung

In der Computer Vision bezeichnet die Kamerakalibrierung den Prozess der Bestimmung der internen und externen Parameter einer Kamera. Die internen Parameter umfassen die Brennweite, den optischen Mittelpunkt und die Verzeichnungskoeffizienten, während die externen Parameter die Position und Orientierung der Kamera im Raum beschreiben. Diese Parameter sind notwendig, um die Geometrie der aufgenommenen Szenen korrekt zu interpretieren. [RS17a] Sturm 2017, S. 1311)

Interne Parameter

Die internen Parameter einer Kamera betreffen die Eigenschaften des Kameraobjektivs und des Bildsensors. Dazu gehören:

Brennweite (f): Bestimmt den Vergrößerungsgrad des Objektivs. Optischer Mittelpunkt (c_x, c_y): Der Punkt auf dem Bildsensor, durch den die optische Achse verläuft.

Verzeichnungskoeffizienten: Korrigieren Verzeichnungen wie die radiale Verzeichnung (siehe Abbildung 39.13) (k_1, k_2, k_3) und tangentiale Verzeichnung (siehe Abbildung 39.14) (p_1, p_2).

Die Kalibrierung dieser Parameter erfolgt typischerweise durch die Aufnahme mehrerer Bilder eines bekannten Kalibrierungsmusters, wie z. B. eines Schachbrettmusters, aus verschiedenen Blickwinkeln. Die Beziehung zwischen den erkannten Punkten im Bild und den bekannten 3D-Koordinaten des Musters ermöglicht die Schätzung der internen Parameter. [GN01]

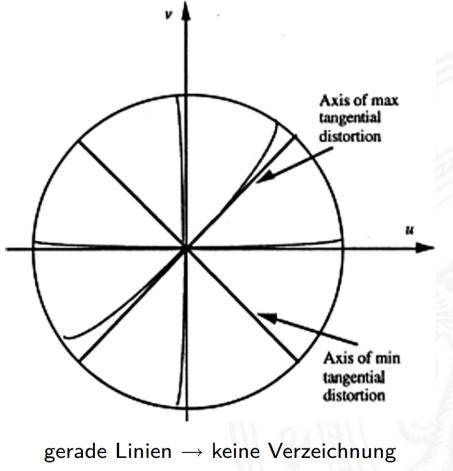


Figure 39.14.: Tangentiale Verzeichnung

Externe Parameter

Die externen Parameter beschreiben die Transformation von dem Weltkoordinatensystem zum Kamerakoordinatensystem, bestehend aus:

- Rotation (R): Beschreibt die Orientierung der Kamera.
- Translation (t): Beschreibt die Position der Kamera im Raum.
- Diese Parameter werden ebenfalls durch die Aufnahme mehrerer Bilder eines Kalibrierungsmusters bestimmt, wobei die Positionen des Musters relativ zur Kamera variiert werden.

Zhang-Kalibrierungsmethode

Ein gängiges Verfahren zur Kamerakalibrierung ist die Verwendung der Zhang-Kalibrierungsmethode, die eine lineare und nicht lineare Optimierung kombiniert, um die Kalibrierungsparameter präzise zu bestimmen. Diese Methode besteht aus folgenden Schritten:

WS:citations

1. Bildaufnahme: Mehrere Bilder des Kalibrierungsmusters aus unterschiedlichen Perspektiven
2. Merkmalserkennung: Identifikation und Zuordnung der Merkmale (Eckpunkte des Schachbrettmusters).
3. Berechnung der Homografie: Bestimmung der Projektionsmatrix, die die Beziehung zwischen 2D-Bildpunkten und 3D-Weltpunkten beschreibt.
4. Schätzung der initialen Parameter: Lineare Schätzung der internen und externen Parameter basierend auf den Homografien.
5. Nicht lineare Optimierung: Feinabstimmung der Parameter durch Minimierung der Reprojektionsfehler.

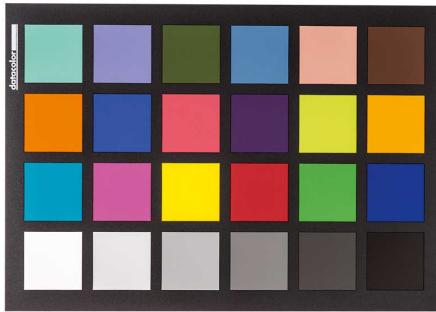


Figure 39.15.: Farbpalette

Anwendung in der Praxis

Eine genaue Kamerakalibrierung ist entscheidend für die Präzision in vielen Anwendungen der Computer Vision:

- 3D-Rekonstruktion: Erzeugung akkurater 3D-Modelle aus 2D-Bildern.
- Robotik: Präzise Navigation von Robotern.
- Augmented Reality: Überlagerung digitaler Inhalte auf reale Szenen in korrekter Perspektive.
- Maschinelles Sehen: Erkennung und Verfolgung von Objekten in Bildern und Videos.

39.10.4. Kamerakalibrierung mit Farbpaletten

Die Farbkamerakalibrierung mit Farbpaletten, siehe Abbildung 39.15, ist eine zentrale Technik in der Bildverarbeitung, welche darauf abzielt, die Farbwiedergabe von Kameras zu standardisieren und zu verbessern. Diese Methode nutzt standardisierte Farbpaletten, um die Kameras so einzustellen, dass sie Farben präzise und realitätsgerecht wiedergeben. Diese Technik ist besonders in Bereichen wichtig, in denen Farbtreue entscheidend ist, wie in der medizinischen Bildgebung, Qualitätskontrolle und Farbmessung.

Methodik

1. Aufnahme der Farbpalette

Der erste Schritt in der Kalibrierung besteht darin, Bilder der Farbpalette unter verschiedenen Beleuchtungsbedingungen aufzunehmen. Dies kann sowohl unter natürlichen als auch unter künstlichen Lichtquellen erfolgen. Wichtig ist, dass die Palette flach und gleichmäßig beleuchtet ist, um Verzeichnungen zu vermeiden.

2. Identifikation der Farbfelder

Nach der Aufnahme werden die einzelnen Farbfelder in den Bildern identifiziert. Dies kann manuell oder automatisch erfolgen, wobei Algorithmen zur Mustererkennung verwendet werden, um die Positionen der Farbfelder zu bestimmen.

3. Vergleich mit Referenzwerten

Die im Bild gemessenen Farbwerte der einzelnen Felder werden mit den bekannten Referenzwerten der Farbpalette verglichen. Dies geschieht häufig im CIE-Lab-Farbraum, der eine perpetuelle Farbskala mit gleichem Abstand bietet.

4. Berechnung der Transformationsmatrix

Basierend auf den Differenzen zwischen den gemessenen und den Referenzfarbwerten wird eine Farbtransformationsmatrix berechnet. Diese Matrix dient dazu, die Farbabweichungen zu korrigieren und die Kamera so zu kalibrieren, dass sie die Farben korrekt wiedergibt.

5. Anwendung der Farbkorrektur

Die berechnete Transformationsmatrix wird dann auf alle zukünftigen Bilder angewendet, die mit der kalibrierten Kamera aufgenommen werden. Dies stellt sicher, dass die Farben in diesen Bildern den Referenzfarben entsprechen.

Herausforderungen

- Beleuchtungseinflüsse

Eine der größten Herausforderungen bei der Farbkamerakalibrierung ist die Variabilität der Beleuchtung. Unterschiedliche Lichtquellen können die Farbwahrnehmung erheblich beeinflussen. Eine Lösung besteht darin, die Kalibrierung unter verschiedenen Beleuchtungsbedingungen durchzuführen und Beleuchtungskorrekturfaktoren zu entwickeln.

- Alterung der Farbpaletten

Farbpaletten können mit der Zeit verblassen oder sich verändern, was die Genauigkeit der Kalibrierung beeinträchtigen kann. Regelmäßige Überprüfungen und der Austausch der Farbpaletten sind notwendig, um eine konstante Kalibrierung zu gewährleisten.

- Kamera interne Verarbeitung

Moderne Kameras führen oft interne Bildverarbeitungen durch, die die Rohdaten verändern. Um dies zu umgehen, sollte die Kalibrierung möglichst auf den Rohdaten (RAW) der Kamera basieren und die Kamera interne Farbkorrekturen deaktiviert werden.

39.10.5. Bibliotheken in der Programmierung

In der Programmierung bezeichnet man Bibliotheken, eine Sammlung von Unterprogrammen, die Lösungswege für thematisch zusammengehörende Problemstellungen anbieten. Im Gegensatz zu eigenständig lauffähigen Programmen sind Bibliotheken keine eigenständigen Einheiten. Stattdessen enthalten sie Hilfsmodule, die von anderen Programmen angefordert werden.

Es ist sinnvoll Bibliotheken zu nutzen, da man thematisch ähnliche Programme nutzen kann und auf seine eigenen Programme anwenden kann. Durch die Nutzung von bereits bestehenden Programmen kann zeitsparend gearbeitet werden.

Verwendete Bibliotheken sind:

Tkinter

[Tkinter](#) ist eine Bibliothek, die es ermöglicht, grafische Benutzeroberflächen (GUIs) zu erstellen. Mithilfe von [Tkinter](#) lassen sich leicht Fenster, Schaltflächen, Textfelder und viele andere GUI-Elemente erstellen. Mit der Funktion Fenster lassen sich Hauptfenster und Dialogfenster erstellen. Widgets wie Schaltflächen, Textfelder, Listenfelder und Radiobuttons können mit [Tkinter](#) erstellt werden. Außerdem kann man mit [Tkinter](#) das Layout der verwendeten Widgets wie die Positionen und das Aussehen individuell einstellen. Es können Befehle und Ereignisse wie Mausklicks

und Tastatureingaben eingebaut werden, die das Steuern von gewissen Befehlen ermöglichen.

Benutzeroberfläche

Mit der Bibliothek Tkinter können Benutzeroberflächen erstellt werden. In diesem Fenster können unterschiedliche Widgets eingebaut werden. Der folgende Code zeigt einen grundlegenden Aufbau einer Benutzeroberfläche.

```
app = tk.Tk()
app.title("Fenstername")
app.geometry("400x300")
app.mainloop()
```

In dem Code ist `app` die Variable, die auf das Hauptfensterobjekt verweist. In der ersten Zeile wird ein neues Hauptfenster erstellt. Mithilfe von `app.title` kann der Name des Fensters geändert werden.

In der dritten Zeile kann die Größe (x-Richtung, y-Richtung) des Fensters verändert werden. In der vierten Zeile wird `app.mainloop()` eingebaut, wodurch das Fenster dauerhaft geöffnet bleibt, bis es geschlossen wird. Ohne diese Eingabe würde sich das Hauptfenster nach kurzer Zeit schließen. Es gibt noch weitere Einstellungen, womit das Hauptfenster bearbeiten werden kann.

Bedienung des Benutzers

Mit der Bibliothek Tkinter können Buttons erstellt werden, wodurch die Bedienung durch den Benutzers ermöglicht wird. Der folgende Code zeigt den grundlegenden Aufbau eines Buttons:

```
def button1():    print("Button clicked!")
    button = tk.Button(root, text="Click Me", command=button1)
    button.pack()
```

Zunächst wird eine Funktion mit dem Schlüsselwort `def`, gefolgt vom Namen der Funktion und runden Klammern, erstellt. Innerhalb der Funktion können Befehle definiert werden, die jederzeit ausgeführt werden können. Im Sample wird der Text `Button Clicked!` in der Konsole ausgegeben. Innerhalb der runden Klammern können zudem noch Parameter eingeführt werden, die als Variablen innerhalb der Funktion dienen.

Nachdem definiert ist, was passiert, wenn die Funktion ausgeführt wird, muss ebenfalls definiert werden, wann die Funktion aktiviert wird. Dazu wird, mithilfe von `Tkinter`, ein Button erstellt. Die Position, das Aussehen und der ausgeführte Command muss hierfür definiert werden. Im Sample bezieht sich `root` auf das Hauptfenster, in dem alle Widgets (auch Buttons) platziert werden. Als Nächstes wird der Name, der auf dem Button steht, mit `text=` benannt. Am Ende wird der Befehl mit `command=` definiert, welcher bei der Aktivierung des Buttons ausgeführt wird. Im Sample wird die Funktion `button1` ausgeführt.

OpenCV

OpenCV ist eine Bibliothek für die Bildverarbeitung und Computer Vision. Funktionen für die Bildverarbeitung sind zum Beispiel die Analyse von Bildern, sowie die Veränderung und Verbesserung dieser.

Mithilfe von Computer Vision-Aufgaben lässt sich Visualisierung der Umgebung erlernen. Es hilft dabei, Bilder zu verarbeiten, Objekte zu erkennen, Gesichter zu identifizieren und Bewegung zu verfolgen.

Da das Erstellen von Programmen mit Bildverarbeitung zeitlich aufwendig werden

kann, ist es sinnvoll bereits bestehende Unterprogramme für die eigene Nutzung zu verwenden.

Kameraübertragung

```
cap = cv2.VideoCapture(0)

if not cap.isOpened():
    print("Fehler: Kamera konnte nicht geöffnet
werden.")
    exit()

while True:
    ret, frame = cap.read()

    if not ret:
        print("Fehler: Frame konnte nicht gelesen
werden.")
        break

    cv2.imshow('Kamera', frame)

    if cv2.waitKey(1)
        if key == 27:
            break

    cap.release()
    cv2.destroyAllWindows()
```

Die erste Zeile öffnet die Kamera. Die Auswahl der Kamera kann mit der runden Klammer gesteuert werden. Die Zahl Null in den runden Klammern steht für die erste angeschlossene Kamera. Die dritte Zeile überprüft, ob die Kamera geöffnet werden kann. Wenn sie nicht geöffnet werden, wird eine Fehlermeldung ausgegeben und das Programm wird mit "exit()" beendet. Als Nächstes wird eine Schleife verwendet, um fortlaufende Frames zu lesen. Der Wert "ret" gibt an, ob das Lesen funktioniert hat und "frame" enthält das aktuelle Bild. Zeile zehn überprüft, ob der aktuelle Frame erfolgreich gelesen werden konnte. War die Überprüfung nicht erfolgreich, wird eine Fehlermeldung ausgegeben und die Schleife wird beendet. Die Zeile 14 zeigt den aktuellen Frame in einem Fenster an. Im Sample wird das Bild in dem Fenster "Kamera" wiedergegeben. Als Nächstes erfolgt eine Überprüfung der Tasteneingabe, wodurch die Schleife und damit auch die Bildanzeige geschlossen werden kann. Im Sample wird überprüft, ob die Taste "Esc" gedrückt wurde, wodurch die Schleife beendet werden würde. Die letzten beiden Zeilen des Codes geben die Kamera frei und schließen alle geöffneten Fenster, sobald das Programm abgebrochen wird.

Bilderfassung

```
if key == ord('c'):
    filename = "captured_image.png"
    cv2.imwrite(filename, frame)
    print(f"Bild wurde als filename
gespeichert.")
    break
```

Der Code startet mit einer Bedingung, bei der überprüft wird, ob die Taste "c" gedrückt wurde. Wird die Taste "C" gedrückt, wird der folgende Code ausgeführt. Zuerst wird der Dateiname des Bildes festgelegt, um das aufgenommene Bild zu speichern. Der Frame, welcher bei der Eingabe zu sehen war, wird als Bild mit dem Dateinamen "filename" gespeichert. Daraufhin erfolgt eine Bestätigung in der Konsole und die aktuelle Schleife wird mit "break" beendet.

Bildspeicherung

```
filename =
f"captured_image_datetime.now().strftime(\"")
.png" cv2.imwrite(filename, frame)
print(f"Bild gespeichert als filename")
```

In der ersten Zeile wird der Dateiname erstellt, der das Datum und die Uhrzeit der Bildaufnahme enthält. In der zweiten Zeile wird das aufgenommene Bild gespeichert. Der Name der Datei ist „filename“, in die das Bild gespeichert werden soll. Der Name „frame“ bezeichnet das Bild, das gespeichert werden soll. Das Bild wurde vorher mit „ret, frame = cap.read()“ aufgenommen und in der Variable „frame“ gespeichert.

Farberkennung

```

hsv_frame = cv2.cvtColor(frame,
cv2.COLOR_BGR2HSV)

height, width, _ = frame.shape

cx = int(width / 2)
cy = int(height / 2)

pixel_center = hsv_frame[cy, cx]
hue_value = pixel_center[0]

color = "Undefined"
if hue_value < 5:
    color = "RED"
elif hue_value < 78:
    color = "GREEN"
elif hue_value < 131:
    color = "BLUE"

else:
    color = "RED"

pixel_center_bgr = frame[cy, cx]
b, g, r = int(pixel_center_bgr[0]),
int(pixel_center_bgr[1]),
int(pixel_center_bgr[2])

cv2.rectangle(frame, (cx - 220, 10), (cx + 200, 120), (255, 255, 255), -1)
cv2.putText(frame, color, (cx - 200, 100), 0,
3, (b, g, r), 5)
cv2.circle(frame, (cx, cy), 5, (25, 25, 25), 3)

cv2.imshow("Frame", frame)
key = cv2.waitKey(1)
if key == 27:
    break

```

In dem Code wird die Farbe eines Pixels im Zentrum der Kamera analysiert und die erkannte Farbe angezeigt. Der Code beginnt mit der Konvertierung des Bildes von BGR (Blue, Green, Red) zu HSV (Hue, Saturation, Value) Farbschema. Durch die Konvertierung wird die Analyse der Farbe erleichtert, da im HSV-Farbmodell der Farbton getrennt von der Helligkeit betrachtet wird. Als Nächstes wird die Höhe, Breite und die Anzahl der Farbkanäle des Bildes bestimmt. Daraufhin werden die X- und Y-Koordinaten des Bildzentrums berechnet, damit im folgenden Schritt der HSV-Wert des zentralen Pixels extrahiert werden kann. Aus dem HSV-Wert wird der Farbton (Hue) des zentralen Pixels extrahiert. Als Nächstes folgt eine Überprüfung unterschiedlicher Bedingungen, um den Farbtonwert "hue_value" zu analysieren und die Farbe entsprechend einzuteilen. Die Einteilung der Farben entspricht den Farbtönen im HSV-Farbmodell.

Außerdem wird der BGR Wert des zentralen Pixels des Bildes bestimmt, um die Textausgabe auf die entsprechende Farbe abzugleichen. Die nächsten Zeilen im Code dienen dazu, ein Kreis in der Mitte des Bildes einzufügen, welcher für die Bestimmung

der Farbe ausgewählt wurde. Außerdem wird in der oberen linken Ecke ein Text mit dem erkannten Farbnamen eingefügt. Am Ende kann das Programm wieder mit der Tasteneingabe "Esc" beendet werden.

Graustufenerkennung

```
def graustufe_prozent(image, x, y):
gray_image = cv2.cvtColor(image,
cv2.COLOR_BGR2GRAY)
grayscale_value = gray_image[y, x]
grayscale_percentage = (255 -
grayscale_value) / 255 * 100
return grayscale_percentage

def graustufenerkennung(kamera_nummer=1):
cap = cv2.VideoCapture(kamera_nummer)

while True:
ret, frame = cap.read()
if not ret:
break

height, width, _ = frame.shape
center_x, center_y = width // 2, height // 2

grayscale_percentage =
get_grayscale_percentage
(frame, center_x, center_y)

grayscale_text = f"Graustufenwert:
grayscale_percentage:.2f%"

cv2.circle(frame, (center_x, center_y), 5,
(0, 255, 0), 2)
text_color = (200, 100, 0)

cv2.putText(frame, grayscale_text, (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 0.7, text_color, 2, cv2.LINE_AA)

cv2.imshow('Grayscale Detection', frame)

if cv2.waitKey(1) & 0xFF == ord('27'):
break

cap.release()
cv2.destroyAllWindows()
```

Die Funktion "graustufe_prozent" konvertiert das aktuelle Bild in Graustufen um. Der Graustufenwert der Pixel an den voreingestellten Koordinaten (x,y) wird bestimmt und dann der Graustufen-Prozentsatz (0% für Weiß und 100% für Schwarz) berechnet. Die Hauptfunktion "graustufenerkennung" öffnet die Kamera und erfasst, während die Schleife läuft, dauerhaft Frames aus dieser Quelle. Für jedes erfasste Frame wird im folgenden Schritt der Graustufen-Prozentsatz in der Mitte des Bildes, mithilfe der ersten Funktion, berechnet und angezeigt. Die Schleife, und damit das Programm,

können mittels der Tasteneingabe "Esc" geschlossen werden.

Easyocr

Easyocr ist eine Python-Bibliothek, die OCR (Optical Character Recognition) Funktionalität bietet. Der Import von Easyocr ermöglicht es, OCR im Python-Code zu verwenden, um Text aus Bildern zu extrahieren.

Texterkennung

```
reader = easyocr.Reader(['de'])

result = reader.readtext(image)

print("Erkannter Text im Bild:")
for (bbox, text, prob) in result:
    print(f"text (Wahrscheinlichkeit: {prob:.2f})")

img_with_text = np.copy(image)
for (bbox, text, prob) in result:

    cv2.putText(img_with_text, text,
               (int(bbox[0][0]),
                int(bbox[0][1]) - 10),
               cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
```

In der ersten Zeile wird ein Easyocr-Objekt initialisiert, das darauf trainiert ist, Text in Bildern zu erkennen. In den eckigen Klammern kann die Sprache, des zu erkennenden Textes, eingestellt werden. Das "de" wird benötigt, damit deutschsprachige Texte erkannt werden. Im nächsten Schritt wird das übertragene Bild "image" auf Text analysiert, welches in der Variable "result" gespeichert wird. Dabei handelt es sich um eine Liste von Tuplen, wobei jede Tuple, den Begrenzungsrahmen, den erkannten Text und die entsprechende Wahrscheinlichkeit enthält. Die Informationen aus jeder Tuple werden mithilfe einer Schleife durchlaufen und ausgegeben. Als Nächstes wird eine Kopie des Originalbildes erstellt, um den erkannten Text darauf abzubilden. Mithilfe einer weiteren Schleife wird jedes Tuple durchgegangen und Informationen auf das Bild übertragen.

Datetime

Die Bibliothek Datetime in Python bietet eine Reihe von Funktionen, die es ermöglichen, mit Datums- und Zeitangaben zu arbeiten.

```
f"captured_image_datetime.now().strftime('%Y%m%d%H%M%S').png"
```

Die gegebene Zeile erzeugt einen Dateinamen für ein Bild, welches das aktuelle Datum und die aktuelle Uhrzeit enthält.

Numpy

Die Bibliothek Numpy ist eine leistungsstarke Bibliothek für numerische Berechnungen. Sie bietet Unterstützung für große, mehrdimensionale Arrays und Matrizen, zusammen mit einer Vielzahl von mathematischen Funktionen, um effizient über diese Arrays zu

arbeiten.

Texterkennung

```
img_with_text = np.copy(image)
```

Im Abschnitt der Texterkennung wird die Bibliothek Numpy verwendet, um eine Kopie des Bildes zu erstellen, damit das Originalbild unverändert bleibt, während der erkannte Text in die Kopie übermittelt wird.

Kalibrierung

```
objp = np.zeros((chessboard_size[0] *  
chessboard_size[1], 3),  
np.float32)  
objp[:, :2] = np.mgrid[0:chessboard_size[0],  
0:chessboard_size[1]].T.reshape(-1, 2)  
  
np.savez('calibration_data.npz',  
**calibration_data)  
  
data = np.load('calibration_data.npz')  
  
known_colors = np.array([ [115, 82, 68], [194, 150, 130], [98, 122,  
157], [87, 108, 67], ...], dtype=np.float32)
```

Im Abschnitt der Kalibrierung wird die Bibliothek Numpy verwendet, um zuerst die bekannten 3D-Koordinaten, der Schachbrett-Eckpunkte, im Raum darzustellen. Dabei wird Numpy verwendet, da die Arbeit mit vielen Datenpunkten effektiver durchgeführt werden kann.

Als Nächstes wird Numpy verwendet, um die Kalibrierungsdaten zu speichern und zu laden. Die Kalibrierungsdaten (Kameramatrix und Verzeichnungskoeffizienten) werden in einer komprimierten ".npz"- Datei gespeichert. Außerdem wird die Bibliothek verwendet, damit die bekannten Farben der Colorcards in einem mehrdimensionalen Array gespeichert werden können.

Os

Die Bibliothek Os bietet eine Vielzahl von Funktionen zum Interagieren mit dem Betriebssystem. Diese Funktionen ermöglichen es, Dateisystemoperationen, Umgebungsvariablen und Prozesse zu verwalten.

```
directory = "dateipfad"
if not os.path.exists(directory):
    os.makedirs(directory)

filename = os.path.join(directory,
f"captured_image_datetime.now().strftime('%Y%m%d%H%M%S').png")
cv2.imwrite(filename, img_with_text)
print(f"Bild mit erkanntem Text wurde
gespeichert als filename.")
```

Der Code beginnt mit der Überprüfung, ob das Verzeichnis "directory" existiert. Falls es nicht existiert, wird es von "os.makedirs(directory)" erstellt. Als Nächstes wird das Verzeichnis mit dem Dateinamen zu einem vollständigen Pfad verbunden. Dann wird das Bild, mithilfe der OpenCV Bibliothek, gespeichert. Am Ende wird eine Bestätigungs Nachricht ausgegeben.

40. Grundlagen der Kamerakalibrierung

Zu Beginn werden die Grundlagen der Kamerakalibrierung, die notwendig sind, um die nachfolgenden Kamerakalibrierungsmethoden zu verstehen, beschrieben [Gra14]. Was versteht man eigentlich unter dem Begriff Kamera und Kalibrierung? Dies wird für folgende Arbeit definiert.

Unter dem Begriff Kamera, versteht man ein optisches Gerät, das in der Lage ist Lichtwellen einzufangen und daraus ein digitales Bild entstehen zu lassen. Darauf wird der Fokus in dieser Arbeit gelegt. Weitere Kameraarten, wie die Wärmebild-, Röntgen- oder Ultraviolettkameras, sind nicht Gegenstand der Arbeit.

Unter dem Begriff der Kalibrierung versteht man den Prozess des Abgleichs der Soll- / Istwerte, um damit eine Korrektur / Anpassung der Kameraparameter zu generieren. Ziel ist, dass das aufgenommene digitale Bild möglichst genau die Realität widerspiegelt.

40.1. Funktionsweise einer Kamera

Um die abstrakte Theorie der Kamerakalibrierung besser nachvollziehen zu können, muss man erst einmal verstehen, wie eine Kamera aufgebaut ist und wie sie funktioniert. Im Folgenden werden die wesentlichen Komponenten und deren Rollen im Prozess beschrieben (siehe Abbildung 40.1). Diese ähneln dem Prozess des „Sehens“ vom Menschen.

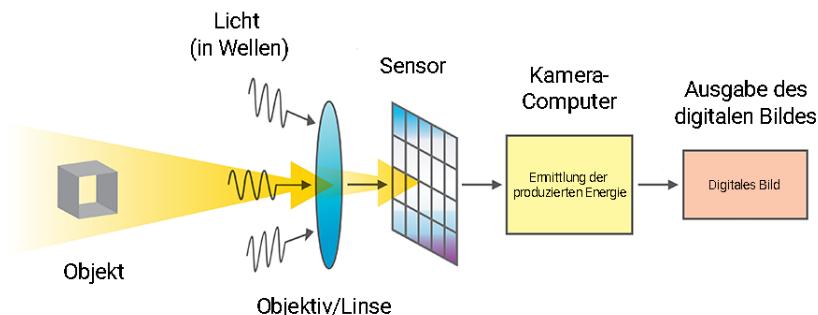


Figure 40.1.: Prinzipdarstellung der Funktionsweise einer Kamera
[Zim21]

Eine Lichtquelle (Sonne oder künstliche Beleuchtung) schickt Lichtwellen aus, die sich als elektromagnetische Wellen ausbreiten. Diese treffen auf das Objekt.[Run24] Das **Objekt** ist die physische Abbildung in der Realität, die später als digitales Bild eingefangen und ausgegeben werden soll. Sie ist also der Ausgangspunkt des Bildaufnahmeprozesses. Diese Lichtwellen werden teilweise reflektiert oder vom Objekt absorbiert, je nachdem, welche Eigenschaften das Objekt hinsichtlich des Materials, der Farbe, der Textur und der Form besitzt.

Die reflektierenden Lichtwellen, mit den Eigenschaftsinformationen treffen dann auf das **Objektiv**. Ein Objektiv besteht aus mehreren unterschiedlich geformten Linsen, die das Licht sammeln, fokussieren und brechen, um es auf den Bildsensor zu projizieren. Die

Brennweite

(Abstände der Linsen zum Sensor), die Blendenöffnung (Größe der Öffnung durch das Licht eindringen kann) und die Belichtungszeit (bestimmt die Helligkeit), bestimmen die Tiefenschärfe, den Bildausschnitt, die Lichtmenge, die Verzerrungen sowie die Bildschärfe. Damit wird eingestellt, welche Informationen auf den Sensor fallen.

Der **Bildsensor** fängt das Licht ein und wandelt es in ein elektrisches Signal um, das über dem Computer zum digitalen Bild umgewandelt wird. Die häufigsten Sensoren, die zum Einsatz kommen sind: CCD-Sensor (Charge-Coupled Device) und CMOS-Sensor (Complementary Metal-Oxide-Semiconductor). [Run24] Beide beruhen auf demselben Prinzip. Die Lichtwellen fallen dabei auf dem Bildsensor, der aus vielen lichtempfindlichen Pixeln besteht. Diese Pixel sind jedoch nur in der Lage Helligkeitswerte (Graustufen) aufzunehmen. Damit trotzdem Farbe dargestellt werden kann, wird ein Farbfilter, z.B. ein Bayer-Filter (siehe Abbildung 40.2) vor dem Sensor platziert.

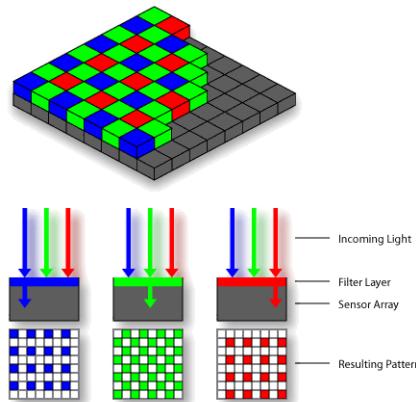


Figure 40.2.: Darstellung des Bayer-Filters Prinzips
[Run24]

Beim Bayer-Filter wird jedem Pixel eine Farbe, also Rot, Grün oder Blau, zugeordnet und deren Intensität in Volt gemessen. Genau genommen kann ein blauer Pixel nur die blauen Lichtwellen aufnehmen. Die anderen Lichtwellen werden nicht durch den Filter zum Sensor durchgelassen.

Der **Computer** wandelt dann das elektrische analoge Signal in ein digitales Signal (Analog-Digital-Wandler) um, indem das „RGB-Mosaik“ mit den entsprechenden Helligkeitswerten zu einem RGB-Farbwert umgerechnet wird.

Die dafür erforderliche Interpolation [Run24] wird in der Abbildung 40.3 dargestellt und ist in die folgenden 3 Fälle untergliedert:

- **Rotes Pixel in der Mitte von 9 Pixeln**

Der neue Rot-Wert ist gleich dem roten Pixel-Wert. Der neue Blau-Wert ergibt sich als Durchschnitt der 4 blauen diagonalen Nachbarpixel B_1, \dots, B_4 . Der neue Grün-Wert ergibt sich als Durchschnitt der 4 grünen direkten Nachbarpixel G_1, \dots, G_4 .

- **Grünes Pixel in der Mitte von 5 Pixeln**

Der neue Grün-Wert ist gleich dem grünen Pixel-Wert. Der neue Blau-Wert ergibt sich als Durchschnitt der 2 blauen horizontalen Nachbarpixel B_1, B_2 . Der neue Rot-Wert ergibt sich als Durchschnitt der 2 roten vertikalen Nachbarpixel R_1, R_2 .

- **Blaues Pixel in der Mitte von 9 Pixeln**

Der neue Blau-Wert ist gleich dem blauen Pixel-Wert. Der neue Rot-Wert ergibt sich als Durchschnitt der 4 roten diagonalen Nachbarpixel R_1, \dots, R_4 . Der neue Grün-Wert ergibt sich als Durchschnitt der 4 grünen direkten Nachbarpixel G_1, \dots, G_4 .



Figure 40.3.: Darstellung der Bayer-Mosaik-Farbinterpolation einzelner Farben [Vis24]

Vorab führen Kameras noch einen internen Weißabgleich, Rauschunterdrückung und Gamma-Korrektur durch, um die Werte zu optimieren. Diese sind bei der eigentlichen Kalibrierung abzuschalten.

Das Endergebnis ist dann eine Matrix aus grauen oder farbigen Pixeln, die als **digitales Bild** wahrgenommen werden. Die Qualität dieses Bildes hängt von der Auflösung (Anzahl der Pixel), des Bildsensors und der folgenden Berechnungen im Computer ab.

40.2. Mathematische Grundlagen: Lösung des nichtlinearen Kleinste-Quadrat-Problems

Da die eigentliche Kalibrierung modellübergreifend häufig zu einem nichtlinearen Kleinste-Quadrat-Problem führt, soll dies in diesem Abschnitt zunächst allgemein dargestellt werden.

Gesucht ist im allgemeinen eine nichtlineare Funktion f , die die IST-Werte einer Kamera möglichst gut auf die Soll-Werte anpasst. Diese Ansatzfunktion wird durch m verschiedene Abbildungsparameter θ_i bestimmt. Diese Parameter sind zusammengefasst in dem Vektor $\theta \in \mathbb{R}^m$. Aus diesem noch zu berechnenden Vektor werden die Einträge der späteren Korrekturmatrixt hergeleitet. Als Ansatz für diese nichtlineare Funktion f wird häufig eine Potenzreihe n-ter Ordnung gewählt (s. [Kue08, S.23 f.]). Entsprechend dem jeweiligen Modell werden durch n verschiedene Messungen die Ist-Werte $IST(i)$ der i-ten Messung erfasst und anschließend der Abstand von $f(IST(i))$ zum bekannten Soll-Wert $SOLL(i)$ gemessen. Danach werden die n Abstände aufsummiert.

Die Minimierung dieser Zielfunktion liefert die Lösungsparameter, die zu einer minimalen Abweichung der Ist- von den Sollwerten führen und somit die neuen Parameterwerte zur Kamerakalibrierung. Die Abstandsmessung der Vektoren wird gewöhnlich in der quadrierten euklidischen Norm $\|\cdot\|_2$ vollzogen, um die Differenzierbarkeit der Zielfunktion sicherzustellen. Damit können effiziente numerische Optimierungsalgorithmen verwendet werden [DS83].

$$\min_{\theta \in \mathbb{R}^m} \sum_{i=1}^n \|f((IST(i)) - SOLL(i)\|_2^2$$

Dieses nichtlineare Kleinste-Quadrat-Problem (Least-Square-Problem) kann nun näherungsweise mit dem Gauß-Newton-Verfahren oder dem Levenberg-Marquardt-Algorithmus berechnet werden. Dabei ist der Levenberg-Marquardt-Algorithmus eine Kombination des Gauß-Newton-Verfahrens mit dem Gradienten-Verfahren. Der Levenberg-Marquardt-Algorithmus geht dabei lokal ins Gauß-Newton-Verfahren über, ist aber robuster, d.h. mit größerer Wahrscheinlichkeit wird auch bei schlechteren Startwerten eine Konvergenz sichergestellt.

Da bei der Kalibrierung die Parameter jedoch nicht neu bestimmt, sondern nur angepasst werden, sollten die Parameter der Voreinstellung wohl gute Startwerte liefern. Damit ist eine Konvergenz zu erwarten. Voraussichtlich werden somit weniger Iterationen benötigt, um eine zufriedenstellende Lösung zu berechnen. Eine ausführliche Darstellung beider Algorithmen ist im Buch [DS83] dargestellt.

Bei den einzelnen Kalibrierungsmethoden ist nun festzulegen, welche Arten von Kamerawerten (z.B. Farbwerte) zu kalibrieren und in welcher Art und Weise diese für den Soll-/Ist-Abgleich zu erfassen sind.

41. Farbkalibrierung und Farberkennung

Die Farbkalibrierung spielt eine entscheidende Rolle bei der realitätsnahen Darstellung von Farben in digitalen Bildern. Unterschiedliche Kameras, Lichtquellen und Einstellungen können dazu führen, dass Abweichungen in der Farbdarstellung entstehen. Dies kann zu vielfältigen Problemen beispielsweise in der industriellen Fertigung führen, wenn in der Qualitätskontrolle die Kamera eingesetzt wird, um Farbfehler bei Produkten zu erkennen. Bei einer schlecht kalibrierten Kamera könnte es dazu führen, dass gute Produkte aussortiert und schlechte in den Verkauf gelangen. In der Medizintechnik könnten schlecht kalibrierte Kameras auch Fehler in der Diagnostik verursachen, die zu Fehlbehandlungen führen.

41.1. Farbkalibrierung durch Farbkarten

Eine Farbkarte ist ein Referenzprodukt, um Farbabweichungen zu identifizieren. Diese besitzt standardisierte Referenzfarben, die mit den aufgenommenen Farben verglichen werden können. Als Beispiel sei hier die GretagMacbeth ColorChecker (siehe Abbildung 41.1) erwähnt, die bereits im Jahre 1976 eingeführt wurde und aus 24 Farbquadrate in einem schwarzen Rahmen besteht. Deren Farbwerte sind in der Tabelle 41.1 explizit angegeben.

Moderne Kameras führen oft bereits interne Bildverarbeitungen durch, die die Rohdaten verändern. Diese sollte vorab intern deaktiviert werden, bevor man mit der Farbkalibrierung beginnt.

Zu Beginn der Kalibrierung wird nun eine passende Farbkarte - wie z.B. die Gretag-Macbeth ColorChecker - ausgewählt. Da Farbpaletten mit der Zeit verblassen und somit nicht mehr den Referenzfarben entsprechen, sollte der Zustand vorab kontrolliert und die Palette ggf. erneuert werden.

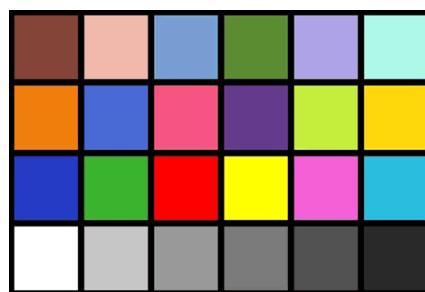


Figure 41.1.: Farbkarte GretagMacbeth ColorChecker [Bal24]

Anschließend wird mit der Kamera, bzw. mit dem Bildsensor (s. Abschnitt 40.1) ein möglichst präzises Bild der Farbpalette aufgenommen. Da unterschiedliche Lichtquellen die Farbwahrnehmung erheblich beeinflussen können, stellt die Variabilität der Beleuchtung eine erhebliche Herausforderung dar. Eine Lösung kann darin bestehen, die Kalibrierung unter verschiedenen Beleuchtungsbedingungen durchzuführen und Beleuchtungskorrekturfaktoren zu entwickeln. Ein weiterer Ansatz

Original Munsell Notation CIE 1976 L*a*b*, D50, 2°					
Name	Hue	Value/Chroma	L*	a*	b*
Dark skin	3.05 YR	3.69/3.20	39.07	13.70	14.37
Light skin	2.2 YR	6.47/4.10	66.36	18.08	18.56
Blue sky	4.3 PB	4.95/5.55	50.66	-4.84	-21.44
Foliage	6.65 GY	4.19/4.15	44.03	-13.01	22.34
Blue flower	9.65 PB	5.47/6.70	55.87	8.88	-24.79
Bluish green	2.5 BG	7/6	71.47	-32.98	0.65
Orange	5 YR	6/11	62.37	36.05	56.58
Purplish blue	7.5 PB	4/10.7	40.77	9.18	-43.58
Moderate red	2.5 R	5/10	51.68	48.21	16.75
Purple	5 P	3/7	31.27	20.00	-20.83
Yellow green	5 GY	7.08/9.1	72.74	-22.60	57.15
Orange yellow	10 YR	7/10.5	72.35	19.57	68.76
Blue	7.5 PB	2.90/12.75	29.61	13.63	-49.65
Green	0.1 G	5.38/9.65	55.61	-37.19	31.81
Red	5 R	4/12	42.23	55.32	27.45
Yellow	5 Y	8/11.1	82.65	4.44	80.52
Magenta	2.5 RP	5/12	52.55	49.33	-14.42
Cyan	5 B	5/8	52.24	-28.24	-27.14
White	N	9.5/	96.37	-0.30	3.26
Neutral 8	N	8/	81.70	-0.56	0.25
Neutral 6.5	N	6.5/	66.49	-0.33	0.03
Neutral 5	N	5/	50.67	-1.06	-0.19
Neutral 3.5	N	3.5/	36.23	-0.48	-0.26
Black	N	2/	20.68	0.17	-0.55

Table 41.1.: Farbwerte des GretagMacbeth ColorCheckers nach [Mye02]

[Kue08] ist den Beleuchtungsabstand möglichst groß zu halten und ggf. polarisiertes Licht zu verwenden.

Nach der Aufnahme werden nun die einzelnen Farbfelder in den Bildern manuell oder automatisch identifiziert, um die Positionen der Farbfelder zu bestimmen.

Die Kalibrierung findet dann manuell durch Eingabe der Werte in die Kamera oder über eine externe Software wie z.B. Photoshop statt. Alternativ kann die Kalibrierung auch automatisiert über die Kamera realisiert werden.

Da es trotzdem nicht möglich ist, ein ideales Kamerabild zu erhalten, bei dem die Helligkeit gleichmäßig konstant über die Farbpalette verteilt ist, werden die Aufnahmen auf verschiedene Weisen messtechnisch vorbearbeitet. Diese Methoden, wie die Gamma Korrektur oder die Farbkontrasteinstellung mit einem Schwarzabgleich [Bal24], werden auf das ursprüngliche Kamerabild angewendet.

41.2. Berechnung des Weißabgleichs

Exemplarisch wird an dieser Stelle der Weißabgleich, als einer dieser Methoden, ausführlich dargestellt [Roh12]. Da das Licht verschiedener Lichtquellen und Tageszeiten unterschiedliche Farben hat, entspricht auch der Weißton des ursprünglichen Bildes nicht dem theoretischen Weiß, also die Summe aller Farben in RGB. Das Bild besitzt somit einen Farbstich, der mit dem Weißabgleich behoben wird.

Dazu wird ein weißer Referenzpunkt aus dem ursprünglichen Bild mit den RGB-Werten

$$\begin{pmatrix} R_w^{alt} \\ G_w^{alt} \\ B_w^{alt} \end{pmatrix}.$$

entnommen. Alternativ ist es auch möglich, mehrere weiße Punkte zu wählen und deren durchschnittliche Rot-, Blau- sowie Grünwerte als Referenzpunkt zu definieren. Nun werden alle alten Werte $(R^{alt}, G^{alt}, B^{alt})^T$ durch folgende Matrix-Multiplikation in die neuen RGB-Werte $(R^{neu}, G^{neu}, B^{neu})^T$ transformiert:

$$\begin{pmatrix} R^{neu} \\ G^{neu} \\ B^{neu} \end{pmatrix} = \begin{pmatrix} \frac{255}{R_w^{alt}} & 0 & 0 \\ 0 & \frac{255}{G_w^{alt}} & 0 \\ 0 & 0 & \frac{255}{B_w^{alt}} \end{pmatrix} \begin{pmatrix} R_w^{alt} \\ G_w^{alt} \\ B_w^{alt} \end{pmatrix}.$$

Durch diese Skalierung wird sichergestellt, dass der ursprüngliche weiße Referenzpunkt auch dem theoretischen RGW-Weiß von

$$\begin{pmatrix} R_w^{neu} \\ G_w^{neu} \\ B_w^{neu} \end{pmatrix} = \begin{pmatrix} \frac{255}{R_w^{alt}} & 0 & 0 \\ 0 & \frac{255}{G_w^{alt}} & 0 \\ 0 & 0 & \frac{255}{B_w^{alt}} \end{pmatrix} \begin{pmatrix} R_w^{alt} \\ G_w^{alt} \\ B_w^{alt} \end{pmatrix} = \begin{pmatrix} 255 \\ 255 \\ 255 \end{pmatrix}$$

entspricht.

41.3. Berechnung der Farbkorrektur-Matrix

Während diese ersten Korrekturschritte mathematisch teilweise durch lineare Transformationen realisiert werden, basiert die folgende Berechnung der Farbkorrekturmatrixt auf ein nichtlineares Optimierungsproblem.

Da sich Farträume durch Transformationen ineinander überführen lassen, ist eine Optimierung grundsätzlich in jedem Farbraum möglich (s. Abbildung 41.2). Bei der Berechnung muss lediglich sichergestellt werden, dass sowohl die Ist- als auch die Soll-Werte im gleichen Farbraum, beispielsweise im CIE-Lab-Farbraum, vorliegen. Ggf. müsste eine Transformation vorgeschaltet werden. Die Transformation von einer RGB-Farbe in den entsprechenden CIE-Koordinaten ist beispielweise in [SR14, S. 246 ff] ausführlich dargestellt.

Jede Farbe in einem beliebigen Farbraum lässt sich durch einen Vektor mit 3 Komponenten darstellen. D.h. bei n unterschiedlichen Farben der ursprünglichen Farbpalette, stehen sowohl für die Ist- als auch für die Sollwerte $3n$ Werte zum Abgleich zur Verfügung.

Für die eigentliche Kalibrierung werden erstmalig die allgemeinen Erläuterungen des Abschnittes 40.2 genutzt, bezogen auf die Farbwerte. Der i-te Ist-Vektor $IST_{Farbe}(i) \in \mathbb{R}^3$ besteht aus den bereits korrigierten i-ten Farbwerten des ursprünglichen Kamerabildes. Der i-te Soll-Vektor $SOLL_{Farbe}(i) \in \mathbb{R}^3$ besteht aus den bekannten Farbwerten der zugrundeliegenden Farbpalette. Dabei liegen alle Werte – ggf. nach vollzogener Transformation – in den Koordinaten des Farbraums vor, der nun für die Kleinst-Quadrat-Approximation verwendet wird.

Da n unterschiedliche Farben der ursprünglichen Farbpalette abgeglichen werden, läuft die Minimierung über die Summierung von 1 bis n :

$$\min_{\theta \in \mathbb{R}^m} \sum_{i=1}^n \|f((IST_{Farbe}(i)) - SOLL_{Farbe}(i))\|_2^2.$$

Als Approximationsfunktion $f : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ ist grundsätzlich jede nichtlineare Funktion, beispielsweise ein Polynom vom Grad n , anwendbar [Kue08] und die Algorithmen aus

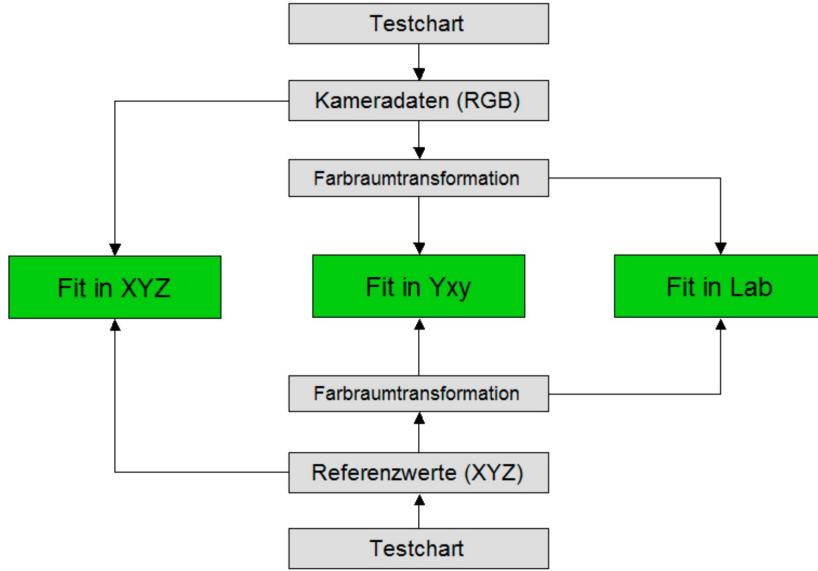


Figure 41.2.: Optimierung in unterschiedlichen Farträumen [Kue08, S. 24]

Abschnitt 40.2 zur Optimierung nutzbar. Eine einfache Möglichkeit wäre auch die Wahl einer linearen Abbildung mit einer Matrix $C \in \mathbb{R}^{3 \times 3}$. Dies führt zum linearen Kleinsten-Quadrate-Problem, das auch durch einfachere Algorithmen berechenbar ist. Als Ergebnis erhält man die Farbkorrekturmatrixt C , die auch häufig als White-point Preserving Least Square (WPPLS) bezeichnet wird. Bei Bedarf kann vor der Farbkalibrierung noch der Parameter für die Farbsättigung eingestellt sowie der Farbraum des Anzeigegerätes berücksichtigt werden. Dies geschieht wiederum in Form einer linearen Abbildung, also einer Matrix-Multiplikation. In diesen Fällen würde sich die Farbkorrektur-Matrix als Multiplikation dieser 3 Matrizen ergeben [Bal24].

Schließlich wird diese Farbkorrekturmatrixt auf die RGB-Werte der Kamera angewendet, um die Farbabweichungen zu korrigieren. Das heißt, die korrigierten RGB-Werte $(R_k, G_k, B_k)^T$ lassen sich als Matrix-Vektor-Multiplikation der Farbkorrekturmatrixt $C \in \mathbb{R}^{3 \times 3}$ mit den ursprünglichen RGB-Vektor berechnen:

$$\begin{pmatrix} R_k \\ G_k \\ B_k \end{pmatrix} = C \begin{pmatrix} R \\ G \\ B \end{pmatrix}.$$

42. Graustufenkalibrierung und Graustufenerkennung

In einigen Anwendungsbereichen – wie z.B. in der biomedizinische Bildanalyse [Kap24] – wird eine Darstellung in Graustufen verwendet. Um die Funktionalität dieser Geräte sicherzustellen, müssen diese in zeitlichen Abständen immer wieder kalibriert werden. Im Gegensatz zu Farbwerten haben Graustufen keine Farbinformationen, sondern sind nur durch ihre Helligkeitswerte definiert. Dabei wird nur ein Byte zur Darstellung des Grauwertes verwendet, d.h. die Zahl 0 (Schwarz) und die Zahl 255 (Weiß) zugewiesen. Farbbilder können auch in Graubilder umgewandelt werden, indem die Werte der RGB-Farben kombiniert werden, um die einzelnen Helligkeitswerte zu berechnen. Theoretisch geschieht dies durch eine einfache Durchschnittsbildung:

$$Grau = \frac{R + G + B}{3}.$$

Dies führt jedoch zu einem nicht zufriedenstellenden Ergebnis, da das menschliche Auge Helligkeit nicht linear empfindet. Aufgrund einer Vielzahl von theoretischen Untersuchungen und psychologischen Tests zum Helligkeitsempfinden von Menschen wird daher die folgende gewichtete Durchschnittsbildung allgemein präferiert (siehe [Bul01] und [Blo08]):

$$Grau = 0,3 \cdot R + 0,59 \cdot G + 0,11 \cdot B.$$

Eine Alternative zur Farbkameraaufnahme ist eine direkte Aufnahme der Grauwerte. Dabei wird nur der Sensor - ohne Farbfilter - verwendet.

42.1. Graustufenkalibrierung durch Graukarten

Zur Graustufenkalibrierung sind wie bei der Farbkalibrierung Testkarten in Gebrauch (siehe Abbildung 42.1).



Figure 42.1.: Graukarte von Datacolor Spyder Checkr 24[Dat24]

Die Kalibrierung der Graustufen verläuft grundsätzlich analog zur Farbkalibrierung. Auch hier sind Vorarbeiten - wie beispielsweise die Gamma-Korrektur oder die Farbkontrasteinstellung mit einem Schwarzabgleich sowie die Anwendung des Weißabgleichs - sinnvoll. Da jedoch lediglich ein Helligkeitswert statt 3 Farbwerte betrachtet wird, verringert sich die Komplexität der Methoden erheblich.

Beispielsweise reduziert sich beim Weißabgleich mit einem ursprünglichen Weißpunkt $Grau_w^{alt}$ die Berechnung der neuen $Grau^{neu}$ aus den alten Grauwerten $Grau_w^{alt}$ auf die einfache Skalierung:

$$Grau^{neu} = \frac{255}{Grau_w^{alt}} Grau_w^{alt}.$$

Analog zur Farbkalibrierung wird bei der Graustufenkalibrierung wiederum eine nichtlineare Funktion $f : \mathbb{R} \rightarrow \mathbb{R}$ nach Abschnitt 40.2 zur Optimierung genutzt. Dabei ist der i-te Ist-Wert $IST_{Grauwert}(i) \in \mathbb{R}$ der korrigierte Wert des ursprünglichen Kamerabildes und der i-te Soll-Wert $SOLL_{Grauwert}(i) \in \mathbb{R}$ der Wert aus der zugrundeliegenden Farbpalette.

Da n unterschiedliche Farben der ursprünglichen Farbpalette abgleichen werden, läuft die Minimierung über die Summierung von 1 bis n.

$$\min_{\theta \in \mathbb{R}^m} \sum_{i=1}^n \|f((IST_{Grauwert}(i)) - SOLL_{Grauwert}(i))\|_2^2.$$

Die Vereinfachung auf eine lineare Funktion bedeutet die Reduktion der Kalibrierungsmatrix $C \in \mathbb{R}^{3 \times 3}$ auf einen linearen Faktor $c \in \mathbb{R}$. Dies ist allerdings nicht praktikabel, da vorher beim Weißabgleich der maximale Weißwert bereits auf 255 gesetzt wurde und dieser Wert bei einer linearen Transformation auf $255 \cdot c$ nicht sinnvoll ist. Aus diesem Grund wird dieser Ansatz hier nicht weiter verfolgt.

43. Bestimmung des Fokus

Die genaue Bestimmung des Fokus besitzt eine sehr große Bedeutung in der Bildgenerierung. Da dieser bestimmt, welche Stelle eines Bildes scharf oder verschwommen dargestellt wird. Es ist technisch nicht möglich und sinnvoll (Gewohnheit menschliches Sehen) ein ganzes Bild gleich scharf dazustellen [Gra14, S. 86]. Es gibt immer eine Schärfeebene, an der ein Detail klar und detailliert abgebildet wird.

Der Fokus ist der Detailpunkt (Fokuspunkt), der auf dem Bild scharf dargestellt werden soll. Dieser wird als Ausgangspunkt gewählt. Darüber hinaus liegt er auf dem Brennpunkt, das ist der Punkt, an dem sich die Lichtstrahlen kreuzen. Dieser wird im Anschlusskapitel (siehe 44.2) mathematisch erläutert. Die Ebene, die parallel zur Bildebene der Kamera liegt und durch den Fokuspunkt läuft, heißt Fokusebene. Alles, was durch diese Ebene verläuft, wird scharf dargestellt (siehe Abbildung 43.1: die rote Linie). Der Rest vor und hinter dieser Ebene wird unscharf dargestellt, dieser Verlauf ist näherungsweise nach der Gauß-Verteilung verteilt. Dies führt dazu, dass kurz vor und hinter der Fokusebene noch ein Schärfentiefenbereich existiert (siehe graues Hintergrunddreieck). Die Größe des Bereichs hängt von mehreren Faktoren ab, der Blendenöffnung, der Brennweite und dem Abstand zum Objekt. Bei der Kalibrierung und Bestimmung des Fokus, wird die Schärfentiefe vernachlässigt. Es ist trotzdem sinnvoll, eine geringere Schärfentiefe zu verwenden, damit man die Fokusebene leichter erkennen kann.

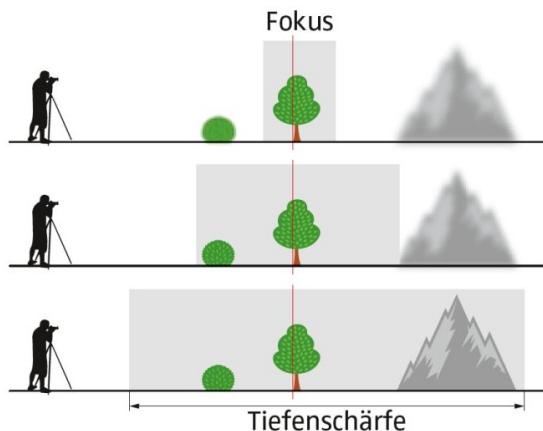


Figure 43.1.: Darstellung des Fokus mit Tiefenschärfe [Koe23]

Beispielhaft wird auf zwei Testschablonen eingegangen. Einmal die Faltbare Testschablone von Arducam [Ard24a] (siehe Abbildung 43.2). Mit dieser Testschablone kann man den Fokus kalibrieren und schätzen, aber auch die Tiefenschärfe messen. Das Testchart ist für kleine Objektive von z.B. Robotern oder Handy gedacht. Mit der Linse wird das Testbild (Siemensstern) eingefangen, bis es scharf gestellt ist. Danach kann man die Parameter einfach ablesen.

Eine andere Testschablone ist der SpyderLensCal von Datacolor [Dat23] (siehe Abbildung 43.3). Damit kann man den Fokus von Digitalkameras kalibrieren. Man platziert die Kamera vor die Testschablone, fokussiert die Mitte der 4 Quadrate und schießt Fotos. Danach wird auf den Bildern überprüft, an welcher Stelle der Fokus auf der



Figure 43.2.: Darstellung der Testschablone Arducam Lens Calibration Tool [Ard24a]

Skalierung liegt. Dieser soll bei einer korrekten Kalibrierung bei null liegen. Falls dies nicht der Fall ist, ändert man die Parameter auf der Kamera so lange ab, bis der Fokus auf null liegt.



Figure 43.3.: Darstellung der Testschablone SpyderLensCal von Datacolor [Dat23]

44. Kalibrierung mit Kalibrierkörper

In diesem Kapitel geht es um die Kalibrierung der geometrischen Perspektive mit Hilfe von Kalibrierkörpern. Anwendungsgebiete finden sich beispielsweise in der Robotik, Augmented Reality und dem normalen Alltag.

Ein Kalibrierkörper ist durch seine Markierungspunkte im Weltkoordinatensystem definiert. Dabei müssen die exakten Koordinaten bekannt sein und eine geringe Toleranz in der Anfertigung des Kalibrierkörpers vorliegen.

Zunächst wird die klassische Methode der Kalibrierung nach [CJP23] mit einem 3D-Kalibrierkörper, wie in der Abbildung 44.1 analysiert.

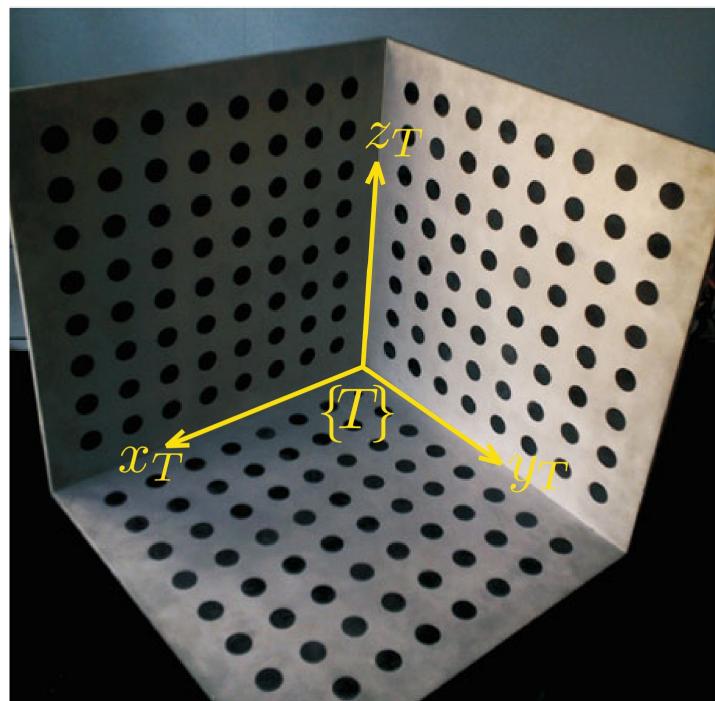


Figure 44.1.: 3D-Kalibrierkörper mit eingezeichneten Achsen nach [CJP23]

Daraus lässt sich die sogenannte Selbstkalibrierung mit 2D-Kalibrierkörper herleiten, indem die spätere Z-Achse des Weltkoordinatensystems eliminiert wird. Dies führt zur sogenannten planaren Homographie [CJP23, S. 583]. Beispiele für 2D-Kalibrierkörper wären das Schachbrett 44.2 oder das Kreisgitter-Muster (Circle Grid). Das Kreisgittermuster ist in der Abbildung 44.1 als 3D-Körper zu erkennen. Genauso wie beim Schachbrett existiert dies sowohl als 2D- als auch 3D- Testschablone.

Häufig werden diese 2D-Kalibrierkörper aus unterschiedlichen Winkeln zur Kalibrierung aufgenommen. Durch die Drehungen im 3D-Weltkoordinatensystem können die Markierungspunkte aus den verschiedenen Ansichten auch als virtuelle 3D-Kalibrierkörper interpretiert werden.

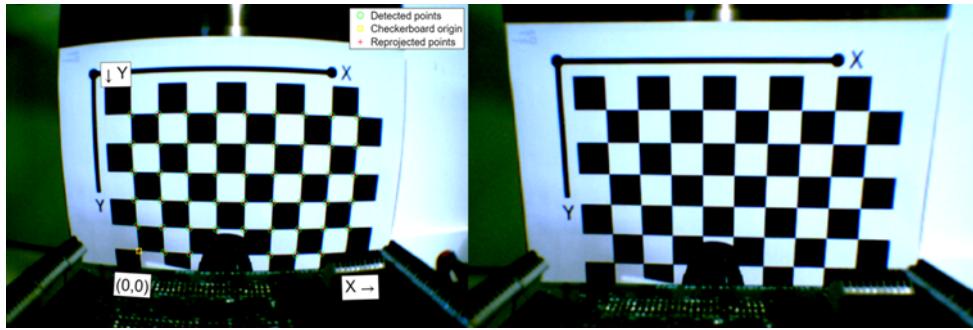


Figure 44.2.: 2D-Kalibrierkörper Schachbrett Testschablone[Kru23]

44.1. Extrinsische Parameter

Zunächst wird die Außendarstellung einer Kameraaufnahme betrachtet. Das Aufnahmeeobjekt, später unser Kalibrierkörper, steht als Objekt "in der realen Welt" und wird von einer Kamera aufgenommen (siehe Abschnitt 44.1), deren Position sich auch ändern kann. D.h. entsprechend der Abbildung 44.3 ist unsere Kamera im allgemeinen nicht bezogen auf das Weltkoordinatensystem, sondern nimmt eine willkürliche Position an. Damit muss eine Transformation des Weltkoordinatensystems in das Kamerakoordinatensystem beschrieben werden.

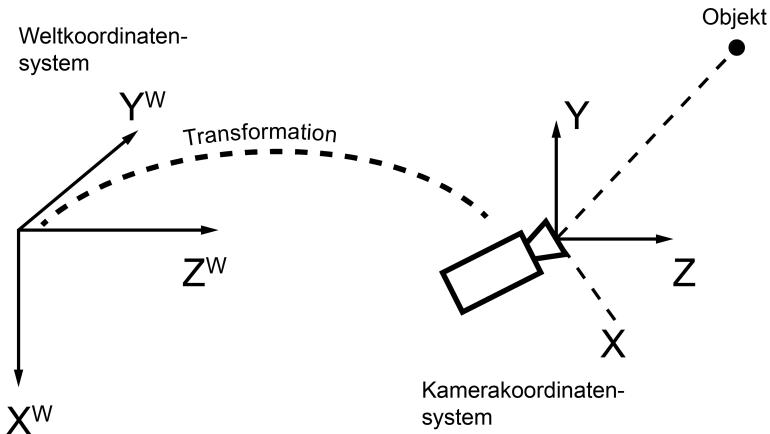


Figure 44.3.: Transformation des Weltkoordinatensystems in das Kamerakoordinatensystem [SR14, S. 294]

Dabei ist $P^W = (X^W, Y^W, Z^W)^T$ ein beliebiger Punkt im Weltkoordinatensystem, der auf einen Punkt $P = (X, Y, Z)^T$ im 3D-Kamerakoordinatensystem abgebildet wird. Jede Transformation lässt sich als Rotation mit einer Matrix $R \in \mathbb{R}^{3 \times 3}$ und einer anschließenden Translation $t \in \mathbb{R}^3$ beschreiben. Damit gilt die Darstellung:

$$P = RP^W + t. \quad (44.1)$$

Jede Rotation und jede Translation wird durch jeweils 3 Parameter eindeutig bestimmt. Somit gibt es insgesamt 6 Parameter, die die Transformation der Kamera beschreiben. Dies sind die 6 extrinsischen Parameter des Kamerasystems. Die Gleichung (44.1) wird nun in homogene Koordinaten dargestellt. Im weiteren Verlauf wird das Tilde-Zeichen immer für homogene Koordinaten verwendet. Durch diese homogenen Koordinaten erhält man den Punkt $\tilde{P} = (X, Y, Z, 1)$ im Kamerakoordinatensystem als Transformation des Punktes $\tilde{P}^W = (X^W, Y^W, Z^W, 1)^T$ in den

homogenen Weltkoordinaten

$$\begin{aligned}\tilde{P} &= \begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix} \begin{pmatrix} X^W \\ Y^W \\ Z^W \\ 1 \end{pmatrix} \\ &= \begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix} \tilde{P}^w.\end{aligned}\quad (44.2)$$

Diese Transformation wird im Kapitel 44.4 verwendet.

44.2. Intrinsische Parameter

Im vorherigen Abschnitt wurde lediglich die Außendarstellung einer Kameraaufnahme mit den extrinsischen Parameter betrachtet. Nun stehen die inneren Vorgänge der Kamera im Fokus der Betrachtung, deren Eigenschaften durch die intrinsischen Parameter bestimmt sind.

Je nach Modell sind maximal die folgenden 6 intrinsischen Parameter zu beachten [Kru23]:

- **Brennweite f** bestimmt das Sichtfeld und die Vergrößerung der Kamera.
- **Hauptpunkt** ist der Punkt, an dem die optische Achse der Kamera den Bildsensor schneidet. Er bestimmt den Bildmittelpunkt.
- **Schiefe** gibt an, ob die Bildachse des Sensors nicht senkrecht zur optischen Achse liegt.
- **Radialer Verzeichnungsparameter** modelliert radiale Verzerrungen der Kameralinse.
- **Tangentialer Verzeichnungsparameter** modelliert tangentiale Verzerrungen der Kameralinse.

Die beiden letzten Parameter entstehen durch fehlerhafte Krümmungen der Linse. Bei einer radialen Verzeichnung verschieben sich die Bildkoordinaten nach innen oder nach außen. Die tangentiale Verzeichnung ist eine Verschiebung der Bildachsen. Bei beiden ist die Stärke der Verzeichnung in der Bildmitte am geringsten und am Bildrand am stärksten. Diese entsprechenden Bildverzeichnungen sind in der Abbildung 44.4 schematisch dargestellt.

44.3. Stetiges Kameramodell der zentralen Projektion

Der Prozess der Bildaufnahme mit einer Kamera ist eine Abbildung von einem 3-dimensionalen Gebilde in der realen Welt auf eine 2-dimensionale Projektionsfläche. Solch eine Transformation von 3 in 2 Dimensionen heißt perspektivische Projektion [Tho23]. Dabei geht die Tiefeinformation verloren, so dass ein nahes kleines Objekt nicht mehr von einem weit entfernten großen Objekt in der Welt unterscheidbar ist. Durch solch eine perspektivische Projektion können parallele Geraden in der 3-dimensionalen Welt nicht in parallele Geraden in der 2D-Bildebene abgebildet werden. Die ursprünglich parallelen Geraden schneiden sich auf der Projektionsebene im Fluchtpunkt. Eine weitere Implikation dieser perspektivischen Projektion ist der Übergang von Kreisen zu Ellipsen.

Die perspektivische Projektion wird auch durch unser Auge realisiert und ist uns Menschen vertraut. Das einfachste perspektivische Projektionsmodell einer Lochkamera

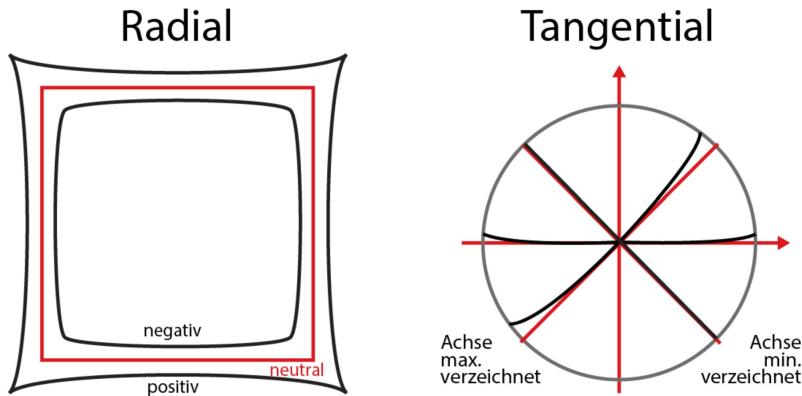


Figure 44.4.: Schematische Darstellung der Linsenverzeichnungsarten [Iac16, S. 8]

wird im weiteren Verlauf nicht betrachtet, sondern direkt zum stetigen Kameramodell der zentralen Projektion übergegangen. Anhand dieses Modells - dargestellt in der Abbildung 44.5 - werden die mathematischen Zusammenhänge hergeleitet.

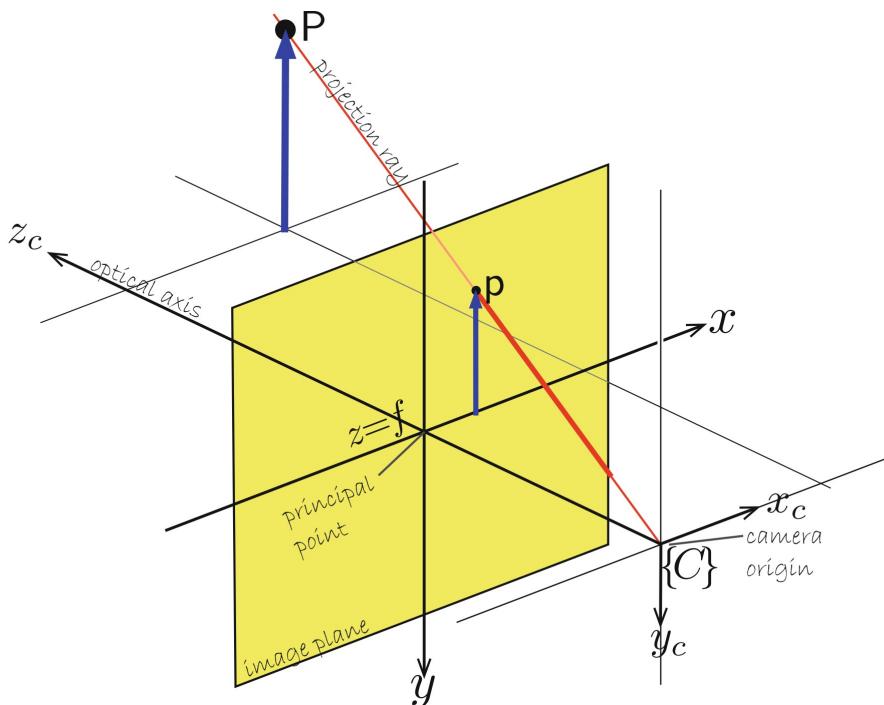


Figure 44.5.: Kameramodell der zentralen Projektion [CJP23, S. 546]

In diesem Kamerakoordinatensystem wird ein Punkt P durch ein Projektionsstrahl (projection ray) auf den Ursprung der Kamera C - dem Brennpunkt bzw. Fokuspunkt - abgebildet. Dieser Projektionsstrahl schneidet die - hier gelb dargestellte - Bildebene in dem Projektionspunkt p . Die x - und y -Achsen des Kamerakoordinatensystems liegen parallel zur gelben Projektionsebene. Die z -Achse geht als optische Achse vom Brennpunkt der Kamera C aus und schneidet die 2-dimensionale Bildebene an der Stelle $z = f$. Somit ist der Abstand der Bildebene vom Kameraursprung f . Dieser Abstand f ist die Brennweite der Kamera, die den

Vergrößerungsgrad eines Objektes bestimmt und somit einer der zu bestimmenden intrinsischen Parameter.

Die Kamerakoordinaten des 3-dimensionalen Punktes $P = (X, Y, Z)^T$ werden so eindeutig auf einen Punkt $p = (x, y)^T$ in der 2-dimensionalen Projektionsebene abgebildet. Mit Hilfe des Strahlensatzes berechnet man folgende Beziehungen:

$$x = f \frac{X}{Z} \quad \text{sowie} \quad y = f \frac{Y}{Z}.$$

Dies sind die Koordinaten auf der 2-dimensionalen Projektionsebene. Für $f = 1$ werden diese als normalisierte Bildkoordinaten der Ebene bezeichnet.

Im Weiteren ist es vorteilhaft, die Koordinaten der Bildebene x und y durch folgende homogene Koordinaten \tilde{x} , \tilde{y} und \tilde{z} zu ersetzen:

$$\tilde{x} = fX, \quad \tilde{y} = fY, \quad \tilde{z} = Z$$

Diese Transformation ist in der folgenden Matrixdarstellung zusammengefasst:

$$\tilde{p} = \begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \end{pmatrix} = \begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}. \quad (44.3)$$

Dabei können die ursprünglichen nicht-homogenen Variablen zurückgerechnet werden mit

$$x = \frac{\tilde{x}}{\tilde{z}} = \frac{fX}{Z} \quad \text{sowie} \quad y = \frac{\tilde{y}}{\tilde{z}} = \frac{fY}{Z}.$$

Nun werden auch die Koordinaten des Vektors $P = (X, Y, Z)^T$ in der homogenen Form $\tilde{P} = (X, Y, Z, 1)^T$ dargestellt. Damit kann die Gleichung (44.3) umgeschrieben werden zu

$$\tilde{p} = \underbrace{\begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}}_{K_{int}^S} \underbrace{\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}}_{\tilde{P}},$$

wobei $K_{int}^S \in \mathbb{R}^{3 \times 4}$ als intrinsische Kameramatrix des stetigen Projektionsmodells bezeichnet wird, da sie die intrinsischen Parameter der Matrix beschreibt.

Diese Kameramatrix kann auch zerlegt werden in die ursprüngliche Matrix sowie der Projektionsmatrix $\Pi \in \mathbb{R}^{3 \times 4}$. Diese Projektionsmatrix projiziert den homogenen 4-dimensionalen Punkt der Kamerakoordinaten in den 3-dimensionalen Raum:

$$\tilde{p} = \underbrace{\begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{=\Pi} \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}}_{=K_{ext}^S} \tilde{P}. \quad (44.4)$$

44.4. Diskretes Kameramodell der zentralen Projektion

Bevor die Transformationen der Weltkoordinaten in die Gleichung (44.4) integriert wird, wird zunächst von dem stetigen zum diskreten Kameramodell der zentralen Projektion gewechselt. da diese bei digitalen Kameras verwendet wird.

Dabei besteht die Bildebene aus einem $W \times H$ Gitter von lichtsensitiven Pixeln, den Photozellen, wie in der Abbildung 44.6 dargestellt. Diese Pixel haben alle die gleiche Gestalt mit der Breite ρ_w und der Höhe ρ_h . Die Pixel-Koordinaten sind nun in einem 2-dimensionalen Vektor $(u^D, v^D)^T$ dargestellt, deren Elemente ganze Zahlen sind. Der

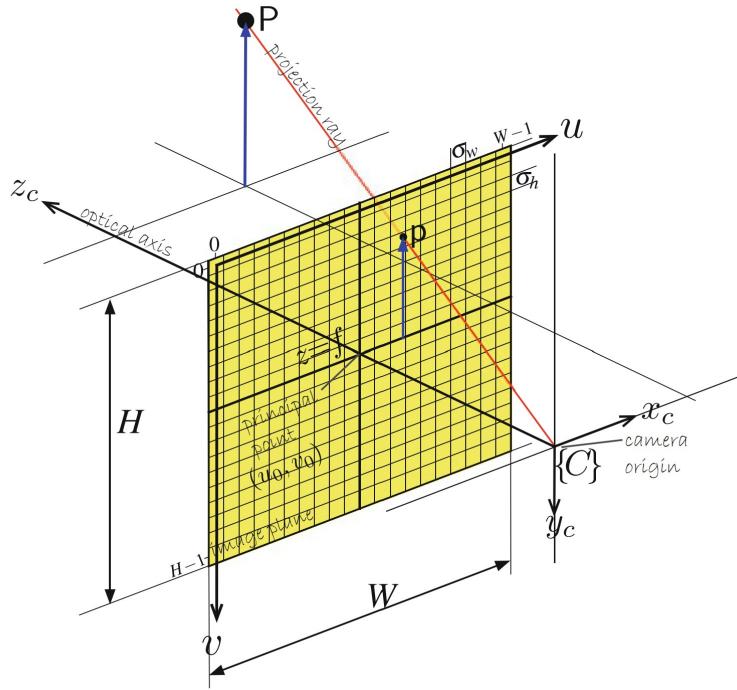


Figure 44.6.: Diskretes Kameramodell der zentralen Projektion
[CJP23, S. 550]

Index D wird eingeführt, um die diskreten Variablen - in Abgrenzung zu den stetigen Variablen - zu kennzeichnen.

Gewöhnlich wird der Ursprung dieser Bildebene in der oberen linken Ecke der Bildebene platziert. Der Referenzpunkt (u_0, v_0) ist der Punkt, indem die optische Achse ausgehend vom Fokus die Bildebene schneidet, angegeben in den Koordinaten des uv-Koordinatensystems. Mit der horizontalen Skalierung um den Faktor $\frac{1}{\rho_w}$ und der Verschiebung um u_0 erhält man den u -Wert:

$$u^D = \frac{x}{\rho_w} + u_0,$$

bzw. mit der vertikalen Skalierung um den Faktor $\frac{1}{\rho_v}$ und der Verschiebung um v_0 nun den entsprechenden v -Wert:

$$v^D = \frac{y}{\rho_h} + v_0.$$

In der homogenen Vektor-Darstellung liefert die folgende Matrixmultiplikation

$$\underbrace{\begin{pmatrix} \tilde{u}^D \\ \tilde{v}^D \\ \tilde{w}^D \end{pmatrix}}_{\tilde{p}^D} = \begin{pmatrix} \frac{1}{\rho_w} & s & u_0 \\ 0 & \frac{1}{\rho_v} & v_0 \\ 0 & 0 & 1 \end{pmatrix} \underbrace{\begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \end{pmatrix}}_{=\tilde{p}}$$
(44.5)

die Transformation von dem homogenen stetigen Bildpunkt $\tilde{p} = (\tilde{x}, \tilde{y}, \tilde{z})^T$ auf die diskretisierte homogene Darstellung $\tilde{p}^D = (\tilde{u}^D, \tilde{v}^D, \tilde{w}^D)^T$.

Der Faktor s ist bei orthogonalem uv-Koordinatensystem stets 0. Falls Verzerrungen zu einem nicht-rechtwinkligen Koordinatensystem führen, können noch weitere Verallgemeinerungen mit $s \neq 0$ integriert werden.

Die nicht-homogenen Pixel-Koordinaten der Bildebene sind nun gegeben durch

$$u^D = \frac{\tilde{u}^D}{\tilde{w}^D} \quad \text{sowie} \quad = \frac{\tilde{v}^D}{\tilde{w}^D}. \quad (44.6)$$

Die Transformation (44.5) von den stetigen zu den diskreten Koordinaten können wir auf unsere Gleichung (44.4) auf beiden Seiten anwenden und anschließend durch Matrix-Matrix-Multiplikation vereinfachen zu:

$$\begin{aligned} \tilde{p}^D &= \begin{pmatrix} \frac{1}{\rho_w} & 0 & u_0 \\ 0 & \frac{1}{\rho_v} & v_0 \\ 0 & 0 & 1 \end{pmatrix} \tilde{p} \\ &= \underbrace{\begin{pmatrix} \frac{1}{\rho_w} & 0 & u_0 \\ 0 & \frac{1}{\rho_v} & v_0 \\ 0 & 0 & 1 \end{pmatrix}}_{K_{intr}} \underbrace{\begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{\Pi} \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}}_{\tilde{P}} \\ &= \underbrace{\begin{pmatrix} \frac{f}{\rho_w} & 0 & u_0 \\ 0 & \frac{f}{\rho_v} & v_0 \\ 0 & 0 & 1 \end{pmatrix}}_{K_{intr}} \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}}_{\Pi} \tilde{P}. \end{aligned} \quad (44.7)$$

Die 3×3 Matrix K_{intr} bestimmt die intrinsischen Parameter der Kamera. So stellen deren Einträge $\frac{f}{\rho_w}$ sowie $\frac{f}{\rho_v}$ die Brennweite der Kamera in den Pixeleinheiten dar. Zum Schluss wird noch die äußere Transformation (44.2) des Abschnittes 44.1 in die Gleichung (44.7) eingesetzt, um die allgemeine Form der Kameraprojektion darzustellen:

$$\begin{aligned} \tilde{p}^D &= \underbrace{\begin{pmatrix} \frac{f}{\rho_w} & 0 & u_0 \\ 0 & \frac{f}{\rho_v} & v_0 \\ 0 & 0 & 1 \end{pmatrix}}_{K_{intr}} \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}}_{\Pi} \tilde{P} \\ &= \underbrace{\begin{pmatrix} \frac{f}{\rho_w} & 0 & u_0 \\ 0 & \frac{f}{\rho_v} & v_0 \\ 0 & 0 & 1 \end{pmatrix}}_{K_{intr}} \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}}_{K_{extr}} \underbrace{\begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix}}_{C} \tilde{P}^w \\ &= C \tilde{P}^w \end{aligned} \quad (44.8)$$

Die allgemeine diskrete Kamerakalibrierungsmatrix $C \in \mathbb{R}^{3 \times 4}$ ist somit die Multiplikation der intrinsischen Kameramatrix $K_{intr} \in \mathbb{R}^{3 \times 3}$ mit der extrinsischen Kameramatrix $K_{extr} \in \mathbb{R}^{3 \times 4}$. Diese Matrix bildet einen beliebigen homogenen Vektor \tilde{P}^w im Weltkoordinatensystem auf den homogenen diskretisierten Vektor des projizierten Punktes in den

Kamerakoordinatensystem ab. Diese Matrix wird im nächsten Abschnitt näherungsweise bestimmt.

44.5. Berechnung der Kalibrierung

In diesem Abschnitt wird die diskrete Kamerakalibrierungsmatrix C näherungsweise aus den Kameraaufnahmen eines Kalibrierkörpers bestimmt.

Die Gleichung (44.8) kann zunächst mit unserer unbekannten Matrix C und den Punkt $\tilde{P}^W = (X^W, Y^W, Z^W, 1)^T$ in den homogenen Weltkoordinaten ausführlich dargestellt werden über

$$\begin{aligned} \underbrace{\begin{pmatrix} \tilde{u}^D \\ \tilde{v}^D \\ \tilde{w}^D \end{pmatrix}}_{\tilde{p}^D} &= \underbrace{\begin{pmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \end{pmatrix}}_C \underbrace{\begin{pmatrix} X^W \\ Y^W \\ Z^W \\ 1 \end{pmatrix}}_{\tilde{P}_w} \\ &= \begin{pmatrix} c_{11}X^W + c_{12}Y^W + c_{13}Z^W + c_{14} \\ c_{21}X^W + c_{22}Y^W + c_{23}Z^W + c_{24} \\ c_{31}X^W + c_{32}Y^W + c_{33}Z^W + c_{34} \end{pmatrix}, \end{aligned} \quad (44.9)$$

wobei im letzten Schritt lediglich die Matrix-Vektor-Multiplikation ausgeführt wurde.

Betrachtet man die Gleichung (44.9) komponentenweise, so liefert die 3. Komponente den Wert

$$\tilde{w}^D = c_{31}X^W + c_{32}Y^W + c_{33}Z^W + c_{34}. \quad (44.10)$$

Bei der 1. Komponente wird auf beiden Seiten \tilde{u}^D abgezogen und anschließend die Beziehung $\tilde{u}^D = u^D\tilde{w}^D$ aus der Gleichung (44.6) genutzt:

$$c_{11}X^W + c_{12}Y^W + c_{13}Z^W + c_{14} - \underbrace{u^D\tilde{w}^D}_{\tilde{u}^D} = 0.$$

Der Wert \tilde{w}^D der Gleichung (44.10) liefert schließlich die erste Gleichung

$$c_{11}X^W + c_{12}Y^W + c_{13}Z^W + c_{14} - c_{31}u^DX^W - c_{32}u^DY^W - c_{33}u^DZ^W - c_{34}u^D = 0.$$

und analog folgt für die 2. Komponente von (44.9) die zweite Gleichung

$$c_{21}X^W + c_{22}Y^W + c_{23}Z^W + c_{24} - c_{31}v^DX^W - c_{32}v^DY^W - c_{33}v^DZ^W - c_{34}v^D = 0$$

zur Bestimmung der unbekannten Matrix C .

Es gibt insgesamt maximal 11 Parameter, die die Kalibrierung einer Kamera bestimmen: 6 extrinsische nach Abschnitt 44.1 sowie 5 intrinsische Parameter nach Abschnitt 44.2. Da die unbekannte Matrix $C \in \mathbb{R}^{3 \times 4}$ insgesamt jedoch $3 \cdot 4 = 12$ unbekannte Koeffizienten aufweist, kann einer dieser 12 Einträge normiert werden, um die restlichen Variablen eindeutig zu bestimmen. Mit der Wahl $c_{34} = 1$ kann auf beiden Seiten der ersten Gleichung u^D sowie v^D auf beiden Seiten der zweiten Gleichung addiert werden. Dies liefert folgende 2 Gleichungen

$$c_{11}X^W + c_{12}Y^W + c_{13}Z^W + c_{14} - c_{31}u^DX^W - c_{32}u^DY^W - c_{33}u^DZ^W = u^D$$

$$c_{21}X^W + c_{22}Y^W + c_{23}Z^W + c_{24} - c_{31}v^DX^W - c_{32}v^DY^W - c_{33}v^DZ^W = v^D.$$

Diese Gleichungen müssen stets erfüllt sein. Also auch für den i-ten Markierungspunkt unseres Kalibrierkörpers mit den Weltkoordinaten $P_i^W = (X_i^W, Y_i^W, Z_i^W)^T$ und dem diskreten gemessenen Kamerabbild $(u_i^D, v_i^D)^T$. Bei N Markierungspunkten führt dies zu einem linearen Gleichungssystem

$$Ac = b$$

mit einer bekannten Koeffizientenmatrix $A \in \mathbb{R}^{2N \times 11}$, mit

$$A =$$

$$\left(\begin{array}{cccccccccc} X_1^W & Y_1^W & Z_1^W & 1 & 0 & 0 & 0 & 0 & -u_1^D X_1^W & -u_1^D Y_1^W & -u_1^D Z_1^W \\ 0 & 0 & 0 & 0 & X_1^W & Y_1^W & Z_1^W & 1 & -v_1^D X_1^W & -v_1^D Y_1^W & -v_1^D Z_1^W \\ \vdots & \vdots \\ X_N^W & Y_N^W & Z_N^W & 1 & 0 & 0 & 0 & 0 & -u_N^D X_N^W & -u_N^D Y_N^W & -u_N^D Z_N^W \\ 0 & 0 & 0 & 0 & X_N^W & Y_N^W & Z_N^W & 1 & -v_N^D X_N^W & -v_N^D Y_N^W & -v_N^D Z_N^W \end{array} \right)$$

und den Vektoren $c = \begin{pmatrix} c_{11} \\ c_{12} \\ \vdots \\ c_{33} \end{pmatrix} \in \mathbb{R}^{11}$ sowie $b = \begin{pmatrix} u_1^D \\ v_1^D \\ \vdots \\ u_N^D \\ v_N^D \end{pmatrix} \in \mathbb{R}^{2N}$.

Um die unbekannten Einträge des Vektors c und somit der Matrix C möglichst genau zu bestimmen, sind viele Markierungspunkte N sinnvoll. Mit $2N$ Gleichungen zur Berechnung der 11 Variablen von C , ist dieses Gleichungssystem überbestimmt und führt somit zum Kleinste-Quadrat-Problem

$$\min_{c \in \mathbb{R}^{11}} \|Ax - b\|_2^2.$$

Zur Berechnung dieses linearen Problems können zwar die Algorithmen des Abschnitts 40.2 verwendet werden, jedoch existieren hierfür auch einfachere Verfahren.

Damit ist die diskrete Kamerakalibrierungsmatrix C berechnet, was für die meisten Anwendungen ausreicht. Falls darüber hinaus noch einzelne Parameter explizit bestimmt werden sollen, muss dies durch eine Rückwärtsrechnung geschehen. Da eine entsprechende Darstellung hier zu ausführlich wäre und die Berechnungen meist nur nähерungsweise, beispielsweise mit MATLAB, vollzogen werden, werden die Betrachtungen hier abgeschlossen.

45. BlueTooth

- <https://www.instructables.com/Arduino-NANO-33-Made-Easy-BLE-Sense-and-IoT/>
- <https://www.hackster.io/sridhar-rajagopal/control-arduino-nano-ble-with-blue>
- <https://docs.arduino.cc/tutorials/nano-33-ble-sense/ble-device-to-device>
- <https://projecthub.arduino.cc/8bitkick/sensor-data-streaming-with-arduino-a6>
- <https://www.okdo.com/getting-started/get-started-with-arduino-nano-33-sense/>
- <https://rootsaid.com/arduino-ble-example/>
- <https://learn.sparkfun.com/tutorials/bluetooth-basics>
- <https://ladvien.com/arduino-nano-33-bluetooth-low-energy-setup/>

45.1. Quick Start

This tutorial shows you how to use the free pfodDesignerV3 V3.0.3774+ Android app to create a general purpose Bluetooth Low Energy (BLE) and WiFi connection for Arduino NANO 33 boards without doing any programming. There are three (3) Arduino NANO 33 boards, the NANO 33 BLE and NANO 33 BLE Sense, which connect by BLE only and the NANO 33 IoT which can connect via BLE or WiFi. Connecting using pfodApp is the most flexible way to connect via BLE (or WiFi). See Using pfodApp to connect to the NANO 33 BLE (Step 4) and
Using pfodApp to connect to the NANO 33 IoT via BLE (Step 7) and
Using pfodApp to connect to the NANO 33 IoT via WiFi (Step 9) below.
However simple sketches are also provided to send user defined command words via Telnet, for WiFi, or via the free Nordic nRF UART 2.0 for BLE. See
Using the Nordic nRF UART 2.0 app to connect to NANO 33 BLE (Step 5) and
Using the Nordic nRF UART 2.0 app to connect to the NANO 33 IoT via BLE (Step 8) and
Using a Telnet terminal program to connect to the NANO 33 IoT via WiFi (Step 10) below
This tutorial is also available on-line at Arduino NANO 33 Made Easy.

45.2. Supplies

Arduino NANO 33 – either BLE or Sense or IoT
optionally pfodDesignerV3 and pfodApp

45.3. Step 1: Introduction

There are a number of problems with BLE. See this page for BLE problems and solutions and there are some BLE trouble shooting tips. The learning curve is steep and the specification has hundreds of specialised connect services each of which requires its own mobile application to connect to. This tutorial shows you how to generate

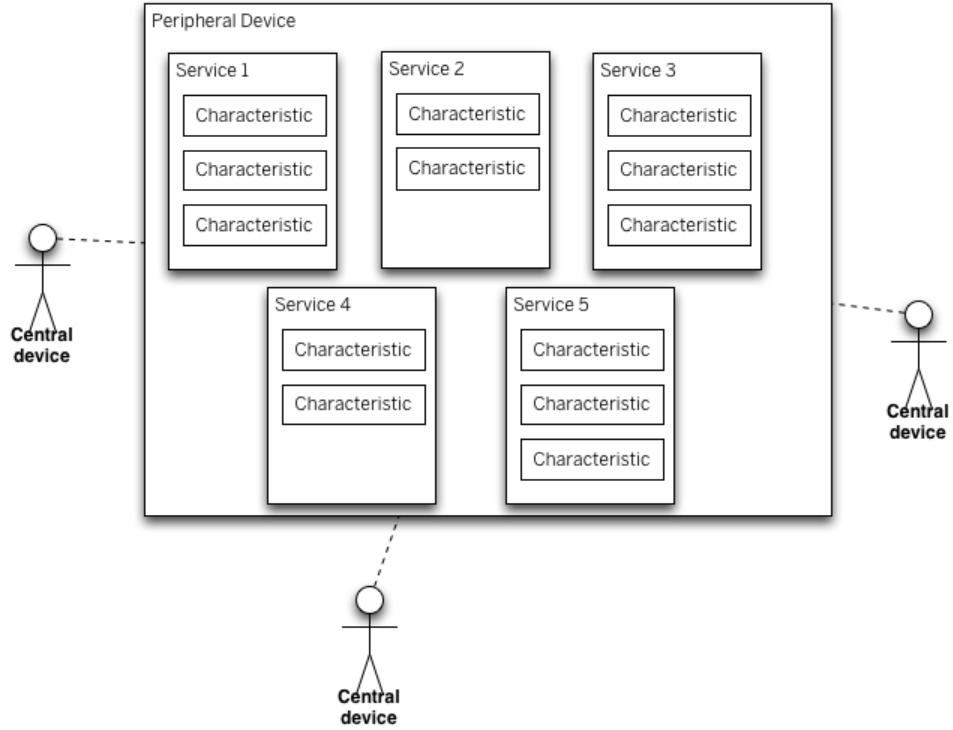


Figure 45.1.: BLE Devices – Peripheral and Central Devices

Arduino code for a general purpose Nordic UART BLE connection over which you can send and receive a stream of commands and data to a general purpose BLE UART mobile application. The free pfodDesignerV3 Android application is used to generate the Arduino code. The output is designed to connect to the paid pfodApp Android application which can display menus, send command, log data and show charts. No Android programming is required. The Arduino code has complete control over what is displayed by pfodApp.

For the NANO 33 IoT you can also connect via WiFi. Again the pfodDesignerV3 generates all the Arduino code and by default is designed to connect to pfodApp with optional 128bit security.

However you do not need to use pfodApp, you can connect to the generated code using the free Nordic nRF UART 2.0 or a Telnet programs (for the WiFi connection). Sketches are included which provide command words to control the boards.

The free pfodDesigner V3.0.3774+ will generate Arduino code for a wide range of boards and connection types including Serial connections, Bluetooth Low Energy (BLE), WiFi, SMS, Radio/LoRa, Bluetooth Classic and Ethernet. For examples Arduino code for of a wide range of BLE boards see Bluetooth Low Energy (BLE) made simple with pfodApp. Here we will be using a BLE connection for NANO 33 BLE, Sense and IoT and a WiFi connection for the IoT.

Each of the NANO 33 boards has extra sensor components that differ between the three (3) boards. The pfodDesignerV3 generates code to read/write the digital outputs and perform analogReads and analogWrites. In the examples below we will turn the board LED on and off and read the voltage at A0 and log and plot it. Once that sketch is running you can add each board's specialised sensor libraries and sent their data in place of the analogRead(A0).

pfodDesigner generates simple menus and charts, however you can also program custom graphical interfaces in your Arduino sketch. Above is an example of slider control

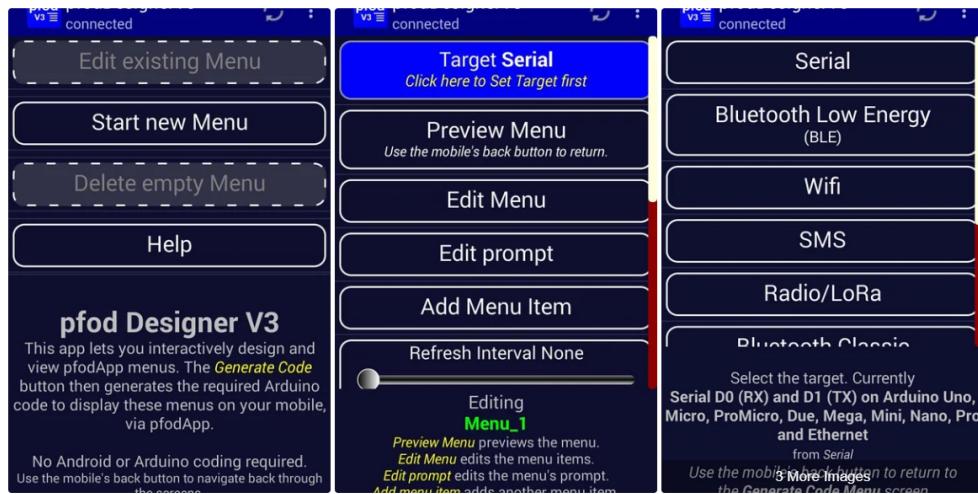


Figure 45.2.: Step 2: Creating the Custom Android Menus and Generating the Code

adjusting a guage. All the code for this control is in your Arduino sketch. No Android programming necessary. See Custom Arduino Controls for more examples.

45.4. How pfodApp is optimised for short BLE style messages

Bluetooth Low Energy (BLE) or Bluetooth V4 is a completely different version of Bluetooth. BLE has been optimised for very low power consumption. pfodApp is a general purpose Android app whose screens, menus, buttons, sliders and plots are completely defined by the device you connect to.

BLE only sends 20 bytes in each message. Fortunately the pfod Specification was designed around very small messages. Almost all of pfod's command are less then 20 bytes. The usual exception is the initial main menu message which specifies what text, menus, buttons, etc. pfodApp should display to the user, but the size of this message is completely controlled by you and you can use sub-menus to reduce the size of the main menu.

The pfod specification also has a number of features to reduce the message size. While the BLE device must respond to every command the pfodApp sends, the response can be as simple as (an empty response). If you need to update the menu the user is viewing in response to a command or due to a re-request, you need only send back the changes in the existing menu, rather then resending the entire menu. These features keep the almost all messages to less than 20 bytes. pfodApp caches menus across re-connections so that the whole menu only needs to be sent once. Thereafter short menu updates can be sent.

45.5. Step 2: Creating the Custom Android Menus and Generating the Code

Before looking at each of these BLE modules, pfodDesignerV3 will first be used to create a custom menu to turn a Led on and off and plot the voltage read at A0. pfodDesignerV3 can then generate code tailored to the particular hardware you select. You can skip over this step and come back to it later if you like. The section on each module, below, includes the completed code sketch for this example menu generated

for that module and also includes sketches that do not need pfodApp

The free pfodDesignerV3 is used to create the menu and show you an accurate preview of how the menu will look on your mobile. The pfodDesignerV3 allows you to create menus and sub-menus with buttons and sliders optionally connected to I/O pins and generate the sketch code for you (see the pfodDesigner example tutorials) but the pfodDesignerV3 does not cover all the features pfodApp supports. See the pfodSpecification.pdf for a complete list including data logging and plotting, multi- and single- selections screens, sliders, text input, etc.

Create the Custom menu to turn the Arduino LED on and off and Plot A0 Start a new menu and select as a target Bluetooth Low Energy (BLE) and then select NANO 33 BLE (and Sense). pfodDesignerV3.0.3770+ has support for NANO 33 boards.

Then follow the tutorial Design a Custom menu to turn the Arduino Led on and off for step by step instructions for creating a LED on/off menu using pfodDesignerV3.

If you don't like the colours of font sizes or the text, you can easily edit them in pfodDesignerV3 to whatever you want and see a WYSIWYG (What You See Is What You Get) display of the designed menu.

Now we will add a Chart button to display the A0 reading. The steps to do this in pfodDesignerV3 are shown in Adding a Chart and Logging Data The AtoD range is 0 to 1023 for 0 to 3.3V

Generating the code from pfodDesignerV3 gives the this sketch [Nano33BLE_Led_A0.ino](#)

45.6. BLE Mobile App “Nordic Semiconductors nRF Connect”

- On your mobile, install the App “Nordic Semiconductors nRF Connect”. There are Android and IOS versions.
- In the Arduino IDE open Serial Monitor by clicking on the magnifying glass icon on the top right.

The Nano will start sending BLE advertising packets and wait for a Central (Mobile Phone) to connect.

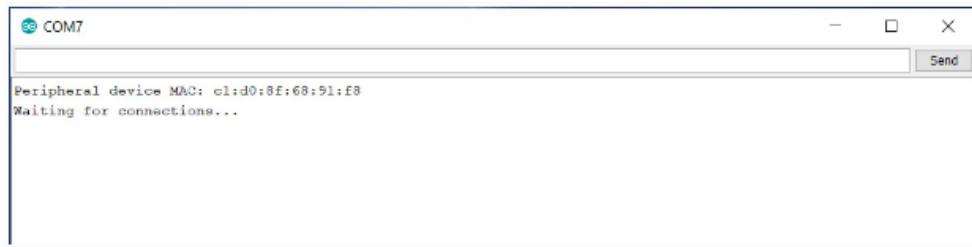


Figure 45.3.: App nRF is searching the Arduino Nano BLE Sense

In the App nRF on your mobile select **Scan** and you should see **Nano33BLE** as a device you can connect to.

Attention: BLE Peripherals can only connect to one device at a time – if the Nano33BLE is already connected it could be to another device nearby.

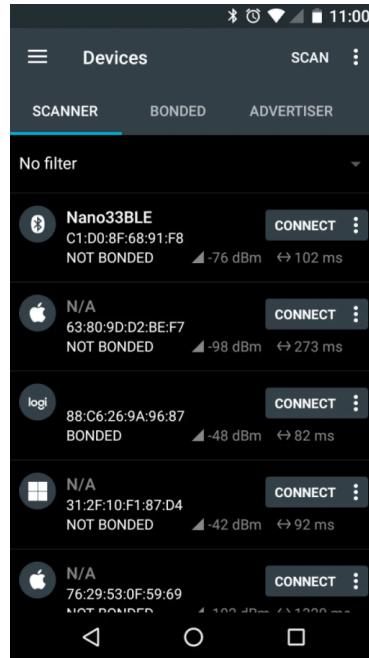


Figure 45.4.: App nRF is searching the Arduino Nano BLE Sense

Connect to the Nano and select the Unknown Service.

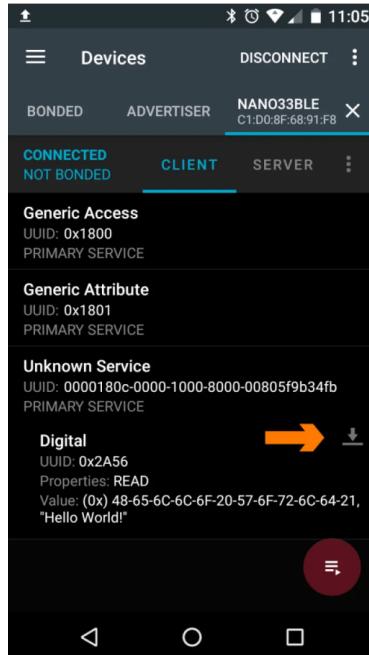


Figure 45.5.: App nRF is getting data the Arduino Nano BLE Sense

Touch the **down arrow** next to the characteristic **Digital** and it will read the message "Hello World!" being broadcast by the Nano33BLE.

45.7. Arduino BLE Example – Explained Step by Step

45.7.1. Arduino BLE Example Code Explained

In this tutorial series, I will give you a basic idea you need to know about Bluetooth Low Energy and I will show you how you can make Arduino BLE Chipset to send and receive data wirelessly from mobile phones and other Arduino boards. Let's Get Started.

Arduino Nano 33 BLE Sense, Nano with BLE connectivity focussing on IOT, which is packed with a wide variety of sensors such as 9 axis Inertial Measurement Unit, pressure, light, and even gestures sensors and a microphone.

It is powered by Nina B306 module that supports BLE as well as Bluetooth 5 connection. The inbuilt Bluetooth module consumes very low power and can be easily accessed using Arduino libraries. This makes it easier to program and enable wireless connectivity to any of your projects in no time. You won't have to use external Bluetooth modules to add Bluetooth capability to your project. Save space and power.

45.7.2. Arduino BLE – Bluetooth Low Energy Introduction

BLE is a version of Bluetooth which is optimized for very low power consuming situations with very low data rate. We can even operate these devices using a coin cell for weeks or even months.

WS:cite Arduino have a wonderful introduction to BLE but here in this post, I will give you a brief introduction for you to get started with BLE communication.

Basically, there are two types of devices when we consider a BLE communication block.

- The Peripheral Device
- The Central Device

Peripheral Device is like a Notice board, from where we can read data from various notices or pin new notices to the board. It posts data for all devices that needs this information.

Central Devices are like people who are reading notices from the notice board. Multiple users can read and get data from the notice board at the same time. Similarly multiple central devices can read data from the peripheral device at the same time.

The information that is given by the Peripheral devices are structured as Services. And These services are further divided into characteristics. Think of Services as different notices in the notice board and services as different paragraphs in each notice board. If Accelerometer is a service, then their values X, Y and Z can be three characteristics. Now let's take a look at a simple Arduino BLE example.

45.7.3. Arduino BLE Example 1 – Battery Level Indicator

In this example, I will explain how you can read the level of a battery connected to pin A0 of an Arduino using a smartphone via BLE. This is the code here. This is pretty much the same as that of the example code for Battery Monitor with minor changes. I will explain it for you.

First you have to install the library ArduinoBLE from the library manager.

Just go to [Sketch -> Include Library -> Manage Library](#) and Search for [ArduinoBLE](#) and simply install it.

```
1000 #include <ArduinoBLE.h>
1001 BLEService batteryService("1101");
1002 BLEUnsignedCharCharacteristic batteryLevelChar("2101", BLERead | BLENotify);
1003
1004 void setup() {
1005     Serial.begin(9600);
1006     while (!Serial);
1007
1008     pinMode(LED_BUILTIN, OUTPUT);
1009     if (!BLE.begin())
1010     {
1011         Serial.println("starting BLE failed!");
1012         while (1);
1013     }
1014
1015     BLE.setLocalName("BatteryMonitor");
1016     BLE.setAdvertisedService(batteryService);
1017     batteryService.addCharacteristic(batteryLevelChar);
1018     BLE.addService(batteryService);
1019
1020     BLE.advertise();
1021     Serial.println("Bluetooth device active, waiting for connections...");
1022 }
1023
1024 void loop()
1025 {
1026     BLEDevice central = BLE.central();
1027
1028     if (central)
1029     {
1030         Serial.print("Connected to central: ");
1031         Serial.println(central.address());
1032         digitalWrite(LED_BUILTIN, HIGH);
1033
1034         while (central.connected()) {
1035
1036             int battery = analogRead(A0);
1037             int batteryLevel = map(battery, 0, 1023, 0, 100);
1038             Serial.print("Battery Level % is now: ");
1039             Serial.println(batteryLevel);
1040             batteryLevelChar.writeValue(batteryLevel);
1041             delay(200);
1042
1043         }
1044     }
1045     digitalWrite(LED_BUILTIN, LOW);
1046     Serial.print("Disconnected from central: ");
1047     Serial.println(central.address());
1048 }
```

Listing 45.1.: Arduino BLE Tutorial Battery Level Indicator Code

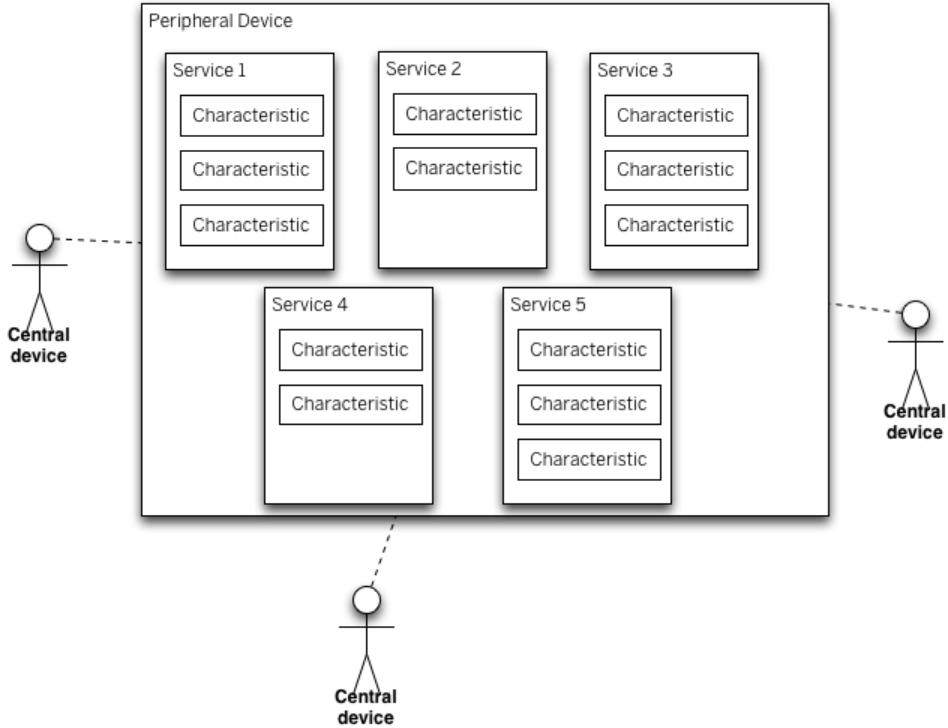


Figure 45.6.: BLE Devices – Peripheral and Central Devices

45.7.4. Arduino BLE Tutorial Battery Level Indicator Code

45.7.5. Arduino Bluetooth Battery Level Indicator Code Explained

```
#include <ArduinoBLE.h>
BLEService batteryService("1101");
BLEUnsignedCharCharacteristic batteryLevelChar("2101", BLERead | BLENotify);
```

The first line of the code is to include the file `ArduinoBLE.h`. Then we will declare the Battery Service as well the battery level characteristics here. Here we will be giving two permissions – `BLERead` and `BLENNotify`.

`BLERead` will allow central devices (Mobile Phone) to read data from the Peripheral device (Arduino). And `BLENNotify` allows remote clients to get notifications if this characteristic changes.

Now we will jump on to the Setup function.

```
Serial.begin(9600);
while (!Serial);
pinMode(LED_BUILTIN, OUTPUT);
if (!BLE.begin()) {
    Serial.println("starting BLE failed!");
    while (1);
}
```

Here it will initialize the Serial Communication and BLE and wait for serial monitor to open.

Set a local name for the BLE device. This name will appear in advertising packets and can be used by remote devices to identify this BLE device.

```
BLE.setLocalName("BatteryMonitor");
```

```
BLE.setAdvertisedService(batteryService);
batteryService.addCharacteristic(batteryLevelChar);
BLE.addService(batteryService);
```

Here we will add and set the value for the Service UUID and the Characteristic.

```
BLE.advertise();
Serial.println("Bluetooth device active , waiting for connections ...");
```

And here, we will Start advertising BLE. It will start continuously transmitting BLE advertising packets and will be visible to remote BLE central devices until it receives a new connection.

```
BLEDevice central = BLE.central();
if (central) {
    Serial.print("Connected to central: ");
    Serial.println(central.address());
    digitalWrite(LED_BUILTIN, HIGH);
```

And here, the loop function. Once everything is setup and have started advertising, the device will wait for any central device. Once it is connected, it will display the MAC address of the device and it will turn on the builtin LED.

```
while (central.connected()) {
    int battery = analogRead(A0);
    int batteryLevel = map(battery, 0, 1023, 0, 100);
    Serial.print("Battery Level % is now: ");
    Serial.println(batteryLevel);
    batteryLevelChar.writeValue(batteryLevel);
    delay(200);
}
```

Now, it will start to read analog voltage from A0, which will be a value in between 0 and 1023 and will map it with in the 0 to 100 range. It will print out the battery level in the serial monitor and the value will be written for the batteryLevelchar characteristics and waits for 200 ms. After that the whole loop will be executed again as long as the central device is connected to this peripheral device.

```
digitalWrite(LED_BUILTIN, LOW);
Serial.print("Disconnected from central: ");
Serial.println(central.address());
Once it is disconnected, a message will be shown on the central device and LE
```

45.7.6. Installing the App for Android

In your Android smartphone, install the app “nRF Connect”. Open it and start the scanner. You will see the device “Battery Monitor” in the device list. Now tap on connect and a new tab will be opened.

Go to that and you will see the services and characteristics of the device. Tap on Battery service and you will the battery levels being read from the Arduino.

45.7.7. Example 2 – Arduino BLE Accelerometer Tutorial

I showed you a very easy example to show you how you can send simple data via Bluetooth. If you are new to this, try this example first. It will help to give you a better understanding of the BLE.

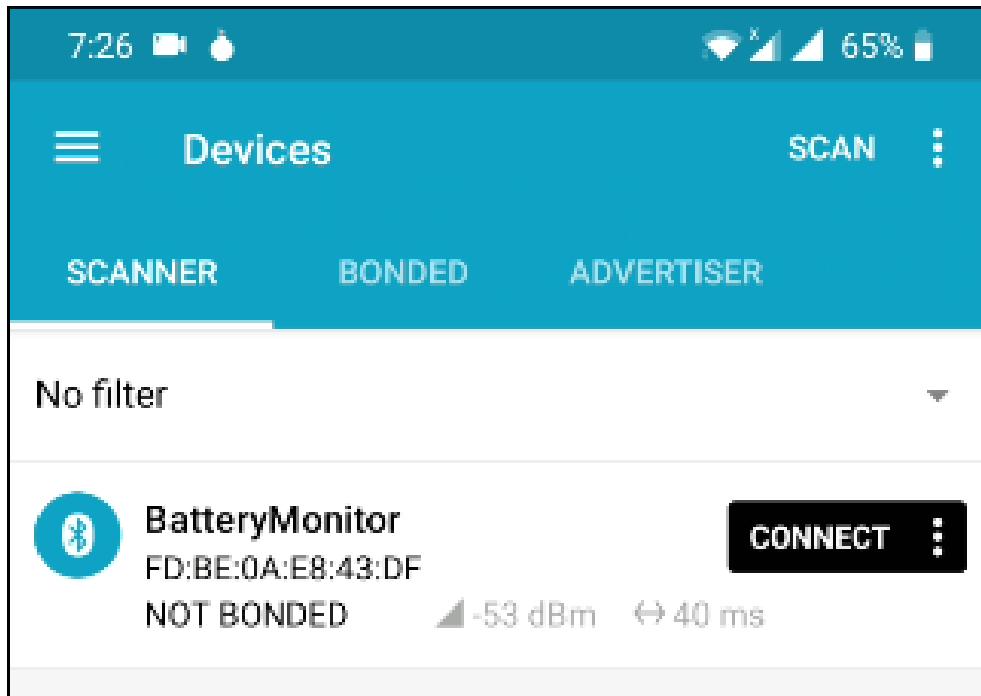


Figure 45.7.: Battery app “nRF Connect”

Step 1 – Installing Libraries

For this example, we will need two libraries

- [ArduinoBLE](#) – To Send Data via Bluetooth
- [LSM9DS1](#) – To Read data from inbuilt Accelerometer

Both these libraries are available in library manager. Simply search for that using the name and click on install.

Step 2 – Test Accelerometer Code (Optional)

Now we will try running the accelerometer code to make sure that these data are being read properly. For that, use the below code.

```
#include <Arduino_LSM9DS1.h>

void setup() {
    Serial.begin(9600);
    while (!Serial);
    Serial.println("Started");

    if (!IMU.begin()) {
        Serial.println("Failed to initialize IMU!");
        while (1);
    }

    Serial.print("Accelerometer sample rate = ");
    Serial.print(IMU.accelerationSampleRate());
    Serial.println(" Hz");
}
```

```

    Serial.println();
    Serial.println("Acceleration in G's");
    Serial.println("XtYtZ");
}

void loop() {
    float x, y, z;

    if (IMU.accelerationAvailable()) {
        IMU.readAcceleration(x, y, z);

        Serial.print(x);
        Serial.print('t');
        Serial.print(y);
        Serial.print('t');
        Serial.println(z);
    }
}

```

Once uploaded, start the serial monitor. You will see the data being populated. Try tilting the board in all direction and you will see the value changes accordingly.

Step 3 – Upload the Code

Now its time to upload the complete code. Copy the code below and paste it in the IDE.

```

#include <ArduinoBLE.h>
#include <Arduino_LSM9DS1.h>

int accelX=1;
int accelY=1;
float x, y, z;

BLEService customService("1101");
BLEUnsignedIntCharacteristic customXChar("2101", BLERead | BLENotify);
BLEUnsignedIntCharacteristic customYChar("2102", BLERead | BLENotify);

void setup() {
    IMU.begin();
    Serial.begin(9600);
    while (!Serial);

    pinMode(LED_BUILTIN, OUTPUT);

    if (!BLE.begin()) {
        Serial.println("BLE failed to initiate");
        delay(500);
        while (1);
    }

    BLE.setLocalName("Arduino Accelerometer");
    BLE.setAdvertisedService(customService);
    customService.addCharacteristic(customXChar);
    customService.addCharacteristic(customYChar);
    BLE.addService(customService);
}

```

```

customXChar.writeValue(accelX);
customYChar.writeValue(accelY);

BLE.advertise();

Serial.println("Bluetooth device is now active , waiting for connections ...");
}

void loop() {
    BLEDevice central = BLE.central();
    if (central) {
        Serial.print("Connected to central:");
        Serial.println(central.address());
        digitalWrite(LED_BUILTIN, HIGH);
        while (central.connected()) {
            delay(200);
            read_Accel();

            customXChar.writeValue(accelX);
            customYChar.writeValue(accelY);

            Serial.print("At Main Function");
            Serial.println("");
            Serial.print(accelX);
            Serial.print(" - ");
            Serial.println(accelY);
            Serial.println("");
            Serial.println("");
        }
    }
    digitalWrite(LED_BUILTIN, LOW);
    Serial.print("Disconnected from central:");
    Serial.println(central.address());
}

void read_Accel() {
    if (IMU.accelerationAvailable()) {
        IMU.readAcceleration(x, y, z);
        accelX = (1+x)*100;
        accelY = (1+y)*100;

    }
}

```

Select the right port and board. Click on upload.

Step 4 – Testing Arduino BLE Accelerometer

In your Android smartphone, install the app “nRF Connect”. Open it and start the scanner. You will see the device “Arduino Accelerometer” in the device list. Now tap on connect and a new tab will be opened.

Go to that and you will see the services and characteristics of the device.

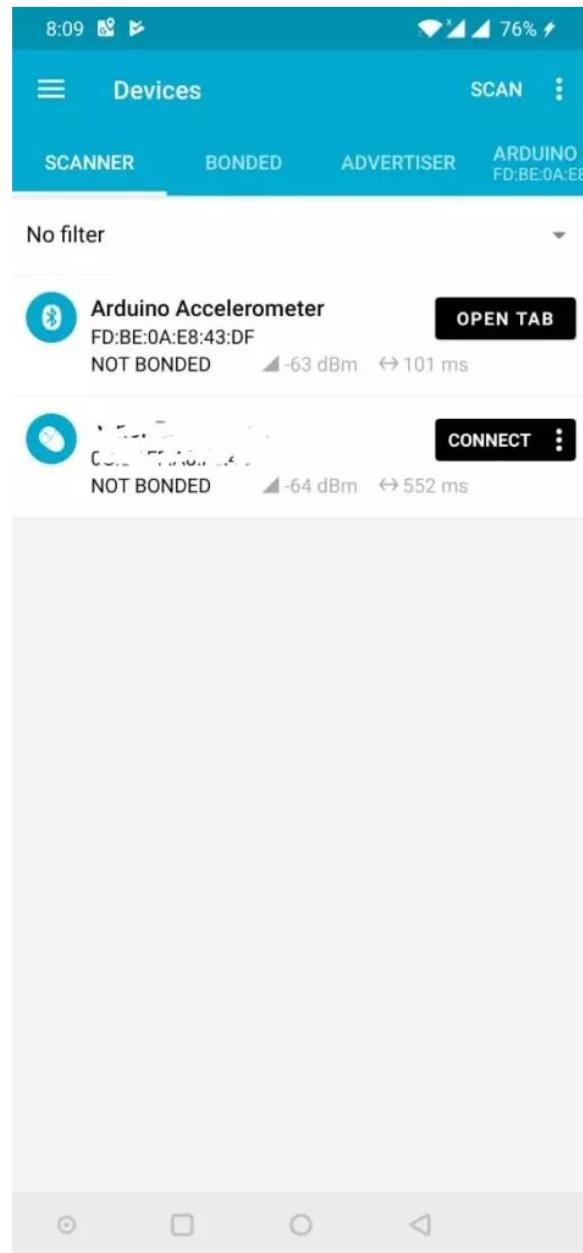


Figure 45.8.: nRF Connect Screen

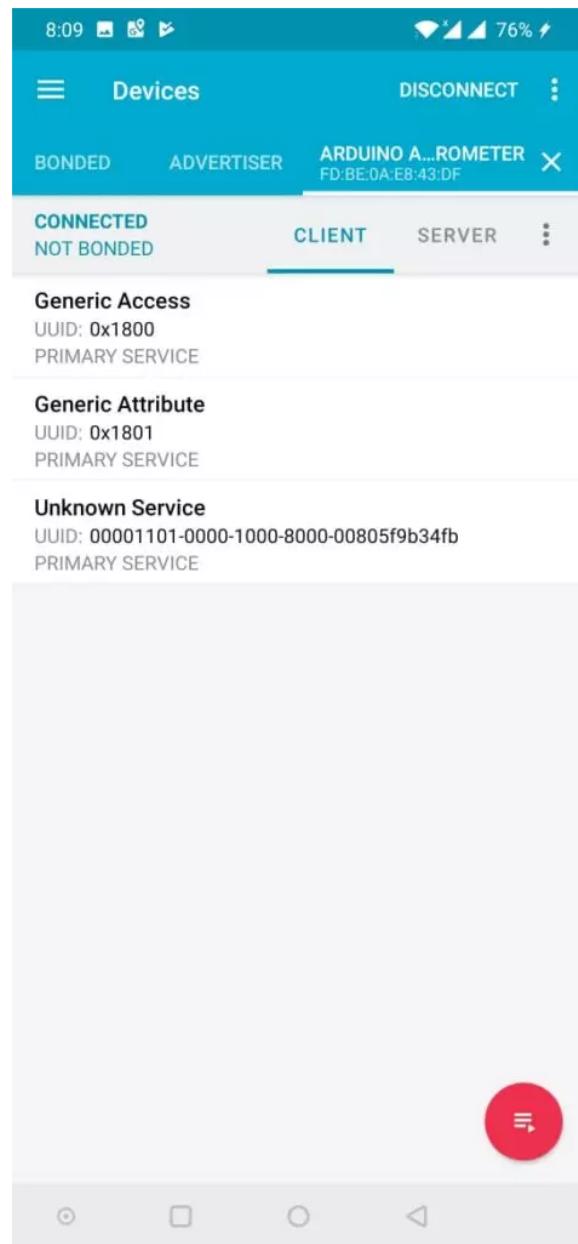


Figure 45.9.: Accelerometer Values Read from Arduino Using BLE

Tap on Unknown Service and you will see the accelerometer values being read from the Arduino.

In the next post, I will show you how you can send inbuilt sensor values such as accelerometer, gyroscope, color sensor and gesture sensor from the Arduino to your phone as well as another Arduino via BLE.

45.8. Arduino Library [BLEAK](#)

Low Energy (BLE) protocol. BLE allows you to exchange data between devices wirelessly and efficiently. You can use the bleak library in Python to interact with [BLE devices](#).

To get started, you need to install the bleak library using pip:

```
pip install bleak
```

Then, you need to write a Python program that can scan for BLE devices, connect to the Arduino Nano 33 BLE 33 Sense, and read or write data to its characteristics. Characteristics are the attributes that define the data and behavior of a BLE device. [For example, the Arduino Nano 33 BLE 33 Sense has a characteristic for each color of its RGB LED.](#)

You can find an example of a Python program that can control the RGB LED of the Arduino Nano 33 BLE 33 Sense using bleak [here3](#). You can also find the corresponding Arduino sketch that sets up the BLE service and characteristics for the Nano 33 BLE 33 Sense [here4](#).

45.9. Test

45.10. Test with Bluetooth Module Connection

The same procedure we need to follow for making the successful bleutooth coonection, the one we follow for on-board sensors. Below figure 45.10 shows the ArduinoBLE library in the example section of Arduino IDE.

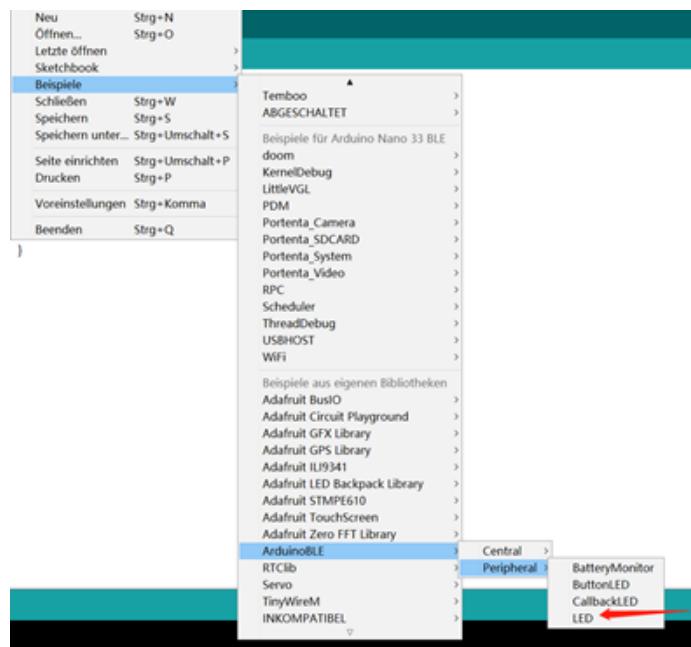
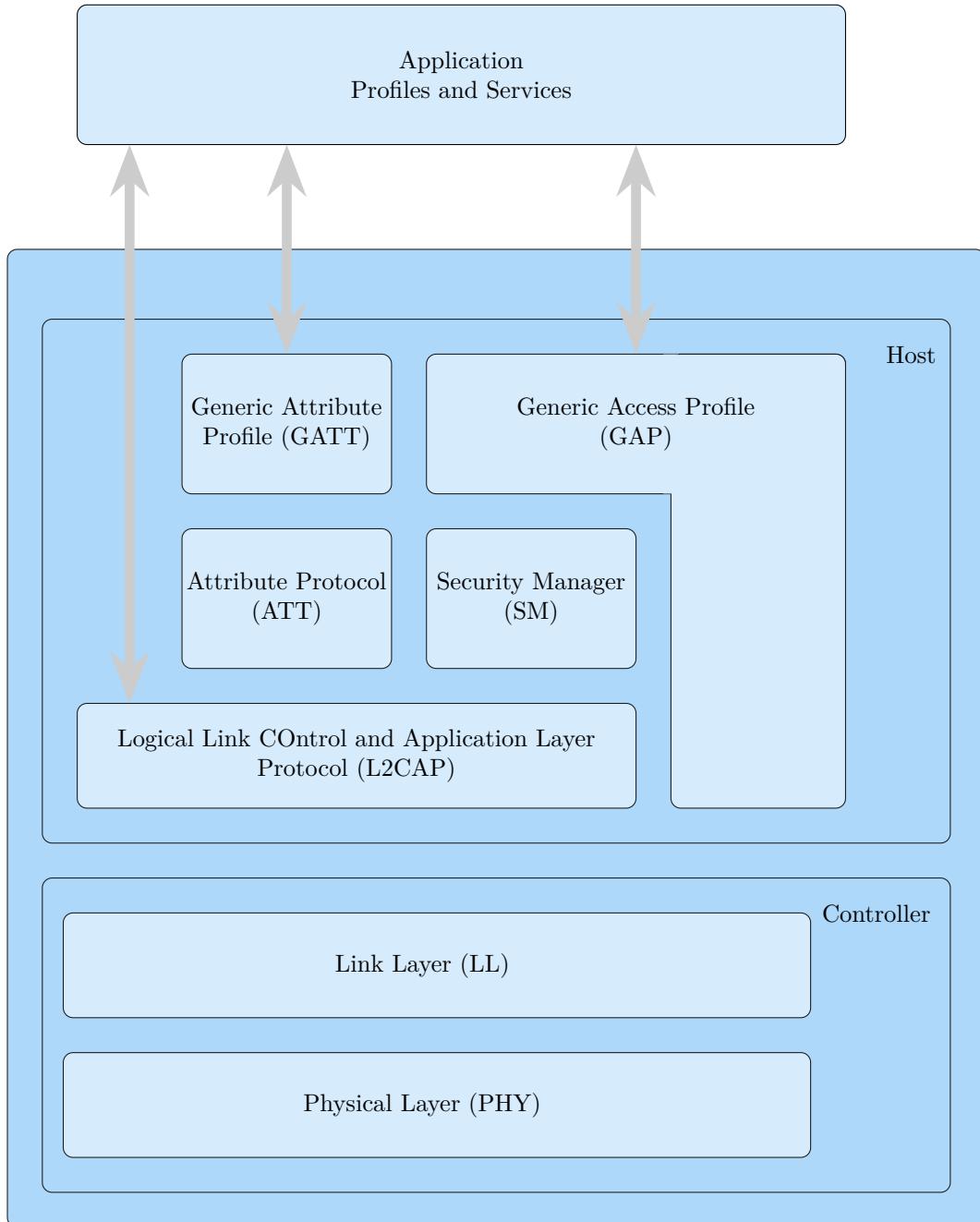


Figure 45.10.: Bluetooth Connection

Bluetooth wireless technology allows us to share the data, the voice, the music, the video, and a lot of information between paired devices. It is built into many products, from mobile phones, cars to medical devices and computers. It has lower power consumption. It is easily upgradeable. It has range better than Infrared communication. Bluetooth is used for voice and data transfer, we can communicate by receiving and sending the data with other bluetooth connected devices. It is also possible to output a value on the phone side or also on the laptop by making the proper pairing.

45.11. BLEStack



45.12. Control Arduino Nano BLE with Bluetooth & Python

<https://www.hackster.io/sridhar-rajagopal/control-arduino-nano-ble-with-bluetooth>
Demonstrated with the on-board RGB LED.

45.13. Peripheral Device aka Arduino Nano

The Generic Attribute Profile (or GATT) defines how Bluetooth LE devices can transfer data back and forth, using the Attribute Protocol (or ATT) which organizes

the data hierarchically into Service and Characteristics.

For our purpose, we define a top level service, as well as 3 characteristics, corresponding to each of the 3 colors of our RGB LED:

```
BLEService nanoService( "13012F00-F8C3-4F4A-A8F4-15CD926DA146" );
BLEByteCharacteristic redLEDCharacteristic( "13012F01-F8C3-4F4A-A8F4-15CD926DA146" );
BLEByteCharacteristic greenLEDCharacteristic( "13012F02-F8C3-4F4A-A8F4-15CD926DA146" );
BLEByteCharacteristic blueLEDCharacteristic( "13012F03-F8C3-4F4A-A8F4-15CD926DA146" );
```

The UUIDs used above have been randomly generated. They must match between the peripheral device (the Nano) and the central device (the Python program).

In our setup function, apart from setting up the pinMode for the RGB LEDs, we also setup our BLE service. Here, we set the local name for our BLE peripheral - central devices will see this when trying to connect to advertising services.

We add the service and characteristics we defined above, set their initial values, and start advertising. Here, at this point, central devices that are listening will see our advertising packets and know that we are ready to accept connections.

```
// set advertised local name and service UUID:
BLE.setLocalName("Arduino_Nano_33_BLE_Sense");
BLE.setAdvertisedService(nanoService);
// add the characteristic to the service
nanoService.addCharacteristic(redLEDCharacteristic);
nanoService.addCharacteristic(greenLEDCharacteristic);
nanoService.addCharacteristic(blueLEDCharacteristic);
// add service
BLE.addService(nanoService);
// set the initial value for the characteristic:
redLEDCharacteristic.writeValue(0);
greenLEDCharacteristic.writeValue(0);
blueLEDCharacteristic.writeValue(0);
// start advertising
BLE.advertise();
```

In our main loop(), we wait for a central device to connect. We then wait to see if any of the characteristics are written to, and the value of that characteristic. In our example, a non-zero value written to the redLEDCharacteristic, for example, will cause us to turn on the RED LED. When we encounter a zero value, we turn off the LED.

```
void loop() {
    BLEDevice central = BLE.central();
    if (central) {
        ...
        while (central.connected()) {
            if (redLEDCharacteristic.written()) {
                if (redLEDCharacteristic.value()) {
                    // any non-zero value
                    Serial.println("RED_LED_on");
                    digitalWrite(RED_PIN, LOW); // LOW will turn LED on
                } else { // a zero value
                    Serial.println(F("RED_LED_off"));
                    digitalWrite(RED_PIN, HIGH); // HIGH will turn LED off
                }
            }
        .... do similar stuff for GREEN and BLUE
    }
}
```

```

    // when the central disconnects, print it out:
    Serial.print(F("Disconnected from central:"));
    Serial.println(central.address());
}
}

```

Here we have to note that the RGB LEDs on the Arduino Nano 33 BLE Sense follow a reverse logic, so setting the pin LOW turns ON the LED and setting it HIGH turns it OFF.

The onboard RGB LED also does not support PWM, so the states of the LEDs can either be ON or OFF, giving us 7 different colors:

R	G	B	
OFF	OFF	OFF	— OFF
ON	OFF	OFF	— RED
OFF	ON	OFF	— GREEN
OFF	OFF	ON	— BLUE
ON	ON	OFF	— ORANGE
ON	OFF	ON	— PURPLE
OFF	ON	ON	— CYAN
ON	ON	ON	— WHITE

The Central device can also read the characteristic's value. This is automatically handled by the BLE library and we don't need to do anything more. We initialize the value appropriately on startup, and maintain the value, so it can be read anytime.

Let's dive into the code - Central Device aka Python On the Python side, we utilize the Python bleak library for BLE support - of course, the machine we are running it on needs to have Bluetooth support as well!

We also use the asyncio asynchronous framework for our program. Our main loop here is the run() function. It goes into discover mode and checks the local names of the Bluetooth devices that are advertising, and connects to the device with the local name of "Arduino Nano 33 BLE Sense" (the local name we chose).

It then reads the RED, GREEN and BLUE Characteristics to know the initial states of the LEDs, and then goes into the setColor() loop:

```

async def run():
    global RED, GREEN, BLUE
    print('ProtoStax\u2014Arduino\u2014Nano\u2014BLE\u2014LED\u2014Peripheral\u2014Central\u2014Service')
    print('Looking\u2014for\u2014Arduino\u2014Nano\u201433\u2014BLE\u2014Sense\u2014Peripheral\u2014Device...')

    found = False
    devices = await discover()
    for d in devices:
        if 'Arduino\u2014Nano\u201433\u2014BLE\u2014Sense' in d.name:
            print('Found\u2014Arduino\u2014Nano\u201433\u2014BLE\u2014Sense\u2014Peripheral')
            found = True
            async with BleakClient(d.address) as client:
                print(f'Connected\u2014to\u2014{d.address}')
                val = await client.read_gatt_char(RED_LED_UUID)
                if (val == on_value):
                    print ('RED\u2014ON')
                    RED = True
                else:
                    print ('RED\u2014OFF')
                    RED = False
    ... do similar stuff for GREEN and BLUE
    while True:
        await setColor(client)

```

```
if not found:  
print('Could not find Arduino Nano 33 BLE Sense Peripheral')
```

In `setColor()`, it prompts for user input, and checks the user input to see if the string has any 'r', 'g' or 'b' values in it. It then toggles the color and writes the appropriate value to the characteristic, and waits for further user input again. This continues until the user hits Control-C to stop the program.

```
async def setColor(client):  
    global RED, GREEN, BLUE  
    val = input('Enter rgb to toggle red, green and blue LEDs: ')  
    print(val)  
    if ('r' in val):  
        RED = not RED  
        await client.write_gatt_char(RED_LED_UUID, getValue(RED))  
    ... do similar stuff for GREEN and BLUE
```

45.13.1. Taking the Project Further

The Arduino Nano has a whole bunch of sensors onboard. You can extend the project by adding characteristics for the other sensors - for example, reading the temperature, humidity and pressure values (there is no need to write these values, so the characteristics can be defined read-only). You'll also need to modify the program to handle the read command - polling the sensor and writing the data with the current value of the sensor.

On the central device (Python Program), you can choose to store the data along with the timestamp of when it was read, to a database or comma-separated file. This way, your central device can act like a data logger, and you can even utilize the data to perform actions, or send the data to a cloud service like AWS to process and run analytics on. Having the central device on Python allows you to utilize the numerous packages and libraries and capabilities of Python.

Happy Coding! If you have any questions, feel free to ask them in the comments below! We'll also be happy to hear how you have taken the project further and what wonderful creations you have made!

46. Ethernetschnittstelle ENC28J60

Die ARD NET ENC28J60 Netzwerk- Schnittstelle 46.1 erweitert den Arduino über eine 10-Pin Buchsenleiste um eine Netzwerkschnittstelle für Ethernet- LAN. Dadurch wird es möglich einen Webserver aufzubauen oder Daten mit anderen Geräten auszutauschen. Das Modul kommuniziert mit dem Arduino über die SPI-Schnittstelle. In der SPI-Kommunikation werden Daten gleichzeitig übertragen und empfangen, wobei die synchronen Taktflanken das Timing für das Senden und Empfangen der Daten steuern. Aufgrund von Taktsignalen bis in den MHz Bereich, ist eine sehr hohe Rate der Datenübertragung möglich. Damit die Schnittstelle benutzt werden kann, wird ein Arduino, wie der 33 BLE Sense, benötigt. Außerdem bedarf es Steckbrücken und ein Netzwerkkabel zum Router oder Switch. Über Stiftkontakte wird die Verbindung zum Arduino hergestellt. Diese Kontakte dienen zum einem dem Datenaustausch über die SPI-Schnittstelle, sowie der Spannungsversorgung und dem Massenanschluss. Damit das Modul betrieben werden kann, muss zusätzlich eine zugehörige "ENC28J60"-Bibliothek in Arduino IDE heruntergeladen und installiert werden. Die Bibliothek beinhaltet zugehörigen Beispiel Code, der unter "Examples" hinzugefügt wird. Dort müssen dann lediglich die Anschlusspins angepasst werden. [Con24] [Dha21]

46.1. Netzadapter



Figure 46.2.: Netzadapter PA0217

Damit das automatisierte System mit Strom versorgt werden kann, benötigt es ein Netzadapter bzw. USB-Ladegerät. 46.2 In diesem Fall handelt es sich um das "XTAR 5VWA USB-Ladegerät 5V 2,1A". Die 5V im Titel beziehen sich auf den Output des Netzteils. Dabei wird eine Stromstärke von 2100mA ausgegeben. Die Input-Spannung darf sich zwischen 100 V und 240 V Wechselstrom befinden. Bei dem Adapter handelt es sich um einen Eurostecker vom Typ C auf der einen Seite und einem weiblichen USB-Port vom Typ A auf der anderen Seite. Das Netzteil hat eine Tiefe von 80 mm, eine Höhe von 25 mm und eine Breite von 39 mm. [Log24]



Figure 46.1.: Netzwerkschnittstelle ENC28J60

46.2. Test des Ethernet-Shield ENC28J60

Der Code [LinkStatus.ino](#) 46.1 überprüft die Verbindung zum ENC28J60 Ethernet Shield, welches per SPI Datenbus angeschlossen wird. Der Status wird jede Sekunde überprüft und über die serielle Schnittstelle ausgegeben. Der Sketch gibt "Unknown" aus, wenn der Verbindungsstatus nicht bestimmt werden kann, "ON" wenn das Kabel angeschlossen ist und "OFF", wenn das Kabel nicht angeschlossen ist.

46.3. Test des Web Servers

Dieser Code 46.2 implementiert einen Webserver auf dem Arduino. Die Kommunikation zwischen Client und Server basiert auf dem HTTP-Protokoll. Er ermöglicht es dem Arduino Anfragen zu empfangen, zu verarbeiten und entsprechend zu antworten. Zunächst wurde die Funktion durch Eingabe in der Adresszeile die LED anzusteuern hinzugefügt. Danach wurden Buttons auf der Website 46.3 per HTML Formular eingefügt. Nachdem dieser Schritt erfolgreich war, wurde der Code erweitert. Durch Betätigen der Schaltfläche "Bildschirmaufnahme", soll der Void Capture aktiviert werden. Überprüft wurde dies zunächst durch das Aufleuchten der LED. Nachdem überprüft wurde, dass der Void Capture im Skript ausgeführt wird, sollte der Livestream erfolgen. Bevor die übertragenen Bytes ausgewertet wurden, trat das Problem auf, dass durch die große Datenmenge der Arbeitsspeicher im Arduino überlastet wurde. Im nächsten Schritt sollte dann die Datenübertragung als einzelne Bilder per Hexadezimal-String realisiert werden.

46.4. Datenübertragung

Um die Datenübertragung zu realisieren, muss die Datenmenge verkleinert oder komprimiert werden 46.3 . Zunächst wurde die Auflösung auf 176x144 Pixel reduziert. Bei einer Farbtiefe von 16 Bit pro Pixel entspricht das 405.504 Bits. Danach wurde von der seriellen Übertragung einzelner Bits auf die Übertragung eines Hexadezimal-Strings umgestellt. Hexadezimalzeichenfolgen sind für Menschen besser lesbar und einfacher zu debuggen als Binärdaten. Jedes Hexadezimalzeichen repräsentiert dabei 4 Bit, was zu einer kompakteren und effizienteren Darstellung von Binärdaten führt.

```

/*
LinkStatus

Dieser Sketch gibt den Ethernet-Verbindungsstatus aus. Wenn das
Ethernet-Kabel angeschlossen ist, sollte der Verbindungsstatus
"ON" anzeigen.

*/
#include <SPI.h>
#include <EthernetENC.h>

void setup() {

    Ethernet.init(10); // Most Arduino shields

    Serial.begin(9600);
}

void loop() {
    auto link = Ethernet.linkStatus();
    Serial.print("Link status: ");
    switch (link) {
        case Unknown:
            Serial.println("Unknown");
            break;
        case LinkON:
            Serial.println("ON");
            break;
        case LinkOFF:
            Serial.println("OFF");
            break;
    }
    delay(1000);
}

```

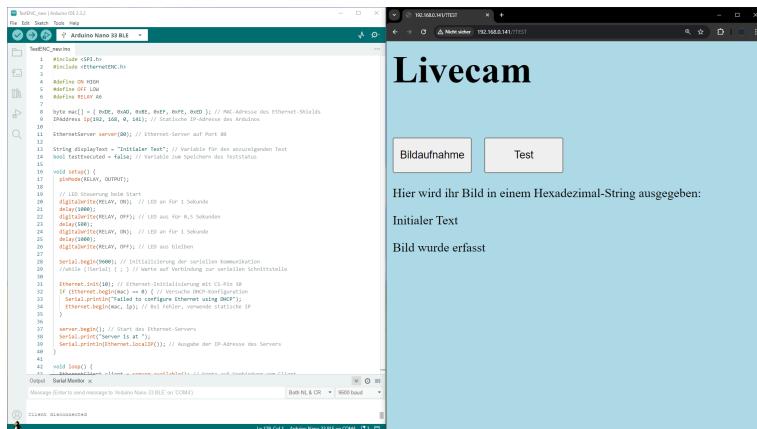
Listing 46.1.: Der Sketch [LinkStatus.ino](#) in Arduino für das Microcontroller Board

Figure 46.3.: GUI

```

/*
TestENC

Dieser Code implementiert einen Webserver auf einem Arduino
mit einem Ethernet-Shield, der es ermöglicht, über eine
Webschnittstelle eine LED zu steuern und Aktionen auszulösen.

*/
#include <SPI.h>
#include <EthernetENC.h>

#define ON HIGH
#define OFF LOW
#define RELAY A6

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED }; // MAC-Adresse des Ethernet-
IPAddress ip(192, 168, 0, 141); // Statische IP-Adresse des Arduinos

EthernetServer server(80); // Ethernet-Server auf Port 80

String displayText = "Initialer Text"; // Variable für den anzuzeigenden Text
bool testExecuted = false; // Variable zum Speichern des Teststatus

void setup() {
    pinMode(RELAY, OUTPUT);

    // LED Steuerung beim Start
    digitalWrite(RELAY, ON); // LED an für 1 Sekunde
    delay(1000);
    digitalWrite(RELAY, OFF); // LED aus für 0,5 Sekunden
    delay(500);
    digitalWrite(RELAY, ON); // LED an für 1 Sekunde
    delay(1000);
    digitalWrite(RELAY, OFF); // LED aus bleiben

    Serial.begin(9600); // Initialisierung der seriellen Kommunikation
    //while (!Serial) { ; } // Warte auf Verbindung zur seriellen Schnittstelle

    Ethernet.init(10); // Ethernet-Initialisierung mit CS-Pin 10
    if (Ethernet.begin(mac) == 0) { // Versuche DHCP-Konfiguration
        Serial.println("Failed to configure Ethernet using DHCP");
        Ethernet.begin(mac, ip); // Bei Fehler, verwende statische IP
    }

    server.begin(); // Start des Ethernet-Servers
    Serial.print("Server is at ");
    Serial.println(Ethernet.localIP()); // Ausgabe der IP-Adresse des Servers
}

void loop() {
    EthernetClient client = server.available(); // Warte auf Verbindung vom Client
    if (client) {
        Serial.println("New client");
        bool currentLineIsBlank = true;
        String request = "";

        while (client.connected()) {
            if (client.available()) {
                char c = client.read();
                request += c;

                if (c == '\n' && currentLineIsBlank) {
                    Serial.println(request);
                    currentLineIsBlank = false;
                }
            }
        }
    }
}

```

```

/*
  CaptureSingleHexImage

  Dieser Code nimmt einzelne Bilder auf die als
  Hexadezimal-String gespeichert werden.

*/
#include <Arduino_OV767X.h>

unsigned short pixels[176 * 144]; // QCIF: 176x144 X 2 bytes per pixel (RGB565)

void setup() {
  Serial.begin(9600);
  while (!Serial);

  Serial.println("OV767X Camera Capture");
  Serial.println();

  if (!Camera.begin(QCIF, RGB565, 1)) {
    Serial.println("Failed to initialize camera!");
    while (1);
  }

  Serial.println("Camera settings:");
  Serial.print("\twidth=");
  Serial.println(Camera.width());
  Serial.print("\theight=");
  Serial.println(Camera.height());
  Serial.print("\tbits_per_pixel=");
  Serial.println(Camera.bitsPerPixel());
  Serial.println();
}

Serial.println("Send the 'c' character to read a frame ...");
Serial.println();
}

void loop() {
  if (Serial.read() == 'c') {
    Serial.println("Reading frame");
    Serial.println();
    Camera.readFrame(pixels);

    int numPixels = Camera.width() * Camera.height();

    for (int i = 0; i < numPixels; i++) {
      unsigned short p = pixels[i];

      if (p < 0x1000) {
        Serial.print('0');
      }

      if (p < 0x0100) {
        Serial.print('0');
      }

      if (p < 0x0010) {
        Serial.print('0');
      }

      Serial.print(p, HEX);
    }
  }
}

```


47. Heart Rate Sensor

47.1. The Heartbeat

47.1.1. Introduction

The heartbeat, also known as pulse or heart rhythm, is a central element of the human circulatory system. The heart is a muscular hollow organ, approximately the size of a fist, located in the thorax between the lungs. It consists of four chambers: the two atria and the two ventricles. This anatomy allows the heart to pump blood efficiently throughout the body, supplying tissues and organs with oxygen and nutrients while removing waste products. [GV17]

47.1.2. Cardiac Cycle

The heartbeat occurs in a regular cycle known as the cardiac cycle. This cycle consists of two main phases: diastole and systole. During diastole, the heart relaxes and fills with blood from the atria. The atria contract to pump blood into the ventricles; the ventricles then contract to pump blood out of the heart and into the circulatory system. This rhythmic alternation between contraction and relaxation enables efficient blood flow throughout the body. [GV17]

47.1.3. Electrical Activity

The electrical activity of the heart plays a crucial role in controlling the heartbeat. The sinoatrial node, also known as the natural pacemaker of the heart, is located in the right atrium and sends out electrical signals that initiate the contraction of the heart muscle. These electrical impulses spread through specialized conduction pathways within the heart, such as the AV node and the bundle of His, stimulating the synchronized contraction of the entire heart muscle. [GV17]

47.1.4. Heart Rate

Heart rate, measured in beats per minute (beats per minute (bpm)), varies depending on factors such as age, fitness level, and individual conditions. The average resting heart rate for adults is typically between 60 and 100 beats per minute. A lower heart rate can indicate good heart health and efficient heart function. An elevated heart rate, however, may result from factors such as stress, physical activity, or illness. [SIB21] Figure 47.1 shows an example frequency. The x-axis represents time in seconds (s), and the y-axis represents the frequency in bpm. The RR interval is the distance between two R peaks in an ECG, from which the heart rate (HR) can be mathematically derived:

WS:R peak? RR? ECG
Picture?

$$HR \left[\frac{1}{\text{min}} \right] = \frac{60}{\text{RR interval [s]}}$$

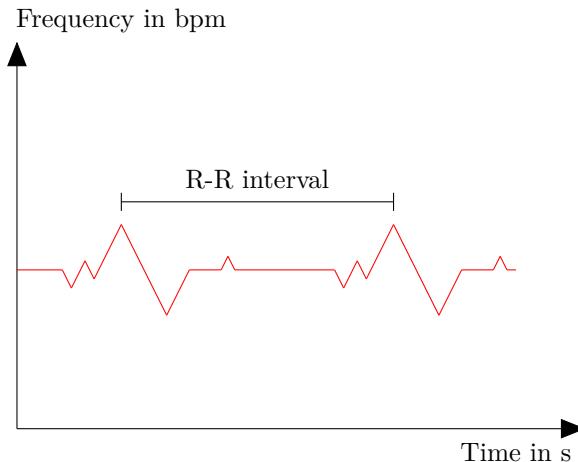


Figure 47.1.:]
Example frequency of a heartbeat

47.1.5. Regulation of the Heartbeat

The heartbeat is regulated by various factors. In addition to the central nervous system, which can increase or decrease the heart rate as needed, hormones like adrenaline and noradrenaline, as well as certain medications, can influence the heartbeat. Emotions such as fear or excitement can also impact the heartbeat by activating the sympathetic nervous system, leading to an increased heart rate. [GV17]

47.1.6. Heartbeat Deviations

Deviations in the heartbeat, such as arrhythmias (irregular heart rhythms), can be caused by various factors, including heart disease, electrolyte imbalances, or medications. An elevated heart rate (tachycardia) or a decreased heart rate (bradycardia) can also indicate medical issues and may require clinical examination and treatment. [GV17]

Sieht man das in den Daten?

47.2. Operation of the Heartbeat Sensor

The sensor “GRV Heart Rate 3” is an optical heart rate sensor that uses photoplethysmography (Photoplethysmographie (PPG)) to measure heart rate. [GM23]

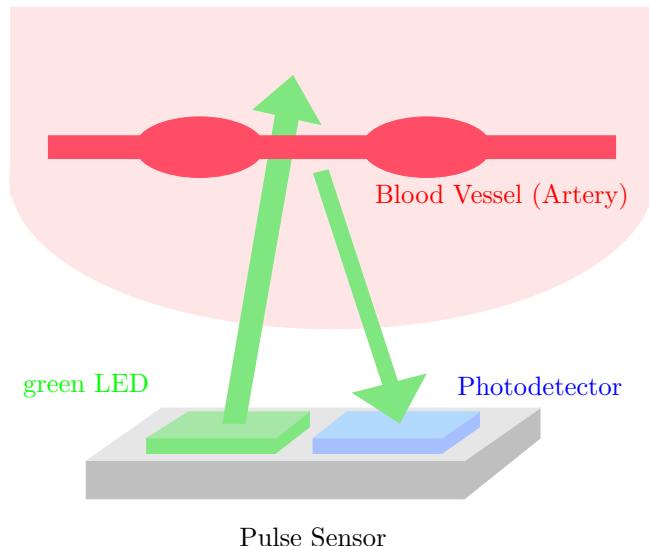


Figure 47.2.: Photoplethysmography

- **Light Source and Photodetector:** The sensor consists of LEDs (usually green light) and a photodetector. The LEDs emit light into the skin.
- **Measurement of Light Reflection:** The light is directed into the skin and is partially absorbed by the blood vessels. The photodetector measures the amount of reflected light.
- **Detection of Blood Flow Changes:** With each heartbeat, the amount of reflected light changes due to the varying blood flow in the vessels. These changes are detected by the photodetector.
- **Calculation of Heart Rate:** The changes in light reflection caused by the pulse are used to calculate the heart rate.

The functional principle of the sensor is shown schematically in Figure 47.2.

WS:more explanations

47.2.1. The Photodetector

The photodetector utilizes the photoelectric effect, a phenomenon where electrons are emitted from a material (usually a metal) when it is exposed to light. [TR14]. Figure 47.3 illustrates this phenomenon.

Output of the Photodetector

The photodetector generates a voltage that is measured over time. This voltage varies periodically, depending on the amount of reflected light, which is influenced by changes in blood volume caused by the heartbeat.

Therefore, the variation in the photodetector's voltage is directly correlated with the frequency of the heartbeat. [TR14].

47.2.2. Voltage Data Processing

The sensor “GRV Heart Rate 3” outputs analog voltage values ranging from 0 to 3.3 volts. These voltages are captured by the Arduino through an analog input. The Arduino uses its built-in Analog to Digital Converter (ADC) to convert these analog voltages into digital values. This process is illustrated in Figure 47.4.

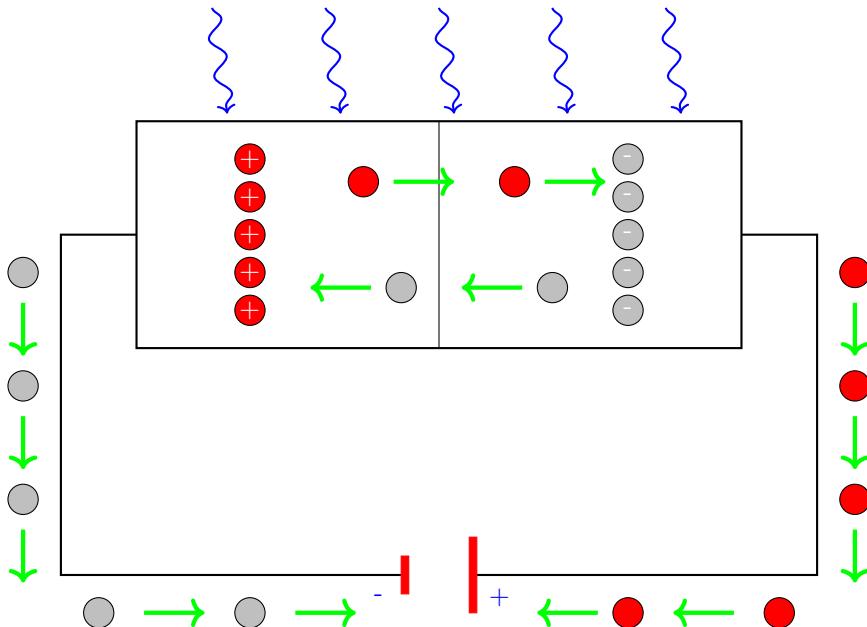


Figure 47.3.: The Photoelectric Effect

The Arduino's ADC has a resolution of 10 bits, meaning it divides the voltage range from 0 to 3.3 volts into 1024 steps. Thus, a voltage of 0 volts is represented as a digital value of 0, and a voltage of 3.3 volts is represented as a digital value of 1023. The voltage values between these two extremes are proportionally converted into corresponding digital values. [Par16]

WS:really proportional?

This conversion allows the Arduino to process the analog voltages output by the sensor as easily manageable digital values. These digital values are then used in the code to calculate the heart rate by counting the number of peaks in the voltage data over a certain period and converting this into beats per minute.

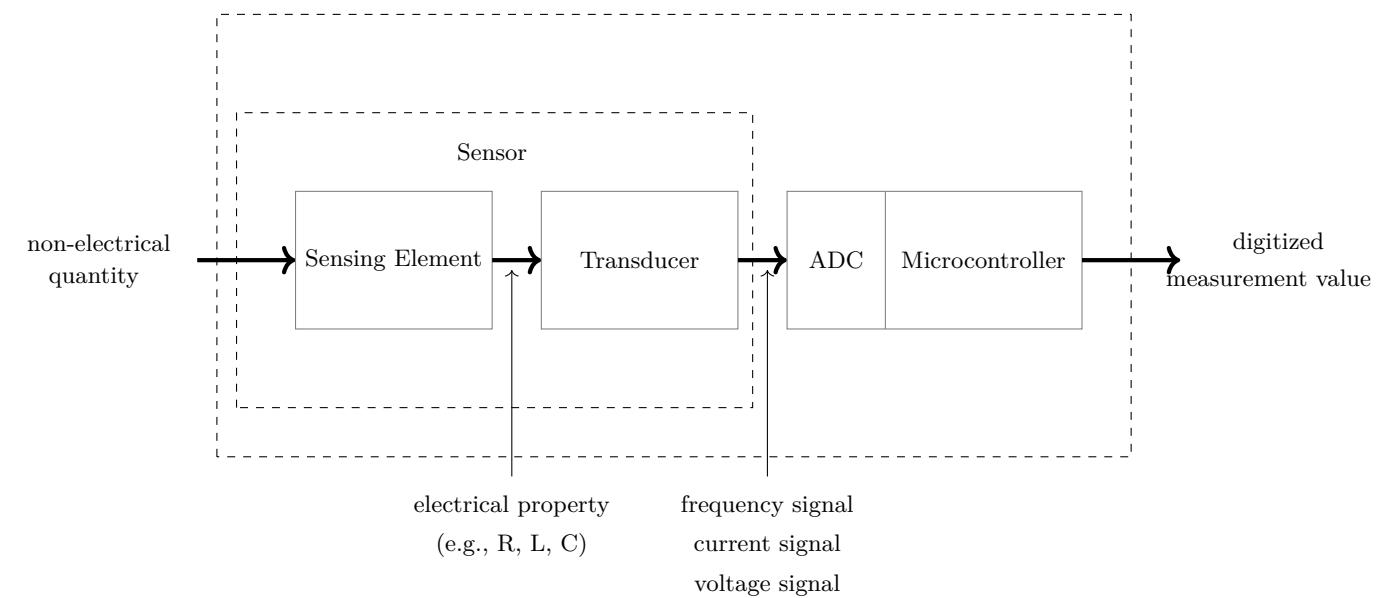


Figure 47.4.: Structure of a Sensor

WS:smaller

47.2.3. Drift and Measurement Inaccuracies

Drift can play a significant role in heartbeat sensors. Drift refers to the gradual change in sensor measurements over time, which is not due to actual changes in the measured parameter. In the context of a heartbeat sensor, various types of drift can occur. A heartbeat sensor converts physiological properties, such as changes in blood flow, into electrical signals for analysis. However, it is possible that the readings fluctuate even though the heart rate remains constant. This unwanted variation in readings is referred to as drift. [TR14]. Drift is time-dependent and occurs due to aging processes and inherent inaccuracies of the sensor. In the worst case, drift can lead to the sensor's functional failure.

WS:Algorithms? Value

Electronic Drift refers to changes in the electronics of the sensor or the Arduino board itself. Electronic components can change their properties over time due to temperature fluctuations, aging, or other factors, leading to drift in the output values.

Optical Drift occurs in heartbeat sensors that operate based on photoplethysmography (PPG) by measuring the light absorption through the blood. Changes in optical components, such as LEDs or photodetectors, can cause drift, for example, due to contamination, aging of the LEDs, or changes in tissue permeability.

Mechanical Drift can be caused by mechanical changes or shifts of the sensor on the skin. Poor or shifting placement of the sensor can affect the accuracy of the measurements.

Calibration Drift occurs in heartbeat sensors that require regular calibration to ensure accurate measurements. Without regular calibration, the sensor can begin to drift and provide inaccurate readings.

Several measures can be taken to minimize the effects of drift:

Regular Calibration: If the sensor requires calibration, it should be performed regularly to ensure the measurements remain accurate.

Continuous Monitoring: Continuous monitoring of output values and comparison with known reference values (e.g., comparison with a clinically validated device) can help detect and correct drift.

Sensor Placement: Consistent and stable placement of the sensor on the body can minimize mechanical drift. Ideally, the sensor should be attached to an area with minimal movement, such as the earlobe.

Environmental Control: Controlling the temperature and environmental conditions can reduce electronic drift.

By considering these measures, the accuracy and reliability of heart rate measurements using the heartbeat sensor can be improved. [TR14].

WS:How to do? Where
WS:How?

47.3. Heart Rate Sensor with Grove Interface and Ear Clip

The GRV Heart Rate 3 sensor is designed to measure heart rate using infrared light. It detects pulse waves in the bloodstream through an ear-mounted clip (see figure 47.5), providing continuous heart rate data. This type of measurement is commonly used in health monitoring systems and fitness devices. [See24]

WS:citations?!

47.3.1. Key Technical Specifications

Along the data sheet, see [See24], the sensor has the following specifications:

- **Measurement Principle:** Optical pulse detection using infrared light
- **Power Supply:** 3.0V (min), 5.0V (typical), 5.25V (max)
- **Output Signal:** Analog pulse frequency signal
- **Current Consumption:** 6.5mA (typical), \leq 10mA
- **Response Time:** Less than 2 seconds
- **Measurement Range:** \geq 30 beats per minute, up to 240 beats per minute
- **Operating Temperature:** 0°C to +50°C
- **Interface Type:** Analog
- **Length of Ear Clip Wire:** 120cm
- **Weight:** 0.064kg
- **Connector Type:** JST-VH connector
- **Connection System:** Grove interface



Figure 47.5.: Heart Rate Sensor with Grove Interface and Ear Clip[See15]

47.4. Simple Function Test

47.4.1. Heart Rate Detection - Manual

To test the heart rate sensor, a simple program is used to detect peaks. The heart rate sensor is connected to the Arduino via the Grove cable through the shield, and the

measured value is sent to the computer every 500 milliseconds via the serial interface. The serial output is displayed on a plotter in a graph that shows periodic heart rate signals. The values fluctuate between 0 and 1023, indicating the detected peaks from the heart rate sensor. This confirms that the sensor is functioning properly and that heart rate data is being successfully captured and visualized.

47.4.2. Heart Rate Detection - Code

The variable `heartRatePin` defines the pin where the heart rate sensor is connected, in this case using analog pin `A0`. The variable `heartRateValue` stores the sensor's analog reading, which ranges from 0 to 1023. The `previousMillis` variable keeps track of the last time the sensor was read, allowing the sketch to manage timing without blocking the loop.

WS:Pin no?

In the setup phase, `Serial.begin(9600)` initializes serial communication at a baud rate of 9600 to send data to the serial monitor, which is useful for debugging. The function `pinMode(heartRatePin, INPUT)` configures the specified pin as an input to read the analog signal from the heart rate sensor.

In the function `loop`, the function `millis()` is used to get the number of milliseconds that have passed since the microcontroller started running. The condition `currentMillis - previousMillis >= interval` checks if the set time interval of `500` milliseconds has passed. If so, it triggers a new sensor reading, ensuring periodic heart rate measurements without blocking the loop.

47.4.3. Heart Rate Detection - File

The program can be found at:

```

1000 // Code to test the GRV Heart Rate 3 sensor.
1001 // The sensor's raw values are output to the serial monitor at an
1002 // interval of 500 ms.
1003
1004 // file: TestHeartRate.ino
1005
1006 int heartRatePin = A0;
1007 int heartRateValue = 0;
1008
1009 // Timing variables for controlling reading intervals
1010 unsigned long previousMillis = 0;
1011 const long interval = 500; // Interval for readings in milliseconds
1012
1013 void setup() {
1014     Serial.begin(9600);
1015     pinMode(heartRatePin, INPUT);
1016 }
1017
1018 void loop() {
1019     // Get the current time in milliseconds
1020     unsigned long currentMillis = millis();
1021
1022     // Check if the specified interval has passed
1023     if (currentMillis - previousMillis >= interval) {
1024         // Update the previous time marker
1025         previousMillis = currentMillis;
1026
1027         // Read the analog value from the heart rate sensor
1028         heartRateValue = analogRead(heartRatePin);
1029         // Output the heart rate value to the serial monitor
1030         Serial.println(heartRateValue);
1031     }
1032 }
```

..../Code/Arduino/HeartRate/TestHeartRateSensor/TestHeartRateSensor.ino

Listing 47.1.: Test sketch for the heart rate sensor

47.5. Kalibrierung

WS:to do!! 47.5.1. Calibration

For accurate measurements, regular calibration of the sensor is recommended. To calibrate:

1. Measure your pulse manually or with an external device.
2. Calculate the calibration factor by dividing your manually measured pulse rate by the monitor's displayed rate.
3. Enter this calibration factor in line 48 of the code, under `calibrationFactor`.

WS:no go

47.6. Further Readings

- Seeed Studio, "Grove - Ear-clip Heart Rate Sensor GitHub Repository." The official GitHub repository with sample code and resources for the Grove Ear-clip Heart Rate Sensor. Available at: https://github.com/Seeed-Studio/Grove_Ear_Clip_Heart_Rate_Sensor
- Reichelt Electronics, "GRV HEART RATE3 - Arduino - Heartbeat Sensor, Ear-Clip." Product page with technical details and specifications for the GRV Heart Rate 3 sensor. Available at: <https://www.reichelt.de/arduino-herzschlagsensor-ohr-clip.html>

VS:this aren't citations!
Find some

48. Application of Heart Rate Monitor with OLED Display

48.1. Overview

The portable Heart Rate Monitor is designed for real-time heart rate tracking and data storage. The device consists of the following components:

- GRV Heart Rate 3 Sensor
- Arduino Nano 33 BLE Sense
- 1.30" IIC OLED Display
- Micro SD Card Reader Module
- Two Wago 221 Clamps
- Power switch
- 9V battery

WS:material list?

48.2. Functionality

This monitor, which can be conveniently attached to items such as a belt using a clamp, provides users with real-time feedback and logging capabilities. Key features include:

- **Real-time Heart Rate Display:** The current heart rate is displayed on the OLED screen.
- **Graphical Heart Rate Tracking:** Heart rate trends are represented graphically on the OLED, showing data over time.
- **Data Logging:** Heart rate values are saved to the micro SD card for future reference and analysis.

WS:Screen shots?
Photos?

48.3. Manual

48.3.1. Setup of the Device

The circuit Heart Rate Monitor is illustrated in the circuit diagram, see Figure 48.1. The entire assembly is housed within an enclosure, with only the ear clip accessible externally. The OLED display is mounted on the enclosure lid for easy readability.

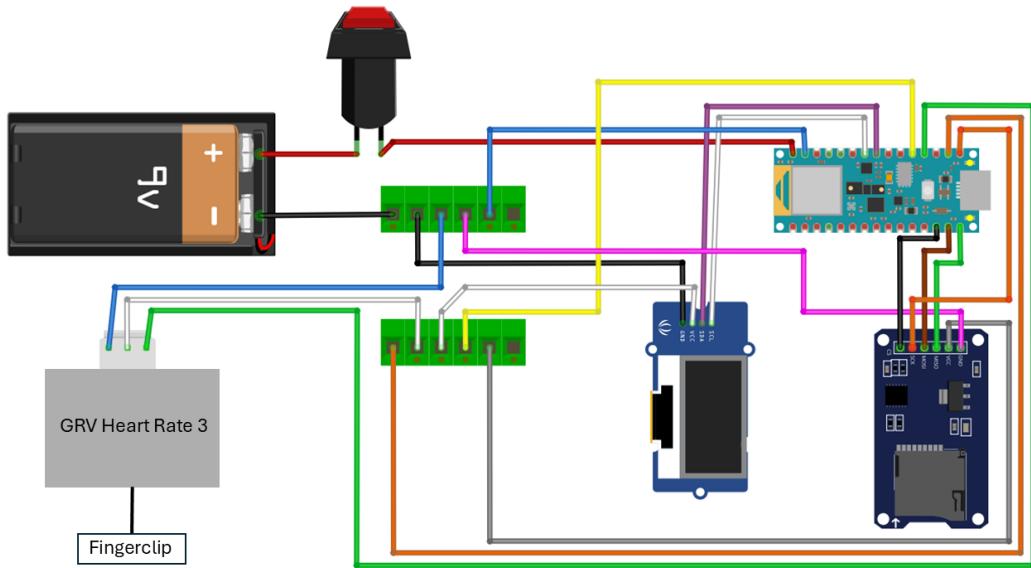


Figure 48.1.: Circuit Diagramm of Application

48.3.2. Programming the Device

To program the Arduino, connect it to a computer and upload the Application-HeartRate code, which can be found at:

Listing 48.1.: Application Heart Rate Sensor

```

1000 // Code for using the GRV Heart Rate 3 Sensor as a heart rate monitor.
1001 // To calibrate , measure the actual pulse and divide it by the pulse
1002 // measured by the GRV Heart Rate 3 Sensor .
1003 // Enter this factor as calibrationFactor (line 48).
1004
1005 // Required components:
1006 // - Arduino Nano 33 BLE Sense
1007 // - GRV Heart Rate 3 Sensor
1008 // - 1.30" IIC OLED Display
1009 // - micro SD card reader with an SD card
1010
1011 // file: ApplicationHeartRate.ino
1012
1013 #include <Arduino.h>
1014 #include <U8g2lib.h>
1015 #include <SPI.h>
1016 #include <SD.h>
1017
1018 // Include SPI and Wire libraries conditionally if hardware SPI/I2C is
1019 // available
1020 #ifdef U8X8_HAVE_HW_SPI
1021 #include <SPI.h>
1022 #endif
1023 #ifdef U8X8_HAVE_HW_I2C
1024 #include <Wire.h>
1025 #endif
1026
1027 // Initialize display object for a 128x64 OLED with I2C interface
1028 U8G2_SH1106_128X64_NONAME_F_HW_I2C u8g2(U8G2_R0, /* reset=*/
1029     U8X8_PIN_NONE, /* clock=*/ A5, /* data=*/ A4);
1030
1031 // Define sensor and SD card pins
1032 const int heartRatePin = A0;

```

```
1030 const int batteryPin = A1;
1031 const int chipSelect = 10;
1032
1033 // Variables for heart rate calculation
1034 int heartRateValue = 0;
1035 int lastHeartRateValue = 0;
1036 unsigned long lastPeakTime = 0;
1037 unsigned long lastHeartbeatTime = 0;
1038 bool peakDetected = false;
1039 int threshold = 512;
1040 int heartRate = 0;
1041 int heartRateArray[10];
1042 int heartRateIndex = 0;
1043 int heartRateSum = 0;
1044 int previousHeartRate = 0;
1045 const unsigned long errorTimeout = 5000; // 5-second timeout for error
1046 detection
1047 bool errorState = false;
1048 double calibrationFactor = 1;
1049
1050 // Variables for display switching
1051 unsigned long lastSwitchTime = 0;
1052 const unsigned long switchInterval = 10000; // 10-second interval for
1053 display mode toggle
1054 bool showGraph = false;
1055 int graphData[108];
1056
1057 // Flags and timers for SD card and initial setup
1058 bool sdCardFull = false;
1059 unsigned long startTime;
1060 bool initialMeasurement = true;
1061
1062 void setup(void) {
1063     Serial.begin(9600);
1064     pinMode(heartRatePin, INPUT);
1065     pinMode(batteryPin, INPUT);
1066
1067     u8g2.begin(); // Initialize display
1068
1069     // SD card initialization
1070     Serial.print("Initializing SD card... ");
1071     if (!SD.begin(chipSelect)) {
1072         Serial.println("SD card initialization failed!");
1073         return;
1074     }
1075     Serial.println("SD card initialized.");
1076
1077     // Display initial message
1078     u8g2.clearBuffer();
1079     u8g2.setFont(u8g2_font_ncenB08_tr);
1080
1081     const char* startText1 = "Measurement starting, ";
1082     const char* startText2 = "please wait.";
1083
1084     int16_t x1 = (128 - u8g2.getStrWidth(startText1)) / 2;
1085     int16_t x2 = (128 - u8g2.getStrWidth(startText2)) / 2;
1086     int16_t y1 = 30;
1087     int16_t y2 = 45;
1088
1089     u8g2.drawStr(x1, y1, startText1);
1090     u8g2.drawStr(x2, y2, startText2);
1091
1092     u8g2.sendBuffer();
1093
1094     startTime = millis();
1095 }
1096
1097 void loop(void) {
1098     // Delay measurement start by 5 seconds
```

```

1098     if ( millis() - startTime < 5000) {
1099         return;
1100     }
1101
1102     // Stop execution if SD card is full
1103     if (sdCardFull) {
1104         return;
1105     }
1106
1107     heartRateValue = analogRead(heartRatePin);
1108     float rawBatteryValue = analogRead(batteryPin);
1109     float voltage = rawBatteryValue * (3.3 / 1023.0); // Calculate battery
1110     voltage
1111
1112     Serial.print("Measured voltage at A1: ");
1113     Serial.print(voltage);
1114     Serial.println(" V");
1115
1116     // Heartbeat detection based on threshold crossing
1117     if (heartRateValue > threshold && lastHeartRateValue <= threshold) {
1118         if (!peakDetected) {
1119             peakDetected = true;
1120             unsigned long peakInterval = millis() - lastPeakTime;
1121             lastPeakTime = millis();
1122             lastHeartbeatTime = millis();
1123             errorState = false;
1124             initialMeasurement = false;
1125
1126             if (peakInterval > 0) {
1127                 heartRate = 60000 / peakInterval; // Calculate heart rate in BPM
1128
1129                 // Update average heart rate
1130                 heartRateSum -= heartRateArray[heartRateIndex];
1131                 heartRateArray[heartRateIndex] = heartRate;
1132                 heartRateSum += heartRate;
1133                 heartRateIndex = (heartRateIndex + 1) % 10;
1134                 int averageHeartRate = (heartRateSum / 10) * calibrationFactor;
1135
1136                 Serial.print("Heart Rate: ");
1137                 Serial.println(averageHeartRate);
1138
1139                 // Update display if heart rate has changed
1140                 if (averageHeartRate != previousHeartRate) {
1141                     previousHeartRate = averageHeartRate;
1142
1143                     if (!showGraph) {
1144                         u8g2.clearBuffer();
1145                         u8g2.setFont(u8g2_font_ncenB08_tr);
1146                         u8g2.drawStr(0, 20, "Heart Rate:");
1147                         u8g2.setFont(u8g2_font_fub25_tr);
1148
1149                         char heartRateStr[10];
1150                         sprintf(heartRateStr, "%d", averageHeartRate);
1151                         int16_t x = (128 - u8g2.getStrWidth(heartRateStr) - u8g2.
1152                         getStrWidth(" BPM")) / 2;
1153                         u8g2.drawStr(x, 50, heartRateStr);
1154                         u8g2.drawStr(x + u8g2.getStrWidth(heartRateStr), 50, " BPM")
1155                     ;
1156
1157                     drawBattery(voltage); // Draw battery level
1158
1159                     u8g2.sendBuffer();
1160                 }
1161
1162                 updateGraph(averageHeartRate); // Update graph data
1163                 saveToSD(averageHeartRate); // Save data to SD card
1164             }
1165         }
1166     }
1167 }
```

```
1164     } else if (heartRateValue <= threshold) {
1165         peakDetected = false;
1166     }
1167
1168     // Check for error state if no heartbeat detected within timeout
1169     if (!initialMeasurement && millis() - lastHeartbeatTime > errorTimeout)
1170     ) {
1171         errorState = true;
1172         showGraph = false;
1173         u8g2.clearBuffer();
1174         u8g2.setFont(u8g2_font_ncenB08_tr);
1175         u8g2.drawStr(0, 20, "Heart Rate:");
1176         u8g2.setFont(u8g2_font_fub25_tr);
1177         u8g2.drawStr(30, 50, "Error");
1178
1179         drawBattery(voltage);
1180
1181         u8g2.sendBuffer();
1182     } else {
1183         errorState = false;
1184     }
1185
1186     // Toggle display between heart rate and graph every 10 seconds
1187     if (!errorState && millis() - lastSwitchTime > switchInterval) {
1188         showGraph = !showGraph;
1189         lastSwitchTime = millis();
1190     }
1191
1192     if (showGraph) {
1193         drawGraph(); // Draw heart rate graph
1194     }
1195
1196     lastHeartRateValue = heartRateValue;
1197     delay(10);
1198 }
1199
1200 // Display battery level as segmented bars
1201 void drawBattery(float voltage) {
1202     int batteryLevel = map(voltage * 10, 0, 90, 0, 4);
1203
1204     u8g2.drawFrame(100, 2, 24, 10);
1205     u8g2.drawBox(124, 4, 2, 6);
1206
1207     for (int i = 0; i < batteryLevel; i++) {
1208         u8g2.drawBox(102 + (i * 5), 4, 4, 6);
1209     }
1210 }
1211
1212 // Update graph data for real-time heart rate display
1213 void updateGraph(int heartRate) {
1214     for (int i = 0; i < 107; i++) {
1215         graphData[i] = graphData[i + 1];
1216     }
1217     graphData[107] = (heartRate > 40) ? heartRate : 0;
1218 }
1219
1220 // Draw heart rate graph on the display
1221 void drawGraph() {
1222     int xOffset = 10;
1223     int yOffset = 10;
1224
1225     u8g2.clearBuffer();
1226     u8g2.drawLine(xOffset, yOffset + 44, xOffset + 108, yOffset + 44);
1227     u8g2.drawLine(xOffset, yOffset, xOffset, yOffset + 44);
1228
1229     u8g2.setFont(u8g2_font_ncenB08_tr);
1230     u8g2.drawStr(xOffset + 60, yOffset + 54, "Time (s)");
1231     u8g2.drawStr(xOffset - 10, yOffset - 2, "BPM");
```

```

1232     for (int i = 1; i < 108; i++) {
1233         int y1 = yOffset + 44 - map(graphData[i - 1], 40, 140, 0, 44);
1234         int y2 = yOffset + 44 - map(graphData[i], 40, 140, 0, 44);
1235         if (graphData[i - 1] > 0 && graphData[i] > 0) {
1236             u8g2.drawLine(xOffset + i - 1, y1, xOffset + i, y2);
1237         }
1238     }
1239     u8g2.sendBuffer();
1240 }

1242 // Save heart rate data to SD card
1243 void saveToSD(int heartRate) {
1244     File dataFile = SD.open("datalog.txt", FILE_WRITE);

1246     if (dataFile) {
1247         dataFile.print("Heart Rate: ");
1248         dataFile.print(heartRate);
1249         dataFile.println(" BPM");
1250         dataFile.close();
1251         Serial.println("Data saved to SD card.");
1252     } else {
1253         Serial.println("Error opening file for writing. SD card may be full.");
1254     }
1255     sdCardFull = true;
1256     u8g2.clearBuffer();
1257     u8g2.setFont(u8g2_font_ncenB08_tr);
1258     u8g2.drawStr(0, 20, "SD Card Full!");
1259     u8g2.sendBuffer();
1260 }

```

..../Code/Arduino/HeartRate/HeartRateApp/HeartRateApp.ino

48.3.3. Using the Heart Rate Monitor

To take a heart rate measurement:

1. Attach the ear clip to your index finger or ear.
2. The OLED display will automatically begin showing real-time heart rate data.

48.3.4. Mobile Operation

The Heart Rate Monitor can be used portably. Switch it on to begin the heart rate monitoring.

48.4. Code Explanation

In the following code section the display is initialized using the U8g2 library for a 128x64 pixel OLED display with an I2C interface. The sensor inputs and the SD card reader are connected to the following pins: `heartRatePin` (A0), `batteryPin` (A1), and `chipSelect` (Pin 10 for the SD card).

VS:check the pins! refer to?

For heart rate calculation, variables are used to track heart rate values, timing for each heartbeat, and a threshold for peak detection. The code includes an error detection routine that is triggered if no pulse is detected within a five-second interval (`errorTimeout`).

The OLED display alternates every ten seconds (`switchInterval`) between displaying the current heart rate and a graphical representation of recent data. Graph data is stored in an array (`graphData`) and continuously updated.

Additionally, flags are used to manage the SD card status and initial setup states to ensure smooth operation of the program. This includes both the display of sensor data and its storage on the SD card.

Listing 48.2.: Application Heart Rate Sensor - Initialization

```

1000 // Code for using the GRV Heart Rate 3 Sensor as a heart rate monitor.
1001 // To calibrate, measure the actual pulse and divide it by the pulse
1002 // measured by the GRV Heart Rate 3 Sensor.
1003 // Enter this factor as calibrationFactor (line 48).

1004 // Required components:
1005 // - Arduino Nano 33 BLE Sense
1006 // - GRV Heart Rate 3 Sensor
1007 // - 1.30" IIC OLED Display
1008 // - micro SD card reader with an SD card

1009 // file: ApplicationHeartRate.ino

1012 #include <Arduino.h>
1013 #include <U8g2lib.h>
1014 #include <SPI.h>
1015 #include <SD.h>

1016 // Include SPI and Wire libraries conditionally if hardware SPI/I2C is
1017 // available
1018 #ifdef U8X8_HAVE_HW_SPI
1019 #include <SPI.h>
1020#endif
1021 #ifdef U8X8_HAVE_HW_I2C
1022 #include <Wire.h>
1023#endif

1024 // Initialize display object for a 128x64 OLED with I2C interface
1025 U8G2_SH1106_128X64_NONAME_F_HW_I2C u8g2(U8G2_R0, /* reset==*/
1026     U8X8_PIN_NONE, /* clock==*/ A5, /* data==*/ A4);

1028 // Define sensor and SD card pins
1029 const int heartRatePin = A0;
1030 const int batteryPin = A1;
1031 const int chipSelect = 10;

1032 // Variables for heart rate calculation
1033 int heartRateValue = 0;
1034 int lastHeartRateValue = 0;
1035 unsigned long lastPeakTime = 0;
1036 unsigned long lastHeartbeatTime = 0;
1037 bool peakDetected = false;
1038 int threshold = 512;
1039 int heartRate = 0;
1040 int heartRateArray[10];
1041 int heartRateIndex = 0;
1042 int heartRateSum = 0;
1043 int previousHeartRate = 0;
1044 const unsigned long errorTimeout = 5000; // 5-second timeout for error
1045 // detection
1046 bool errorState = false;
1047 double calibrationFactor = 1;

1048 // Variables for display switching
1049 unsigned long lastSwitchTime = 0;
1050 const unsigned long switchInterval = 10000; // 10-second interval for
1051 // display mode toggle
1052 bool showGraph = false;
1053 int graphData[108];

1054 // Flags and timers for SD card and initial setup
1055 bool sdCardFull = false;

```

```

1058     unsigned long startTime;
      bool initialMeasurement = true;

      ..../Code/Arduino/HeartRate/HeartRateApp/HeartRateApp.ino

```

In the next code section, the initial setup for the heart rate monitor is conducted. The function `setup()` configures the serial communication, input pins, display, and SD card. Specifically, `Serial.begin(9600)` initializes the serial monitor, while `pinMode()` sets up the heart rate sensor and battery pins as input.

The OLED display is initialized through the command `u8g2.begin()`, preparing it for data visualization. Subsequently, the SD card is initialized, with a message confirming successful initialization or failure displayed in the serial monitor. If the SD card setup is unsuccessful, the program outputs an error message and stops further SD card-related processes.

WS:sd-card is an own topic

An initial message, “Measurement starting, please wait”, is then displayed on the OLED. This message provides user feedback, centered on the display with coordinates calculated to ensure proper alignment. Finally, `startTime = millis()` captures the initial time, marking the beginning of the measurement period.

Listing 48.3.: Application Heart Rate Sensor - Setup

```

1000 void setup(void) {
1001   Serial.begin(9600);
1002   pinMode(heartRatePin, INPUT);
1003   pinMode(batteryPin, INPUT);

1004   u8g2.begin(); // Initialize display
1005
1006   // SD card initialization
1007   Serial.print("Initializing SD card... ");
1008   if (!SD.begin(chipSelect)) {
1009     Serial.println("SD card initialization failed!");
1010     return;
1011   }
1012   Serial.println("SD card initialized.");
1013
1014   // Display initial message
1015   u8g2.clearBuffer();
1016   u8g2.setFont(u8g2_font_ncenB08_tr);
1017
1018   const char* startText1 = "Measurement starting ,";
1019   const char* startText2 = "please wait .";
1020
1021   int16_t x1 = (128 - u8g2.getStrWidth(startText1)) / 2;
1022   int16_t x2 = (128 - u8g2.getStrWidth(startText2)) / 2;
1023   int16_t y1 = 30;
1024   int16_t y2 = 45;

1025   u8g2.drawStr(x1, y1, startText1);
1026   u8g2.drawStr(x2, y2, startText2);
1027
1028   u8g2.sendBuffer();
1029
1030   startTime = millis();
1031 }

1032 }
```

```
..../Code/Arduino/HeartRate/HeartRateApp/HeartRateApp.ino
```

In the next code section, the main measurement and display loop for the heart rate monitor is implemented. Initially, a 5-second delay is applied to ensure that measurements do not start immediately. If the SD card is full, the loop halts further execution, preventing additional data logging.

The loop reads heart rate and battery values through the analog inputs. The battery voltage is calculated from the raw sensor data and displayed in the serial monitor.

The heartbeat detection is based on threshold crossing. A peak in the heart rate signal indicates a heartbeat if it exceeds the threshold value, marking the time interval between beats.

When a peak is detected, the heart rate is calculated in beats per minute (BPM) by dividing 60,000 milliseconds by the time interval since the last peak. An average heart rate is then computed from the last ten readings, adjusted by the `calibrationFactor`, and displayed if it differs from the previous reading.

The OLED display updates based on the current heart rate or displays a graph of recent heart rate data. The battery level is shown alongside the heart rate. The graph data is updated using `updateGraph()` and saved to the SD card via `saveToSD()`. If no heartbeat is detected within a specified timeout period (`errorTimeout`), an error message appears on the display, and the graph is suppressed. The display toggles every 10 seconds between the heart rate and graph view, providing comprehensive feedback on the user's heart rate status.

Listing 48.4.: Application Heart Rate Sensor - Loop

```

1000 void loop(void) {
1001     // Delay measurement start by 5 seconds
1002     if (millis() - startTime < 5000) {
1003         return;
1004     }
1005
1006     // Stop execution if SD card is full
1007     if (sdCardFull) {
1008         return;
1009     }
1010
1011     heartRateValue = analogRead(heartRatePin);
1012     float rawBatteryValue = analogRead(batteryPin);
1013     float voltage = rawBatteryValue * (3.3 / 1023.0); // Calculate battery
1014     voltage
1015
1016     Serial.print("Measured voltage at A1: ");
1017     Serial.print(voltage);
1018     Serial.println(" V");
1019
1020     // Heartbeat detection based on threshold crossing
1021     if (heartRateValue > threshold && lastHeartRateValue <= threshold) {
1022         if (!peakDetected) {
1023             peakDetected = true;
1024             unsigned long peakInterval = millis() - lastPeakTime;
1025             lastPeakTime = millis();
1026             lastHeartbeatTime = millis();
1027             errorState = false;
1028             initialMeasurement = false;
1029
1030             if (peakInterval > 0) {
1031                 heartRate = 60000 / peakInterval; // Calculate heart rate in BPM
1032
1033                 // Update average heart rate
1034                 heartRateSum -= heartRateArray[heartRateIndex];
1035                 heartRateArray[heartRateIndex] = heartRate;
1036                 heartRateSum += heartRate;
1037                 heartRateIndex = (heartRateIndex + 1) % 10;
1038                 int averageHeartRate = (heartRateSum / 10) * calibrationFactor;
1039
1040                 Serial.print("Heart Rate: ");
1041                 Serial.println(averageHeartRate);
1042
1043                 // Update display if heart rate has changed
1044                 if (averageHeartRate != previousHeartRate) {
1045                     previousHeartRate = averageHeartRate;
1046
1047                     if (!showGraph) {
1048                         u8g2.clearBuffer();
1049
1050                     }
1051
1052                     if (showGraph) {
1053                         updateGraph();
1054
1055                     }
1056
1057                     if (errorState) {
1058                         u8g2.drawString(10, 10, "No Heartbeat");
1059
1060                     }
1061
1062                     if (showGraph) {
1063                         saveToSD();
1064
1065                     }
1066
1067                     if (errorState) {
1068                         u8g2.drawString(10, 10, "Error");
1069
1070                     }
1071
1072                     if (showGraph) {
1073                         updateGraph();
1074
1075                     }
1076
1077                     if (errorState) {
1078                         u8g2.drawString(10, 10, "No Heartbeat");
1079
1080                     }
1081
1082                     if (showGraph) {
1083                         saveToSD();
1084
1085                     }
1086
1087                     if (errorState) {
1088                         u8g2.drawString(10, 10, "Error");
1089
1090                     }
1091
1092                     if (showGraph) {
1093                         updateGraph();
1094
1095                     }
1096
1097                     if (errorState) {
1098                         u8g2.drawString(10, 10, "No Heartbeat");
1099
1100                     }
1101
1102                     if (showGraph) {
1103                         saveToSD();
1104
1105                     }
1106
1107                     if (errorState) {
1108                         u8g2.drawString(10, 10, "Error");
1109
1110                     }
1111
1112                     if (showGraph) {
1113                         updateGraph();
1114
1115                     }
1116
1117                     if (errorState) {
1118                         u8g2.drawString(10, 10, "No Heartbeat");
1119
1120                     }
1121
1122                     if (showGraph) {
1123                         saveToSD();
1124
1125                     }
1126
1127                     if (errorState) {
1128                         u8g2.drawString(10, 10, "Error");
1129
1130                     }
1131
1132                     if (showGraph) {
1133                         updateGraph();
1134
1135                     }
1136
1137                     if (errorState) {
1138                         u8g2.drawString(10, 10, "No Heartbeat");
1139
1140                     }
1141
1142                     if (showGraph) {
1143                         saveToSD();
1144
1145                     }
1146
1147                     if (errorState) {
1148                         u8g2.drawString(10, 10, "Error");
1149
1150                     }
1151
1152                     if (showGraph) {
1153                         updateGraph();
1154
1155                     }
1156
1157                     if (errorState) {
1158                         u8g2.drawString(10, 10, "No Heartbeat");
1159
1160                     }
1161
1162                     if (showGraph) {
1163                         saveToSD();
1164
1165                     }
1166
1167                     if (errorState) {
1168                         u8g2.drawString(10, 10, "Error");
1169
1170                     }
1171
1172                     if (showGraph) {
1173                         updateGraph();
1174
1175                     }
1176
1177                     if (errorState) {
1178                         u8g2.drawString(10, 10, "No Heartbeat");
1179
1180                     }
1181
1182                     if (showGraph) {
1183                         saveToSD();
1184
1185                     }
1186
1187                     if (errorState) {
1188                         u8g2.drawString(10, 10, "Error");
1189
1190                     }
1191
1192                     if (showGraph) {
1193                         updateGraph();
1194
1195                     }
1196
1197                     if (errorState) {
1198                         u8g2.drawString(10, 10, "No Heartbeat");
1199
1200                     }
1201
1202                     if (showGraph) {
1203                         saveToSD();
1204
1205                     }
1206
1207                     if (errorState) {
1208                         u8g2.drawString(10, 10, "Error");
1209
1210                     }
1211
1212                     if (showGraph) {
1213                         updateGraph();
1214
1215                     }
1216
1217                     if (errorState) {
1218                         u8g2.drawString(10, 10, "No Heartbeat");
1219
1220                     }
1221
1222                     if (showGraph) {
1223                         saveToSD();
1224
1225                     }
1226
1227                     if (errorState) {
1228                         u8g2.drawString(10, 10, "Error");
1229
1230                     }
1231
1232                     if (showGraph) {
1233                         updateGraph();
1234
1235                     }
1236
1237                     if (errorState) {
1238                         u8g2.drawString(10, 10, "No Heartbeat");
1239
1240                     }
1241
1242                     if (showGraph) {
1243                         saveToSD();
1244
1245                     }
1246
1247                     if (errorState) {
1248                         u8g2.drawString(10, 10, "Error");
1249
1250                     }
1251
1252                     if (showGraph) {
1253                         updateGraph();
1254
1255                     }
1256
1257                     if (errorState) {
1258                         u8g2.drawString(10, 10, "No Heartbeat");
1259
1260                     }
1261
1262                     if (showGraph) {
1263                         saveToSD();
1264
1265                     }
1266
1267                     if (errorState) {
1268                         u8g2.drawString(10, 10, "Error");
1269
1270                     }
1271
1272                     if (showGraph) {
1273                         updateGraph();
1274
1275                     }
1276
1277                     if (errorState) {
1278                         u8g2.drawString(10, 10, "No Heartbeat");
1279
1280                     }
1281
1282                     if (showGraph) {
1283                         saveToSD();
1284
1285                     }
1286
1287                     if (errorState) {
1288                         u8g2.drawString(10, 10, "Error");
1289
1290                     }
1291
1292                     if (showGraph) {
1293                         updateGraph();
1294
1295                     }
1296
1297                     if (errorState) {
1298                         u8g2.drawString(10, 10, "No Heartbeat");
1299
1300                     }
1301
1302                     if (showGraph) {
1303                         saveToSD();
1304
1305                     }
1306
1307                     if (errorState) {
1308                         u8g2.drawString(10, 10, "Error");
1309
1310                     }
1311
1312                     if (showGraph) {
1313                         updateGraph();
1314
1315                     }
1316
1317                     if (errorState) {
1318                         u8g2.drawString(10, 10, "No Heartbeat");
1319
1320                     }
1321
1322                     if (showGraph) {
1323                         saveToSD();
1324
1325                     }
1326
1327                     if (errorState) {
1328                         u8g2.drawString(10, 10, "Error");
1329
1330                     }
1331
1332                     if (showGraph) {
1333                         updateGraph();
1334
1335                     }
1336
1337                     if (errorState) {
1338                         u8g2.drawString(10, 10, "No Heartbeat");
1339
1340                     }
1341
1342                     if (showGraph) {
1343                         saveToSD();
1344
1345                     }
1346
1347                     if (errorState) {
1348                         u8g2.drawString(10, 10, "Error");
1349
1350                     }
1351
1352                     if (showGraph) {
1353                         updateGraph();
1354
1355                     }
1356
1357                     if (errorState) {
1358                         u8g2.drawString(10, 10, "No Heartbeat");
1359
1360                     }
1361
1362                     if (showGraph) {
1363                         saveToSD();
1364
1365                     }
1366
1367                     if (errorState) {
1368                         u8g2.drawString(10, 10, "Error");
1369
1370                     }
1371
1372                     if (showGraph) {
1373                         updateGraph();
1374
1375                     }
1376
1377                     if (errorState) {
1378                         u8g2.drawString(10, 10, "No Heartbeat");
1379
1380                     }
1381
1382                     if (showGraph) {
1383                         saveToSD();
1384
1385                     }
1386
1387                     if (errorState) {
1388                         u8g2.drawString(10, 10, "Error");
1389
1390                     }
1391
1392                     if (showGraph) {
1393                         updateGraph();
1394
1395                     }
1396
1397                     if (errorState) {
1398                         u8g2.drawString(10, 10, "No Heartbeat");
1399
1400                     }
1401
1402                     if (showGraph) {
1403                         saveToSD();
1404
1405                     }
1406
1407                     if (errorState) {
1408                         u8g2.drawString(10, 10, "Error");
1409
1410                     }
1411
1412                     if (showGraph) {
1413                         updateGraph();
1414
1415                     }
1416
1417                     if (errorState) {
1418                         u8g2.drawString(10, 10, "No Heartbeat");
1419
1420                     }
1421
1422                     if (showGraph) {
1423                         saveToSD();
1424
1425                     }
1426
1427                     if (errorState) {
1428                         u8g2.drawString(10, 10, "Error");
1429
1430                     }
1431
1432                     if (showGraph) {
1433                         updateGraph();
1434
1435                     }
1436
1437                     if (errorState) {
1438                         u8g2.drawString(10, 10, "No Heartbeat");
1439
1440                     }
1441
1442                     if (showGraph) {
1443                         saveToSD();
1444
1445                     }
1446
1447                     if (errorState) {
1448                         u8g2.drawString(10, 10, "Error");
1449
1450                     }
1451
1452                     if (showGraph) {
1453                         updateGraph();
1454
1455                     }
1456
1457                     if (errorState) {
1458                         u8g2.drawString(10, 10, "No Heartbeat");
1459
1460                     }
1461
1462                     if (showGraph) {
1463                         saveToSD();
1464
1465                     }
1466
1467                     if (errorState) {
1468                         u8g2.drawString(10, 10, "Error");
1469
1470                     }
1471
1472                     if (showGraph) {
1473                         updateGraph();
1474
1475                     }
1476
1477                     if (errorState) {
1478                         u8g2.drawString(10, 10, "No Heartbeat");
1479
1480                     }
1481
1482                     if (showGraph) {
1483                         saveToSD();
1484
1485                     }
1486
1487                     if (errorState) {
1488                         u8g2.drawString(10, 10, "Error");
1489
1490                     }
1491
1492                     if (showGraph) {
1493                         updateGraph();
1494
1495                     }
1496
1497                     if (errorState) {
1498                         u8g2.drawString(10, 10, "No Heartbeat");
1499
1500                     }
1501
1502                     if (showGraph) {
1503                         saveToSD();
1504
1505                     }
1506
1507                     if (errorState) {
1508                         u8g2.drawString(10, 10, "Error");
1509
1510                     }
1511
1512                     if (showGraph) {
1513                         updateGraph();
1514
1515                     }
1516
1517                     if (errorState) {
1518                         u8g2.drawString(10, 10, "No Heartbeat");
1519
1520                     }
1521
1522                     if (showGraph) {
1523                         saveToSD();
1524
1525                     }
1526
1527                     if (errorState) {
1528                         u8g2.drawString(10, 10, "Error");
1529
1530                     }
1531
1532                     if (showGraph) {
1533                         updateGraph();
1534
1535                     }
1536
1537                     if (errorState) {
1538                         u8g2.drawString(10, 10, "No Heartbeat");
1539
1540                     }
1541
1542                     if (showGraph) {
1543                         saveToSD();
1544
1545                     }
1546
1547                     if (errorState) {
1548                         u8g2.drawString(10, 10, "Error");
1549
1550                     }
1551
1552                     if (showGraph) {
1553                         updateGraph();
1554
1555                     }
1556
1557                     if (errorState) {
1558                         u8g2.drawString(10, 10, "No Heartbeat");
1559
1560                     }
1561
1562                     if (showGraph) {
1563                         saveToSD();
1564
1565                     }
1566
1567                     if (errorState) {
1568                         u8g2.drawString(10, 10, "Error");
1569
1570                     }
1571
1572                     if (showGraph) {
1573                         updateGraph();
1574
1575                     }
1576
1577                     if (errorState) {
1578                         u8g2.drawString(10, 10, "No Heartbeat");
1579
1580                     }
1581
1582                     if (showGraph) {
1583                         saveToSD();
1584
1585                     }
1586
1587                     if (errorState) {
1588                         u8g2.drawString(10, 10, "Error");
1589
1590                     }
1591
1592                     if (showGraph) {
1593                         updateGraph();
1594
1595                     }
1596
1597                     if (errorState) {
1598                         u8g2.drawString(10, 10, "No Heartbeat");
1599
1600                     }
1601
1602                     if (showGraph) {
1603                         saveToSD();
1604
1605                     }
1606
1607                     if (errorState) {
1608                         u8g2.drawString(10, 10, "Error");
1609
1610                     }
1611
1612                     if (showGraph) {
1613                         updateGraph();
1614
1615                     }
1616
1617                     if (errorState) {
1618                         u8g2.drawString(10, 10, "No Heartbeat");
1619
1620                     }
1621
1622                     if (showGraph) {
1623                         saveToSD();
1624
1625                     }
1626
1627                     if (errorState) {
1628                         u8g2.drawString(10, 10, "Error");
1629
1630                     }
1631
1632                     if (showGraph) {
1633                         updateGraph();
1634
1635                     }
1636
1637                     if (errorState) {
1638                         u8g2.drawString(10, 10, "No Heartbeat");
1639
1640                     }
1641
1642                     if (showGraph) {
1643                         saveToSD();
1644
1645                     }
1646
1647                     if (errorState) {
1648                         u8g2.drawString(10, 10, "Error");
1649
1650                     }
1651
1652                     if (showGraph) {
1653                         updateGraph();
1654
1655                     }
1656
1657                     if (errorState) {
1658                         u8g2.drawString(10, 10, "No Heartbeat");
1659
1660                     }
1661
1662                     if (showGraph) {
1663                         saveToSD();
1664
1665                     }
1666
1667                     if (errorState) {
1668                         u8g2.drawString(10, 10, "Error");
1669
1670                     }
1671
1672                     if (showGraph) {
1673                         updateGraph();
1674
1675                     }
1676
1677                     if (errorState) {
1678                         u8g2.drawString(10, 10, "No Heartbeat");
1679
1680                     }
1681
1682                     if (showGraph) {
1683                         saveToSD();
1684
1685                     }
1686
1687                     if (errorState) {
1688                         u8g2.drawString(10, 10, "Error");
1689
1690                     }
1691
1692                     if (showGraph) {
1693                         updateGraph();
1694
1695                     }
1696
1697                     if (errorState) {
1698                         u8g2.drawString(10, 10, "No Heartbeat");
1699
1700                     }
1701
1702                     if (showGraph) {
1703                         saveToSD();
1704
1705                     }
1706
1707                     if (errorState) {
1708                         u8g2.drawString(10, 10, "Error");
1709
1710                     }
1711
1712                     if (showGraph) {
1713                         updateGraph();
1714
1715                     }
1716
1717                     if (errorState) {
1718                         u8g2.drawString(10, 10, "No Heartbeat");
1719
1720                     }
1721
1722                     if (showGraph) {
1723                         saveToSD();
1724
1725                     }
1726
1727                     if (errorState) {
1728                         u8g2.drawString(10, 10, "Error");
1729
1730                     }
1731
1732                     if (showGraph) {
1733                         updateGraph();
1734
1735                     }
1736
1737                     if (errorState) {
1738                         u8g2.drawString(10, 10, "No Heartbeat");
1739
1740                     }
1741
1742                     if (showGraph) {
1743                         saveToSD();
1744
1745                     }
1746
1747                     if (errorState) {
1748                         u8g2.drawString(10, 10, "Error");
1749
1750                     }
1751
1752                     if (showGraph) {
1753                         updateGraph();
1754
1755                     }
1756
1757                     if (errorState) {
1758                         u8g2.drawString(10, 10, "No Heartbeat");
1759
1760                     }
1761
1762                     if (showGraph) {
1763                         saveToSD();
1764
1765                     }
1766
1767                     if (errorState) {
1768                         u8g2.drawString(10, 10, "Error");
1769
1770                     }
1771
1772                     if (showGraph) {
1773                         updateGraph();
1774
1775                     }
1776
1777                     if (errorState) {
1778                         u8g2.drawString(10, 10, "No Heartbeat");
1779
1780                     }
1781
1782                     if (showGraph) {
1783                         saveToSD();
1784
1785                     }
1786
1787                     if (errorState) {
1788                         u8g2.drawString(10, 10, "Error");
1789
1790                     }
1791
1792                     if (showGraph) {
1793                         updateGraph();
1794
1795                     }
1796
1797                     if (errorState) {
1798                         u8g2.drawString(10, 10, "No Heartbeat");
1799
1800                     }
1801
1802                     if (showGraph) {
1803                         saveToSD();
1804
1805                     }
1806
1807                     if (errorState) {
1808                         u8g2.drawString(10, 10, "Error");
1809
1810                     }
1811
1812                     if (showGraph) {
1813                         updateGraph();
1814
1815                     }
1816
1817                     if (errorState) {
1818                         u8g2.drawString(10, 10, "No Heartbeat");
1819
1820                     }
1821
1822                     if (showGraph) {
1823                         saveToSD();
1824
1825                     }
1826
1827                     if (errorState) {
1828                         u8g2.drawString(10, 10, "Error");
1829
1830                     }
1831
1832                     if (showGraph) {
1833                         updateGraph();
1834
1835                     }
1836
1837                     if (errorState) {
1838                         u8g2.drawString(10, 10, "No Heartbeat");
1839
1840                     }
1841
1842                     if (showGraph) {
1843                         saveToSD();
1844
1845                     }
1846
1847                     if (errorState) {
1848                         u8g2.drawString(10, 10, "Error");
1849
1850                     }
1851
1852                     if (showGraph) {
1853                         updateGraph();
1854
1855                     }
1856
1857                     if (errorState) {
1858                         u8g2.drawString(10, 10, "No Heartbeat");
1859
1860                     }
1861
1862                     if (showGraph) {
1863                         saveToSD();
1864
1865                     }
1866
1867                     if (errorState) {
1868                         u8g2.drawString(10, 10, "Error");
1869
1870                     }
1871
1872                     if (showGraph) {
1873                         updateGraph();
1874
1875                     }
1876
1877                     if (errorState) {
1878                         u8g2.drawString(10, 10, "No Heartbeat");
1879
1880                     }
1881
1882                     if (showGraph) {
1883                         saveToSD();
1884
1885                     }
1886
1887                     if (errorState) {
1888                         u8g2.drawString(10, 10, "Error");
1889
1890                     }
1891
1892                     if (showGraph) {
1893                         updateGraph();
1894
1895                     }
1896
1897                     if (errorState) {
1898                         u8g2.drawString(10, 10, "No Heartbeat");
1899
1900                     }
1901
1902                     if (showGraph) {
1903                         saveToSD();
1904
1905                     }
1906
1907                     if (errorState) {
1908                         u8g2.drawString(10, 10, "Error");
1909
1910                     }
1911
1912                     if (showGraph) {
1913                         updateGraph();
1914
1915                     }
1916
1917                     if (errorState) {
1918                         u8g2.drawString(10, 10, "No Heartbeat");
1919
1920                     }
1921
1922                     if (showGraph) {
1923                         saveToSD();
1924
1925                     }
1926
1927                     if (errorState) {
1928                         u8g2.drawString(10, 10, "Error");
1929
1930                     }
1931
1932                     if (showGraph) {
1933                         updateGraph();
1934
1935                     }
1936
1937                     if (errorState) {
1938                         u8g2.drawString(10, 10, "No Heartbeat");
1939
1940                     }
1941
1942                     if (showGraph) {
1943                         saveToSD();
1944
1945                     }
1946
1947                     if (errorState) {
1948                         u8g2.drawString(10, 10, "Error");
1949
1950                     }
1951
1952                     if (showGraph) {
1953                         updateGraph();
1954
1955                     }
1956
1957                     if (errorState) {
1958                         u8g2.drawString(10, 10, "No Heartbeat");
1959
1960                     }
1961
1962                     if (showGraph) {
1963                         saveToSD();
1964
1965                     }
1966
1967                     if (errorState) {
1968                         u8g2.drawString(10, 10, "Error");
1969
1970                     }
1971
1972                     if (showGraph) {
1973                         updateGraph();
1974
1975                     }
1976
1977                     if (errorState) {
1978                         u8g2.drawString(10, 10, "No Heartbeat");
1979
1980                     }
1981
1982                     if (showGraph) {
1983                         saveToSD();
1984
1985                     }
1986
1987                     if (errorState) {
1988                         u8g2.drawString(10, 10, "Error");
1989
1990                     }
1991
1992                     if (showGraph) {
1993                         updateGraph();
1994
1995                     }
1996
1997                     if (errorState) {
1998                         u8g2.drawString(10, 10, "No Heartbeat");
1999
2000                     }
2001
2002                     if (showGraph) {
2003                         saveToSD();
2004
2005                     }
2006
2007                     if (errorState) {
2008                         u8g2.drawString(10, 10, "Error");
2009
2010                     }
2011
2012                     if (showGraph) {
2013                         updateGraph();
2014
2015                     }
2016
2017                     if (errorState) {
2018                         u8g2.drawString(10, 10, "No Heartbeat");
2019
2020                     }
2021
2022                     if (showGraph) {
2023                         saveToSD();
2024
2025                     }
2026
2027                     if (errorState) {
2028                         u8g2.drawString(10, 10, "Error");
2029
2030                     }
2031
2032                     if (showGraph) {
2033                         updateGraph();
2034
2035                     }
2036
2037                     if (errorState) {
2038                         u8g2.drawString(10, 10, "No Heartbeat");
2039
2040                     }
2041
2042                     if (showGraph) {
2043                         saveToSD();
2044
2045                     }
2046
2047                     if (errorState) {
2048                         u8g2.drawString(10, 10, "Error");
2049
2050                     }
2051
2052                     if (showGraph) {
2053                         updateGraph();
2054
2055                     }
2056
2057                     if (errorState) {
2058                         u8g2.drawString(10, 10, "No Heartbeat");
2059
2060                     }
2061
2062                     if (showGraph) {
2063                         saveToSD();
2064
2065                     }
2066
2067                     if (errorState) {
2068                         u8g2.drawString(10, 10, "Error");
2069
2070                     }
2071
2072                     if (showGraph) {
2073                         updateGraph();
2074
2075                     }
2076
2077                     if (errorState) {
2078                         u8g2.drawString(10, 10, "No Heartbeat");
2079
2080                     }
2081
2082                     if (showGraph) {
2083                         saveToSD();
2084
2085                     }
2086
2087                     if (errorState) {
2088                         u8g2.drawString(10, 10, "Error");
2089
2090                     }
2091
2092                     if (showGraph) {
2093                         updateGraph();
2094
2095                     }
2096
2097                     if (errorState) {
2098                         u8g2.drawString(10, 10, "No Heartbeat");
2099
2100                     }
2101
2102                     if (showGraph) {
2103                         saveToSD();
2104
2105                     }
2106
2107                     if (errorState) {
2108                         u8g2.drawString(10, 10, "Error");
2109
2110                     }
2111
2112                     if (showGraph) {
2113                         updateGraph();
2114
2115                     }
2116
2117                     if (errorState) {
2118                         u8g2.drawString(10, 10, "No Heartbeat");
2119
2120                     }
2121
2122                     if (showGraph) {
2123                         saveToSD();
2124
2125                     }
2126
2127                     if (errorState) {
2128                         u8g2.drawString(10, 10, "Error");
2129
2130                     }
2131
2132                     if (showGraph) {
2133                         updateGraph();
2134
2135                     }
2136
2137                     if (errorState) {
2138                         u8g2.drawString(10, 10, "No Heartbeat");
2139
2140                     }
2141
2142                     if (showGraph) {
2143                         saveToSD();
2144
2145                     }
2146
2147                     if (errorState) {
2148                         u8g2.drawString(10, 10, "Error");
2149
2150                     }
2151
2152                     if (showGraph) {
2153                         updateGraph();
2154
2155                     }
2156
2157                     if (errorState) {
2158                         u8g2.drawString(10, 10, "No Heartbeat");
2159
2160                     }
2161
2162                     if (showGraph) {
2163                         saveToSD();
2164
2165                     }
2166
2167                     if (errorState) {
2168                         u8g2.drawString(10, 10, "Error");
2169
2170                     }
2171
2172                     if (showGraph) {
2173                         updateGraph();
2174
2175                     }
2176
2177                     if (errorState) {
2178                         u8g2.drawString(10, 10, "No Heartbeat");
2179
2180                     }
2181
2182                     if (showGraph) {
2183                         saveToSD();
2184
2185                     }
2186
2187                     if (errorState) {
2188                         u8g2.drawString(10, 10, "Error");
2189
2190                     }
2191
2192                     if (showGraph) {
2193                         updateGraph();
2194
2195                     }
2196
2197                     if (errorState) {
2198                         u8g2.drawString(10, 10, "No Heartbeat");
2199
2200                     }
2201
2202                     if (showGraph) {
2203                         saveToSD();
2204
2205                     }
2206
2207                     if (errorState) {
2208                         u8g2.drawString(10, 10, "Error");
2209
2210                     }
2211
2212                     if (showGraph) {
2213                         updateGraph();
2214
2215                     }
2216
2217                     if (errorState) {
2218                         u8g2.drawString(10, 10, "No Heartbeat");
2219
2220                     }
2221
2222                     if (showGraph) {
2223                         saveToSD();
2224
2225                     }
2226
2227                     if (errorState) {
2228                         u8g2.drawString(10, 10, "Error");
2229
2230                     }
2231
2232                     if (showGraph) {
2233                         updateGraph();
2234
2235                     }
2236
2237                     if (errorState) {
2238                         u8g2.drawString(10, 10, "No Heartbeat");
2239
2240                     }
2241
2242                     if (showGraph) {
2243                         saveToSD();
2244
2245                     }
2246
2247                     if
```

```

1048     u8g2.setFont(u8g2_font_ncenB08_tr);
1050     u8g2.drawStr(0, 20, "Heart Rate:");
1052     u8g2.setFont(u8g2_font_fub25_tr);

1054     char heartRateStr[10];
1055     sprintf(heartRateStr, "%d", averageHeartRate);
1056     int16_t x = (128 - u8g2.getStrWidth(heartRateStr) - u8g2.
1057     getStrWidth(" BPM")) / 2;
1058     u8g2.drawStr(x, 50, heartRateStr);
1059     u8g2.drawStr(x + u8g2.getStrWidth(heartRateStr), 50, " BPM")
1060     ;

1061     drawBattery(voltage); // Draw battery level
1062     u8g2.sendBuffer();
1063 }

1064     updateGraph(averageHeartRate); // Update graph data
1065     saveToSD(averageHeartRate); // Save data to SD card
1066   }
1067 }

1068 } else if (heartRateValue <= threshold) {
1069   peakDetected = false;
1070 }

// Check for error state if no heartbeat detected within timeout
1071 if (!initialMeasurement && millis() - lastHeartbeatTime > errorTimeout)
1072 {
1073   errorState = true;
1074   showGraph = false;
1075   u8g2.clearBuffer();
1076   u8g2.setFont(u8g2_font_ncenB08_tr);
1077   u8g2.drawStr(0, 20, "Heart Rate:");
1078   u8g2.setFont(u8g2_font_fub25_tr);
1079   u8g2.drawStr(30, 50, "Error");

1080   drawBattery(voltage);

1081   u8g2.sendBuffer();
1082 } else {
1083   errorState = false;
1084 }

1085 // Toggle display between heart rate and graph every 10 seconds
1086 if (!errorState && millis() - lastSwitchTime > switchInterval) {
1087   showGraph = !showGraph;
1088   lastSwitchTime = millis();
1089 }

1090 if (showGraph) {
1091   drawGraph(); // Draw heart rate graph
1092 }

1093 lastHeartRateValue = heartRateValue;
1094 delay(10);
1095 }
```

.../Code/Arduino/HeartRate/HeartRateApp/HeartRateApp.ino

In the next code section, additional functionalities are implemented to display battery level, update the heart rate graph, and save data to the SD card. The function `drawBattery()` visualizes the battery level on the OLED as segmented bars by mapping the voltage value to a range between 0 and 4 segments. The battery icon frame and bars are drawn at specified positions on the display.

The function `updateGraph()` is responsible for refreshing the heart rate data used in real-time graph display. It shifts all previous data points left by one position and

inserts the current heart rate at the end of the array `graphData`. This structure supports continuous real-time visualization of the heart rate trend.

The function `drawGraph()` renders the heart rate graph on the OLED display. It first sets up the graph axes, labeled “Time (s)” and “BPM”, and then draws a line connecting each consecutive heart rate value. The y-coordinates of each point are calculated based on a range from 40 to 140 BPM, allowing the display to scale appropriately with typical heart rate values.

Finally, the function `saveToSD()` writes the current heart rate to an SD card in a text file named `datalog.txt`. If the file opens successfully, the heart rate value is recorded, providing a timestamped record of heart rate data. Should the SD card be full or unavailable, an error message is displayed on the OLED, and a flag is set to prevent further write attempts.

Listing 48.5.: Application Heart Rate Sensor - Functions

```

1000 // Display battery level as segmented bars
1001 void drawBattery(float voltage) {
1002     int batteryLevel = map(voltage * 10, 0, 90, 0, 4);
1003
1004     u8g2.drawFrame(100, 2, 24, 10);
1005     u8g2.drawBox(124, 4, 2, 6);
1006
1007     for (int i = 0; i < batteryLevel; i++) {
1008         u8g2.drawBox(102 + (i * 5), 4, 4, 6);
1009     }
1010 }
1011
1012 // Update graph data for real-time heart rate display
1013 void updateGraph(int heartRate) {
1014     for (int i = 0; i < 107; i++) {
1015         graphData[i] = graphData[i + 1];
1016     }
1017     graphData[107] = (heartRate > 40) ? heartRate : 0;
1018 }
1019
1020 // Draw heart rate graph on the display
1021 void drawGraph() {
1022     int xOffset = 10;
1023     int yOffset = 10;
1024
1025     u8g2.clearBuffer();
1026     u8g2.drawLine(xOffset, yOffset + 44, xOffset + 108, yOffset + 44);
1027     u8g2.drawLine(xOffset, yOffset, xOffset, yOffset + 44);
1028
1029     u8g2.setFont(u8g2_font_ncenB08_tr);
1030     u8g2.drawStr(xOffset + 60, yOffset + 54, "Time (s)");
1031     u8g2.drawStr(xOffset - 10, yOffset - 2, "BPM");
1032
1033     for (int i = 1; i < 108; i++) {
1034         int y1 = yOffset + 44 - map(graphData[i - 1], 40, 140, 0, 44);
1035         int y2 = yOffset + 44 - map(graphData[i], 40, 140, 0, 44);
1036         if (graphData[i - 1] > 0 && graphData[i] > 0) {
1037             u8g2.drawLine(xOffset + i - 1, y1, xOffset + i, y2);
1038         }
1039     }
1040
1041     u8g2.sendBuffer();
1042 }
1043
1044 // Save heart rate data to SD card
1045 void saveToSD(int heartRate) {
1046     File dataFile = SD.open("datalog.txt", FILE_WRITE);
1047
1048     if (dataFile) {
1049         dataFile.print("Heart Rate: ");
1050         dataFile.print(heartRate);

```

```

1052     dataFile.println(" BPM");
1053     dataFile.close();
1054 } else {
1055     Serial.println("Data saved to SD card.");
1056     sdCardFull = true;
1057     u8g2.clearBuffer();
1058     u8g2.setFont(u8g2_font_ncenB08_tr);
1059     u8g2.drawStr(0, 20, "SD Card Full!");
1060     u8g2.sendBuffer();
1061 }
1062 }
```

..../Code/Arduino/HeartRate/HeartRateApp/HeartRateApp.ino

48.5. Application Test

The Heart Rate Monitor measures the pulse with high accuracy, allowing the `calibrationFactor` to remain at 1. Additionally, the heart rate display on the OLED screen is very clear, making it intriguing to observe changes in pulse, such as those induced by rapid breathing. These pulse changes are effectively displayed by the graphical function.

48.6. Improvement Suggestions

Another enhancement could involve transmitting the heart rate data via Bluetooth to a smartphone. This would allow for easier data analysis and could facilitate tracking long-term trends with regular measurements, which would be medically beneficial.

- Data structure
- Flow chart
- modules: SD card, sensor, display, serial monitor
- Message handler
- error handler
- no magic numbers/text
- doxygen
- How to get the SD card?
- How can I access the SD card?
- List of Materials

49. Ultraschallsensor HC-SR04

Introduction

WS:cite books,
applications, board

49.1. Domänenwissen

Im folgenden Abschnitt wird der namensgebende Sensor für diese Arbeit näher betrachtet. Dazu zählen seine physikalischen Grundlagen, seine Funktionsweise und Anwendungsbeispiele aus Industrie und Medizin.

49.2. Grundlagen zur Verwendung eines Ultraschallsensors

49.2.1. Grundlagen Schall

Als Schall bezeichnet man die Ausbreitung von lokalen Druckschwankungen in einem elastischen Medium. Die Schallwelle breitet sich dabei longitudinal aus, die Auslenkung der Moleküle erfolgt also in Ausbreitungsrichtung [TM24]. Die Einteilung der Schallbereiche erfolgt auf Grundlage der Frequenz:

- *Infraschall*: bis 16 Hz,
- *Hörbarer Schall*: von 16 Hz bis 20 kHz,
- *Ultraschall*: von 20 kHz bis 1 GHz,
- *Hyperschall*: über 1 GHz [HS23].

WS:Grafik fehlt

49.2.2. Schallgeschwindigkeit und Wellenlänge

Die Geschwindigkeit, mit der sich Schall in Luft ausbreitet, ist von der Lufttemperatur T und der Luftfeuchtigkeit φ abhängig. Da die Luftfeuchtigkeit einen erheblich geringeren Einfluss auf den Wert hat

$$\Delta v_{Schall,\varphi} \approx 1,2 \frac{\text{m}}{\text{s}}; 0 < \varphi < 1, T_C = 20^\circ\text{C}, \quad (49.1)$$

[HS23] lässt sich die Schallgeschwindigkeit mit ausreichender Genauigkeit mit der Formel 49.2

$$v_{Schall} = \sqrt{\kappa RT} \quad (49.2)$$

und der Annahme $\varphi = 0$ berechnen [TM24]. Dabei ist $\kappa = 1,4$ der Adiabatenexponent von Luft, $R = 287 \frac{\text{J}}{\text{kg K}}$ die spezifische Gaskonstante von Luft und T die absolute Temperatur in Kelvin [TM24; Wil22]. Für eine angenommene Temperatur von $T_C = 20^\circ\text{C}$ ergibt sich damit eine Geschwindigkeit von $v_{Schall} = 343,2 \frac{\text{m}}{\text{s}}$. In Abb. 49.1 ist die Schallgeschwindigkeit in Luft im Temperaturbereich von $T_C = -10^\circ\text{C}$ bis 60°C dargestellt.

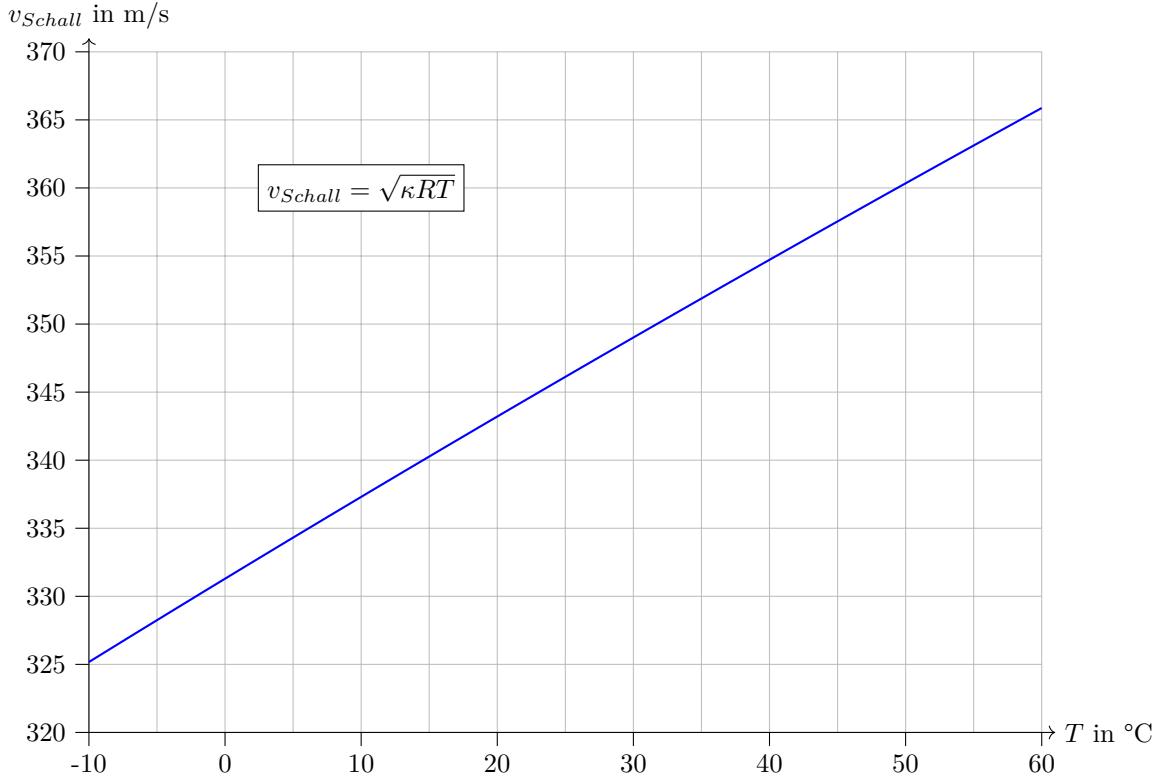


Figure 49.1.: Ausbreitungsgeschwindigkeit von Schall in Luft

Über den Formelzusammenhang 49.3

$$v_{Schall} = \lambda f \quad (49.3)$$

lässt sich bei gegebener Frequenz f und mit der berechneten Schallgeschwindigkeit v_{Schall} die Wellenlänge des Schalls berechnen [Wil22]. Diese beträgt bei einer angenommenen Frequenz von $f = 40\text{kHz}$ und der in Abschnitt 49.2.2 berechneten Schallgeschwindigkeit $\lambda = 8,58 * 10^{-3}\text{ m}$.

49.2.3. Absorption und Reflexion

Absorption beschreibt die Abnahme der Intensität I einer Schallwelle, während sie Weg durch das sie leitenden Medium zurücklegt. Sie ist vom Quadrat der Frequenz f des Schalls und den Absorptionseigenschaften a des Ausbreitungsmediums abhängig und lässt sich über die Formel 49.4

$$I(x) = I_0 \times \exp(-\alpha x); \alpha = af^2 \quad (49.4)$$

berechnen [HS23; HMS21]. In Abbildung 49.2 ist die Abhängigkeit des Verhältnisses der Intensität I zu I_0 von der Frequenz dargestellt. Die Wegstrecke von einem Meter und die Umgebungsbedingungen sind entsprechend konstant.

Für eine Frequenz von $f = 40\text{ kHz}$ ergibt sich ein Verhältnis von 0,95.

Trifft die Schallwelle auf die Grenzschicht zweier Medien mit verschiedenen Schallkennimpedanzen Z_n , wird sie zum Teil reflektiert und zum anderen Teil transmittiert. Der Reflexionsgrad ist bei großen Differenzen zwischen den Impedanzen ($Z_1 \ll Z_2$ oder $Z_1 \gg Z_2$) der Medien besonders hoch. Sind die Impedanzen hingegen ähnlich ($Z_1 \approx Z_2$) ist der Absorptionsgrad sehr hoch [HMS21]. Da viele Flüssigkeiten

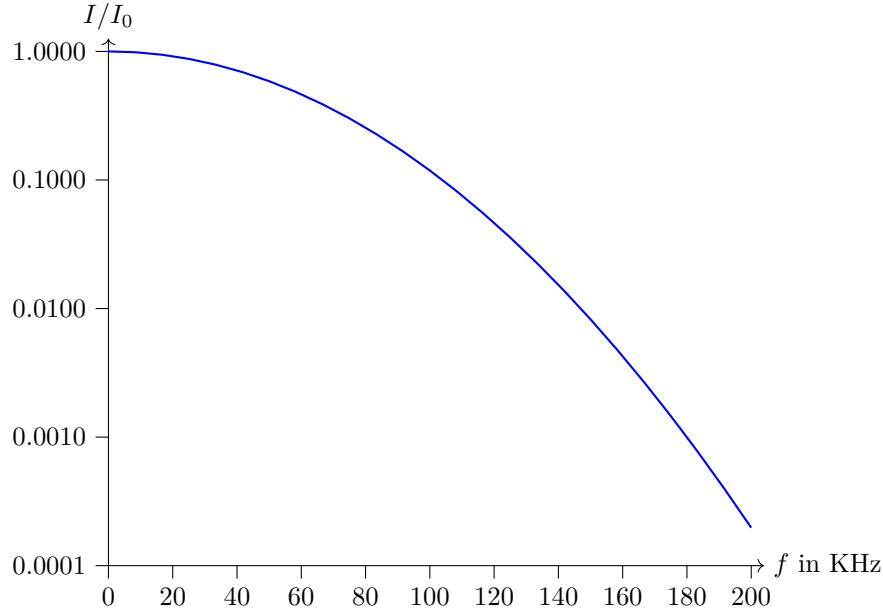


Figure 49.2.: Abnahme der Schallintensität in Abhängigkeit der Frequenz bei einem Meter, nach [HS23]

und feste Stoffe eine um den Faktor 10^4 bis 10^5 höhere Impedanz im Vergleich zu Luft vorweisen, ist der Reflexionsgrad in diesen Fällen $\gg 99\%$ [HMS21; HS23].

49.2.4. Weg- und Abstandsmessung mit Ultraschall

Um eine Entfernung oder einen Abstand eines Objekts zu dem Sensor zu ermitteln, wird der Sensor im sog. *Tastbetrieb mit Echo-Laufzeit-Messung*, wie in Abb. 49.3 zu sehen, verwendet. Dabei sendet der Sensor zunächst ein Schallpuls in Richtung des zu messenden Objekts aus (Lautsprecher), welche zum Teil von diesem reflektiert wird und somit zurück zum Sensor gelangt (Echo). Dieses Echo kann der Sensor detektieren (Mikrofon) und über die gemessene Laufzeit des Echos sowie die Schallgeschwindigkeit im dem den Sensor umgebenden Medium die Distanz berechnen [HS23].

In der Anwendung einköpfiger Sensoren ist für die Messung sehr kleiner Distanzen auf die Ausschwingzeit des Wandlers sowie die Umschaltzeit vom Lautsprecher- in den Mikrofonbetrieb zu achten, da in dieser Zeit kein Echosignal aufgenommen werden kann. Bei zweiköpfigen Systemen, also je einem dedizierten Lautsprecher und Mikrofon, sind die Öffnungswinkel der Schallkeulen in Verbindung mit dem radialen Abstand maßgeblich für die minimale Distanz.

Bei großen Distanzen sind sowohl die Genauigkeit der Ausrichtung des Sensors (bei kleinen Öffnungswinkeln der Schallkeulen) als auch die Absorption der Schallintensität in dem Ausbreitungsmedium relevant [HS23].

Ebenfalls störend auswirken können sich die Geometrie des zu messenden Objektes (z. B. bei stark konkaven oder konvexen Oberflächen), absorbierendes/diffus reflektierendes Material (Filz, Watte, Schaumstoff) oder Umwelteinflüsse orthogonal zur Messrichtung wirkender Wind. Verunreinigungen wie Staub und Rauch oder leichter Niederschlag in Form von Regen und Schnee beeinträchtigen die Messung hingegen nicht [HS23].

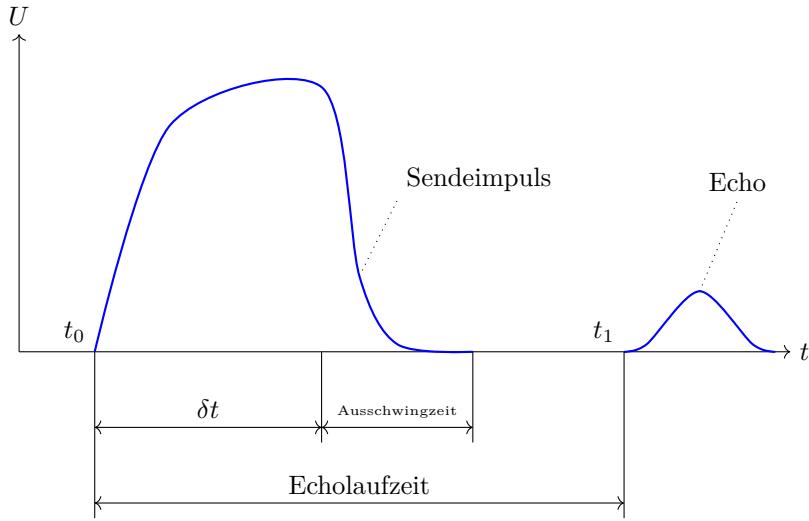


Figure 49.3.: Spannungsverlauf Schallwandler bei einer Echo-Laufzeit-Messung, nach [HS23]

49.2.5. Anwendungen

Ultraschallsensoren haben in der Praxis ein breites Einsatzgebiet [HS23]. Basierend auf dem Ausbreitungsmedium lassen sie sich in drei Kategorien einteilen.

Die erste Kategorie wird dem Luftschall zugeschrieben. Typische Anwendungsbeispiele sind Füllstandssensoren, Durchmessererfassung für Auf- und Abwickelsteuerung oder Kollisionsschutz/Objekterkennung [HS23].

Bei Sensoren der zweiten Kategorie breitet sich der Schall über Flüssigkeiten aus, wodurch diese vor allem im maritimen Bereich anzutreffen sind. Als Beispiele sind Sonarsysteme zum Orten von Unterseebooten oder Fischschwärmern und das Echolot zur Kartografie des Meeresbodens zu nennen [HMS21].

Die dritte Kategorie umfasst Körperschallsensoren. Viel verwendetes Beispiel in der Industrie sind Sensoren zur zerstörungsfreien Materialprüfung. In der Medizin ist die bildgebende Ultraschalldiagnostik ein wichtiges Werkzeug zur Überprüfung von Organen oder Fötten [HMS21].

49.2.6. Herausforderungen

Dadurch, dass die Schallgeschwindigkeit von der Umgebungstemperatur abhängig ist, muss diese für die Berechnung der Distanz mit einbezogen werden, wenn der Sensor in einem größeren Temperaturbereich eingesetzt werden soll. In Abbildung 49.1 ist zu erkennen, dass die Differenz der Geschwindigkeit in dem Temperaturbereich, in dem alle Komponenten des Sensors betrieben werden könnten ($-10^\circ\text{C} \leq T_C \leq 55^\circ\text{C}$), etwa $38 \frac{\text{m}}{\text{s}}$ beträgt. Dies entspricht, ausgehend von dem Standardwert für die Schallgeschwindigkeit $v_{\text{Schall},20} = 343,2 \frac{\text{m}}{\text{s}}$, einer Streuung von $-5,2\%$ und $+5,8\%$. Hinzu kommt die Unsicherheit aufgrund der Luftfeuchtigkeit. Hier liegt die Streuung, bezogen auf die Schallgeschwindigkeiten bei $\varphi = 0$, in einem Bereich zwischen $+0,37\%$ für $T_C = -10^\circ\text{C}$ und $+0,33\%$ für $T_C = 55^\circ\text{C}$.

Physikalische Effekte wie die Absorption, welche unter Laborbedingungen maßgeblich die Reichweite des Sensors einschränkt, oder einen schlechten Reflexionsgrad bestimmter Materialien, schränken das System in seiner Vielseitigkeit ein. Selbiges gilt für Attribute des Sensors wie die Ausschwingzeit des Schallwandlers oder die

geometrische Anordnung der Schallkeulen bei zweiköpfigen Systemen. Auch die genannten Umwelteinflüsse können dafür sorgen, dass eine Messung fehlschlägt oder ein falsche Ergebnis liefert.

49.2.7. Lösungsansätze

Um große mögliche Messfehler aufgrund einer fest hinterlegten Schallgeschwindigkeit zu vermeiden, soll die Temperatur von dem internen Temperatursensor des Arduino Nano 33 BLE Sense Lite bei einer Abstandsmessung erfasst und auf dessen Grundlage die Schallgeschwindigkeit berechnet werden. Da der Sensor jedoch auf der Platine ist, diese sich bei Benutzung selbst erwärmt und zusätzlich in einem Gehäuse verbaut ist, dessen Material gute wärmeisolierende Eigenschaften hat, ist die von ihm gemessene Temperatur zeitabhängig. Daraus folgt, dass seine Messwerte nicht für die Berechnung der Schallgeschwindigkeit verwendet werden können. Die Verwendung eines zusätzlichen Temperatursensors wäre eine mögliche Lösung. Daher wird, um eine zu große Abweichung des angezeigten Messwertes zu verhindern, der für den Sensor zugelassene Temperaturbereich eingeschränkt. Ein möglicher Temperaturbereich ist zwischen 17 °C und 23 °C. In diesem Bereich liegt die Abweichung der Schallgeschwindigkeit bei unter 0,5%. Ob diese Genauigkeit ausreichend ist, ist von der Anwendung abhängig.

49.3. Ultraschall Abstandssensor

Der Ultraschallabstandssensor in Abbildung 49.4 vom Hersteller JOY-it enthält den Sensor HC-SR04. Er ist als zweiköpfiger Sensor ausgeführt, wodurch er über je einen Lautsprecher und ein Mikrofon verfügt.

und ist zur Erfassung von Abständen in einem Bereich von 3 bis 400 cm spezifiziert. Dafür nutzt er eine Schallfrequenz von 40 kHz. Er ist als zweiköpfiger Sensor ausgeführt, wodurch er über je einen Lautsprecher und ein Mikrofon verfügt. Die Abmessungen des Sensors betragen in der Breite 43 mm, in der Höhe 15 mm und in der Tiefe 20 mm. Die Auflösung des Sensors beträgt ± 1 mm. Angeschlossen wird der Abstandssensor über den VCC-Pin, Trig/Serial Clock (SCL)-Pin, Echo/Serial Data (SDA)-Pin und dem GND-Pin. Der Sensor kann über die drei Schnittstellen GPIO, Universal Asynchronous Receiver Transmitter (UART) und Inter-Integrated Circuit (I^2C) verbunden werden [Simc].



Figure 49.4.: Ultraschallabstandssensor der Marke JOY-IT[Simc]

49.4. Specification

Der Sensor HC-SR04 ist zur Erfassung von Abständen in einem Bereich von 3 bis 400 cm mit einer Auflösung von ± 1 mm spezifiziert. Dafür nutzt er eine Schallfrequenz

von 40 kHz. Die Abmessungen des Sensors betragen in der Breite 43 mm, in der Höhe 15 mm und in der Tiefe 20 mm.

Angeschlossen wird der Abstandssensor über den SCL-Pin für das Triggersignal, den SDA-Pin fürs Echo, den VCC-Pin zur Stromversorgung und dem GND-Pin.

Der Sensor kann über die drei Schnittstellen GPIO, UART und I²C verbunden werden [Simc].

Bei den folgenden Beispielprogrammen wird der Arduino Nano 33 BLE Sense verwendet. In den Beispielen werden die Pins D6 und D9 für die Datenübertragung verwendet. In der Tabelle 49.1 ist dies für die Grove-Schnittstelle festgehalten.

Table 49.1.: Pinbelegung für den Ultraschallabstandssensor [Simc]

Arduino Pin	Ultraschall Abstandssensor
D9	Trig
D6	Echo
3V3	VCC
GND	GND

49.5. Bibliothek

49.5.1. Description

49.5.2. Installation

49.5.3. Functions

49.5.4. Example - Manual

49.5.5. Example

49.5.6. Example - Code

49.5.7. Example - Files

49.6. Calibration

49.7. Kalibrierung

Der Ultraschallabstandssensor wird für die Verwendung bei der Temperatur von 20 °C ausgelegt. Die berechnete Schallgeschwindigkeit $v_{Schall} = 343,2 \frac{\text{m}}{\text{s}}$ bei 20 °C wird für die Umrechnung der Schalldauer in die Distanz zur Reflexionsfläche verwendet.

WS:cite method

49.8. Simple Code

49.9. Einfacher Beispiel zur Verwendung des Ultraschallabstandssensors

Mit dem Sketch `TestHCSR04_UltraschallsensorTest.ino` wird die Funktionalität des Ultraschallsensors getestet.

49.9.1. Durchführung

Für den Test werden die folgenden Hardware-Komponenten benötigt:

```

1000 /**
1001 * @file UltraschallsensorTest.ino
1002 * @author Wings
1003 * @date 20.05.2024
1004 * @details The sketch is used to test the functionality of the
1005 *         ultrasonic sensor. The transit time of the ultrasonic signal is
1006 *         measured
1007 *         The distance is then calculated from the transit time and the speed of
1008 *         sound and displayed on the serial monitor.
1009 * Projekt: Abstandssensor
1010 */
1011
1012 /**
1013 * Pin D6 is referenced as an echo pin for the ultrasonic sensor. */
1014 #define Echopin D6
1015 /**
1016 * Pin D9 is referenced as the trigger pin for the ultrasonic sensor.
1017 */
1018 #define Triggerpin D9
1019
1020 /**
1021 * Der Parameter MaxReichweite definiert die obere Grenze des
1022 * Messbereichs von dem Ultraschallsensor und ist hier in cm angegeben.
1023 */
1024 int MaxReichweite = 215;
1025 /**
1026 * Der Parameter MinReichweite definiert die untere Grenze des
1027 * Messbereichs von dem Ultraschallsensor und ist hier in cm angegeben.
1028 */
1029 int MinReichweite = 3;
1030 /**
1031 * The parameter Anstand stands for the distance between a wall and
1032 * the ultrasonic sensor and is shown on the display in cm. */
1033 double Abstand;
1034 /**
1035 * The duration parameter stands for the runtime of the ultrasonic
1036 * signal and is used here with the unit microseconds. */
1037 double Dauer;
1038 /**
1039 * The parameter Schallgeschwindigkeit stands for the speed of sound
1040 * in the medium air at a temperature of 20C and is specified in m/s.
1041 */
1042 double Schallgeschwindigkeit = 343.2;
1043
1044 /**
1045 * @brief Starten der seriellen Kommunikation und festlegen der Pinmodi
1046 * @details The function is executed when the programme is started. The
1047 *         serial interface to the computer is initialised at 9600 baud, the
1048 *         pin mode for the trigger pin is defined as OUTPUT
1049 *         and the pin mode for the echo pin is defined as INPUT
1050 *         There is no return value.
1051 */
1052 void setup() {
1053     Serial.begin(9600);
1054     pinMode(Triggerpin, OUTPUT);
1055     pinMode(Echopin, INPUT);
1056 }
1057
1058 /**
1059 * @brief Abstand messen und anzeigen
1060 * @details The function is performed continuously. An ultrasonic signal
1061 *         is sent out and the runtime is measured until the signal arrives
1062 *         back at the ultrasonic sensor.
1063 * The distance in cm is calculated using the duration of the runtime and
1064 *         the speed of sound and then displayed on the serial monitor.
1065 * There is no return value.
1066 */
1067 void loop() {
1068     digitalWrite(Triggerpin, LOW); // Der Triggerpin wird auf Low gestellt
1069     delayMicroseconds(2); // Es wird 2 Millisekunden gewartet
1070     digitalWrite(Triggerpin, HIGH); // Das Senden eines Ultraschallsignals
1071         wird initiiert
1072     delayMicroseconds(10); // Es wird 10 Millisekunden gewartet
1073     digitalWrite(Triggerpin, LOW); // Der Triggerpin wird auf Low gestellt
1074     Dauer = pulseIn(Echopin, HIGH); // Measure the time until the signal
1075         returns
1076     Abstand = 0.5 * Schallgeschwindigkeit * Dauer * pow(10,-6) * 100; //
1077         The distance to the reflective surface is calculated in cm
1078     Serial.print(Abstand);
1079     Serial.println(" cm");
1080     delay(5000);
1081 }

```

- Arduino Nano 33 BLE Sense Lite
- Tiny Machine Learning Shield
- USB-A auf USB-Mikro Verbindungskabel
- Grove Jumper zu Grove 4 Pin Kabel
- Ultraschallsensor

Die Hardware-Komponenten werden verbunden und anschließend der Arduino Nano 33 BLE Sense Lite mit einem Computer verbunden. Dann wird der Sketch [TestHCSR04.ino](#) auf den Arduino Nano 33 BLE Sense Lite geladen und der serielle Monitor in der Arduino IDE geöffnet. Ein Gegenstand wird ungefähr 20 cm orthogonal vor den Ultraschallsensor gehalten.

49.9.2. Ergebnisse

Der gemessene Abstand wird in Abbildung 49.5 dargestellt und liegt in dem erwarteten Bereich.



The screenshot shows the Arduino Serial Monitor window. The title bar says "Message (Enter to send message to 'Arduino Nano 33 BLE' on 'COM3')". The main area displays three lines of text output:

```
23:55:54.755 -> 21.14 cm
23:55:59.770 -> 21.16 cm
23:56:04.762 -> 21.19 cm
```

Figure 49.5.: Testoutput des Ultraschallsensors

49.10. Simple Application

49.11. Tests

49.11.1. Simple Function Test

49.11.2. Test all Functions

49.12. Simple Application

49.13. Further Readings

50. Hypatia - Li-Po batteries, Pins and board LEDs

50.1. Battery capacity

Li-Po batteries are charged up to 4,2V with a current that is usually half of the nominal capacity (C/2). For Hypatia we use a specialized chip that has a preset charging current of **350mAh**. This means that the **minimum** capacity of the Li-Po battery should be 700 mAh. Smaller cells will be damaged by this current and may overheat, develop internal gasses and explode, setting on fire the surroundings. We strongly recommend that you select a Li-Po battery of at least 700mAh capacity. A bigger cell will take more time to charge, but won't be harmed or overheated. The chip is programmed with **4** hours of charging time, **then it goes into automatic sleep mode**. This will limit the amount of charge to maximal **1400 mAh** per charging round.

50.2. Battery connector

If you want to connect a battery to your Hypatia be sure to search one with female 2 pin JST PHR2 Type connector.

Polarity : looking at the board connector pins, polarity is Left = Positive, Right = GNDDownload

On the Hypatia, connector is a male 2 pin JST PH Type Vin. This pin can be used to power the board with a regulated 5V source. If the power is fed through this pin. If USB power source is connected, then the battery is not in use. The onboard energy management checks whether the battery is full. If the battery is not full, it will be loaded.

This is the only way you can supply 5V (range is 5V to maximum 6V) to the board not using USB. This pin is an INPUT.

WS:which pin

50.3. 5V

This pin outputs 5V from the the board when powered from the USB connector or from the pin VIN of the board. It is unregulated and the voltage is taken directly from the inputs. When powered from battery it supplies around 3.7 V. As an OUTPUT, it should not be used as an input pin to power the board.

50.4. VCC

This pin outputs 3.3V through the on-board voltage regulator. This voltage is the same regardless the power source used (USB, Vin and Battery).

50.5. LED ON

This LED is connected to the 5V input from either USB or VIN. It is not connected to the battery power. This means that it lits up when power is from USB or VIN, but

stays off when the board is running on battery power. This maximizes the usage of the energy stored in the battery. It is therefore normal to have the board properly running on battery power without the LED ON being lit.

50.6. CHARGE LED

The CHARGE LED on the board is driven by the charger chip that monitors the current drawn by the Li-Po battery while charging. Usually it will light up when the board gets 5V from VIN or USB and the chip starts charging the Li-Po battery connected to the JST connector. There are several occasions where this LED will start to blink at a frequency of about 2Hz. This flashing is caused by the following conditions maintained for a long time (from 20 to 70 minutes):

- No battery is connected to JST connector.
- Overdischarged/damaged battery is connected. It can't be recharged.
- A fully charged battery is put through another unnecessary charging cycle. This is done disconnecting and reconnecting either VIN or the battery itself while VIN is connected.

50.7. Onboard LED

On MKR1000 the onboard LED is connected to D6 and not D13 as on the other boards. Blink example or other sketches that uses pin 13 for onboard LED may need to be changed to work properly.

50.8. Pin Assignment

The voltage of the battery is connected to pin 10. The actual voltage can not be measured.

Built-in Battery Voltage: `PinBatteryVoltage = 10u`

The Voltage is analogue value with 10 bits.

The voltage of the battery can be measured by reading the pin: `analogRead(PinBatteryVoltage)`.

50.9. Checking the Battery Voltage with Hypatia

50.10. Battery Example

<https://ampul.eu/it/batterie/4079-batteria-li-pol-8000mah-37v-126090>
Batteria Li-Pol 8000mAh, 3,7V, 126090

Wiederaufladbare Lithium-Polymer-Akkus werden in allen elektronischen Geräten für den Privatgebrauch verwendet (z. B. Mobiltelefone, Kameras, Laptops, RC-Modelle, ...). Es ist nicht notwendig, den Akku vor dem Aufladen vollständig zu entladen, es gibt keinen Memory-Effekt. Einfache und nahtlose Verbindung vieler Zellen in Reihe.

Integrierter Schutzchip.

Spannung: 3,7V

Kapazität: 8000mAh

Ladespannung: 3,7-4,2V DC

2000 Ladezyklen

Minimale Selbstentladung (ca. 5

Betriebstemperatur: -10 - 50°C

Abmessungen: 90x60x12 mm

Typenbezeichnung: 126090

WARNUNG: Fällt die Spannung unter 2,7 V, kann die Batterie irreversibel zerstört werden.

51. Battery

51.1. Appendix - Powering Your TinyML System

Now that you have your Arduino development board up and running, let's talk about how you could deploy it independent of your computer! While some embedded systems call on AC-DC converters (or "wall warts," colloquially) to provide low voltage power to their electronics (the Google home speaker, for example), others are battery powered. Both of these paradigms are applicable to real-world deployment of tinyML and both are achievable using your TinyML kit.

51.1.1. USB Power Delivery

To this point, we have provided power over USB to our microcontroller via the microB port on the Nano 33 BLE Sense. The 5V that USB carries is then down regulated on the development board to 3.3V, the logical reference for the MCU. While there's nothing wrong with this in development, a prototype of your application ought not depend on drawing power from your computer. Instead, you could call on an AC-DC converter with USB output. This has the added benefit of likely raising the current capacity of the 5V power rail, from at most 500 mA via a computer to whatever the specification happens to be for a given wall-bound converter. This could be meaningful for driving certain power hungry actuators, like speakers.

51.2. Battery

While the above solution removes the need for the computer, you're still tethered to a wall. To go fully mobile you'll want to call on a battery. So what are our options? The idea of using a voltage regulator (specifically, the MPM3610) to cut down an input voltage to a nice, stable 3.3V level applies here as well. If you were to take a closer look at the linked datasheet, you'd find that the MPM3610 accepts input voltages from 4.5V to 21V. The 5V delivered over USB is within this range, and any compatible battery will need to be as well. This unfortunately eliminates the possibility of calling on single cell 3.7V lithium batteries, but makes the selection of a 9V alkaline battery fairly obvious.

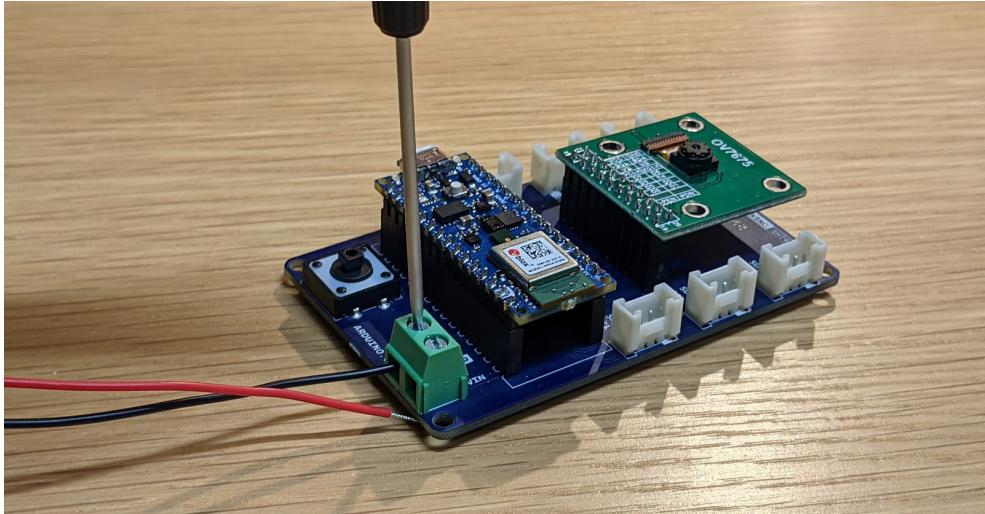
You might be wondering how any battery might connect to the boards in front of you, but never fear, we've got you covered. At one corner of the Tiny Machine Learning Shield you'll find a green terminal block with silkscreen labels that read, "VIN" and "GND," where GND is our reference voltage and as such should be connected to the negative terminal of any compatible battery. This green terminal block is where you'd want to screw in wires carrying 4.5V to 21V, and we'll add that 9V clip, like this, that terminates in pre-stripped hook-up wire makes this quite easy!

Assembly Steps

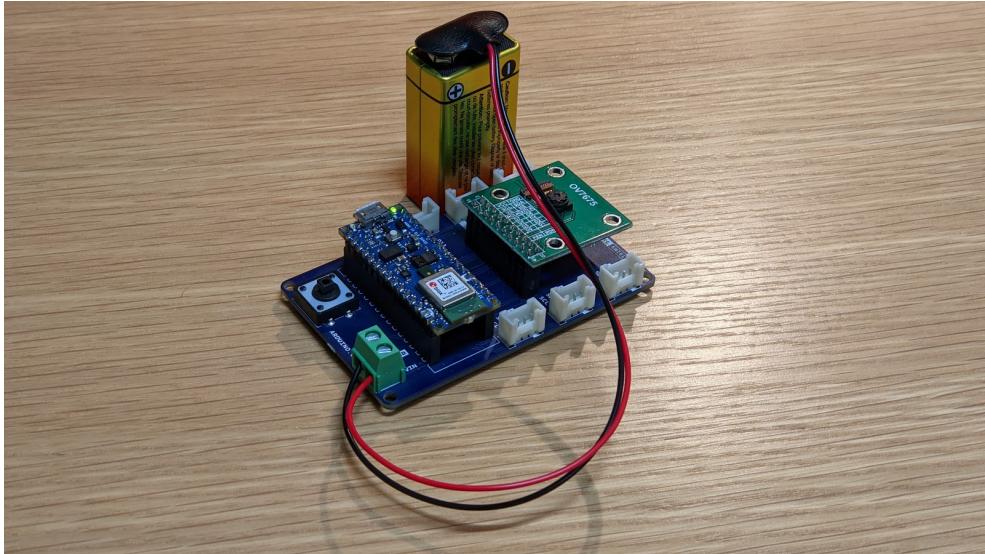
1. Screw down a wire leading from the negative battery terminal (black) to GND (Most < 3mm flat-head screwdrivers will suffice here).
2. Repeat this process for the positive battery terminal (red) to VIN. And that's it you're all set to power your Arduino from a battery!

Important Notes

1. While there is clever circuitry on board to handle such an exception, it is generally good practice to avoid having competing power sources, so we'd recommend that you unplug the Nano 33 BLE Sense from USB power before connecting a battery
2. With about 550 mAh capacity, a 9V battery can source 15 mA for about 37 hours before you will need to get a new one.



An image of screwing down the black negative terminal to attach a wire.



An image of the attached battery powering the device.

51.3. Checking the Battery Voltage

We use an analog input pin to read the voltage. As we are running from a 3.7V volt battery, we need to adjust the reference voltage used by the pin as otherwise it would be comparing the voltage to itself. The statement `analogReference(INTERNAL)` sets the pin to compare the input voltage to a regulated 1.1V. We therefore need to reduce the voltage on the input pin to less than 1.1V for this to work. This is done by dividing the voltage using 2 resistors, 1m and 330k ohms. This divides the voltage by

approximately 4 so when the battery is fully charged, which is 4.2V, the voltage at the pin input is $4.2/4 = 1.05V$.

```
// Battery Monitor
#define MONITOR_PIN A0           // Pin used to monitor supply voltage
const float voltageDivider = 4.0; // Used to calculate the actual voltage from
                                  // Using 1m and 330k ohm resistors divides the
voltage by approx 4              // You may want to substitute
actual values of resistors in an equation (R1 + R2)/R2
                                  // E.g. (1000 + 330)/330 = 4.03
                                  // Alternatively take the voltage reading across
                                  // the 2 resistors to ground and divide one

// Read the monitor pin and calculate the voltage
float BatteryVoltage()
{
    float reading = analogRead(MONITOR_PIN);
    // Calculate voltage - reference voltage is 1.1v
    return 1.1 * (reading/1023) * voltageDivider;
}
```

The function `BatteryVoltage()`, reads the analog pin, which will range from 0 for 0V to 1,023 for 1.1V and using this reading calculates the actual voltage coming from the battery.

The function `DrawScreenSave()` function calls this then selects the appropriate bitmap to display based on the following:

- If voltage is greater than 3.6V - full
- Voltage between 3.5 and 3.6V - 3/4
- Voltage between 3.4 and 3.5V - half
- Voltage between 3.3 and 3.4V - 1/4
- Voltage < 3.3V - empty

51.4. Batterieclip

Der Batterieclip in Abb. 51.1, der vom Hersteller *reichelt* ist, kann vertikal an einen 9-Volt-Block angeschlossen werden. Die dazugehörigen Anschlussdrähte haben eine Länge von 150 mm. Der Anschlussclip ist in der I-Form ausgeführt, weshalb er sich platzsparend ins Gehäuse einbinden lässt [Rei].

51.5. Spannungssensor

Der Spannungssensor in Abb. ?? von dem Hersteller *Shenzhen Global Technology Co., Ltd* kann bei der Versorgungsspannung von 3,3 V Spannungen in dem Bereich von 0 V bis 16,5 V messen. Dieser wird genutzt, um den Ladestand der Batterie zu überwachen. Die analoge Auflösung des Sensors liegt bei 10 Bit. Damit kann bei dem angegebenen Messbereich die Spannung mit der Auflösung von 0,016 V gemessen werden. Zur Eingangsschnittstelle gehört der VCC-Anschluss und der GND-Anschluss [She]. Die Bauteilmaße betragen 13 mm x 27 mm.



Figure 51.1.: Batterieclip für 9-Volt-Block[Rei24b]

51.6. Spannungssensor

Mit dem Sketch [TestBattery.ino](#) soll der Spannungssensor getestet werden.

51.6.1. Durchführung

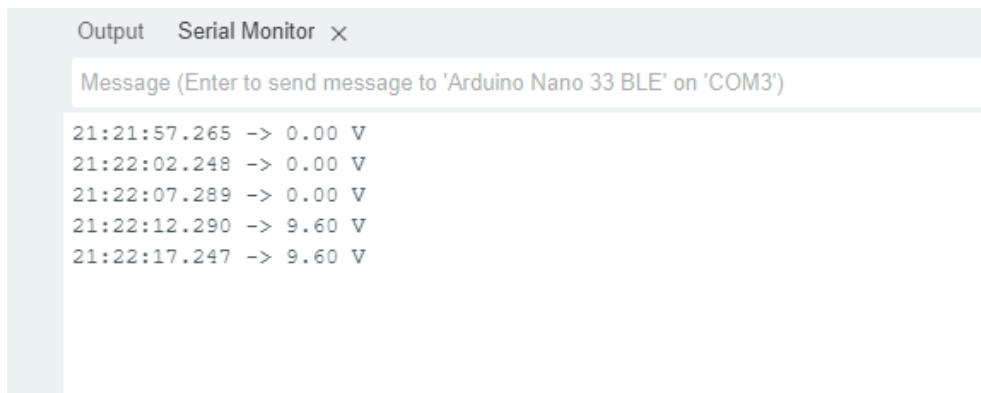
Für den Test werden die folgenden Hardware-Komponenten benötigt:

- Arduino Nano 33 BLE Sense Lite
- Tiny Machine Learning Shield
- USB-A auf USB-Mikro Verbindungskabel
- Grove Jumper zu Grove 4 Pin Kabel
- Spannungssensor
- Batterie
- Batterieclip

Die Hardware-Komponenten werden zusammengebaut, aber die Batterie wird noch nicht angeschlossen. Dann wird der Arduino Nano 33 BLE Sense Lite mit einem Computer verbunden. Anschließend wird der Sketch [TestBattery.ino](#) auf den Arduino Nano 33 BLE Sense Lite geladen und der serielle Monitor in der Arduino IDE geöffnet. Die Batterie wird während des Tests an den Batterieclip angeschlossen und der gemessene Wert bei dem seriellen Monitor ausgelesen.

51.6.2. Ergebnisse

Zu Beginn des Tests zeigt der serielle Monitor die Spannung 0 V an. Nach dem Anschließen der Batterie an den Spannungssensor wird die Spannung 9,6 V angezeigt. Abbildung 51.2 zeigt den gemessenen Spannungsverlauf. Die angezeigte Spannung liegt in dem erwarteten Bereich einer 9 V Batterie.



The screenshot shows the Arduino Serial Monitor window. The title bar says "Output" and "Serial Monitor X". Below the title bar is a message input field with the placeholder "Message (Enter to send message to 'Arduino Nano 33 BLE' on 'COM3')". The main area displays the following serial output:

```
21:21:57.265 -> 0.00 V
21:22:02.248 -> 0.00 V
21:22:07.289 -> 0.00 V
21:22:12.290 -> 9.60 V
21:22:17.247 -> 9.60 V
```

Figure 51.2.: Testoutput des Spannungssensors

51.7. Lithium Battery

Betrieben wird der Arduino Nano 33 BLE Sense mit einer Lithium Batterie CR2032:
[Ene]

<https://data.energizer.com/pdfs/cr2032.pdf>

51.7.1. Attention

Der Arduino Nano 33 BLE Sense lässt sich über die eingebaute Micro-USB Schnittstelle programmieren.

Dazu muss der Jumper geschlossen (gesteckt werden).

ACHTUNG vor dem Stecken des Jumpers muss der Stab ausgeschaltet sein und er darf nicht eingeschaltet werden, solange der Jumper steckt!!

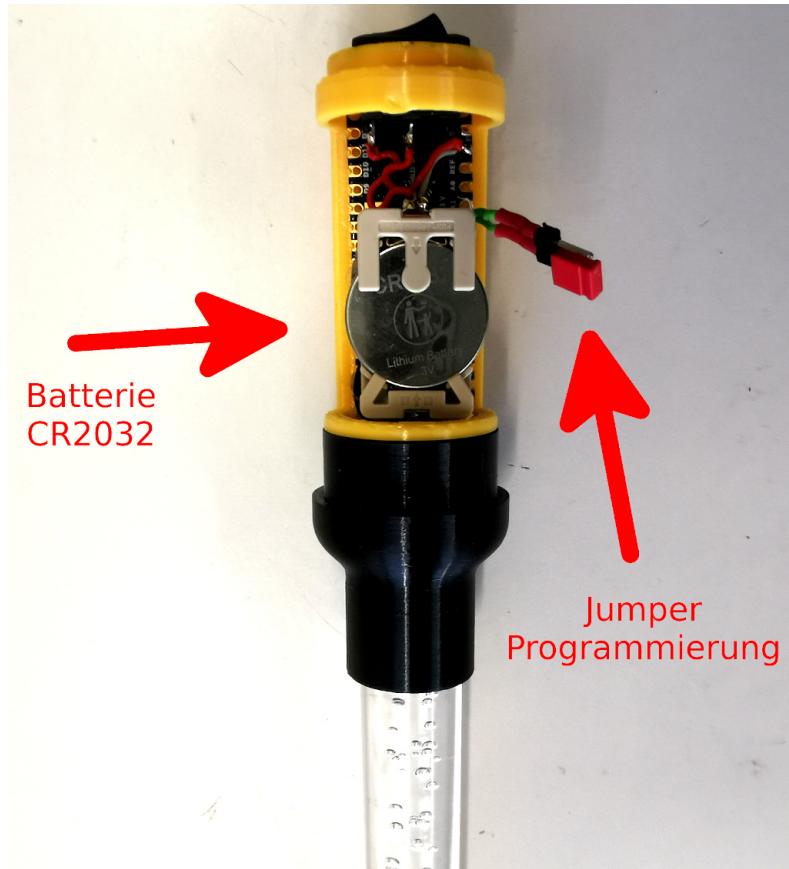


Figure 51.3.: Example with Battery and Jumper

51.7.2. Battery

The battery of the Arduino Nano 33 BLE Sense is a button battery CR2032 on Lithium, to used correctly the battery we need a specific connection. This battery connection is also connect on Power button.

The battery need some description about the capacity, the current voltage for Arduino Nano 33 BLE Sense battery is about 3.3V. The button cell batteries can have a capacity of up to 3.7V, making them ideal for the Arduino Nano, since they are only used occasionally. The voltage will then vary between 3V when it's full and 2.6V when it's empty.

The connection between battery and card is something particular with the power button, we connect the battery and the card like this :

- The positive part of the battery is connected to one part of the button ON/OFF.
- The second part of the button ON/OFF button is connected to the +3.3V battery.
- The negative go to Arduino GND to button battery GND.

51.7.3. Jumper

The jumper consists of two wires that never touch. When the jumper is not activated, either the two wires are touching, or there is no soldering and the jumper is not cut; this case is effective when there is no battery.

When a battery is added, the jumper is set 51.5, preventing the creation of a short circuit.

The battery's GND is on the board's GND (in black) and the battery's 3V is on the board's 3.3V. The jumper, on the other hand, is "in a vacuum", and the battery and jumper connections are defined by the board below 51.4.

The creation and installation of this jumper is essential to the operation of the battery. Without the jumper in place, the arduino board would be in danger.

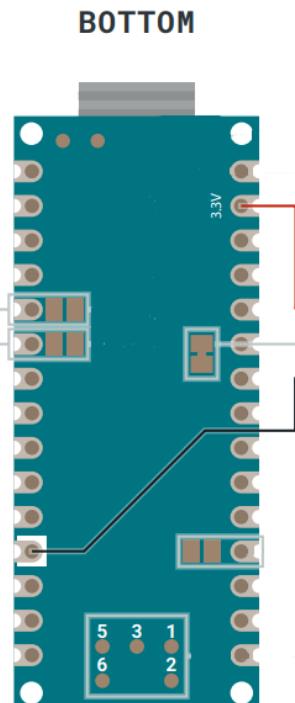


Figure 51.4.: Jumper connection Arduino card

WS:tikz picture with numbers

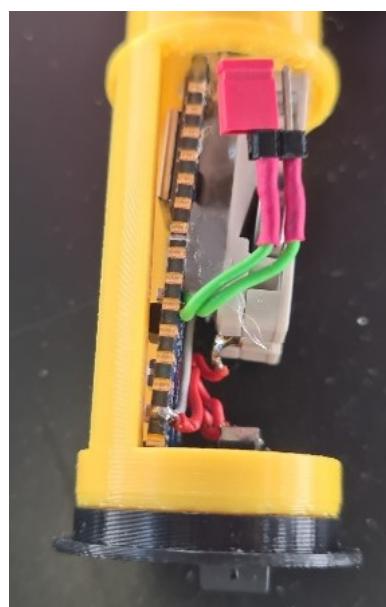


Figure 51.5.: Photo of the jumper on the project

<https://forum.arduino.cc/t/can-i-power-the-nano-33-ble-via-the-3v3-pin-with-3-3v/614634>

I tried both ways to power nano 33 ble and as you can imagine powering from Vin is less efficient than from 3.3v pin. To be more specific for a basic sketch where nano acts like a peripheral and sends gyro + accel values to a central device, powering from vin through a 5v booster the power consumption was 18mAh. The same sketch consumes 9.30mAh powering from 3.3v pin without any low power configuration. Just have in mind that in order to use direct the 3.3v pin you have to cut out the relative jumper on the bottom layer. Also, if you want to re flush nano board you have to connect the jumper again, otherwise usb connection does not work. I suppose for that you can do something with the Vusb jumper but i did not try it yet.

So as if i understand correctly I can wire 3.3V to the 3V3 pin under the condition: Just have in mind that in order to use direct the 3.3v pin you have to cut out the relative jumper on the bottom layer.

Which means I have to break this connection marked with the blue square?

[EDIT: While doing more research on this I could not find any information on the SJ4 connection on the internet. The the only thing I was able to find and is mentioned everywhere is information about SJ1. Could you give me the source of the information about SJ4? What is the orientation of these pads? I am using the references from the official datasheet.]

https://content.arduino.cc/assets/NANO33BLE_V2.0_sch.pdf

<https://forum.arduino.cc/t/power-nano-33-ble-sense-with-3-3v/654217>

<https://dumblebots.com/2020/03/27/getting-started-with-arduino-nano-33-ble-and-b>

<https://community.element14.com/products/roadtest/b/blog/posts/review-of-development-board-arduino-nano-33-ble-sense-roadtest>

https://tinyml.seas.harvard.edu/assets/other/4D/22.03.11_Marcelo_Rovai_Handout.pdf

<https://learn.adafruit.com/circuitpython-on-the-arduino-nano-rp2040-connect?view=all>

<https://support.arduino.cc/hc/en-us/articles/360014735580-Nano-boards-that-can-be-powered-via-3v3-or-vin>

```
1000 /**
1001 * @file TestBattery.ino
1002 * @author Wings
1003 * @date 20.05.2024
1004 * @details The sketch is used to test the voltage sensor. For the test,
1005 *          the voltage is measured on a 9 V block battery and output on the
1006 *          serial monitor.
1007 */
1008 /**
1009 * Pin A7 is referenced as the voltage pin for the voltage sensor. */
1010 #define VoltagePin A7
1011 /**
1012 * The parameter SignalEingang is later assigned the value read in from
1013 *          the VoltagePin pin. */
1014 long SignalEingang;
1015 /**
1016 * The parameter Spannung stands for the voltage that is measured with
1017 *          the voltage sensor and has the unit volt. */
1018 double Voltage;
1019 /**
1020 * @brief Starting serial communication
1021 *
1022 * @details The function is executed when the programme is started. The
1023 *          serial interface to the computer is initialised at 9600 baud
1024 *          There is no return value.
1025 */
1026 void setup() {
1027     Serial.begin(9600);
1028 }
1029 /**
1030 * @brief Display of the measured voltage
1031 *
1032 * @details The function is executed continuously. The signal at the pin
1033 *          VoltagePin is read out and a voltage value is converted.
1034 *          The value read in at the voltage pin is between 0 and 1023. The
1035 *          voltage measurement range is between 0 and 16.5 V. The read-in value
1036 *          can be converted into a voltage using the function map().
1037 *          The voltage is displayed in volts on the serial monitor. There is a 5-
1038 *          second wait after each cycle to avoid flooding the serial monitor.
1039 *          There is no return value.
1040 */
1041 void loop() {
1042     SignalEingang = analogRead(VoltagePin);
1043     Voltage      = map(SignalEingang, 0, 1023, 0, 165); // Converting the
1044     // input signal 165 for 16.5 V
1045     Voltage      = Voltage/10; // Converting the input signal 165 for
1046     // 16.5 V
1047     Serial.print(Voltage);
1048     Serial.println(" V");
1049     delay(5000);
1050 }
```

..../Code/Arduino/Battery/TestBattery.ino

52. Powerbank

Die Spannungsversorgung erfolgt über die Powerbank. Sie verfügt über einen Taster, mit dem sie ein- und ausgeschaltet werden kann. Zum Einschalten muss der Taster für eine Sekunden gedrückt werden. Bei kurzer Betätigung wird dem Nutzer der Ladezustand angezeigt. Zum Ausschalten muss der Taster drei Sekunden lang gedrückt werden.

WS:Text fehlt



Figure 52.1.: Die verwendete Powerbank

WS:Quelle fehlt

53. Entwicklerboard - Motorsteuerung DEBO MotoDriver2 L298N

Um die zwei verwendeten Motoren ansteuern zu können, wird die Erweiterungsplatine DEBO MotoDriver2 L298N benutzt. Diese erlaubt die Steuerung und Versorgung von bis zu zwei Gleichstrommotoren. Die Motoren können mit Spannungen zwischen 5V und 35V angetrieben werden. Der verwendete Chipsatz ist L298N. Es wird auf dem Logiklevel von 5V gearbeitet. Die Ausgangsleistung beträgt 25W bei einem maximalen Treiberstrom von bis zu 2A. Die Abmaße der Platine sind 43mm × 43mm × 27mm. In der folgenden Abbildung 53.1 ist die Erweiterungsplatine zu sehen.

In der folgenden Abbildung 53.2 sind die Pins des Treibers beschriftet und in der Tabelle 53.1 ist die Pinbelegung dargestellt. [Simb]

PIN	Belegung
1	DC Motor 1 / Stepper Motor +
2	DC Motor 1 / Stepper Motor GND
3	12V Jumper
4	Stromversorgung +
5	Stromversorgung GND
6	5V Ausgang (wenn Jumper 3 gesetzt)
7	DC Motor 1 Jumper
8	Input 1
9	Input 2
10	Input 3
11	Input 4
12	DC Motor 2 Jumper
13	DC Motor 2 / Stepper Motor +
14	DC Motor 2 / Stepper Motor GND

Table 53.1.: Pinbelegung des Treibers

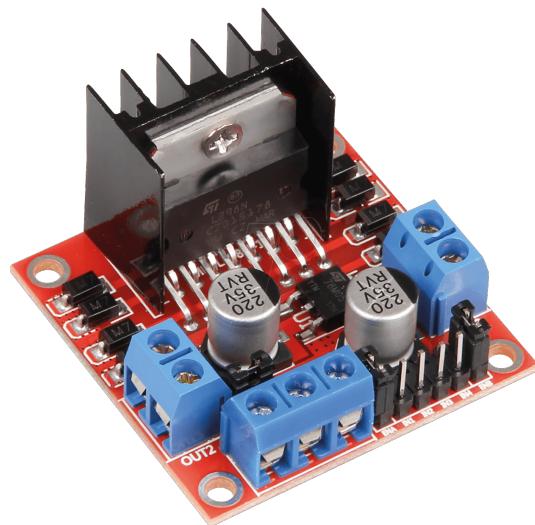


Figure 53.1.: Motorsteuerung DEBO MotoDriver2 L298N

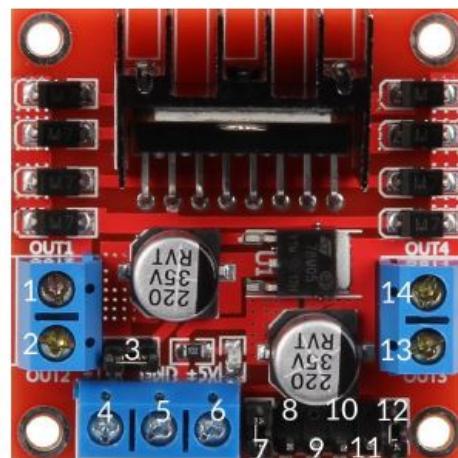


Figure 53.2.: Treiber mit Beschriftung

54. Terminal Adapter Board mit Schraubklemmen kompatibel mit Nano V3 und Arduino

Das verwendete Shield führt alle Anschlüsse des Arduinos auf außenliegende Schraubklemmen. Dies ermöglicht die Nutzung aller Anschlüsse durch das Verwenden von Jumper-Kabeln. Das Shield ist in der folgenden Abbildung 54.1 zu sehen. Die Abmessungen des Shields sind 54x43x13mm und es hat ein Gewicht von 21 Gramm. [Az c]

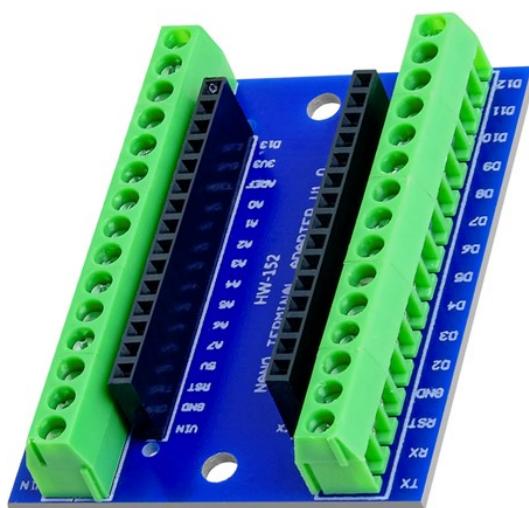


Figure 54.1.: Terminal Adapter Board mit Schraubklemmen kompatibel mit Nano V3 und Arduino

55. Getriebemotor mit Rad

Für den Antrieb werden zwei Getriebemotoren mit Rädern von JOY-IT verwendet. Es handelt sich um einen Getriebemotor mit großem Spannungsbereich und beidseitiger Welle. Der Motor ist in Abbildung 55.1 zu sehen. Die technischen Daten sind in den Tabellen 55.1 und 55.2 dargestellt. [Sima]

55.1. Funktion Motoren

Nachdem der Roboter nach Anleitung aufgebaut ist, kann durch Laden des folgenden Codes 55.1 auf den Arduino getestet werden, ob die Motoren richtig angeschlossen sind und funktionieren. Es ist wichtig, dass beide Motoren in die selbe Richtung drehen, ansonsten ist die Verkabelung zu überprüfen.

Listing 55.1: Arduino Motor Test

```
// Motorsteuerungs-Pins
const int in1 = 6;
const int in2 = 9;
const int in3 = 10; const int in4 = 11;

// Funktion zum Vorwaertsfahren der Motoren
void forward(int geschw) {
    analogWrite(in1, geschw);
    analogWrite(in2, 0);
    analogWrite(in3, geschw);
    analogWrite(in4, 0);

    Serial.print("Motoren_vorwaerts:_Geschwindigkeit_= ");
    Serial.println(geschw);
}

// Funktion zum Rueckwaertsfahren der Motoren
void backward(int geschw) {
```



Figure 55.1.: Getriebemotor

Eigenschaft	Wert
Welle	3,6mm beidseitig mit Loch 1,9mm
Spannungsversorgung	3 - 9V DC (empfohlen: 4,5V)
Abmessungen (Rad)	65mm
Durchmesser	27mm
Breite	
Abmessungen (Motor)	37,6 x 64,2 x 22,5mm

Table 55.1.: Allgemeine Informationen des Getriebemotors mit Rad

Spannung DC (V)	4,5	6	7,2	9
Stromstärke im Leerlauf (mA)	190	160	180	200
Drehzahl pro Min. im Leerlauf ($\pm 10\%$)	90	190	230	300
Drehmoment (gf/cm)	800	800	1000	1200

Table 55.2.: Technische Daten des Getriebemotors mit Rad

```

analogWrite(in1, 0);
analogWrite(in2, geschw);
analogWrite(in3, 0);
analogWrite(in4, geschw);

Serial.print("Motoren\u2190rueckwaerts : \u2190Geschwindigkeit\u2190 ");
Serial.println(geschw);
}

// Funktion zum Stoppen der Motoren
void stopMotors() {
    analogWrite(in1, 0);
    analogWrite(in2, 0);
    analogWrite(in3, 0);
    analogWrite(in4, 0);

    Serial.println("Motoren\u2190gestoppt");
}

void setup() {
    // Initialisierung der seriellen Kommunikation
    // fuer Debugging
    Serial.begin(9600);

    // Konfigurieren der Motorsteuerungspins als Ausgaenge
    pinMode(in1, OUTPUT);
    pinMode(in2, OUTPUT);
    pinMode(in3, OUTPUT);
    pinMode(in4, OUTPUT);
}

void loop() {
    // Motoren fuer 5 Sekunden vorwaerts fahren
    Serial.println("Motoren\u2190fahren\u2190vorwaerts");
    forward(255); // Volle Geschwindigkeit
    delay(5000); // 5 Sekunden warten
}

```

```
// Motoren fuer 5 Sekunden rueckwaerts fahren
Serial.println("Motoren_fahren_rueckwaerts");
backward(255); // Volle Geschwindigkeit
delay(5000); // 5 Sekunden warten

// Motoren fuer 5 Sekunden anhalten
Serial.println("Motoren_anhalten");
stopMotors();
delay(5000); // 5 Sekunden warten
}
```


56. Drehwinkel-Encoder

Der Demonstrator soll mehrere Programme fahren können, deswegen wurde ein Drehwinkel-Encoder der Steuerung hinzugefügt. Dieser wird über drei Pins am Arduino angeschlossen und über zwei weitere Pins wird er mit Spannung versorgt. Durch drehen des Drehschalters werden nacheinander drei Kontakte geschlossen oder geöffnet (ein Kontakt ist immer geschlossen). Dieser dadurch entstehende Signalfloss, bestehend aus zwei um 90 Grad versetzte Sinus bzw. Cosinusschwingungen werden ausgewertet. Daraus wird bestimmt, in welche Richtung (im oder gegen Uhrzeigersinn) und wie weit (inkrementell) gedreht wurde. Mithilfe dieser Logik kann durch ein Menü eine Bewegungsstufe ausgewählt werden, die der Schrittmotor fahren soll.[Bas16] Bei einer Drehung im Uhrzeigersinn wird im Menü eine Bewegungsstufe höher und bei einer Drehung gegen den Uhrzeigersinn eine Bewegungsstufe niedriger ausgewählt. Zusätzlich zum Drehwinkel-Encoder ist auch noch ein Schaltfunktion im Bauteil selbst integriert. Durch eindrücken des Encoders wird ein Taster betätigt, durch denn der eingestellte Wert bestätigt und an den Arduino zur weiteren Verarbeitung weitergegeben wird. Zur Besseren Handhabung des Drehwinkel-Encoders wurde noch ein Drehgriff angefertigt und auf dem Drehgeber montiert. Weitere Details:

- **Abmessungen** ($b \times l \times h$): $18\text{ mm} \times 31\text{ mm} \times 30\text{ mm}$
- **Betriebsspannung:** 3,3 - 5 V [Sim19]

56.1. Encoder

Die Library Encoder ermöglicht das Zählen von Impulsen aus bestimmten Signalen, die häufig von Drehknöpfen, Motor- oder Wellensensoren sowie anderen Positionsgebern stammen. Diese Bibliothek ist für Arduino und Teensy Boards optimiert und bietet verschiedene Zählmodi, darunter 4X counting für präzise Positionserfassung. Die Verwendung erfolgt durch die Initialisierung eines Encoder-Objekts mit zwei Pins, wobei der erste Pin idealerweise über Interruptfähigkeit verfügt für optimale Leistung. Die Bibliothek unterstützt sowohl I2C- als auch SPI-Schnittstellen und bietet verschiedene Hardware- und Softwareoptionen zur Anpassung an die Anforderungen des Benutzers. Sie wird in diesem Projekt in der Version 1.4.4 verwendet.[Sto24]

56.2. Drehwinkel-Encoder

Um die Funktion und korrekte Ansteuerung des Drehwinkel-Encoders zu testen, wird das Testprogramm 56.1 verwendet. Das Programm wertet die Drehung des Encoders aus und gibt den neuen Zustand im seriellen Monitor der Arduino IDE aus [Sto24].

```
#include <Encoder.h>

const int CLK = 9;
const int DT = 2;
const int SW = 3;
long altePosition = -999;

Encoder meinEncoder(DT, CLK);

void setup()
{
    Serial.begin(9600);
    pinMode(SW, INPUT);
}

void loop()
{
    long neuePosition = meinEncoder.read();

    if (neuePosition != altePosition)
    {
        altePosition = neuePosition;
        Serial.println(neuePosition);
    }

    int StatusTaster = digitalRead(SW);
    if (StatusTaster == 0) {
        Serial.println("Switch unbetaetigt");
        Serial.println("Switch betätigte");
    }
}
```

Listing 56.1.: Testprogramm für den Encoder

57. Arduino - SD-Karten-Modul

Ein SPI-Lesegerät ist eine Art Modul, das die Kommunikation zwischen einem Mikrocontroller und einer Speicherplatte, z. B. einer Secure Digital (SD)- oder TensorFlow (tf)-Karte, ermöglicht. Das Modul verwendet das SPI-Protokoll, um Daten zwischen den beiden Geräten zu übertragen. Das SD/tf-Card-Shield-Modul ist ein spezieller Typ von SPI-Leser, der für die Verwendung mit Mikrocontroller-Boards konzipiert ist. Es ermöglicht einen einfachen Zugriff auf die auf der Speicherplatte gespeicherten Daten und kann für eine Vielzahl von Anwendungen wie Datenprotokollierung, Dateispeicherung und Multimedia-Projekte verwendet werden. Das Shield wird über die SPI-Schnittstelle mit dem Mikrocontroller verbunden.

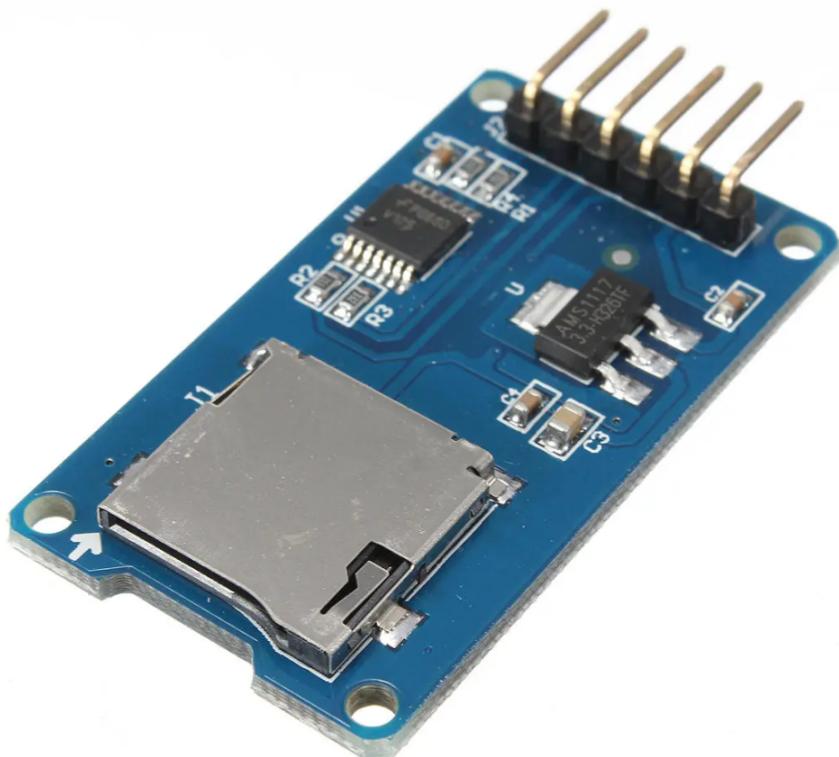


Figure 57.1.: SD/TF-Card-Shield-Modul

57.1. Eigenschaften

Leichte und kostengünstige Erweiterung: Der SD-Karten Slot ermöglicht eine einfache und kostengünstige Erweiterung des Speicherplatzes für Ihr Arduino-Pro-

jekt mittels SD-Karte.

Einfache Kommunikation: Das Modul kommuniziert mit dem Mikrocontroller über das SPI-Protokoll, welches eine zuverlässige und schnelle Datenübertragung gewährleistet.

Unterstützung von Speicherkarten: Es unterstützt sowohl Micro-SD-Karten (bis zu 2GB) als auch Micro-SDHC-Karten (bis zu 32GB), inklusive High-Speed-Karten.

Spannungsunterstützung: Das Modul ist kompatibel mit einer Versorgungsspannung von 3,3-5V, was die Integration in verschiedene Projekte erleichtert.

57.2. Technische Spezifikationen

Schnittstelle: SPI

Unterstützte Karten: Micro-SD-Karte (2GB), Micro-SDHC-Karte (bis zu 32GB)

Versorgungsspannung: 3,3-5V

Kompatibilität: Arduino und andere Mikrocontroller-Boards

[Az b]

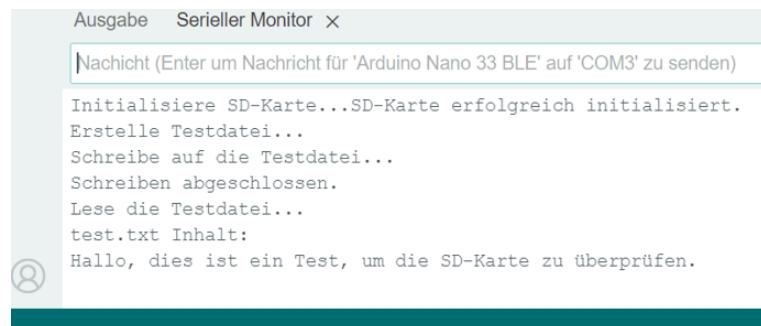
57.3. Bibliothek [SD.h](#)

Die SD.h Bibliothek ist eine essenzielle Bibliothek für Arduino-Entwickler, die es ermöglicht, SD-Karten in ihren Projekten zu verwenden. Sie bietet eine einfache und effiziente Möglichkeit, Daten auf SD-Karten zu speichern und zu lesen. Die Bibliothek unterstützt das FAT16 und FAT32 Dateisystem, was sie kompatibel mit den meisten SD-Karten macht. Mit der SD.h Bibliothek können Dateien erstellt, gelöscht, umbenannt und bearbeitet werden. Zudem gibt es Funktionen zum Lesen und Schreiben von Daten in Dateien. Die Bibliothek ermöglicht das Erstellen und Löschen von Verzeichnissen (Ordnern) und unterstützt das Navigieren durch Verzeichnisse, was das Arbeiten mit komplexen Dateistrukturen erleichtert.

Die SD.h Bibliothek ist kompatibel mit den meisten Arduino-Boards, einschließlich Arduino Uno, Mega, Nano und anderen. Sie unterstützt sowohl Standard-SD-Karten als auch microSD-Karten mit einem geeigneten Adapter. Die Integration der Bibliothek in Arduino-Sketches ist einfach und erfolgt durch leicht verständliche Methodenaufrufe. Die Arduino IDE enthält zudem Beispiele und umfangreiche Dokumentation, die den Einstieg in die Nutzung der SD.h Bibliothek erleichtern.

57.4. Testen des SD-Karten Moduls

Um das SD-Karten Modul zu testen, haben wir ein Beispiel erstellt, das zeigt, wie man mit einem Arduino Daten auf einer SD-Karte speichert und wieder ausliest. Basierend auf der Anleitung Funduino - Das SD-Karten Modul haben wir den dortigen Beispielcode leicht angepasst [Az b]. Zunächst wird die SD-Karte initialisiert, um sicherzustellen, dass sie ordnungsgemäß funktioniert. Danach wird eine Testdatei erstellt, in die ein kurzer Text geschrieben wird. Anschließend wird der Inhalt dieser Datei ausgelesen und über den seriellen Monitor angezeigt.



Ausgabe Serieller Monitor ×

Nachricht (Enter um Nachricht für 'Arduino Nano 33 BLE' auf 'COM3' zu senden)

```
Initialisiere SD-Karte...SD-Karte erfolgreich initialisiert.  
Erstelle Testdatei...  
Schreibe auf die Testdatei...  
Schreiben abgeschlossen.  
Lese die Testdatei...  
test.txt Inhalt:  
Hallo, dies ist ein Test, um die SD-Karte zu überprüfen.
```

Figure 57.2.: Ausgabe SD-Kartentest Serieller Monitor

```

1000 #include <SPI.h>
1001 #include <SD.h>
1002 const int chipSelect = 10;
1004 void setup() {
1006     Serial.begin(9600);
1008     while (!Serial) {
1009         ;
1010     }
1012     Serial.print("Initialisiere SD-Karte... ");
1014     if (!SD.begin(chipSelect)) {
1015         Serial.println("SD-Karten Initialisierung fehlgeschlagen!");
1016         return;
1017     }
1018     Serial.println("SD-Karte erfolgreich initialisiert.");
1020     Serial.println("Erstelle Testdatei... ");
1021     File testFile = SD.open("test.txt", FILE_WRITE);
1022     if (testFile) {
1023         Serial.println("Schreibe auf die Testdatei... ");
1024         testFile.println("Hallo, dies ist ein Test, um die SD-Karte zu
1025             ueberpruefen.");
1026         testFile.close();
1027         Serial.println("Schreiben abgeschlossen.");
1028     } else {
1029         Serial.println("Fehler beim Oeffnen der Testdatei.");
1030     }
1032     Serial.println("Lese die Testdatei... ");
1033     testFile = SD.open("test.txt");
1034     if (testFile) {
1035         Serial.println("test.txt Inhalt:");
1036         while (testFile.available()) {
1037             Serial.write(testFile.read());
1038         }
1039         testFile.close();
1040     } else {
1041         Serial.println("Fehler beim Oeffnen der Testdatei.");
1042     }
1043 }
1044 void loop() {
1045 }
```

..../Code/Arduino/SDCard/TestSDCard.ino

Listing 57.1.: Testprogramm SD-Karten-Modul

58. Sensor DEBO DHT22

Der Sensor DEBO DHT22 wird verwendet, um den fehlenden Sensor im Arduino Nano 33 BLE Sense Lite auszugleichen. Der DHT22 Sensor ermittelt die Temperatur und Luftfeuchtigkeit. Angeschlossen wird der Sensor mit einem 4.7KOhm Pull-Up Widerstand. Angeschlossen wird der Sensor mit Steckbrückenkabel, welche in einen Grovestecker übergehen und mit dem TMLS verbunden sind. Er verfügt über die Anschlüsse GND für die Masse, VCC zur Spannungsversorgung und einen Datenpin zur Datenübertragung. Der Widerstand wird parallel zum Sensor geschaltet.

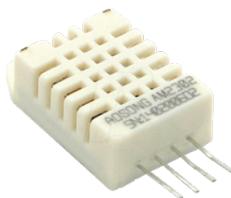


Figure 58.1.: Sensor DHT22

58.1. Luftdruck

Der Luftdruck wird durch einen ultrakompakten piezoresistiven Absolutdrucksensor ermittelt. Für die elektronische Druckmessung ist ein Sensor erforderlich, der den zu messenden absoluten Luftdruck aufnimmt und in ein elektrisches Signal umwandelt. Unterschieden wird hier zwischen der resistiven und piezoresistiven Druckmessung.

58.1.1. Resistive Druckmessung

Die resistive Druckmessung ist die klassische Druckmessung und funktioniert über einen dünnen Metallstreifen, dessen Widerstandswert sich bei Verformung ändert. Bei Dehnung wird der Streifen länger und dünner, sodass der elektrische Widerstand steigt. Umgekehrt folgt daraus, dass bei Stauchung der Streifen kürzer wird und der Querschnitt steigt, was wiederum einen verringerten Widerstand aufweist. Um den zu messenden Druck in eine kontrollierte mechanische Verformung zu übersetzen, wird der Dehnungsmessstreifen (DMS) auf eine elastische Membran mittels Klebstoff aufgebracht. Wirkt nun auf einer Seite dieser Membran ein Überdruck, so verformt sich diese und führt je nach Position zur Stauchung oder Dehnung des DMS, siehe 58.2.

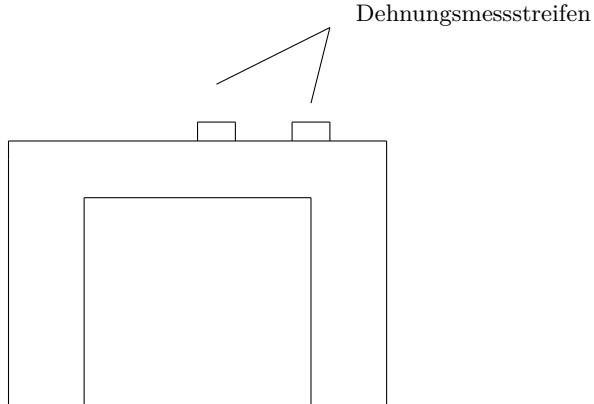


Figure 58.2.: Positionierung der DMS auf Membran

58.1.2. Piezoresistive Druckmessung

Die piezoresistive Druckmessung basiert auf einem ähnlichen Prinzip. Auch hier bewirkt eine Verlängerung oder Verkürzung eine Änderung des Widerstandes. Zusätzlich führt in einem piezoresistiven Material die mechanische Spannung, die bei Dehnung oder Stauchung auftritt, auch zu einer Änderung der elektrischen Leitfähigkeit. Dieser Effekt basiert auf Verschiebungen der Atompositionen zueinander, welche sich direkt auf den elektrischen Ladungstransport auswirken. Die aus der Änderung der elektrischen Leitfähigkeit resultierende Widerstandsänderung kann deutlich größer ausfallen als jene, die durch reine Verformung bedingt ist. Grundlage für einen piezoresistiven Sensorchip sind weniger als einen Millimeter dünne, kristalline Siliziumscheiben, sogenannte Wafer. In dessen Oberfläche werden an bestimmten Stellen Fremdatome eingebracht, die örtlich gezielt die Leitfähigkeit beeinflussen. Dieser Prozess ist das sogenannte Dotieren, und diese dotierten Gebiete im Silizium bilden die piezoresistiven Widerstände.

In einem nachfolgenden Prozessschritt wird dann das Siliziumwafer örtlich so abgedünnt, dass Membranen direkt im Silizium entstehen und die piezoresistiven Widerstände, ähnlich wie in 58.2 gezeigt, an bestimmten Positionen liegen. Wirkt nun auf eine Seite dieser Membran ein Druck, verformt sich diese und bewirkt so eine mechanische Spannung in den piezoresistiven Widerständen. Je nach Position nimmt der Widerstandswert zu oder ab. Über die Dicke der verbleibenden Membran lässt sich die Druckempfindlichkeit des Sensorchips einstellen. Anschließend wird die Rückseite des Siliziums fest mit einem Glas verbunden, siehe 58.3. Dabei entsteht für Absolutdrucksensoren ein abgeschlossener Referenzraum unter Vakuum. Für die Messung eines Relativdrucks enthält das rückseitige Glas ein Referenzloch.

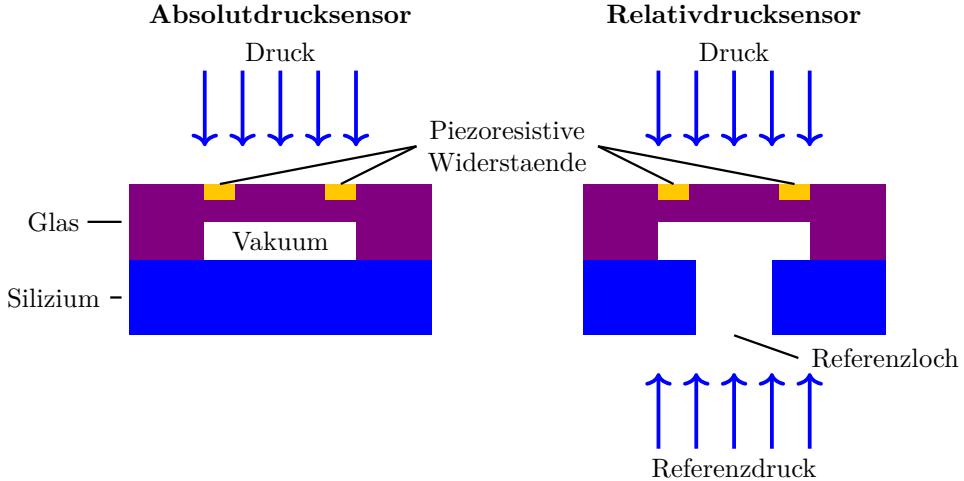


Figure 58.3.: Aufbau des piezoresistiven Sensorchips

Bei piezoresistiven Druckmesszellen sind die Messwiderstände also im Gegensatz zum DMS in die Membran integriert. Bei dieser Technologie entfällt somit das Aufkleben, was eine wichtige Voraussetzung für Alterungs- und Temperaturbeständigkeit sowie Hysteresefreiheit (Hysterese = Nachwirkung des vorherigen Verformungszustands) ist. Das piezoresistive Messprinzip wird auch in dem DHT22-Sensor verwendet, was zu guten und genauen Messwerten führt [Dru19].

58.2. Temperatur

Die Temperaturmessung kann auf unterschiedliche Weise erfolgen. Die verbreitetste ist das Flüssigkeitsthermometer, welches aus einem Vorratsgefäß und einem angeschlossenen Steigrohr, auch Kapillare genannt, besteht. Das Vorratsgefäß ist mit einer thermometrischen Flüssigkeit gefüllt, welche sich bei steigender Temperatur ausdehnt und das Steigrohr füllt.

Als alternative Möglichkeiten werden NTC- und PTC-Thermistoren verwendet. Auch genannt Heißleiter und Kaltleiter. Der NTC-Thermistor ist ein variabler elektrischer, wärmeempfindlicher Widerstand, dessen Wert sich mit der Temperatur reproduzierbar ändert. Die Widerstände sind nicht linear und der Widerstand nimmt bei steigender Temperatur ab. Die Art und Weise, wie der Widerstand abnimmt, hängt mit einer Konstante zusammen, welche in der Elektroindustrie als Beta bekannt ist. Beta wird in Kelvin gemessen. Ein NTC ist also ein elektronisches Bauteil, welches den Widerstand temperaturabhängig verändert. Diese elektrischen Signale können dann Aufschluss über die Temperatur geben. Der PCT funktioniert vom Prinzip her gleich, jedoch steigt der Widerstand mit zunehmender Temperatur.

Leider wurde auf keiner Seite und in keinem Datenblatt veröffentlicht, um welches Prinzip es sich beim DHT22-Sensor handelt. Jedoch wurde nach der Recherche angenommen, dass ein NTC-Thermistor verbaut ist. Dies wird dadurch begründet, dass keine Flüssigkeit im Sensor ist, und somit die erste Möglichkeit entfällt. Die Entscheidung gegen den PTC-Thermistor wurde getroffen, da PTC-Thermistoren Verhältnismäßig sehr teuer sind, und da der Sensor gerade einmal 7€ kostet, es sehr unwahrscheinlich ist, dass das Prinzip des PTC verwendet wurde [Ame24].

58.3. Feuchtigkeit

Es gibt vier Arten von Feuchtigkeitssensoren, die auf den Funktionsprinzipien und Sensormaterialien basieren: Kapazitiv, resistiv, Wärmeleitfähigkeit, psychometrisch.

58.3.1. Kapazitive Feuchtigkeitssensoren

Kapazitive Feuchtigkeitssensoren gehören zu den am häufigsten verwendeten Arten. Dieses Prinzip ist auch im DHT22-Sensor verbaut. Die Sensoren funktionieren, indem sie Änderungen der Dielektrizitätskonstante eines Materials als Reaktion auf Änderung der Luftfeuchtigkeit messen. Die Dielektrizitätskonstante misst die Fähigkeit eines Materials, elektrische Energie in einem elektrischen Feld zu speichern.

Sie bestehen aus zwei Elektronen, wovon eins mit hygroskopischen Material beschichtet ist, dass Wasserdampf aus der Luft absorbiert. Wenn das hygroskopische Material Wasserdampf aufnimmt, führt dies zu einer Änderung der Dielektrizitätskonstante zwischen den beiden Elektroden, die vom Sensor gemessen wird.

58.3.2. Resistive Feuchtigkeitssensoren

Resistive Feuchtigkeitssensoren, auch Hygrometer genannt, messen Änderungen im elektrischen Widerstand eines Materials als Reaktion auf der Änderung der Luftfeuchtigkeit. Die gebräuchlichste Art von Widerstandsfeuchtigkeitssensoren ist der polymerbasierte Sensor, welcher aus einem leitfähigen Polymerfilm besteht, der seinen Widerstand ändert, wenn er Wasserdampf ausgesetzt wird.

Wenn der Polymerfilm Wasserdampf aus der Luft aufnimmt, schwollt er an und wird leitfähiger, wodurch der durch den Sensor fließende elektrische Strom zunimmt. Die Widerstandsänderung ist proportional zur Wasserdampfmenge in der Luft und kann zur Bestimmung der Luftfeuchtigkeit gemessen werden. Dieser Sensortyp ist auch in dem DHT22-Sensor verbaut und basiert auf diesem System.

58.3.3. Wärmeleitfähigkeits Feuchtigkeitssensor

Wärmeleitfähigkeits-Feuchtigkeitssensoren messen die Wärmeleitfähigkeit eines Gasgemisches als Reaktion auf Änderungen der Luftfeuchtigkeit. Sie bestehen aus einem beheizten Sensorelement und einem Temperatursensor, der den Temperaturunterschied zwischen den Beiden misst. Wenn das Sensorelement Wasserdampf absorbiert, verringert es seine Wärmeleitfähigkeit, was zu einer Temperaturänderung führt, die der Temperatursensor messen kann. Diese Temperaturänderung ist proportional zur Menge an Wasserdampf in der Luft und kann zur Bestimmung der Luftfeuchtigkeit verwendet werden.

58.3.4. Psychrometrische Feuchtigkeitssensoren

Psychrometrische Feuchtigkeitssensoren, auch Taupunktspiegelsensoren genannt, messen die Temperatur, bei der Wasserdampf auf einer Oberfläche kondensiert. Sie bestehen aus einem gekühlten Spiegel, bis sich auf seiner Oberfläche Tau oder Reif bildet. Die Temperatur, bei der diese Kondensation auftritt, ist eine Funktion der relativen Luftfeuchtigkeit der Luft um den Spiegel [Hen24].

58.4. Sensor DHT22

Der extern angeschlossene Sensor DHT22 ist für die Temperatur- und Luftfeuchtigkeitsmessung verantwortlich. Für den Betrieb wird die Bibliothek [DHT.h](#) benötigt, welche von Adafruit angeboten wird. Sie ermöglicht das starten und stoppen des Sensors, sowie

das Auslesen der Temperatur und der Luftfeuchtigkeit. Die Bibliothek ist geeignet für die Sensoren vom Type DHT11 und DHT22.

58.5. Schaltplan

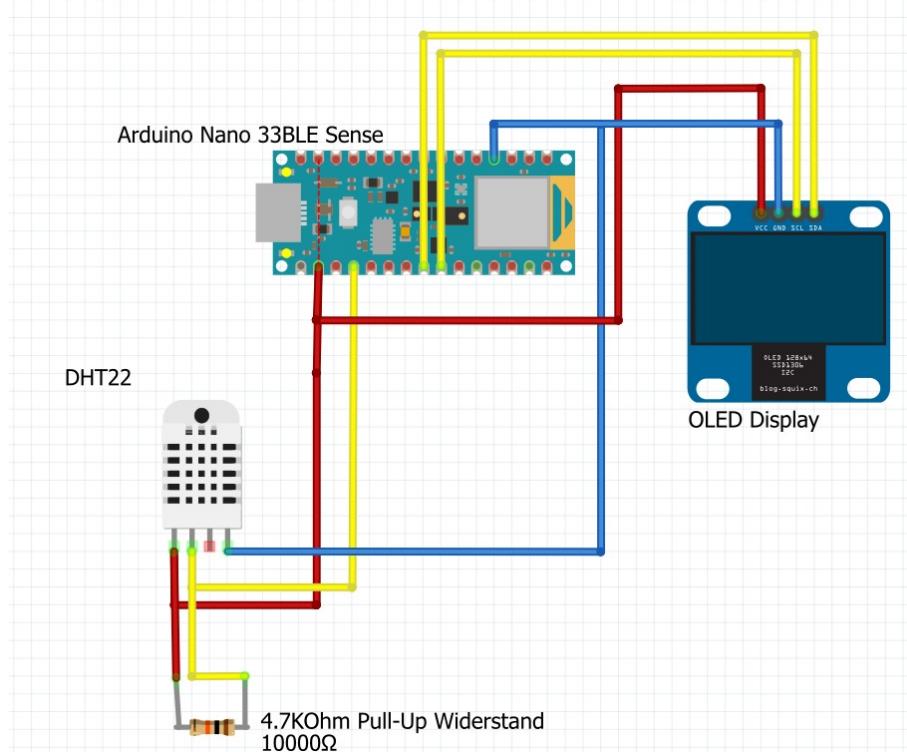


Figure 58.4.: Schaltplan des Raumklimamessgeräts

58.6. Testen des Sensors DHT22

Um die Funktion des LPS22HB-Sensors zu gewährleisten, wurde dieser an den Arduino angeschlossen und mit dem folgenden Programm getestet. Die gemessenen Werte werden seriell an den Computer weitergegeben und im Serial Monitor in der Arduino IDE angezeigt. Dieses Programm wurde zudem zur Kalibrierung verwendet.

```

1 // Einbinden von verwendeten Bibliotheken
2 #include <DHT.h>
3
4 // Einstellen des DHT22-Sensors
5 #define DHTPIN D11
6 #define DHTTYPE DHT22
7 DHT dht(DHTPIN, DHTTYPE);
8
9 void setup() {
10     // Einschalten des Sensors
11     pinMode(DHTPIN, INPUT);
12     dht.begin();
13 }
14

```

```

15 void loop() {
16     // Einstellen der Verzögerung zwischen den
17     // Messvorgaengen
18     delay(5000);
19
20     // Auslesen der Messwerte durch den Sensor
21     float humidity = dht.readHumidity();
22     float temp = dht.readTemperature();
23
24     // Messwerte seriell ausgeben
25     Serial.print("Luftfeuchte:_");
26     Serial.println(humidity);
27     Serial.println();
28
29     Serial.print("Temperatur:_");
30     Serial.println(temp);
31     Serial.println();
31 }
```

58.7. Kalibrierung

58.7.1. Opus20 THI

Das Opus 20 THI ist ein hochpräziser LAN-Datenlogger, der speziell zur Überwachung des Gebäudeklimas und zur Kontrolle klimasensitiver Produktionsprozesse entwickelt wurde. Es wird in EDV-Rechenzentren, Schaltschränken, Windturbinen, Lagerräumen und Museen eingesetzt. Das Gerät zeichnet sich durch seine Genauigkeit, Zuverlässigkeit und einfache Bedienung aus, siehe 58.5.



Figure 58.5.: Opus 20 THI

58.7.2. Funktionen

Messparameter

- Temperatur
- Relative Feuchte

Messtechnologie

- Temperatur: NTC
- Relative Feuchte: Kapazitive Messung

Produkt-Highlights

- LAN-Datenlogger mit eingebauten Sensoren
- Höchste Messgenauigkeit
- Firmware online aktualisierbar
- Verschiedene Stromversorgungsoptionen (USB, Batterien, PoE)
- Inklusive Auswertungssoftware SmartGraph3

58.7.3. Schnittstellen

- USB (inkl. Kabel und SmartGraph3 Software)
- LAN

58.7.4. Technische Daten

Allgemeine Daten

- Abmessungen: 166 x 78 x 32 mm
- Gewicht: ca. 250 g
- Gehäusematerial: Kunststoff
- Datenspeicher: 16 MB (3.200.000 Messwerte)
- Betriebsdauer mit Batterie: > 1 Jahr
- LC-Display: 90 x 64 mm
- Abtastintervalle: 10/30s, 1/10/12/15/30min, 1/3/6/12/24h
- Speicherintervalle: 1/10/12/15/30min, 1/3/6/12/24h
- Stromversorgung: 4 x LR6 AA Mignon, USB
- Betriebstemperaturbereich: -20 bis 50 °C
- Relative Feuchte: 0 bis 100 r.F., < 20g/m³ (nicht kondensierend)
- Maximale Höhe: 10.000m über Normal Null

Temperaturmessung

- Messprinzip: NTC
- Messbereich: -20 bis 50 °C
- Einheit: °C
- Genauigkeit: ±0,3 °C (0 bis 40 °C), sonst ±0,5 °C
- Auflösung: 0,1 °C

Feuchtemessung

- Messprinzip: Kapazitiv
- Messbereich: 0 bis 100 r.F.
- Einheit: r.F.
- Genauigkeit: ±2 r.F.
- Auflösung: 0,1 r.F. [Luf18]

58.7.5. Anwendung

Das Opus 20 THI ist ideal für die Überwachung und Aufzeichnung von Klimadaten in kritischen Umgebungen, um optimale Bedingungen zu gewährleisten und Abweichungen frühzeitig zu erkennen. Die hohe Genauigkeit und die umfangreichen Schnittstellenoptionen machen es zu einem vielseitigen Werkzeug in der Klimakontrolle. Dies haben wir als Vorteil gesehen, um Referenzwerte für unser Gerät zu erlangen, um Anhand dessen unser Programm anpassen zu können, um Korrekte Werte messen zu können. Unsere Kalibrierung erfolgte also auf Basis des hier genannten Opus 20 THI Datenloggers [DS24].

58.7.6. Kalibrierung

Für die Kalibrierung des Raumklimamessgeräts wurden vorerst die Testprogramme für den DHT22- und den LPS22HB-Sensor verwendet. Es wurde in Abständen von ca. einer Stunde die Werte des OPUS20 THI und die des Raumklimamessgeräts abgelesen und in einer Tabelle gesammelt. Die Messergebnisse befinden sich in Documents/Messergebnisse_OPUS.xlsx. Bei einer ausreichenden Anzahl an Werten wurde eine mittlere Abweichung der Messwerte berechnet und ausgegeben. Zudem wurden für die Messreihen Graphen erstellt. Es ist zu erkennen, dass in den vorliegenden Messbereichen eine relativ gleichbleibende Abweichung besteht, daher wurde die berechnete mittlere Abweichung einfach mit den gemessenen Werten der Sensoren addiert. Für die Messung der Temperatur wurde zusätzlich ein Messgerät von Philips verwendet, welches aber nicht weiter berücksichtigt wurde.

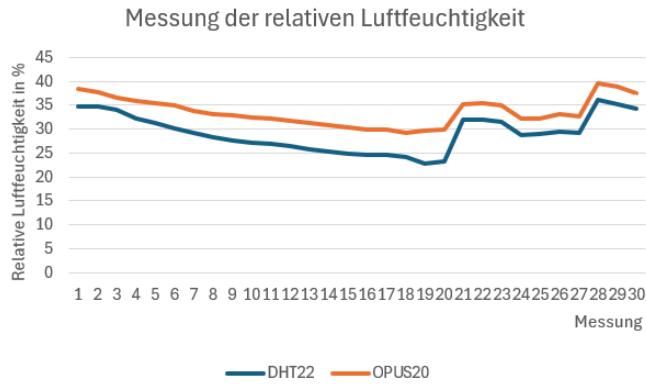


Figure 58.6.: Graph für Abweichung der Luftfeuchtigkeit

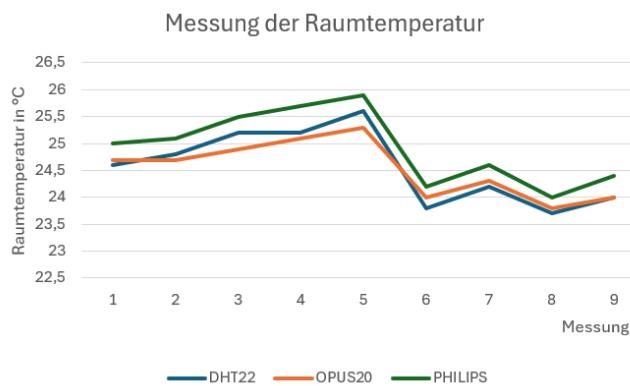


Figure 58.7.: Graph für Abweichung der Temperatur

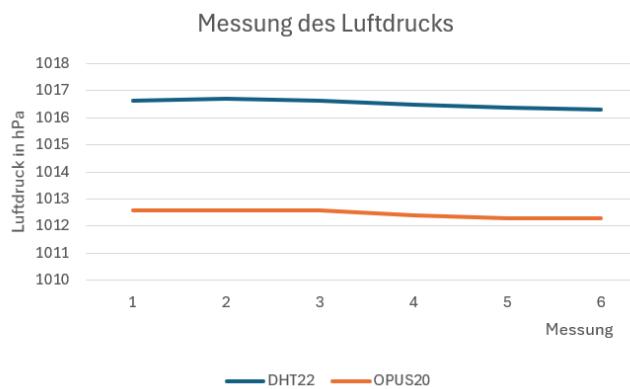


Figure 58.8.: Graph für Abweichung des Luftdrucks

DC 4.5V 0.11A 10g Fähigkeit 3mm Hub Ziehen Typ Magnet Elektromagnet
 Produktnname DC Magnet Elektromagnet Nennspannung DC 4.5V Typ Ziehen Nennstrom
 0.11A Leistung 0.5W Fähigkeit and Hub 10g/3mm
 Angetriebene Rate 50Körpergröße 20 x 12 x 11mm / 0,78 "x 0,47" x 0,43 "(L * B* D)
 Stößel Stab Größe 4 x 8mm / 0,16" x 0,3" (D*L)

Kabellänge 10cm/3.9" Material Metall, Elektronische Teile Hauptfarbe Als Bild Zeigen Gewicht 15g Lieferumfang 1 x DC Solenoid Elektromagnet Beschreibung:

DC Solemoid Elektromagneten hauptsächlich in Verkaufautomaten, Transportausrüstung, Büroausstattung Haushaltsgerät, mechanisch usw. verwendet Wenn erregt ist, mache die Arbeit durch das Ziehen des Kolbens.

Spezifikation Marke Uxcell Herstellernummer nicht zutreffend upc 702105986189

Arduino Nano 33 BLE Sense Lite hat keine HTS221-Temperatur- und Feuchtigkeitssensoren, sondern nur den LPS22HB-Drucksensor, der allerdings auch die Temperatur messen kann

Arduino Nano 33 BLE Sense Rev2 hat einen HTS221-Temperatur- und Feuchtigkeitssensor und den LPS22HB-Drucksensor, der allerdings auch die Temperatur messen kann. Unterschiede:

Inertiale Messeinheit (IMU): Rev 1: Verfügt über eine einzelne 9-Achsen-IMU. Rev 2: Hat eine Kombination aus zwei IMUs: eine 6-Achsen-IMU (BMI270) und eine 3-Achsen-IMU (BMM150), was die Möglichkeiten zur Bewegungserkennung erweitert.¹ Design und Zugänglichkeit: Rev 2: Integriert neue Pads und Testpunkte für USB, SWDIO und SWCLK, was den Zugang zu diesen wichtigen Punkten auf der Platine erleichtert¹. Rev 1: Hat diese zusätzlichen Pads und Testpunkte nicht. Stromversorgung:

Rev 2: Verwendet den MP2322 als Stromversorgungskomponente, was die Leistung verbessert¹.

Rev 1: Nutzt eine andere Stromversorgungskomponente. Der Arduino Nano 33 BLE Sense Rev 1 verwendet den MPS MP2144 als Stromversorgungskomponente

59. Geschwindigkeitssensor LM393

59.1. Allgemeine Beschreibung des Sensors

Der LM393 ist ein einfacher und kostengünstiger Geschwindigkeitssensor, der nach dem Prinzip der optischen Lichtschranke funktioniert. Er besteht aus einem LM393-Komparator, einem Infrarot-Sender-Empfänger-Paar und einem digitalen Ausgang. Der Sensor kann in Kombination mit einer auf einer rotierenden Welle angebrachten Lochscheibe verwendet werden.

59.2. Spezifische Beschreibung des Sensors

Der LM393 Geschwindigkeitssensor besteht aus folgenden Komponenten:

- Infrarot-LED (Sender)
- Fototransistor (Empfänger)
- Lichtschranken-Nut
- LM393 Komparator
- Status-LED
- PCB mit Pins
- Lochscheibe

Wie diese Komponenten die Funktion des Sensors ermöglichen, wird im Folgenden erklärt.

59.2.1. Funktionsweise der Lichtschranke

Die Infrarot-LED und der Fototransistor bilden eine optoelektronische Lichtschranke und sind sich gegenüber auf dem Sensor-PCB angebracht. Die beiden Komponenten sind durch die Lichtschranken-Nut getrennt, welche 5 mm breit ist. In dieser Nut rotiert die Lochscheibe, welche auf der Motorwelle montiert ist und mit 20 Löchern ausgestattet ist. Die Infrarot-LED dient als Sender der Lichtschranke und emittiert kontinuierlich Licht im infraroten Bereich (940 nm) in Richtung des Empfängers. Der Fototransistor ist der Empfänger. Dieser Fototransistor besteht aus einem Halbleitermaterial. Trifft das Infrarotlicht des Senders, durch ein Loch der Lochscheibe auf den Empfänger, so werden dort Elektronen-Loch-Paare erzeugt. Dadurch steigt der Stromfluss durch den Fototransistor. Wird das Infrarotlicht durch die Lochscheibe blockiert sinkt der Strom im Fototransistor.

Die hier verwendete Lochscheibe weist 20 Löcher auf, somit werden pro Umdrehung 20 Signale erzeugt. Da der Sensor aber auf Signaländerungen reagiert werden pro Loch zwei Impulse erzeugt, es werden also 40 Impulse pro Umdrehung gezählt.

59.2.2. Signalverarbeitung mit dem LM393 Komparator

Der LM393 ist ein Komparator, der zwei Analoge Spannungen vergleicht. Eingang A wird mit einer Referenzspannung beschaltet. Eingang B ist mit dem Phototransistor verbunden. Wenn Infrarotlicht auf dem Empfänger auft trifft (durch Loch der Lochscheibe) ändert sich die Spannung am Eingang B. Wenn die Spannung im Phototransistor durch den Lichteinfall höher wird als die Referenzspannung, dann schaltet der Komparator den digitalen Ausgang D0 auf LOW. Wenn der Lichteinfall blockiert ist und somit die Spannung im Phototransistor niedriger ist als die Referenzspannung, dann schaltet der Komparator den digitalen Ausgang D0 auf HIGH. Der LM393 Komparator wird also dazu genutzt kleine Änderungen im Analogsignal zu verstärken und in ein klares digitales Rechtecksignal umzuwandeln.

Am digitalen Ausgang D0 liegt das durch den LM393 erzeugte Signal an. Dieses kann direkt mit einem Mikrocontroller, in diesem Fall einem Arduino Nano 33 BLE Sense verbunden und ausgewertet werden.

59.2.3. Anschluss des Sensors mit dem Arduino Nano 33 BLE Sense

In Abbildung xx ist gezeigt wie der Sensor korrekt an den Arduino Nano 33 BLE Sense angeschlossen wird:

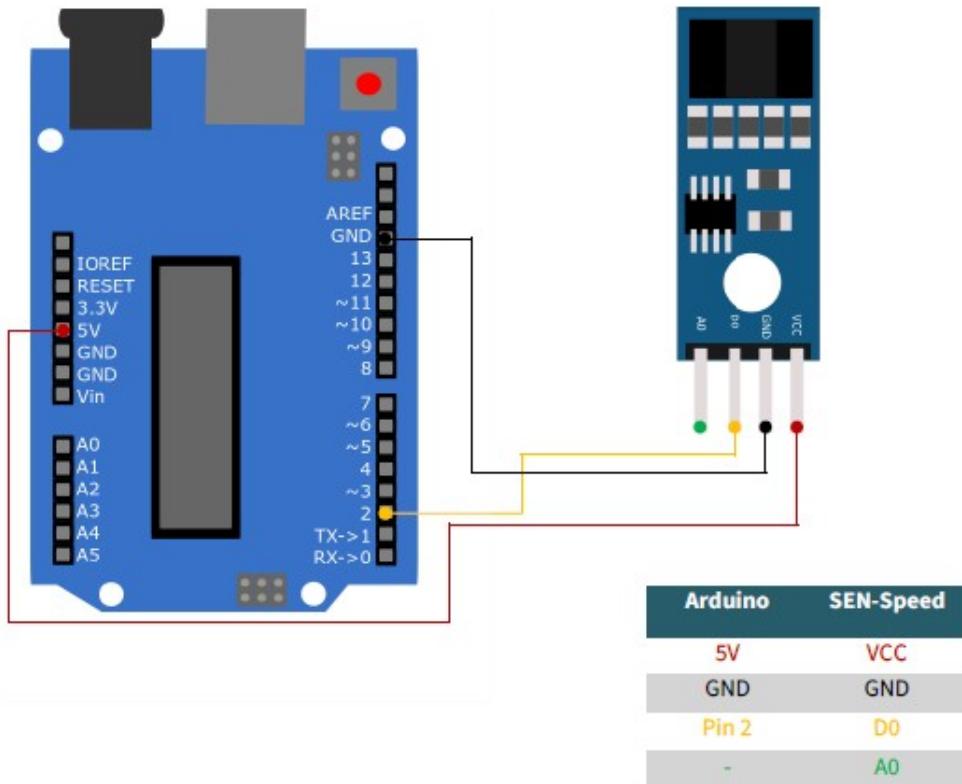


Figure 59.1.: Anschluss des Sensors ans den Arduino Nano 33 BLE Sense

59.3. Spezifikationen

Spezifikation	Wert
Modell	Speedsensor LM393 mit Lochscheibe
Versorgungsspannung	3,3 V - 5 V DC
Signalausgabe	Digital
Nutbreite (Lichtschranke)	5 mm
PCB-Abmessungen	32 mm x 14 mm
Lochscheiben-Durchmesser	25,5 mm (20 Löcher)
Besonderheiten	LED-Anzeige, Montageloch

Table 59.1.: Spezifikationen des Speedsensor LM393

59.4. Bibliothek

Für die Verwendung des Sensors mit dem Arduino Nano 33 BLE Sense wird die TimerOne-Bibliothek benötigt. Die TimerOne-Bibliothek wird verwendet, um die Zeitmessung zu ermöglichen, um aus den erzeugten Impulsen des Sensors die Drehzahl zu berechnen.

Welche Funktionen werden verwendet + Erklärung

59.4.1. Installation

Im Folgenden wird erklärt, wie die TimerOne Bibliothek installiert wird.

1. Öffnen der Arduino IDE
2. Klick auf Sketch, Bibliothek einbinden, Bibliotheken verwalten (Abbildung xx)
3. In der Suchleiste TimerOne eingeben
4. Klick auf Installieren (Version aus Abbildung xx entnehmen)

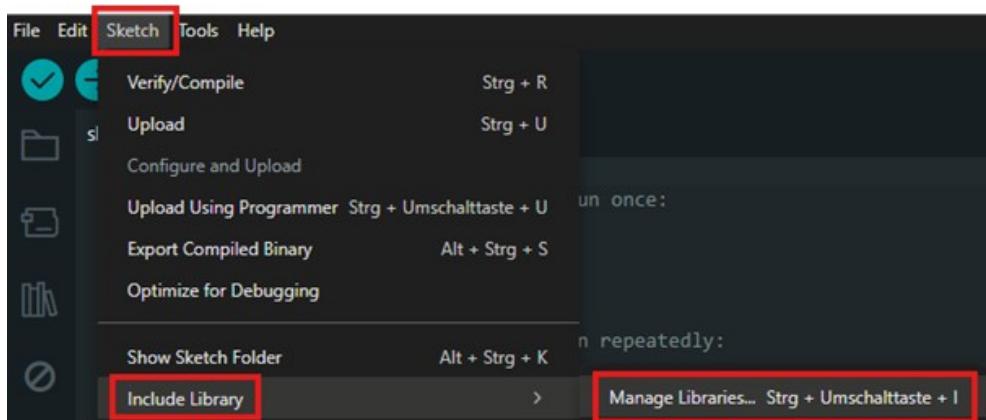


Figure 59.2.: Aufrufen der Bibliotheken in der Arduino IDE



Figure 59.3.: Installation der TimerOne Bibliothek

59.4.2. Code-Beispiel

Im Folgenden ist ein Code-Beispiel gegeben, in dem die verwendeten Funktionen erklärt werden. Für weitere Funktionen der Bibliothek kann man über Datei, Beispiele, TimerOne auf verschiedene Beispiel-Codes zugreifen (Abbildung xx).

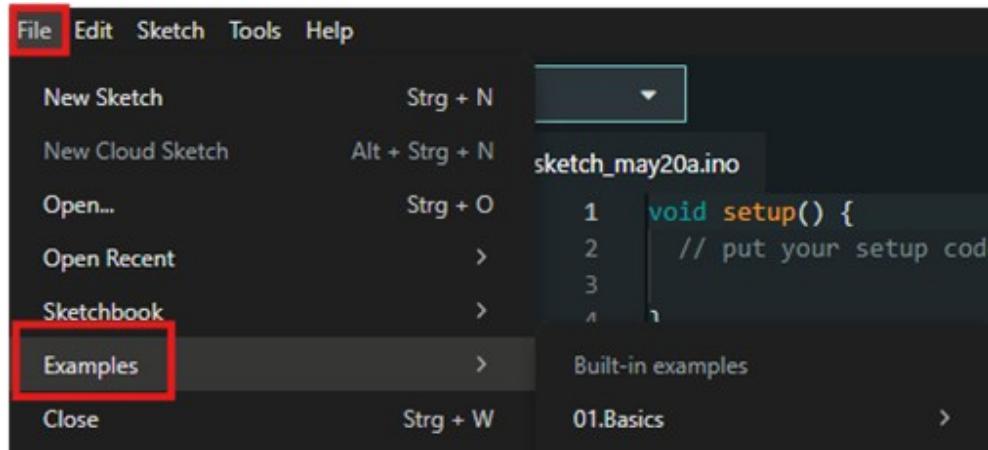


Figure 59.4.: Aufrufen der Beispiele für die TimerOne Bibliothek

59.5. Kalibrierung

Für den LM393 Geschwindigkeitssensor mit Lochscheibe ist keine spezielle Kalibrierung notwendig. Optional kann aber eine empirische Kalibrierung durchgeführt werden, indem man die Umdrehungen mit einem bekannten Drehzahlmesser vergleicht. Bei der Positionierung des Sensors muss darauf geachtet werden, dass die Lichtschranke präzise auf die Mitte der Lochscheibe ausgerichtet ist. Die Lochscheibe muss plan und fest

an der Motorachse montiert werden. Außerdem muss die Variable „wheel“ im Code an die Anzahl der Löcher in der Lochscheibe angepasst werden. Da der Sensor auf Flankenwechsel reagiert, muss die Variable auf die doppelte Anzahl der Löcher gesetzt werden. Die Lochscheibe, welche für den Demonstrator verwendet wird, hat 20 Löcher, somit muss „wheel“ auf 40 gesetzt werden, wie in Abbildung xx unter Abschnitt xx zu sehen.

59.6. Einfaches Beispiel mit Code

Das Beispiel geht davon aus, dass die Lochscheibe auf einer sich rotierenden Welle befestigt ist. Mit dem folgenden Code wird die Drehzahl der Welle alle 5 Sekunden im seriellen Monitor ausgegeben.

```

// sen-speed Demo
// Der Code misst ueber einen Zeitraum (5 Sekunden voreingestellt)
// die Umdrehungen der Encoder-Scheibe, rechnet diese dann auf Umdrehungen
// pro Minute um und gibt diese ueber die serielle Schittstelle aus.

// Import einer Bibliothek
#include "TimerOne.h"
#define pin 2

// Benoetigte Variablen
int interval, wheel, counter;
unsigned long previousMicros, usInterval, calc;

void setup()
{
    counter = 0; // counter auf 0 setzen
    interval = 5; // 5 Sekunden Intervall
    wheel = 20; // Loecher in der Encoder-Scheibe

    calc = 60 / interval; // Intervall auf 1 Minute hoch rechnen
    usInterval = interval * 1000000; // Intervallzeit fuer den Timer in
                                    // Mikrosekunden umrechnen
    wheel = wheel * 2; // Anzahl der Loecher in der Encoder-Scheibe mit 2
                        // multiplizieren, da der Interrupt bei jeder
                        // Aenderung des Signals ausgefuehrt wird

    pinMode(pin, INPUT); // Setzen des analogen Pins als Input
    Timer1.initialize(usInterval); // Timer initialisieren auf dem Intervall
    attachInterrupt(digitalPinToInterrupt(pin), count, CHANGE);
    // fuehrt count aus, wenn sich das Signal am Pin 2 aendert

    Timer1.attachInterrupt(output); // fuehrt nach Intervall output aus
    Serial.begin(9600); // starten der seriellen Schnittstelle mit 9600 Baud
}

// Zaeht Loecher der Encoder-Scheibe (mit Filter)
void count(){
    if (micros() - previousMicros >= 700) {
        counter++;
        previousMicros = micros();
    }
}

// Ausgabe im seriellen Monitor
void output(){
    Timer1.detachInterrupt(); // Unterbricht Timer
    Serial.print("Drehzahl pro Minute: ");
    int speed = ((counter)*calc) / wheel;
    // Berechnung der Umdrehungen pro Minute

    Serial.println(speed);
    counter = 0; // zuruecksetzen des Zaehlers
    Timer1.attachInterrupt(output); // startet Timer erneut
}

void loop(){
    // keine loop notwendig
}

```

Figure 59.5.: wird noch in Code umgewandelt!

59.7. Tests

Um sicherzugehen, dass der Sensor zuverlässig funktioniert, wird empfohlen folgende Tests anhand des einfachen Beispiels mit Code durchzuführen:

- Lichtschrankenprüfung: Drehe die Lochscheibe manuell und beobachte ob die Status-LED bei jedem Loch kurz aufblinkt.
- Signaltest: Starte den Beispielcode und überprüfe ob im eingestellten Zeitintervall Drehzahlen im seriellen Monitor ausgegeben werden.
- Stabilitätstest: Führe längere Messungen durch um sicherzustellen, dass der Sensor stabile Werte liefert.
- Reaktion auf Lichtverhältnisse: Teste den Sensor bei verschiedenen Lichtverhältnissen um externe Störungen der Lichtschranke zu erkennen.

59.8. Weiterführende Literatur

einfügen

60. Sensor XY

Introduction

WS:cite books,
applications, board

60.1. General

General description

cite books

60.2. Specific Sensor/Actor

cite board

60.3. Specification

- Cite the data sheet
- Extract te information from the data sheet
- Circuit Diagram

60.4. Library

60.4.1. Description

60.4.2. Installation

- data types
- data structure
- data quantity
- data quality

60.4.3. Functions

60.5. Example

In this section, a simple example is described in detail, which demonstrates how the library supports the sensor/actor.

60.5.1. Manual**60.5.2. Inside the Example****60.5.3. Code****60.5.4. Files****60.6. Calibration**

cite method

60.7. Simple Code**60.8. Simple Application****60.9. Tests****60.9.1. Simple Function Test****60.9.2. Test all Functions****60.10. Further Readings**

61. Actor XY

Introduction

WS:cite books,
applications, board

61.1. General

General description

cite books

61.2. Specific Sensor/Actor

cite board

61.3. Specification

- Cite the data sheet
- Extract te information from the data sheet
- Circuit Diagram

61.4. Library

61.4.1. Description

61.4.2. Installation

- data types
- data structure
- data quantity
- data quality

61.4.3. Functions

61.5. Example

In this section, a simple example is described in detail, which demonstrates how the library supports the sensor/actor.

61.5.1. Manual**61.5.2. Inside the Example****61.5.3. Code****61.5.4. Files****61.6. Calibration**

cite method

61.7. Simple Code**61.8. Simple Application****61.9. Tests****61.9.1. Simple Function Test****61.9.2. Test all Functions****61.10. Further Readings**

62. Getting Started

62.1. Download and Install the nRF Connect App on Your Mobile

To download and install the **nRF Connect** app on Android or iOS devices, follow these simple steps:

62.1.1. For Android

- Open the **Google Play Store** on your Android device.
- In the search bar, type "*nRF Connect for Mobile*".
- Select the app by **Nordic Semiconductor ASA**.
- Tap the button “**Install**” to download and install the app.
- Once installed, you can open the app from your home screen or app drawer.

62.1.2. For iOS

- Open the **App Store** on your iOS device.
- In the search bar, type “*nRF Connect for Mobile*”.
- Select the app by **Nordic Semiconductor ASA**.
- Tap the button “**Get**” to download and install the app.
- Once installed, you can open the app from your home screen or app drawer.

The **nRF Connect** app allows you to interact with **Bluetooth Low Energy (BLE)** devices, such as the Arduino Nano 33 BLE, and monitor their services and data.

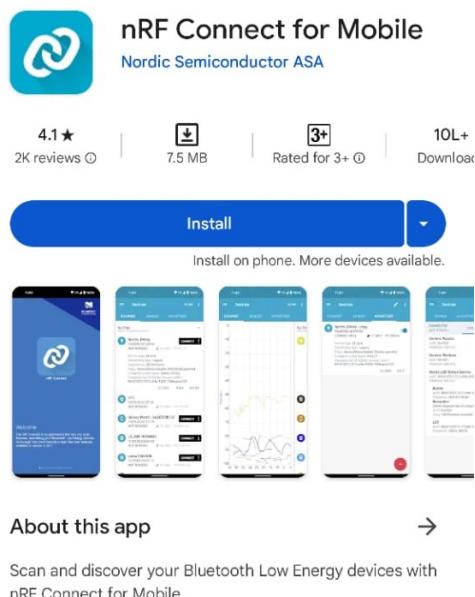


Figure 62.1.: NRF Connect Application

62.2. Power ON the Arduino Nano 33 BLE Sense Lite

To power on the Arduino Nano 33 BLE, connect it to your computer or a USB power adapter using the micro-USB cable provided in the box.

62.2.1. Connect to Arduino Nano 33 BLE Sense Lite through NRFconnect

62.2.2. For Android

- **Open the NRFconnect App:** Open the app and start scanning for nearby BLE devices.
- **Find The Arduino:** You should see a device named “MyProctName” in the list of available devices.
- **Connect:** Tap on the ArduinoNano33 device to connect. The Arduino Nano will now show “Connected” in the Mobile application.

62.2.3. For iOS

- **Open the NRFconnect App:** Open the app and start scanning for nearby BLE devices.
- **Find Your Arduino:** Look for “MyProctName” in the list of detected BLE devices.
- **Connect:** Tap the device name to connect. You will be able to see connection status in the Mobile application.

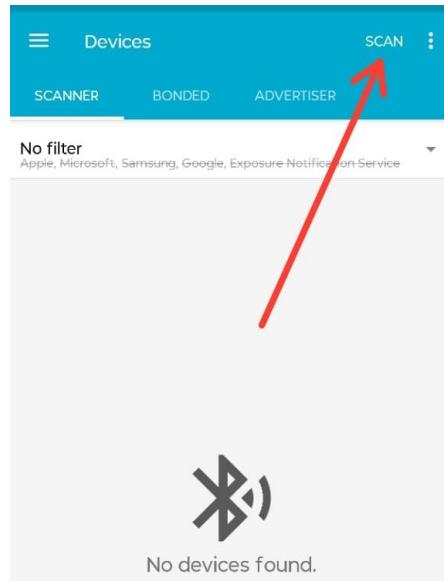


Figure 62.2.: Landing page

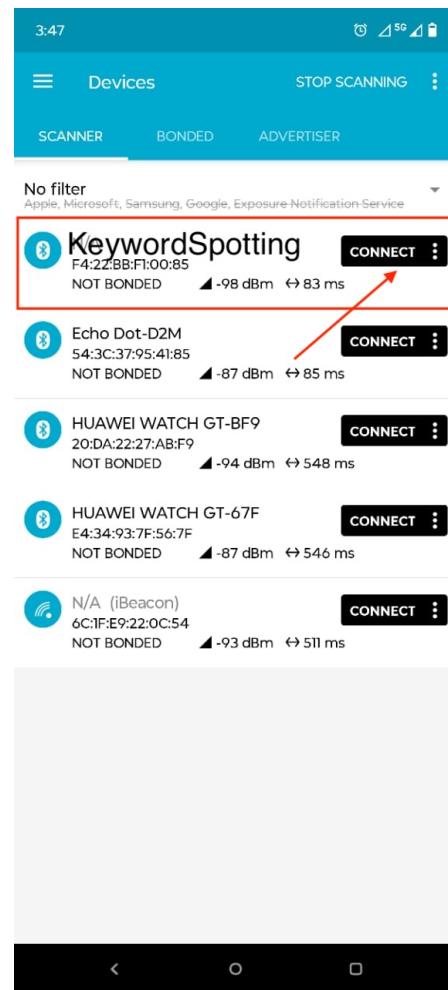


Figure 62.3.: List of devices

Part IV.

Applikation <Title>

63. Application

64. Conclusion XY

65. To DoX Y

Part V.

Templates

66. Sensor

Introduction

WS:cite books,
applications, board

66.1. General

General description

cite books

66.2. Specific Sensor

cite board

66.3. Specification

- cite data sheet

- Circuit Diagram

66.4. Library

66.4.1. Description

66.4.2. Installation

66.4.3. Functions

66.4.4. Example's Manual

66.4.5. Inside the Example

66.4.6. Example's Code

66.4.7. Example's Files

66.5. Calibration

cite method

66.6. Simple Code

66.7. Simple Application

66.8. Tests

66.8.1. Simple Function Test

66.8.2. Test all Functions

66.9. Simple Application

66.10. Further Readings

67. Package Example

67.1. Introduction

67.2. Description

67.3. Installation

`pip packageExample`

67.4. Example - Manual

67.5. Example

67.6. Example - Code

67.7. Example - Files

`PackageExample.py`

67.8. Further Reading

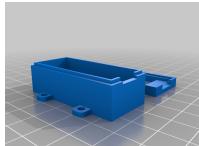
References

- [Aba+16] M. Abadi et al. “TensorFlow: A System for Large-Scale Machine Learning”. In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA: USENIX Association, Nov. 2016, pp. 265–283. ISBN: 978-1-931971-33-1. URL: <https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi>.

Part VI.

Anhang

68. Materialliste

Anzahl	Bezeichnung	Link	Preis
1	ARD NANO 33BLE H - Arduino Nano 33 BLE with header	www.reichelt.de - ARD_NANO_33BLE_H	29,50 €
			
1	ARD NANO 33BSH2 - Arduino Nano 33 BLE Sense Rev.2 with Header	www.reichelt.de - ARD_NANO_33BSH2	44,80 €
			
1	ARD KIT TINYML - Arduino Lern-Kit Tiny Machine	www.reichelt.de - ARD_Kit_TinyML	49,24 €
			
1	Arducam B0226 Lens Calibration Tool	www.welectron.com - Arducam_B0226_Lens_Calibration_Tool	11,90 €
			
1	3D Printed Case - A protective case for Arduino Nano 33 BLE Sense, customizable and available at Thingiverse	Thingiverse	./
			

Stand: 03.02.2025

Part VII.

Hints - Examples for LaTeX - Read and Use

69. Hints

Please, use **biber.exe**.

If you want to create a symbol directory, call makeindex:

makeindex %.nlo -s nomencl.ist

69.1. Allgemeine mathematische Beschreibung Bézier-Kurve

Bézier curves are formed with the help of Bernstein polynomials. If at least two points and two tangents are known, control points can be determined with which a control polygon is formed. The course of the curve is oriented to this control polygon. The number of control points depends on the degree of the Bézier curve. A Bézier curve of degree n has $n+1$ control points. The Bézier curve is calculated using the De Casteljau algorithm.

Definition. In the interval $[0; 1]$ the **Bernstein polynomial of n^{th} degree** is defined by:

$$b_{i,n}(\lambda) = \binom{n}{i} (1-\lambda)^{n-i} \lambda^i, \quad \lambda \in [0, 1], \quad i = 0, \dots, n \quad (69.1)$$

Definition. The binomial coefficient is defined by:

$$\binom{n}{i} = \frac{n!}{i!(n-i)!}, \quad i = 0, \dots, n \quad (69.2)$$

Here the Bernstein polynomials $b_{i,n}$ form a basis of the vector space $\mathbb{P}^n(I)$ of polynomials of degree at most n over I . Thus every polynomial of degree at most n can be written uniquely as a linear combination. [Far02]

Beispiel. Determination of Bernstein polynomials for $n = 3$ holds:

$$b_{3,0}(\lambda) = \binom{3}{0} (1-\lambda)^{3-0} \lambda^0 = (1-\lambda)^3$$

$$b_{3,1}(\lambda) = \binom{3}{1} (1-\lambda)^{3-1} \lambda^1 = 3\lambda(1-\lambda)^2$$

$$b_{3,2}(\lambda) = \binom{3}{2} (1-\lambda)^{3-2} \lambda^2 = 3\lambda^2(1-\lambda)$$

$$b_{3,3}(\lambda) = \binom{3}{3} (1-\lambda)^{3-3} \lambda^3 = \lambda^3$$

$b_{3,i}(\lambda)$ with $i = 0, 1, 2, 3$ are the cubic Bernstein polynomials of degree 3.

Definition. Given are the points

$$Q_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix} \quad \text{mit} \quad Q_i \in \mathbb{R}^2, \quad i = 0, 1, \dots, n$$

A **Bézier curve** is then defined by

$$C(\lambda) = \sum_{i=0}^n Q_i \cdot b_{i,n}(\lambda) \quad (69.3)$$

The points $Q_i, i = 0, \dots, n$ are called **control points**.

The control points of a Bézier curve form the so-called control polygon.

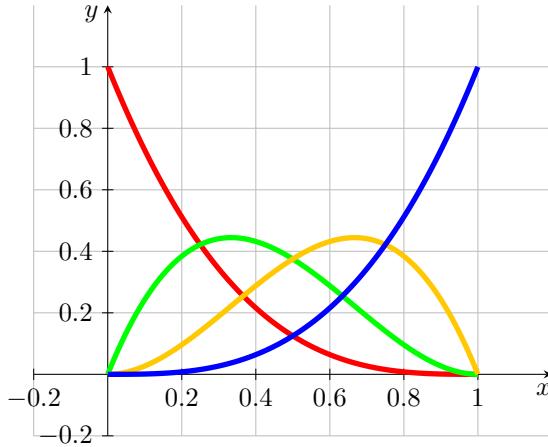


Figure 69.1.: Bernstein polynomial of degree 3

Bemerkung. Let the starting point P_0 and the end point P_1 , as well as the tangents \vec{t}_0 and \vec{t}_1 be given. The tangents are not necessarily normalised.

The control points Q_0, Q_1, Q_2 and Q_3 of the associated Bézier curve are given by the following equations:

$$Q_0 = P_0, \quad Q_1 = P_0 + \lambda_0 \vec{t}_0, \quad Q_2 = P_1 - \lambda_1 \vec{t}_1, \quad Q_3 = P_1 \quad (69.4)$$

[Jak+10]

Beispiel. Given are

$$P_0 = \begin{pmatrix} 2 \\ 4 \end{pmatrix}, \quad \vec{t}_0 = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad \text{und} \quad P_1 = \begin{pmatrix} 6 \\ 1 \end{pmatrix}, \quad \vec{t}_1 = \begin{pmatrix} 1 \\ -2 \end{pmatrix}$$

Then, according to theorem ?? for control points of the associated Bézier curve results:

$$\begin{aligned} Q_0 &= P_0 = \begin{pmatrix} 2 \\ 4 \end{pmatrix}, & Q_1 &= P_0 + \vec{t}_0 = \begin{pmatrix} 2 \\ 4 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 3 \\ 5 \end{pmatrix}, \\ Q_2 &= P_1 - \vec{t}_1 = \begin{pmatrix} 6 \\ 1 \end{pmatrix} - \begin{pmatrix} 1 \\ -2 \end{pmatrix} = \begin{pmatrix} 5 \\ 3 \end{pmatrix}, & Q_3 &= P_1 = \begin{pmatrix} 6 \\ 1 \end{pmatrix} \end{aligned}$$

The Bézier curve and its control points are shown in the figure ??.

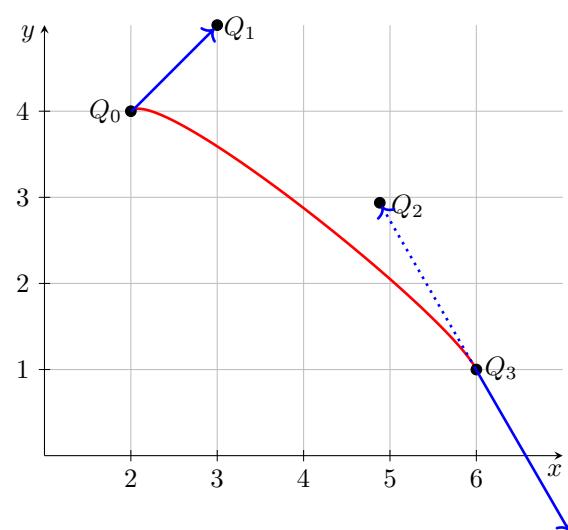


Figure 69.2.: Bézier curve for example 69.1

70. Verrundung mit einem Kreisbogen

70.1. Gleichungen

Blending with an arc is a simple variant of smoothing corners. This variant enables the processing and the generation of files according to DIN 66025. For smoothing, three points P_0 and S and P_1 are always considered, thus a symmetric Hermite problem results. According to theorem ?? it is assumed to be in the standard form $HP(L, \alpha)$.

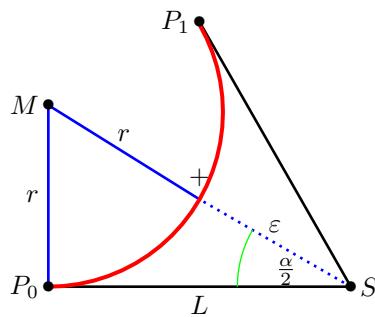


Figure 70.1.: Smoothing a corner with the help of an arc - triangle

The figure ?? represents the situation. The points P_0 , S and P_1 are given. The included angle $\alpha = \angle(P_0; S; P_1)$ as well as the distance $L = \|S - P_0\| = \|P_1 - S\|$ are shown in the graph. The red arc is the desired result. The distance of its centre M to S is then $r + \varepsilon$, where r is the radius of the arc and ε is the given tolerance. Thus we obtain a right triangle P_0SM whose edge lengths are L , $r + \varepsilon$ and r .

According to the definition of the sine and the tangent, it follows:

$$\begin{aligned} \sin\left(\frac{\alpha}{2}\right) &= \frac{L}{r + \varepsilon} & \text{und} & \tan\left(\frac{\alpha}{2}\right) = \frac{L}{r} \\ \Leftrightarrow \quad \varepsilon &= \frac{L}{\sin\left(\frac{\alpha}{2}\right)} - r & \text{und} & \quad r = \cot\left(\frac{\alpha}{2}\right)L \end{aligned}$$

The two equations can be combined to give the following conditions:

$$\varepsilon = L \cdot \frac{1 - \cos\left(\frac{\alpha}{2}\right)}{\sin\left(\frac{\alpha}{2}\right)} \quad \text{bzw.} \quad L = \varepsilon \cdot \frac{\sin\left(\frac{\alpha}{2}\right)}{1 - \cos\left(\frac{\alpha}{2}\right)}$$

This gives the equation for the radius:

$$r = L \cdot \cot\left(\frac{\alpha}{2}\right) = L \cdot \sqrt{\frac{1 - \cos(\alpha)}{1 + \cos(\alpha)}} = L \cdot \frac{\sin(\alpha)}{1 + \cos(\alpha)} = L \cdot \frac{P_{1,x}}{P_{1,y}}$$

The factor for converting L and ε can be simplified using trigonometric transformations.

$$\frac{1 - \cos\left(\frac{\alpha}{2}\right)}{\sin\left(\frac{\alpha}{2}\right)} = \frac{1 - \sqrt{\frac{1-\cos(\alpha)}{2}}}{\sqrt{\frac{1+\cos(\alpha)}{2}}} = \frac{\sqrt{2} - \sqrt{1 - \cos(\alpha)}}{\sqrt{1 + \cos(\alpha)}} = \frac{\sqrt{1 + \cos(\alpha)} - \sin(\alpha)}{1 + \cos(\alpha)}$$

By comparing this with the symmetric Hermite problem in standard form, the following notation is then obtained:

$$\frac{1 - \cos\left(\frac{\alpha}{2}\right)}{\sin\left(\frac{\alpha}{2}\right)} = \frac{\sqrt{P_{1,x}} - P_{1,y}}{P_{1,x}}$$

The previous considerations are now summarised in the following sentence.

Satz. Let a symmetric Hermite problem $(P_0, \vec{t}_0, P_1, \vec{t}_1, S, L)$ be given. Without restriction of generality, it is in the standard form

$$\left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} L + L \cdot \cos(\alpha) \\ L \cdot \sin(\alpha) \end{pmatrix}, \begin{pmatrix} \cos(\alpha) \\ \sin(\alpha) \end{pmatrix}, \begin{pmatrix} L \\ 0 \end{pmatrix}, L \right\}$$

with $L \in \mathbb{R}^{>0}$ and $\alpha \in (-\pi; \pi]$.

Then an arc can be found that connects the points P_0 and P_1 , has the same tangent directions at the two points.

For the circular arcs applies:

$$r = L \cdot \left| \frac{P_{1,x}}{P_{1,y}} \right|; \quad \phi_0 = -\operatorname{sign}(\alpha) \cdot \frac{\pi}{2}; \quad \phi_1 - \phi_0 = \operatorname{sign}(\alpha) \cdot \pi - \alpha = \beta;$$

$$M = P_0 + \operatorname{sign}(\alpha) \cdot r \cdot \vec{t}_0^\perp = \operatorname{sign}(\alpha) \cdot r \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

From the specification of the distance L , the maximum error can be calculated, as shown in theorem ???. According to the derivation, it is also possible to specify the maximum error ε and determine the maximum distance L from it.

Satz. Let a symmetric Hermite problem $(P_0, \vec{t}_0, P_1, \vec{t}_1, S, L)$ be given. Without restriction of generality, it is in the standard form

$$\left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} L + L \cdot \cos(\alpha) \\ L \cdot \sin(\alpha) \end{pmatrix}, \begin{pmatrix} \cos(\alpha) \\ \sin(\alpha) \end{pmatrix}, \begin{pmatrix} L \\ 0 \end{pmatrix}, L \right\}$$

with $L \in \mathbb{R}^{>0}$ and $\alpha \in (-\pi; \pi]$.

- a) Let the maximum error ε be given. The maximum distance L for which an arc exists according to the theorem ?? that takes the error into account is given by:

$$L(\varepsilon, \alpha) = \varepsilon \cdot \frac{P_{1,x}}{\sqrt{P_{1,x}^2 - P_{1,y}^2}} = \varepsilon \cdot \frac{L + L \cdot \cos(\alpha)}{\sqrt{L + L \cdot \cos(\alpha)^2 - L + L \cdot \cos(\alpha)}}$$

- b) When the distance L is specified, the following maximum error results:

$$\varepsilon(L, \alpha) = L \cdot \frac{\sqrt{P_{1,x}^2 - P_{1,y}^2}}{P_{1,x}} = L \cdot \frac{\sqrt{L + L \cdot \cos(\alpha)^2 - L + L \cdot \cos(\alpha)}}{L + L \cdot \cos(\alpha)}$$

These considerations result in limitations for the use of this strategy, which are summarised in the following comment.

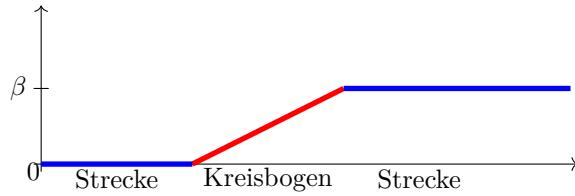


Figure 70.2.: Angular change with blending arc

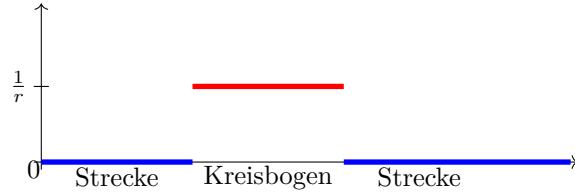


Figure 70.3.: Curvature progression for a blending arc

Bemerkung. The following conditions for applying the strategy of a circle must be fulfilled:

a)

$$P_0 \neq P_1$$

b)

$$\vec{t}_0 = -\vec{t}_1 \Leftrightarrow \alpha = 0$$

c)

$$\vec{t}_0 = \vec{t}_1 \Leftrightarrow \alpha = \pi$$

70.2. Bewertung

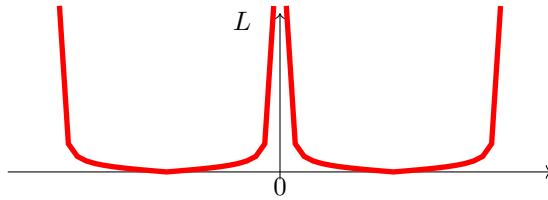
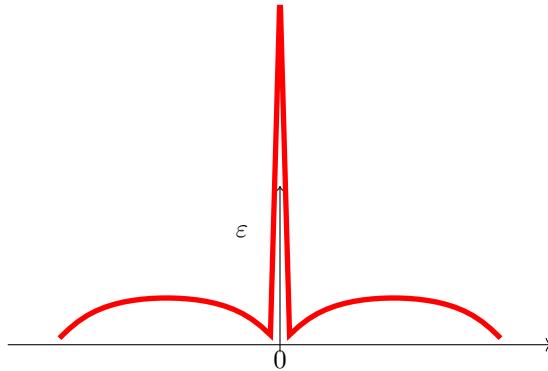
Rounding by means of a circular arc leads to a continuous course of the angle change. The figure ?? illustrates this.

Due to the rounding with a circular arc, the curvature of the curve is not continuous. The figure ?? shows that the curvature is constant in the range of distances 0 and on the circular arc with the value $\frac{1}{r}$, where r is the radius of the circular arc used. Due to the formula $a = \frac{v^2}{r}$ for a movement on a circular arc, the acceleration course exhibits continuity jumps at the transitions for a constant path velocity.

Depending on the angle change β at the corner S and the tolerance ε , the maximum length of the shortening of the lines can be calculated. The figure ?? shows the corresponding diagram, where the tolerance ε is set to the value 1.

The area provided can also be specified. Depending on the distance L from the corner point S , the deviation ε can be calculated. The figure ?? represents the function as a function of L and the angle α .

The figures ?? and ?? show the curves of the length as a function of the angle change for a fixed tolerance and the error as a function of the angle change for a given length. If the angle change is minimal, then the error or length L is extreme. In this case,

Figure 70.4.: Maximum range for a blending arc of a circle - $L(\varepsilon = \text{const}, \alpha)$ Figure 70.5.: Deviation when specifying the distance L when rounding with a circular arc

rounding by means of an arc is not possible. In order to develop a sensible strategy, a minimum angle change must be specified from which a blending arc is permitted. Likewise, a maximum angle change must also be defined. For an angular change of $\pm\pi$ is a bend. In this case, a blending is also not possible.

70.3. Examples

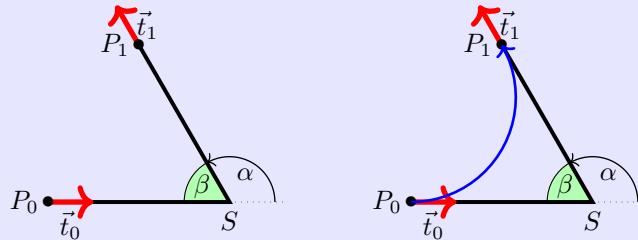
Beispiel. Let it be the symmetric Hermite problem

$$\left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 6.0 + 6.0 \cdot \cos\left(\frac{2}{3}\pi\right) \\ 6.0 \cdot \sin\left(\frac{2}{3}\pi\right) \end{pmatrix}, \begin{pmatrix} \cos\left(\frac{2}{3}\pi\right) \\ \sin\left(\frac{2}{3}\pi\right) \end{pmatrix}, \begin{pmatrix} 6.0 \\ 0 \end{pmatrix}, 6.0 \right\}$$

with $L = 6$ and $\alpha = \frac{2}{3}\pi$.

For the blending arc follows:

$$M = \begin{pmatrix} 0 \\ 3,46412 \end{pmatrix}; \quad r = 3,46412; \quad \phi_0 = -\frac{\pi}{2}; \quad \alpha = \frac{2}{3}\pi$$



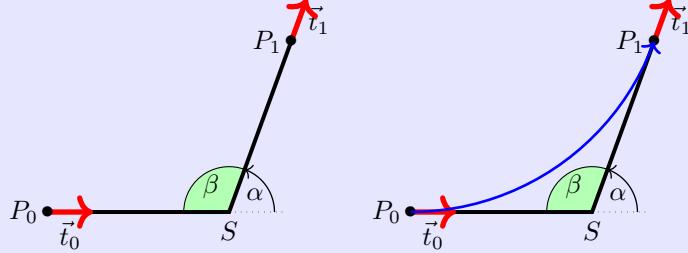
Beispiel. Let it be the symmetric Hermite problem

$$\left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 6.0 + 6.0 \cdot \cos(\frac{7}{18}\pi) \\ 6.0 \cdot \sin(\frac{7}{18}\pi) \end{pmatrix}, \begin{pmatrix} \cos(\frac{7}{18}\pi) \\ \sin(\frac{7}{18}\pi) \end{pmatrix}, \begin{pmatrix} 6.0 \\ 0 \end{pmatrix}, 6.0 \right\}$$

with $L = 6$ and $\alpha = \frac{7}{18}\pi$.

For the blending arc follows:

$$M = \begin{pmatrix} 0 \\ 8,5689 \end{pmatrix}; \quad r = 8,5689; \quad \phi_0 = -\frac{\pi}{2}; \quad \alpha = \frac{7}{18}\pi$$



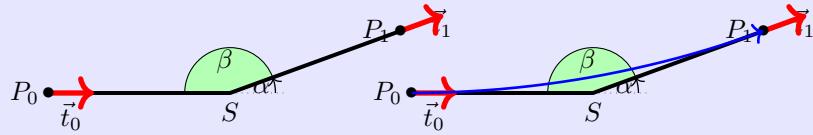
Beispiel. Let it be the symmetric Hermite problem

$$\left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 6.0 + 6.0 \cdot \cos(\frac{1}{9}\pi) \\ 6.0 \cdot \sin(\frac{1}{9}\pi) \end{pmatrix}, \begin{pmatrix} \cos(\frac{1}{9}\pi) \\ \sin(\frac{1}{9}\pi) \end{pmatrix}, \begin{pmatrix} 6.0 \\ 0 \end{pmatrix}, 6.0 \right\}$$

with $L = 6$ and $\alpha = \frac{1}{9}\pi$.

For the blending arc follows:

$$M = \begin{pmatrix} 0 \\ 34,0277 \end{pmatrix}; \quad r = 34,0277; \quad \phi_0 = -\frac{\pi}{2}; \quad \alpha = \frac{1}{9}\pi$$

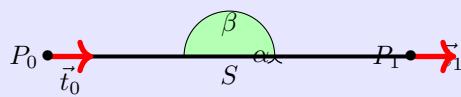


Beispiel. Let it be the symmetric Hermite problem

$$\left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 12.0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 6.0 \\ 0 \end{pmatrix}, 6.0 \right\}$$

with $L = 6$ and $\alpha = 0$.

There is no blending arc here.



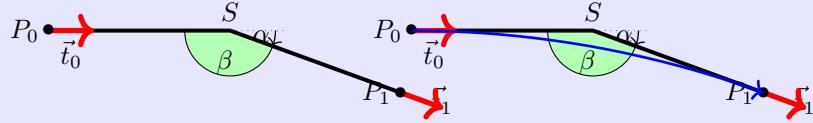
Beispiel. Let it be the symmetric Hermite problem

$$\left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 6.0 + 6.0 \cdot \cos(-\frac{1}{9}\pi) \\ 6.0 \cdot \sin(-\frac{1}{9}\pi) \end{pmatrix}, \begin{pmatrix} \cos(-\frac{1}{9}\pi) \\ \sin(-\frac{1}{9}\pi) \end{pmatrix}, \begin{pmatrix} 6.0 \\ 0 \end{pmatrix}, 6.0 \right\}$$

with $L = 6$ and $\alpha = -\frac{1}{9}\pi$.

For the blending arc follows:

$$M = \begin{pmatrix} 0 \\ -34,0277 \end{pmatrix}; \quad r = 34,0277; \quad \phi_0 = \frac{\pi}{2}; \quad \alpha = \frac{1}{9}\pi$$



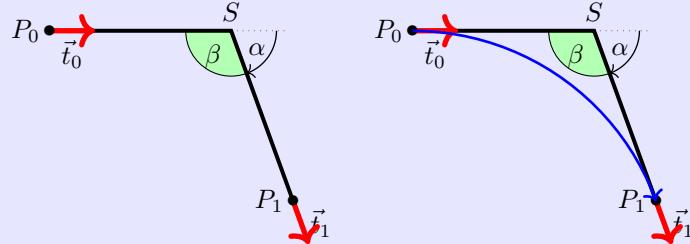
Beispiel. Let it be the symmetric Hermite problem

$$\left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 6.0 + 6.0 \cdot \cos(-\frac{7}{18}\pi) \\ 6.0 \cdot \sin(-\frac{7}{18}\pi) \end{pmatrix}, \begin{pmatrix} \cos(-\frac{7}{18}\pi) \\ \sin(-\frac{7}{18}\pi) \end{pmatrix}, \begin{pmatrix} 6.0 \\ 0 \end{pmatrix}, 6.0 \right\}$$

with $L = 6$ and $\alpha = -\frac{7}{18}\pi$.

For the blending arc follows:

$$M = \begin{pmatrix} 0 \\ -8,5689 \end{pmatrix}; \quad r = 8,5689; \quad \phi_0 = \frac{\pi}{2}; \quad \alpha = -\frac{7}{18}\pi$$



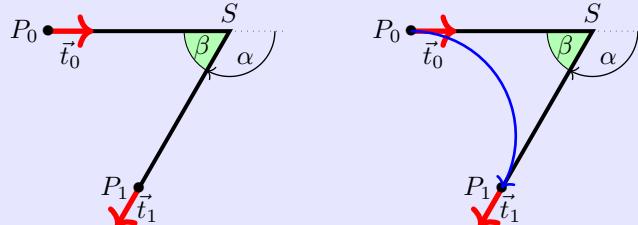
Beispiel. Let it be the symmetric Hermite problem

$$\left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 6.0 + 6.0 \cdot \cos(-\frac{2}{3}\pi) \\ 6.0 \cdot \sin(-\frac{2}{3}\pi) \end{pmatrix}, \begin{pmatrix} \cos(-\frac{2}{3}\pi) \\ \sin(-\frac{2}{3}\pi) \end{pmatrix}, \begin{pmatrix} 6.0 \\ 0 \end{pmatrix}, 6.0 \right\}$$

with $L = 6$ and $\alpha = -\frac{2}{3}\pi$.

For the blending arc follows:

$$M = \begin{pmatrix} 0 \\ -3,4641 \end{pmatrix}; \quad r = 3,46412; \quad \phi_0 = \frac{\pi}{2}; \quad \alpha = -\frac{2}{3}\pi$$



71. Maple files

[Wat17c; Wat17d; Wat17b; Wat17e; Wat17a; Wat17f]

71.1. Geometries with Maple

The algorithms presented can be well represented using geometries. Two aspects can be investigated. On the one hand, the effort for an implementation, the computational accuracy and the stability of an algorithm are interesting; on the other hand, the methods are to be evaluated and compared with regard to their usefulness. For this purpose, modules for the formula manipulation system Maple [Wat17c] were developed. Maple offers the environment to implement algorithms easily and quickly. Furthermore, graphical representation is possible with simple means. However, a simple workbook of Maple is not designed for very large software projects. Here, one must resort to the possibility of creating and using libraries. On the one hand, Maple offers the possibility of creating one's own libraries, so-called modules. In this way, one remains within the environment and syntax of Maple. This path will be pursued further here. Another possibility is the use of libraries created by means of a higher programming language, e.g. C++, the so-called DLLs.

In the following, the creation and use of a test environment with the help of modules is described. First, the geometry elements that are stored in various modules and their use are described. The structure and use of a module in Maple is then explained so that own extensions and additions are possible.

71.2. Geometry elements used

This is a test environment for the algorithms presented, only geometries that have been described are also used.

- points ([MPoint](#))
- lines ([MLine](#))
- arcs ([MArc](#))
- Bézier curves ([Bezier](#))
- polygon courses ([MPolygon](#))
- Geometry List ([MGeoList](#))
- Hermite problems ([MHermitProblem](#))
- Symmetric Hermite Problems ([MHermitProblemSym](#))

For each geometry element a corresponding module has been created, the name of which is given in the list above. The list of which functions are available is presented in another section.

When implementing such a project, it quickly becomes clear that when using several geometry elements, a clear data structure and well-defined access to the data is essential. For example, when representing straight lines, the decision must be made

whether the representation should be point-directional or by means of two points. The choice is generally made depending on the application. Here, the path is taken that, due to a well-defined access to the data, the use of both representations is possible.

71.3. Building the data structure

The idea is to use a data structure that is as uniform and simple as possible. Maple does offer the possibility to define objects. However, it is difficult to use within a procedural environment. Therefore, all data is basically represented as lists. The first list element always contains an identifier of the element `??`. This is followed by the data, which in turn can be geometry elements. It should be noted that direct access to the data cannot be prevented; here the user is responsible.

Access to the data should be exclusively via procedures of a module. The following is an example of the data structure for points:

```
[MVPOINT, [x, y]]
```

The creation of a point then takes place via a procedure `New`:

```
NeuerPunkt := MPoint:-New(10,15);
```

When called up in the test file, the following is then output:

```
TestPO := ["Point", [10, 15]]
```

These data structures are used for the calculation of geometries. They are used in functions or procedures. Unlike variables, the data structures and procedures are defined globally and not locally. Under the command `export` the procedures are declared at the beginning of the module and can thus also be used outside the module. If, for example, a data structure from `MPoint` is to be used in another module or outside the modules, it is called as follows:

```
P0 := MPoint:-New(x, y);
```

In addition to the modules for the geometries, there is a module (`MConstant`) for saving constants and names. There, for `MVPOINT`, for example. „Point“ or e.g. a constant for the comparison to zero for real numbers.

Finally there is the test file, which is not a module, in which the modules are loaded, called and tested in their function. More about this file later in „1.4 Maple test file“.

71.4. Structure of a module

The following section deals with the purpose of modules and their structure.

The project is about capturing the above geometries in a data structure and representing them in Maple. However, the calculations and formulas that have to be used are a hindrance to the final representation. Modules are very well suited for summarising and hiding the calculations that take place in functions. This way, the important functions can be accessed in Maple without seeing what is in them.

Example:

The function `Angle` from the module `MPoint`: Function to calculate an angle between location vector and x-axis:

```
Angle := proc(P)
local alpha, x, y;
```

```

x := GetX(P);
y := GetY(P);
alpha := 0;
if abs(x) < 0.00001
then
    alpha := 3*Pi/2;
else
    alpha := Pi/2;
end if;
else
    alpha := arctan(y/x);
    if x < 0
    then
        alpha := alpha + Pi;
    else
        if y < 0
        then
            alpha := alpha + 2*Pi;
        end if;
    end if;
end if;
return factor(alpha);
end proc;

```

This function is long, but it only represents a small part of the programme for the representation in question. That is why it is in the module and can thus be called externally with a single command:

```
Winkel := MPoint:-Angle(P)
```

The following section describes the structure of a module. Since Maple offers a wide range of possibilities, a restriction is made. Only the structure of the modules used is described, here on the basis of the module **MPoint**. For more detailed possibilities, please refer to the Maple manual [Wat17c].

structure:

The module must first be started. This is done with the name (here always a capital M and the geometry) of the module and the following command:

```
MPoint := module()
```

Then, similar to procedures, the variables and functions must be defined. You can declare them either locally or globally. If the variables or procedures are only used and changed in the module, the declaration is made with the command **local** as follows:

```
local Variable names, with, comma, separated;
```

If the variables or procedures are also to be usable outside the module, this is defined with **export**:

```
export Function names, with, comma, separated;
```

This is followed by the specification of the options:

```
option package;
```

the description of the module:

```
description "Self-selected module description, e.g. module for  
points ";
```

and the initialisation of the module:

```
ModuleLoad := proc  
    MVPOINT:=MConstant:-GetPoint();  
    print("Module MPoint is loaded");  
end proc;
```

From here on, programming is done as usual in Maple. The previously defined procedure and variable names are used and programming is done with commands that are also used outside of a module in Maple. It is important to know that with modules, each function can be called constantly, so the order of the functions is irrelevant.

To finally end the module, use the following command:

```
end module;
```

71.4.1. General structure of a module

In summary, the modules have the following structure:

```
Modulname := module()  
  
    local Names, with, comma, separated;  
  
    export Names, with, comma, separated;  
  
    global Names, with, comma, separated;1  
  
    option package;  
  
    description „ Self-selected module description“;  
  
    ModuleLoad := proc()2  
        MVPOINT:=MConstant:-GetPoint();  
        print("Modul MPoint is loaded");  
    end proc;  
  
    Procedure1 := proc (Passing parameters)  
        inhalt;  
    end proc;  
  
    Procedure2 := proc (Passing parameters)  
        content;  
    end proc;  
  
    ...  
  
end module;
```

¹Possible, but not used in the modules

²Example `MPoint`

71.4.2. Saving a module

The management of modules is done automatically by Maple. However, since the modules are passed on and have to be edited individually, some settings have to be made. Therefore, the following prefix is used for each Maple file of the project:

```
1 restart;
2 with(LibraryTools);
3 lib := "C:/FH/Tools/Maple/MyLibs/Blending.mla";
4 march('open', lib);
5 ThisModule := 'MArc';
```

The first line initialises the system. The next 3 lines enable working with archives. In the second line, the tools are loaded so that the variable `lib` can be occupied. All modules are stored in an archive; in this example it is the file `Blending.mla` in the directory `C:/FH/Tools/Maple/MyLibs/`. The command `ThisModule := 'MArc'`; contains the name of the current module.

A module is then saved using the command `savelib('ModuleName')`. A file `ModuleName.mla` is then created. Depending on the configuration of Maple, the set path where the file is automatically created may not be writable. Then you can configure the system so that the mla file is stored in the current directory or in a directory of your choice.

If the above prefix is used for a Maple file, all that is required to save the module is `savelib(ThisModule, lib);`

71.4.3. Verwendung eines Moduls

A module that has been saved can now be used in other Maple worksheets. To do this, the command

`with(ModuleName)`

is used. If you have used a special path, Maple cannot find the file `ModuleName.mla`. Then the path must be made known, e.g.:

```
savelibname := "c:/Maple/MyLibs";", savelibname;
```

Now the procedures of the module can be accessed. An overview of all exported procedures is provided by the command

`Describe(ModuleName)`

is displayed. A list of the names including the names of the transfer parameters is displayed. If a procedure is equipped with the field `description`, this text is also displayed.

71.4.4. Creating a Maple module

Creating your own module is done quickly if you use the framework above. However, one should make some considerations beforehand and maintain a standard.

Before creating an own module, the data model and the corresponding procedures should be worked out. A programme flow chart is useful here. The central task of the module is usually quickly determined. In addition, however, the following rules should be observed.

Rule 1 Basically, both the module and each procedure receive a description. This is not limited to the optional argument `description`, but is placed in front of each procedure. The description basically contains the description of the task. The prerequisite is also mentioned or an error handling is described. Then follows the description of all input parameters, their function and their data structure. The return value is then described.

rule 2 Each module receives a procedure `Version()`, which returns the current version number.

rule 3 Data is not global. To access data, procedures `Set*` and `Get*` are provided. These procedures are also used within the procedures of a module.

rule 4 At least one test function is written for each procedure. The test function can be used to illustrate the use and any special features.

71.5. Functions of the modules

In the following, the individual modules are listed. The data structure of each module and all functions with associated tasks are listed.

71.5.1. Module `MPoint`

`MPoint` is a module for points and works with a data structure and with procedures/functions. The data structure `New` for a point is a list, which is structured as follows:

`[MVPOINT, [x,y]]`

Its first element is a name to identify the module. Your second element is again a list containing the elements of the geometry. In this case the name is "Point" and the elements are the x and y coordinates for a point. No distinction is made here between point and vector. A point can also be seen as a vector from the coordinate origin to the point.

Via the Get functions `GetX` and `GetY` one can capture the coordinates of the point and use them in other functions. The other functions can then be used to calculate with the points/vectors.

`MPoint`

E	<code>New</code>	Data structure for a point
E	<code>GetX</code>	Reading the x-coordinate
E	<code>GetY</code>	reading the y-coordinate
E	<code>Angle</code>	calculating the angle between the x-axis and the point
E	<code>Add</code>	Calculates a linear combination of two points/vectors
E	<code>Sub</code>	Calculates the difference of two points/vectors
E	<code>Cos</code>	Calculates the cosine between two vectors
E	<code>Sin</code>	Calculates the sine between two vectors
E	<code>Scale</code>	Scales a vector with a factor
E	<code>Perp</code>	Calculates a vector that is orthogonal to the given vector
L	<code>IllustrateXY</code>	Plot function to illustrate a blue point
E	<code>Illustrate</code>	Plot of a blue point
E	<code>Plot</code>	Plot of a green point
E	<code>Plot2D</code>	Plot of a point with own options
E	<code>Length</code>	Calculates the distance of the point to the coordinate origin
E	<code>Uniform</code>	Normalises the vector to length 1
E	<code>LinetoVector</code>	Calculates vector from a distance
E	<code>Distance</code>	Calculates the distance between two points

71.5.2. Module MLine

MLine is a module for lines and works with a data structure and with procedures/functions. The data structure **New** for a line is a list which is structured as follows:

```
[MVLINE, [P0,P1]]
```

Its first element is a name to identify the module. Its second element is again a list containing the elements of the geometry. In this case the name is „Line“ and the elements are the start and end points for a line.

The data structure **NewPointVector** is also a data structure for a line, but here the line is defined by a start point and a direction vector.

The Get functions **StartPoint** and **EndPoint** can be used to capture the start and end points of the route and use them in other functions. The other functions can then be used to calculate with the routes.

MLine

E	New	Data structure for a line (two-point form)
E	NewPointVector	creation of a route (point-direction form)
E	StartPoint	reading the starting point
E	EndPoint	reading the end point
E	Position	Calculate a point that lies on the line
E	Plot2D	Plot a part of the route starting from the starting point
E	Plot2DTangent	Plot of a point with tangent
E	Plot2DTangentArrow	Plot of a point with tangent (as arrow)
E	LineLine	Calculation of the intersection of two straight lines.
E	AngleLine	Calculation of the angle between two straight lines

71.5.3. Module MArc

MArc is a module for circular arcs and works with a data structure and with procedures/functions. The data structure **New** for an arc is a list that is structured as follows:

```
[MVARC, [mx,my,r,phi,alpha]]
```

Its first element is a name to identify the module. Your second element is again a list containing the elements of the geometry. In this case the name is "Arc" and the elements are the x- and y-coordinate for the centre, the radius, the start angle and the angle change for an arc.

The Get functions **GetM**, **GetMX**, **GetMY**, **GetR**, **GetPhi**, and **GetAlpha** can be used to collect the data for an arc and use it in other functions. The other functions can then be used to calculate with the arcs.

MArc

E	New	Data structure for an arc
E	GetMX	Reading the x-coordinate of the centre point.
E	GetMY	read the y-coordinate of the centre point
E	GetR	read the radius
E	GetPhi	reading the start angle
E	GetAlpha	Reading the change of angle
E	GetM	reading the centre point
E	Position	Calculating a point on the arc
E	Plot2D	Plot a part of the arc starting from the start angle
E	Blend	calculating an arc from a symmetric Hermite problem

71.5.4. module MBezier

The **New** data structure for a polygon is a list constructed as follows:

[MVBEZIER, [PointList]]

Within the module **MBezier** exist the procedures listed in the following table.

MBezier

E New	Manual input of control points for a Bézier curve
E Version	Output of the verions
E BlendCurvature	Determination of control points from symmetric Hermite problem
E BlendCurvatureEpsilon	determination of control points from symmetric Hermite problem with given error
E Position	position on the Bézier curve
E GetTheta	Reading the angle
E GetEpsilon	reading the tolerance
E GetControlPoint	Read the control points
E Plot2D	Read the Bézier curve
E PlotControlPoints	representation of all control points

71.5.5. Module MPolygon

MPolygon is a module for polygons and works with a data structure and with procedures/functions. The data structure **New** for a polygon is a list that is structured as follows:

[MVPOLYGON, [PointList]]

Its first element is a name to identify the module. Your second element is again a list containing the elements of the geometry. In this case the name is "Polygon" and the elements are any number of points.

Using the Get functions **GetPoint** and **GetN** you can get the number of points and the points themselves and use them in other functions. The other functions can then be used to calculate with the points or the polygon course.

MPolygon

E New	Data structure for a list of points
E GetPoint	Reading the ith point from the point list
E GetN	Determine the number of points in the point list
E Length	Determination of the Euclidean length of the polygon
E Position	Calculation of a point on the polygon course
E Tangents	calculation of the tangent
L Plot2DAll	Plot list of all points
E Plot2D	Plot of all points as polygonal plots.
E Plot2DTangent	representation of the polygon course with tangent
E PlotPoints	representation of all points

71.5.6. module MGeoList

MGeoList is a module for a geometry list and works with a data structure and with procedures/functions. The data structure **New** for a geometry list is a list that is structured as follows:

[MVGEOLIST, []]

Its first element is a name to identify the module. Its second element is again a list containing the elements of the geometry. In this case the name is „GeoList“ and the elements are any number of individual geometries.

Using the Get functions **GeoGeo** and **GetN**, the i-th element of the list and the number of elements in the list can be captured and used in other functions. The other functions can then be used to calculate with the geometries or the list. In the functions **Length**, **Plot2DAll**, **Plot2D** and **Position** the functions are called in themselves. This is possible because the functions work independently of each other.

MGeoList

E	New	Data structure for a geometry list
E	Append	Append a geometry element to the data structure
E	Prepend	Inserting a geometry element as the first element of the list
E	Replace	Replace a geometry element with another one.
E	GetN	Determine the number of geometry elements in the list
E	GeoGeo	Reading the i-th geometry element
E	Length	Calculate the Euclidean length of the geometry elements
E	Position	Calculates a point on the geometry
L	Plot2DAll	Plot function for plotting the geometry elements
E	Plot2D	Plot a part of the geometry list starting from the first element

71.5.7. Module MHermitProblem

MHermitProblem is a module for Hermite problems and works with a data structure and with procedures/functions. The data structure **New** for a route is a list, which is structured as follows:

```
[MVHERMITPROBLEM, [P0, T0n, P1, T1n]]
```

Your first element is a name to identify the module. Its second element is again a list containing the elements of the geometry. In this case the name is „HermiteProb“ and the elements are two points with associated tangents (lines).

Using the Get functions **StartPoint**, **EndPoint**, **StartTangent** and **EndTangent**, the points and their tangents can be captured and used in other functions. The other functions can then be used to calculate with the data for the Hermite problem.

MHermitProblem

E	New	Data structure for a Hermite problem
E	StartPoint	reading the start point
E	EndPoint	Reading the end point command
E	StartTangent	reading the start tangent
E	EndTangent	Reading the end tangent
E	Plot2D	Plotting the Hermite problem

71.5.8. Module MHermitProblemSym

MHermitProblemSym is a module for symmetric Hermite problems and works with a data structure and with procedures/functions. The data structure **New** for a symmetric Hermite problem is a list built as follows:

```
[MVHERMITPROBLEMSYM, [P0, T0n, P1, T1n, S, L]]
```

Your first element is a name to identify the module. Its second element is again a list containing the elements of the geometry. In this case the name is „SymHermiteProb“

and the elements are two points with associated tangents, their intersection, and the distance of the points to the intersection.

Via the Get functions `StartPoint`, `EndPoint`, `StartTangent`, `EndTangent`, `CrossPoint` and `ParameterL` one can acquire the data and use it in other functions. The other functions can then be used to calculate with the data for the symmetric Hermite problem.

MHermiteProblemSym

E <code>New</code>	Data structure for a symmetric Hermite problem
E <code>StartPoint</code>	reading the start point
E <code>EndPoint</code>	Reading the end point command
E <code>StartTangent</code>	reading the start tangent
E <code>EndTangent</code>	Reading the end tangent
E <code>ParameterL</code>	reading of the distance
E <code>CrossPoint</code>	reading of the intersection point
E <code>Plot2D</code>	Plotting of the symmetrical Hermite problem
E <code>Create</code>	creation of a Sym. Hermite problem by 3 points
E <code>BlendArc</code>	rounding of the corner point by an arc

71.5.9. Module MConstant

`MConstant` is a module for storing fixed constants and names to identify data structures. In the locally declared functions, starting with CV, the constants for declaring the different geometries are defined. These are names for recognising the geometry elements in the test file. In the globally declared functions, starting with Get, the names for identifying the geometry elements are returned.

MConstant

L <code>NULLEPS</code>	constant to compare to zero
L <code>CVPOINT</code>	constant for geometry elements: Point
L <code>CVLINE</code>	constant for geometry elements: Line
L <code>CVARC</code>	constant for geometry elements: Arc
L <code>CVPOLYGON</code>	constant for geometry elements: polygon
L <code>CVGEOLIST</code>	constant for geometry elements: GeoList
L <code>CVHERMITEPROBLEM</code>	constant for geometry elements: HermiteProb
L <code>CVHERMITEPROBLEMSYMMETRIC</code>	Const. for geometry elements: SymHermiteProb
L <code>CVBIARC</code>	Const. for geometry elements: Biarc
E <code>GetNullEps</code>	return of the zero comparison command.
E <code>GetPoint</code>	identifier for points
E <code>GetLine</code>	Identifier for lines
E <code>GetArc</code>	identifier for circular arcs
E <code>GetPolygon</code>	identifier for polygons
E <code>GetGeoList</code>	Identifier for geometry lists
E <code>GetHermiteProblemSymmetric</code>	Identifier for symmetric Hermite problems
E <code>GetHermiteProblem</code>	ID for Hermite problems
E <code>GetBiarc</code>	identifier for Biarcs

71.5.10. Module MGeneralMath

`MGeneralMath` is a module for general mathematical functions and works with the data structures and procedures.

MeneralMath

E	MPoint	Data structure for a point
E	MPointX	Reading of the x-coordinate for a point
E	MPointY	reading the y-coordinate for one point
L	MPointIllustrateXY	plot structure for a blue point
E	MPointIllustrate	plot structure for a blue point
E	MPointPlot	Illustration of a green point
E	MLine	Data structure for a line
E	MLineStartPoint	Reading of the start point for a draw frame
E	MLineEndPoint	Reading the end point for a line
E	MPointOnLine	Calculation of a point on the line
E	MLinePlot2D	Plot the part of a line starting at the starting point
. E	MLineLine	calculating the intersection of two linesline

71.5.11. Module Biarc**Data Structure**

Requirements and specifications:

The Biarc is to be used to solve a Hermite problem. A Hermite problem is defined as follows:

Given two points P_0 and P_1 with associated normalised tangents \vec{t}_0 and \vec{t}_1 . The biarc must connect the points such that the tangents of the start and end points of the biarc coincide with the tangents of the Hermites problem.

Data structure:

The data structure **New** for a biarc must contain two arcs.

So the data structure for one arc is needed: **MArc:-New**.

This in turn contains the coordinates for the centre, the radius, the start angle and the angle change.

71.5.12. Module MBiarc

MBiarc is a module for Biarcs and works with a data structure and with procedures/functions. The data structure **New** for a Biarc is a list which is structured as follows:

[MVBIARC, [Arc0, Arc1]]

Its first element is a name to identify the module. Your second element is again a list containing the elements of the geometry. In this case the name is „Biarc“ and the elements are two arcs.

Using the Get functions **GetArc0** and **GetArc1** one can capture the two arcs for the biarc. The functions listed below can be used to calculate the data for the biarc.

MBiarc

E New	Data structure for a biarc
E GetArc0	Reading of the first arc
E GetArc1	Reading the second arc
E Circle	calculating the circle K_J from the Hermite problem data.
E Plot2DCircle	plot of the circle K_J
E angle	Calculate the angle from the centre of the circle to points on the circle
E Plot2D	Plot of the biarc
E ConnectionPoint	Calculation of the connection point J (Equal Chord)
E TangentTj	Calculation of the tangent to J
E Tangent Biarc	rotation of T_j for the biarc.
E BiarcCenter	Calculate the centres of the arcs of the biarc
E Biarc	calculate the centre of the biarc.
E Position	Determine a point on the biarc
E Blend	Calculate the biarc only from the Hermite problem

71.6. Programme Flowchart

71.6.1. Overall Flow

71.6.2. Verrundung der Kurve

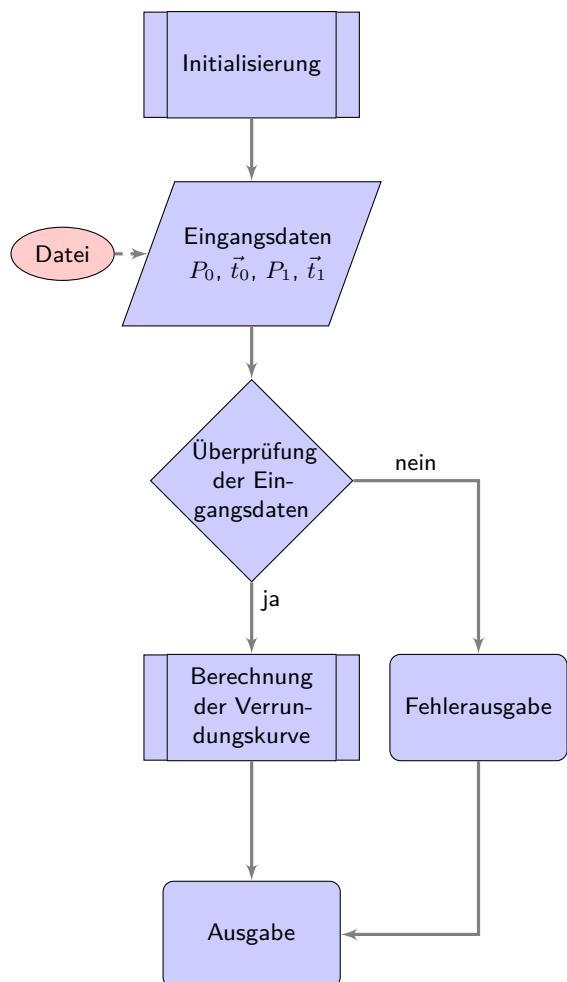


Figure 71.1.: Programme flow chart „Corner rounding“

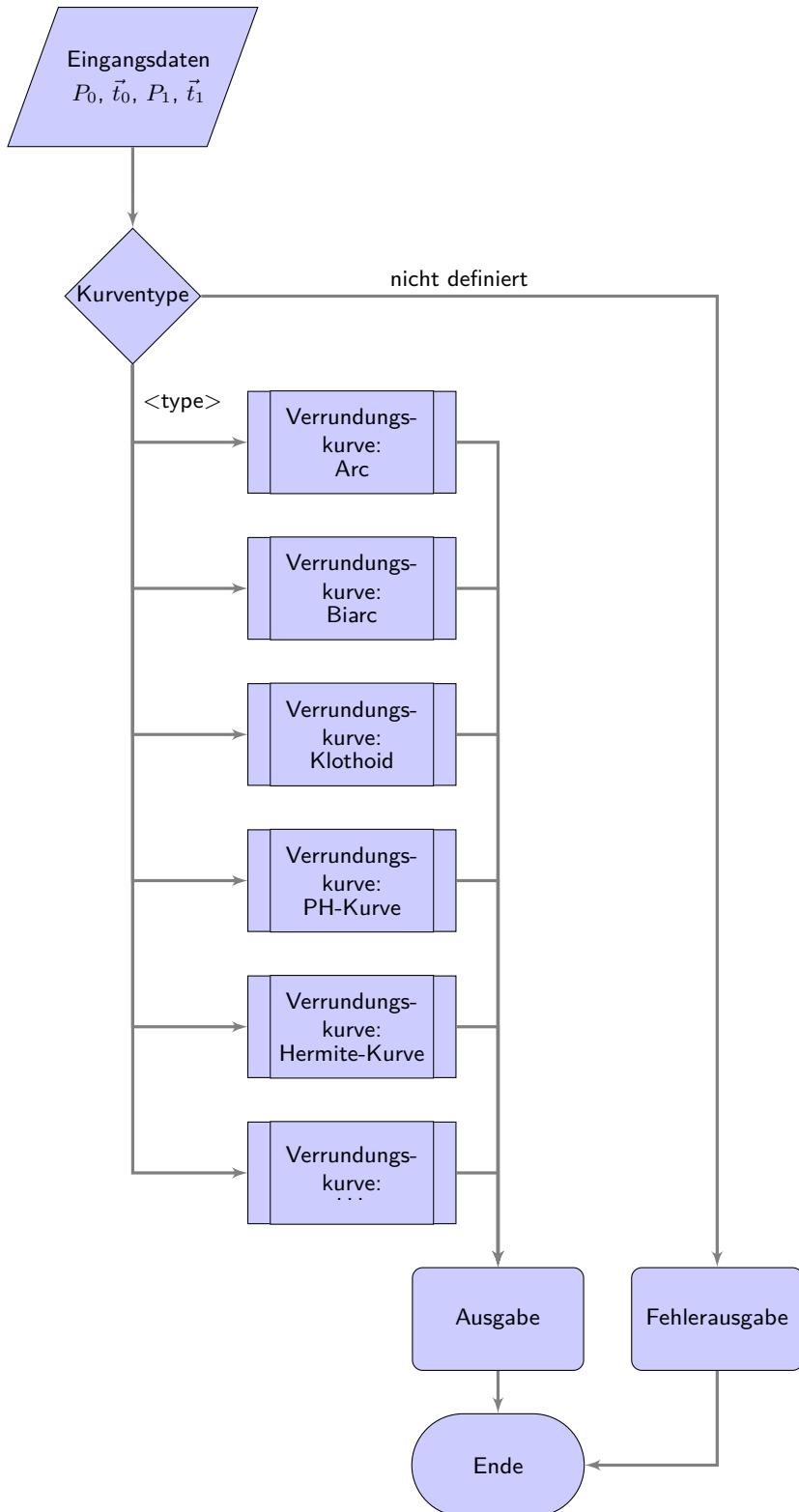


Figure 71.2.: Programme flowchart „Selection of the rounding strategies“

72. Representation of Python Programs

It is possible to integrate a part into the document, see 72.1. This is the most elegant method and is also preferable. Individual lines and line ranges can also be selected in the list of options. If a file is integrated, it is always as up-to-date as the document. It may also be useful to integrate program lines: `redLED = pyb.LED(1)`.

Attention: The integration of programs with the help of images is pointless.

Listing 72.1.: The program “Hello World” in Python for microcontroller boards is inserted from the file `Blink.py`.

```

1000 %% This is file 'rename-to-empty-base.tex',
1002 %% generated with the docstrip utility.
1003 %%
1004 %% The original source files were:
1005 %%
1006 %% fileerr.dtx (with options: 'return')
1007 %%
1008 %% This is a generated file.
1009 %%
1010 %% The source is maintained by the LaTeX Project team and bug
1011 %% reports for it can be opened at https://latex-project.org/bugs/
1012 %% (but please observe conditions on bug reports sent to that address!)
1013 %%
1014 %%
1015 %% Copyright (C) 1993–2024
1016 %% The LaTeX Project and any individual authors listed elsewhere
1017 %% in this file.
1018 %%
1019 %% This file was generated from file(s) of the Standard LaTeX ‘Tools
1020 %% Bundle’.
1021 %% -----
1022 %%
1023 %% It may be distributed and/or modified under the
1024 %% conditions of the LaTeX Project Public License, either version 1.3c
1025 %% of this license or (at your option) any later version.
1026 %% The latest version of this license is in
1027 %% https://www.latex-project.org/lppl.txt
1028 %% and version 1.3c or later is part of all distributions of LaTeX
1029 %% version 2005/12/01 or later.
1030 %%
1031 %% This file may only be distributed together with a copy of the LaTeX
1032 %% ‘Tools Bundle’. You may however distribute the LaTeX ‘Tools Bundle’
1033 %% without such generated files.
1034 %%
1035 %% The list of all files belonging to the LaTeX ‘Tools Bundle’ is
1036 %% given in the file ‘manifest.txt’.
1037 %%
1038 \message{File ignored}
1039 \endinput
1040 %% End of file 'rename-to-empty-base.tex'.

```

`..../Code/HelloWorld/Blink.py`

73. Representation of Programs written for Arduino Boards

It is possible to integrate a part into the document, see 73.1. This is the most elegant method and is also preferable. Individual lines and line ranges can also be selected in the list of options. If a file is integrated, it is always as up-to-date as the document.

Attention: The integration of programs with the help of images is pointless.

Listing 73.1.: The program “Hello World” for an Arduino microcontroller board is inserted from the file [Blink.ino](#).

```

1000 /**
1001 *
1002 * @file Blink.ino
1003 *
1004 * @brief Simple program for testing the configuration
1005 *
1006 *
1007 * Turns an LED on for one second, then off for one second, repeatedly.
1008 *
1009 * Most Arduinos have an on-board LED you can control. On the UNO, MEGA
1010 * and ZERO
1011 * it is attached to digital pin 13, on MKR1000 on pin 6. LED_BUILTIN is
1012 * set to
1013 * the correct LED pin independent of which board is used.
1014 * If you want to know what pin the on-board LED is connected to on your
1015 * Arduino
1016 * model, check the Technical Specs of your board at:
1017 *
1018 * https://www.arduino.cc/en/Main/Products
1019 *
1020 * modified 8 May 2014
1021 * by Scott Fitzgerald
1022 * modified 2 Sep 2016
1023 * by Arturo Guadalupi
1024 * modified 8 Sep 2016
1025 * by Colby Newman
1026 *
1027 * This example code is in the public domain.
1028 *
1029 * https://www.arduino.cc/en/Tutorial/BuiltInExamples/Blink
1030 */
1031
1032 /**
1033 * @brief the setup function runs once when you press reset or power the
1034 * board
1035 *
1036 * standard function of Arduino sketches
1037 *
1038 * Initialization of the pin LED_BUILTIN as output
1039 *
1040 * @param —
1041 *
1042 * @return void
1043 */
1044 void setup() {
1045     // initialize digital pin LED_BUILTIN as an output.
1046     pinMode(LED_BUILTIN, OUTPUT);
1047 }
1048 /**
1049 * @brief the loop function runs over and over again forever
1050 *
1051 * standard function of Arduino sketches
1052 *
1053 * swichting the led on / off for 1sec.
1054 *
1055 * @param —
1056 *
1057 * @return void
1058 */
1059 void loop() {
1060     digitalWrite(LED_BUILTIN, HIGH);      // turn the LED on (HIGH is the
1061         // voltage level)
1062     delay(1000);                      // wait for a second
1063     digitalWrite(LED_BUILTIN, LOW);       // turn the LED off by making the
1064         // voltage LOW
1065     delay(1000);                      // wait for a second
1066 }
```

74. First Chapter

...

A Speicherprogrammierbare Steuerung (SPS) is ...

A Computerized Numerical Control (CNC) needs a SPS (PLC) to ...

75. CAGD

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

The book CAGD by Gerald Farin is a classic on splines. [Far02]

The standard 66025 for programming CNC machines is also a classic; however, it does not deal with splines. [DIN66025-2]

Mr. F. Farouki has dealt with both CNC machine programming and splines. His article¹ on a real-time interpolator also shows this. [FS17]

A new dimension of machine tools have emerged with the invention of 3D printers. [Rus+07]

Another aspect of automation technology is communication. Another milestone has been reached with 5G technology. another milestone has been reached.²

¹co-author is J. Srinathu

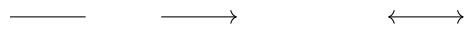
²Zaf+20.

A. drawings with tikz

The package tikz is a powerful tool for creating graphics. Many introductions exist. Here only the first steps are shown, so that you can easily create flowcharts.

Drawing a line and arrows

```
\begin{tikzpicture}
  \draw (0,0) -- (1,0);
  \draw[->] (2,0) -- (3,0);
  \draw[<->] (5,0) -- (6,0);
\end{tikzpicture}
```



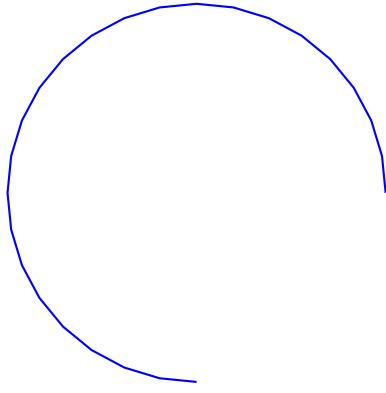
Drawing a thick blue line

```
\begin{tikzpicture}
  \draw [line width=2pt, blue] (0,0) — (1,0);
  \draw [line width=2pt, red, dotted] (2,0) — (3,0);
  \draw [line width=2pt, dashed, green] (4,0) — (5,0);
  \draw [thick,dash dot] (0,1) — (5,1);
  \draw [thick,dash pattern={on 7pt off 2pt on 1pt off 3pt}] (0,2) — (5,2);
\end{tikzpicture}
```



Drawing an arc

```
\begin{tikzpicture}
  \draw [blue,thick,domain=0:270] plot ({5+2.5*cos(\x)}, {1+2.5*sin(\x)});
\end{tikzpicture}
```



Draw a function

```
\begin{tikzpicture}[
    declare function={%
        F(\x) = 3-2*pow(2.7979,-0.8*\x);
    }
]

% Zeichnen der Funktion
\draw [red, line width=2pt, domain=0:4] plot ({\x},{F(\x)}); 

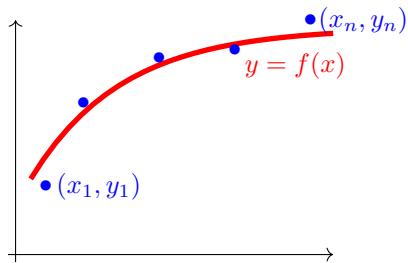
% Bezeichnung
\node [red] (O) at (3.5,2.5) {$y=f(x)$};

% Punkte
\node [blue] (P1n) at (0.9,0.9) {$(x_1,y_1)$};
\node [blue] (P1) at (0.2,0.9) {$\bullet$};

\node [blue] (P2) at (2.7,2.7) {$\bullet$};
\node [blue] (P3) at (1.7,2.6) {$\bullet$};
\node [blue] (P4) at (0.7,2) {$\bullet$};

\node [blue] (P5n) at (4.4,3.1) {$(x_n,y_n)$};
\node [blue] (P5) at (3.7,3.1) {$\bullet$};

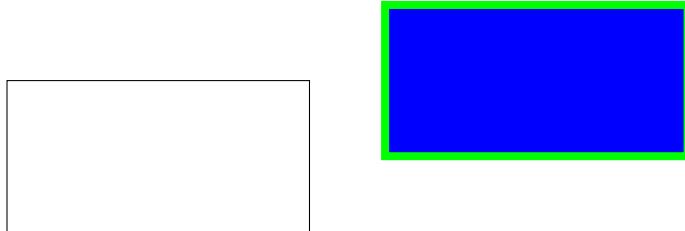
% Koordinatensystem
\draw [color=black,->] (-0.2,-0.1) -- (-0.2,3.1);
\draw [color=black,->] (-0.3,0) -- (4,0);
\end{tikzpicture}
```



Drawing rectangles and moving objects

```
\begin{tikzpicture}
\draw (0,0) -- (4,0) -- (4,2) -- (0,2) -- cycle;

\begin{scope}[shift={(5,1)}]
\draw[green, fill=blue, line width=3pt] (0,0) -- (4,0) -- (4,2) -- (0,2) --
\end{scope}
\end{tikzpicture}
```



Use of variables

```
\begin{tikzpicture}
\pgfmathsetmacro{\PHI}{-15}
% Now use \PHI anywhere you want -15 to appear,
% can also be used in calculations like 2*\PHI
\def\x{10};

\draw[red] (0,4) -- (1-\PHI*0.5,4);
\draw[green] (0,2) -- (1+1/\x,2);
\draw[blue] (0,0) -- ({1+3*(\x/5+1)},0);
\end{tikzpicture}

\bigskip

% \usetikzlibrary{math} %needed tikz library

\begin{tikzpicture}
\pgfmathsetmacro{\drehpunkt}{11.63815573}
%Variables must be declared in a tikzmath environment but
% can be used outside
\tikzmath{\x1 = 1; \y1 = 1;
%computations are also possible
\x2 = \x1 + 1; \y2 = \y1 +3; }
\draw[->] (\x1, \y1)--(\x2, \y2);
\end{tikzpicture}
```

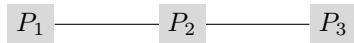


Use of points

```
%\usetikzlibrary{backgrounds} % is needed

\begin{tikzpicture}
    \node [fill=gray!30] (P1) at (0,0) { $P_1$ };
    \node [fill=gray!30] (P2) at (2,0) { $P_2$ };
    \node [fill=gray!30] (P3) at (4,0) { $P_3$ };

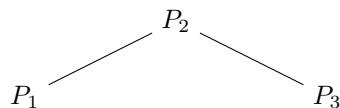
    \begin{scope}[on background layer]
        \draw (P1) -- (P3);
    \end{scope}
\end{tikzpicture}
```



Use of nodes

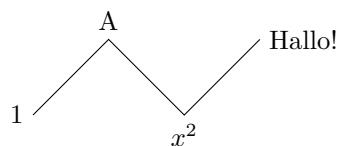
```
\begin{tikzpicture}
    \node (P1) at (0,0){$P_1$};
    \node (P2) at (2,1){$P_2$};
    \node (P3) at (4,0){$P_3$};

    \begin{scope}
        \draw (P1) -- (P2) -- (P3);
    \end{scope}
\end{tikzpicture}
```



Use of nodes

```
\begin{tikzpicture}
    \draw (0, 0) node[left]{1}
          -- +(1, 1) node[above]{A}
          -- +(1,-1) node[below]{$x^2$}
          -- +(1, 1) node[right]{Hallo!};
\end{tikzpicture}
```



B. Criteria for a good L^AT_EX project

A good report is not only characterised by good content, but also fulfils formal aspects. The following list should help to comply with basic rules. Before submitting, all points should be checked and ticked off.

- Is a suitable directory structure used?
- Is an appropriate division into files used?
- Are meaningful directory and file names used?
 - Are directory and file names such as report or term paper avoided?
 - Are meaningful names used for images and not „image1“?
 - Are directory and file names not too long?
 - Are special characters and spaces avoided?
- Are commands like \newline and \\ avoided?
- Are the L^AT_EXfiles clearly laid out?
 - Are indentations used in the L^AT_EXfiles?
 - Are free lines inserted?
 - Is the project mentioned in the header of the files?
 - Does the header of the files mention the main sources?
 - Is the author mentioned in the header of the files?
- Are all necessary files given?
- Are the temporary files deleted?
- Are graphics created by yourself?
- Are the sources of the images given?
- Are citations made with the command \cite?
- Is a bib file used?
 - Are the entries of the bib-file clearly arranged?
 - Are the entries of the bib-file edited to show them correctly?
 - Are the correct types used for the bib entries?
 - Are meaningful keys used in the bib files?

Unfortunately, the list cannot be complete, but it provides some clues.

Literaturverzeichnis

- [Aba+16] M. Abadi et al. “TensorFlow: A System for Large-Scale Machine Learning”. In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA: USENIX Association, Nov. 2016, pp. 265–283. ISBN: 978-1-931971-33-1. URL: <https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi>.
- [Ada] *Adafruit GFX Library*. 2023. URL: <https://github.com/adafruit/Adafruit-GFX-Library> (visited on 06/14/2023).
- [Ame24] Ametherm. *What Is An NTC Thermistor*. letzter Zugriff 23.06.2024. 2024. URL: <https://www.ametherm.com/thermistor/what-is-an-ntc-thermistor>.
- [Arda] *Arduino ABX00031 Nano 33 BLE Sense Module Benutzerhandbuch*. [pdf]. 2022. URL: <https://de.manuals.plus/arduino/abx00031-nano-33-ble-sense-module-manual> (visited on 06/26/2023).
- [Ardb] *Arduino Libraries – LCD_I2C*. 2023. URL: https://www.arduino.cc/reference/en/libraries/lcd_i2c (visited on 07/09/2023).
- [Ardc] *Arduino Libraries – SSD1306Ascii*. 2023. URL: <https://reference.arduino.cc/reference/en/libraries/ssd1306ascii/> (visited on 06/26/2023).
- [Ardd] *Arduino Libraries - Arduino_LPS22HB*. 2024. URL: https://www.arduino.cc/reference/en/libraries/arduino_lps22hb/ (visited on 06/14/2024).
- [Arde] *Arduino Libraries - PDM*. 2023. URL: <https://docs.arduino.cc/learn/built-in-libraries/pdm> (visited on 06/14/2023).
- [Ardf] *Arduino Reference – Interrupt*. 2019. URL: <https://www.arduino.cc/reference/de/language/functions/external-interrupts/attachinterrupt/> (visited on 06/26/2024).
- [Ardg] *Arduino Reference – Wire*. 2022. URL: <https://www.arduino.cc/reference/en/language/functions/communication/wire/> (visited on 06/26/2023).
- [Ardh] *Arduino Reference – Wire.setClock()*. 2022. URL: <https://www.arduino.cc/reference/en/language/functions/communication/wire/setclock/> (visited on 06/26/2023).
- [Ardi] *Arduino Reference – loop()*. 2019. URL: <https://www.arduino.cc/reference/en/language/structure/sketch/loop/> (visited on 06/26/2023).
- [Ardj] *Arduino Reference – setup()*. 2019. URL: <https://www.arduino.cc/reference/en/language/structure/sketch/setup/> (visited on 06/26/2023).
- [Ardk] *Arduino Referenz – pinMode()*. 2019. URL: <https://reference.arduino.cc/reference/de/language/functions/digital-io/pinmode/> (visited on 06/26/2023).

-
- [Ardl] *Arduino Tiny Machine Learning Kit*. [pdf]. 2022. URL: <https://store.arduino.cc/products/arduino-tiny-machine-learning-kit> (visited on 06/23/2023).
- [Ard19] Arduino, ed. *Arduino Library – Kalman*. 2019. URL: <https://www.arduino.cc/reference/en/libraries/kalman/> (visited on 07/09/2023).
- [Ard21a] Arduino 2021, ed. *Arduino OV767X Library for Arduino*. Version 2. 2021. URL: https://github.com/arduino-libraries/Arduino_OV767X/blob/master/examples/CameraCapture/CameraCapture.ino (visited on 04/06/2024).
- [Ard21b] Arduino, ed. *Arduino Nano 33 BLE Sense Rev2 with headers*. 2021. URL: <https://store.arduino.cc/products/nano-33-ble-sense-rev2-with-headers>.
- [Ard21] Arducam, ed. *Featured Camera Modules Supported*. 2021. URL: <https://www.arducam.com/docs/usb-cameras/featured-camera-modules-supported/> (visited on 06/16/2021).
- [Ard21] Arduino, ed. *Check out Arduino Docs!* 2021. URL: <https://www.arduino.cc/en/Guide>.
- [Ard23a] Arduino, ed. *Arduino Nano 33 BLE Sense Rev1 - Product Reference Manual*. [pdf]. 2023. URL: <https://docs.arduino.cc/resources/datasheets/ABX00031-datasheet.pdf>.
- [Ard23b] Arduino, ed. *Arduino Nano 33 BLE Sense Rev2 - Product Reference Manual*. [pdf]. 2023. URL: <https://docs.arduino.cc/resources/datasheets/ABX00069-datasheet.pdf>.
- [Ard24a] ArduCam, ed. *OV7675 VGA Color 20-pin DVP Camera Module for Arduino GIGA R1 WIFI Board*. [pdf]. 2024. URL: <https://www.arducam.com/product/ov7675-vga-color-20-pin-dvp-camera-module-for-arduino-giga-r1-wifi-board/> (visited on 04/05/2024).
- [Ard24a] Arduino, ed. *Getting started with the Arduino Nano 33 BLE Sense*. 2024. URL: <https://wiki-content.arduino.cc/en/Guide/NANO33BLESense>.
- [Ard24b] Arduino Documentation. *Getting Started with Arduino IDE 2*. 2024. URL: <https://docs.arduino.cc/software/ide-v2/tutorials/getting-started-ide-v2/>.
- [Ard24b] Arduino, ed. *Language Reference*. 2024. URL: <https://docs.arduino.cc/language-reference/>.
- [Ard24c] Arduino. *Arducam 0.3MP OV7675*. letzter Zugriff 22.06.2024. 2024. URL: <https://store.arduino.cc/products/arducam-camera-module?queryID=undefined>.
- [Ard24] Arduino, ed. *Arduino APDS9960*. 2024. URL: https://docs.arduino.cc/libraries/arduino_apds9960/ (visited on 09/24/2024).
- [Ari21] D. Aristi. *LCD_I2C library on GitHub*. 2021. URL: https://github.com/blackhack/LCD_I2C (visited on 07/09/2023).
- [Ava15] Avago Technologies, ed. *APDS-9960 - Digital Proximity, Ambient Light, RGB and Gesture Sensor*. [pdf]. 2015. URL: <https://docs.broadcom.com/doc/AV02-4191EN>.
- [Az a] *0,96 Zoll OLED Display - Datenblatt*. [pdf]. AZ-Delivery, 2023. URL: <https://www.az-delivery.de/products/0-96zolldisplay>.

- [Az b] *AZ-Delivery SPI Reader Micro Speicherkartenmodul.* [pdf (de)] [pdf (en)]. AZ-Delivery, 2021. URL: <https://www.az-delivery.de/en/products/terminal-adapter-board-mit-schraubklemmen>.
- [Az c] *Terminal Adapter für Nano V3.* [pdf (de)] [pdf (en)]. AZ-Delivery, 2024. URL: <https://www.az-delivery.de/en/products/terminal-adapter-board-mit-schraubklemmen>.
- [BN11] H. Babovsky and W. Neundorf. *Numerische Approximation von Funktionen.* Tech. rep. TU Ilmenau, 2011. URL: <https://www.tu-ilmenau.de/fileadmin/media/num/neundorf/Dokumente/Preprints/NumApp1.pdf> (visited on 01/13/2017).
- [Bab+23] N. R. Babu et al. “Case Study on Ni-MH Battery”. In: *2023 2nd International Conference on Applied Artificial Intelligence and Computing (ICAAIC)* (2023), pp. 1559–1564.
- [Bai18] J. Baichtal. *10 LED Projects fpr Geeks.* William Pollock, 2018. ISBN: 978-59327-825-0.
- [Bal24] Balluff GmbH. *Farbkorrekturen für natuerliche Farbwiedergaben.* letzter Zugriff 24.08.2024. 2024. URL: <https://www.balluff.com/de-de/whitepapers/farbkorrekturen-fuer-natuerliche-farbwiedergaben>.
- [Bas16] S. Basler. *Encoder und Motor-Feedback-Systeme.* Wiesbaden: Springer Fachmedien Wiesbaden, 2016. ISBN: 978-3-658-12843-2. doi: 10.1007/978-3-658-12844-9. URL: <https://link.springer.com/book/10.1007/978-3-658-12844-9>.
- [Ben17] J. Beningo. “Documenting Firmware with Doxygen”. In: (2017), pp. 121–148. doi: 10.1007/978-1-4842-3297-2_5.
- [Ber18] H. Bernstein. *Elektrotechnik/Elektronik für Maschinenbauer - Einfach und praxisgerecht.* 3rd ed. Berlin Heidelberg New York: Springer-Verlag, 2018, pp. 235–236. ISBN: 978-3-658-20838-7.
- [Blo08] Block, Marco. *Bildverarbeitung.* letzter Zugriff 26.08.2024. 2008. URL: <http://www.inf.fu-berlin.de/lehre/SS08/PI01/Folien11.pdf>.
- [Bosa] *BME280 – Combined humidity and pressure sensor.* [pdf]. Bosch, 2015. URL: https://cdn-reichelt.de/documents/datenblatt/B400/BST-BME280_DS001-10.pdf.
- [Bosb] *BME280 – Integrated Environmental Unit.* [pdf]. Bosch, 2021. URL: https://cdn-reichelt.de/documents/datenblatt/B400/BOSCH_SENSORTEC_FLYER_BME280.pdf.
- [Box21] J. Boxall. *Arduino Workshop - A Hands-On Introduction with 65 Projects.* 2. No Starch Press, 2021. ISBN: 9781593274481.
- [Bro24] Broadcom. *APDS-9960.* 2024. URL: <https://www.broadcom.com/products/optical-sensors/integrated-ambient-light-and-proximity-sensors/apds-9960>.
- [Bul01] Bulova, Viktor. *Farbmodelle.* letzter Zugriff 26.08.2024. 2001. URL: https://homepages.thm.de/~hg10013/Lehre/MMS/SS01_WS0102/Farbmodelle/Kapitel/Kapitel4.html.
- [CJP23] P. Corke, W. Jachimczyk, and R. Pillat. *Robotics, Vision and Control. Fundamental Algorithmus in MATLAB.* 3. Cham, Switzerland: Springer Nature Switzerland AG, 2023. ISBN: 978-3-031-07261.

- [Chi+06] H.-H. Chiang et al. “The Human-in-the-loop Design Approach to the Longitudinal Automation System for the Intelligent Vehicle, TAIWAN iTS-i”. In: *2006 IEEE International Conference on Systems, Man and Cybernetics*. [pdf]. IEEE. 2006, pp. 2839–2844. doi: 10.1109/ICSMC.2006.384384. URL: <https://ieeexplore.ieee.org/abstract/document/4273859>.
- [Con24] Conrad. *Netzwerkschnittstelle ENC28J60 - Datenbaltt.* letzter Zugriff 22.06.2024. 2024. URL: <https://asset.conrad.com/media10/add/160267/c1/-/de/002268123ML01/manual-2268123-tru-components-tc-9072492-rozsirujici-modul-vhodne-pro-vyvojove-sady-arduino.pdf>.
- [DIN66025-2] DIN Deutsches Institut für Normung e.V. *DIN 66025-2:1988-09: Programmablauf für numerisch gesteuerte Arbeitsmaschinen: Wegbedingungen und Zusatzfunktionen*. Norm. Berlin, Sept. 1988.
- [DS24] Datenlogger-Store. *OPUS20 THI Datenlogger*. letzter Zugriff 23.06.2024. 2024. URL: <https://www.datenlogger-store.de/opus20thi-datenlogger.html>.
- [DS83] J. E. J. Dennis and R. Schnabel. *Numerical methods for unconstrained optimization and nonlinear equations*. Englewood Cliffs: Prentice-Hall Series in Computational Mathematics, 1983. ISBN: 978-0898713640.
- [Dat23] Datacolor. *Spyder LensCal*. letzter Zugriff 01.09.2024. 2023. URL: <https://www.datacolor.com/spyder/de/produkte/spyder-lenscal/>.
- [Dat24] Datacolor. *Spyder Checkr 24*. letzter Zugriff 01.09.2024. 2024. URL: <https://www.datacolor.com/spyder/de/produkte/>.
- [Dej18] Dejan. *How to Control Servo Motors with Arduino – Complete Guide*. 2018.
- [Dev24] S. Developers. *Sphinx*. 2024. URL: <https://www.sphinx-doc.org/en/master/> (visited on 12/09/2024).
- [Dha21] P. Dhaker. *So funktioniert das Serial Peripheral Interface*. Link: www.elektronikpraxis.de. letzter Zugriff 22.06.2024. 2021. URL: <https://www.elektronikpraxis.de/so-funktioniert-das-serial-peripheral-interface-a-e33ec4bfff798375112b4d7ea81fc0d1f/>.
- [Din] DIN EN ISO 13850: *Sicherheit von Maschinen - Not-Halt-Funktion - Gestaltungsleitsätze (ISO 13850:2015)*. 2016. URL: <https://www.din.de/de/mitwirken/normenausschuesse/nam/veroeffentlichungen/wdc-beuth:din21:233572513>.
- [Dru19] K. Druckmesstechnik. *Die piezoresistive Druckmesstechnik*. letzter Zugriff 22.06.2024. 2019. URL: <https://keller-druck.com/de/unternehmen/news/die-piezoresistive-druckmesstechnik>.
- [Ene] Energizer *CR2032 – Product Datasheet*. [pdf]. 2018.
- [FAD18] M. Fezari and A. Al Dahoud. “Integrated Development Environment “IDE” For Arduino”. In: (Oct. 2018).
- [FD22] P. Filipi and Dozie. *A Difference between Arduino Nano 33 BLE Sense vs. Arduino Nano 33 BLE Sense Lite*. [pdf]. 2022. URL: <https://forum.arduino.cc/t/a-difference-between-a-n-33-ble-sense-vs-sense-lite/1030305>.
- [FH14] R. Faragher and R. Harle. “An Analysis of the Accuracy of Bluetooth Low Energy for Indoor Positioning Applications”. In: *Proceedings of the 27th International Technical Meeting of The Satellite Division of the Institute of Navigation (ION GNSS+ 2014)*. 2014, pp. 2018–2027.

- [FS17] R. Farouki and J. Srinathu. "A real-time CNC interpolator algorithm for trimming and filling planar offset curves". In: *Computer-Aided Design* 86 (Jan. 2017). doi: 10.1016/j.cad.2017.01.001.
- [Far02] G. Farin. *Curves and Surfaces for CAGD*. 5. [pdf]. San Diego, CA: Academic Press, 2002.
- [Fet21] R. J. Fetick. *Kalman*. 2021.
- [Fou24a] P. S. Foundation, ed. *threading: Thread-based parallelism*. 2024. URL: <https://docs.python.org/3/library/wave.html> (visited on 06/13/2024).
- [Fou24b] P. S. Foundation, ed. *wave: Read and write WAV files*. 2024. URL: <https://docs.python.org/3/library/wave.html> (visited on 06/13/2024).
- [Fun4] *BME280 Luftdruck-, Luftfeuchtigkeits- und Temperatursensor*. 2023. URL: <http://funduino.de/nr-23-bme280-luftdruck-luftfeuchtigkeits-und-temperatursensor>.
- [Funb] *OLED Display SSD1306 128 × 64 / 128 × 32*. 2023. URL: [https://funduino.de/nr-42-oled-display-ssd1306-128x64-128x32](http://funduino.de/nr-42-oled-display-ssd1306-128x64-128x32).
- [GM23] Y. Gitman and J. Murphy. *Heartbeat Sensor Projects with PulseSensor. Prototyping Devices with Biofeedback*. 1st ed. Professional and Applied Computing, Apress Access Books, Professional and Applied Computing (R0). [pdf]. Berkeley, CA: Apress, 2023, pp. XXII, 271. ISBN: 978-1-4842-9324-9 (Softcover), 978-1-4842-9325-6 (eBook). DOI: 10.1007/978-1-4842-9325-6. URL: <https://doi.org/10.1007/978-1-4842-9325-6>.
- [GN01] M. Grossberg and S. Nayar. In: *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001, title = A General Imaging Model and a Method for Finding its Parameters*. Vol. 2. [pdf]. 2001, pp. 108–115. doi: 10.1109/ICCV.2001.937611.
- [GV17] S. Gesenberg and I. Voigt. "Anatomie und Physiologie von Herz und Lunge". In: *Pflegewissen Kardiologie*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2017, pp. 1–24. ISBN: 978-3-662-53979-8. doi: 10.1007/978-3-662-53979-8_1. URL: https://doi.org/10.1007/978-3-662-53979-8_1.
- [GW22] W. Gehrke and M. Winzker. *Digitaltechnik – Grundlagen, VHDL, FPGAs, Mikrocontroller*. Wiesbaden: Springer Vieweg, 2022. ISBN: 978-3-662-63953-5.
- [Gan24] T. Gan. "Capacitive Flexible Pressure Sensors and Their Application". In: *E3S Web of Conferences* 553 (July 2024). [pdf]. doi: 10.1051/e3sconf/202455305038.
- [Gra14] M. Gradias. *Das Nikon D5300 Handbuch*. Heidelberg: dpunkt.verlag GmbH, 2014. ISBN: 978-3-86490-159-1.
- [Gua21] A. Guadalupi. *Machine LEarning Carrier V1.0*. [pdf]. 2021.
- [HEG21] E. Hering, J. Endres, and J. Gutekunst. *Elektronik für Ingenieure und Naturwissenschaftler*. 8. [pdf]. Springer Vieweg, 2021. ISBN: 978-3-662-62697-9. doi: 10.1007/978-3-662-62698-6.
- [HII16] E. Hasche and P. Ingwer. *Game of Colors: Moderne Bewegtbildproduktion - Theorie und Praxis für Film, Video und Fernsehen*. X.media.press. Springer Vieweg Berlin, Heidelberg, 2016. ISBN: 978-3-662-43888-6. doi: 10.1007/978-3-662-43889-3.

-
- [HMS21] E. Hering, R. Martin, and M. Stohrer. *Physik für Ingenieure*. 13th ed. Springer Vieweg, Aug. 2021, p. 548. ISBN: 978-3-662-63177-5.
- [HS23] E. Hering and G. Schönfelder. *Sensoren in Wissenschaft und Technik*. 3. [pdf]. Springer Vieweg, 2023. ISBN: 978-3-658-39490-5. doi: 10.1007/978-3-658-39491-2_978-3-662-62697-9.
- [Hee24a] D. van Heesch. *doxygen – Manual for version 1.12.0*. Version 1.12.0. [pdf]. 2024. URL: <https://www.doxygen.nl/> (visited on 12/09/2024).
- [Hee24b] D. van Heesch. *doxygen*. 2024. URL: <https://www.doxygen.nl/> (visited on 12/09/2024).
- [Hen24] Hengko. *Wie der Feuchtigkeitssensor funktioniert*. letzter Zugriff 23.06.2024. 2024. URL: <https://www.hengko.com/de/news/how-humidity-sensor-works-all-you-should-know>.
- [Hil22] G. Hiller. *Color measurement - the CIE color space*. [pdf]. Datacolor, 2022.
- [Hui] *Technical Data Sheet & Specifications - 10003R1D-EHC-B*. [pdf]. 2017.
- [Hym24] S. Hymel. *APDS-9960 RGB and Gesture Sensor Hookup Guide*. [pdf]. 2024. URL: <https://learn.sparkfun.com/tutorials/apds-9960-rgb-and-gesture-sensor-hookup-guide/all>.
- [Iac16] J. M. Iacobi. *Smartphone-basiertes Bildmesssystem*. Bachelorarbeit im Studiengang Informatik der Fakultät Technik und Informatik. letzter Zugriff 26.08.2024. 2016. URL: file:///C:/Users/manfr/Downloads/BA_Iacobi.pdf.
- [Ibr18] D. Ibrahim. *Motorsteuerung mit Arduino & Raspberry Pi*. Aachen: Elektor, 2018. ISBN: 978-3-895-76336-6.
- [Iot] *BME280 Temperatursensor – Arduino Sketch*. 2020. URL: <https://iotspace.dev/bme280-temperatursensor-arduino-sketch/> (visited on 06/23/0023).
- [JCI14] R. Jana, A. Chowdhury, and S. Islam. “Optical Character Recognition from Text Image”. In: *International Journal of Computer Applications Technology and Research* 3 (Apr. 2014). [pdf], pp. 240–244. doi: 10.7753/IJCATR0304.1009.
- [Jak+10] G. Jaklic et al. “On Interpolation by Planar Cubic G^2 Pythagorean-Hodograph Spline Curves”. In: *Mathematics of Computation* (2010). [pdf].
- [Jam] *Servo High End Micro*. 033212. [pdf]. Jamara e.K., 2018.
- [Jsta] *Corporate Profile*. 2021. URL: <https://www.jst-mfg.com/product/index.php?series=262>.
- [Jstb] *List of JST Connectors Series*. 2020. URL: <http://www.jst.com/home8.html>.
- [Jstc] *PH connector*. [pdf]. J.S.T. Deutschland GmbH, 2022. URL: <https://www.jst-mfg.com/product/index.php?series=199>.
- [Jstd] *VH connector*. [pdf]. J.S.T. Deutschland GmbH, 2021. URL: <https://www.jst-mfg.com/product/index.php?series=262>.
- [KKV14] K. Karvinen, T. Karvinen, and V. Valtokari. *Sensoren – messen und experimentieren mit Arduino und Raspberry Pi*. dpunkt, 2014. ISBN: 97833864901607.
- [Kai09] B. Kainka. *Schnellstart LEDs*. 2. Franzis Verlag GmbH, 2009.

- [Kap24] Kapelan Bio-Imaging GmbH. *Kalibrierservice fuer Graustufen Charts*. letzter Zugriff 26.08.2024. 2024. URL: <https://www.kapelanbio.com/de/loesungen/kalibrierservice-fuer-graustufen-charts/>.
- [Kin] *T-1 3/4 (5mm) – Full Color LED Lamp.*
- [Koe23] Koester, Davis. *Landschaftsfotografie Tipps*. letzter Zugriff 01.09.2024. 2023. URL: <https://www.davidkoester.de/landschaftsfotografie-lernen/landschaftsfotografie-tipps/>.
- [Kru23] Kruse, Tim. *Kalibrierung der Kamera*. letzter Zugriff 29.08.2024. 2023. URL: https://wiki.hshl.de/wiki/index.php/Kalibrierung_der_Kamera.
- [Kue08] N. Kuerten. *Farbkorrektur und Profilierung einer Digitalkamera mittels Approximation*. Diplomarbeit im Fachbereich Photoingenieurwesen und Medientechnik. Koeln: Fachhochschule Koeln, 2008.
- [Küh24] C. Kühnel. *Arduino - Das umfassende Handbuch*. 3rd ed. [pdf]. Rheinwerk Verlag GmbH, 2024. ISBN: 978-3-367-10279-2.
- [Kur21] A. Kurniawan. *IoT Projects with Arduino Nano BLE Sense: Step-By-Step Projects for Beginners*. [pdf]. Berkeley, CA: Apress, 2021. ISBN: 978-1-4842-6457-7. DOI: 10.1007/978-1-4842-6458-4. URL: <https://doi.org/10.1007/978-1-4842-6458-4>.
- [LAH22] C. Liu, M. A. Alif, and G. He. “Shoulder Motion Detection Algorithm Based on MPU6050 Sensor and XGBoost Model”. In: *2022 International Conference on Computing, Communication, Perception and Quantum Technology (CCPQT)*. [pdf]. IEEE. 2022, pp. 1–5. DOI: 10.1109/CCPQT54534.2022.9939285. URL: <https://ieeexplore.ieee.org/document/9939285>.
- [LBSK05] X. Lin, Y Bar-Shalom, and T Kirubarajan. “Multisensor-multitarget bias estimation for general asynchronous sensors”. In: *IEEE Transactions on Aerospace and Electronic Systems* 41.1 (2005). [pdf], pp. 24–45. DOI: 10.1109/TAES.2005.1419586.
- [LP18] R. Lukac and K. N. Plataniotis. *Color Image Processing: Methods and Applications*. Image Processing Series. CRC Press, 2018. ISBN: 9781420009781. URL: <https://books.google.de/books?id=oD8qBgAAQBAJ>.
- [Las] *Interface BME280 Temperature, Humidity and Pressure Sensor with Arduino*. 2023. URL: <https://lastminuteengineers.com/bme280-arduino-tutorial/> (visited on 06/23/0023).
- [Lib21] A. Libraries, ed. *Arduino_LSM6DSOX Library for Arduino*. 2021. URL: https://github.com/arduino-libraries/Arduino_LSM6DSOX (visited on 07/09/2023).
- [Log24] LogiLink, ed. *USB Wall Charger, 10.5W - Datenblatt*. letzter Zugriff 22.06.2024. 2024. URL: https://cdn-reichelt.de/documents/datenblatt/D500/PA0217_DS-EN.pdf.
- [Luf18] Lufft. *Technische Daten OPUS 20 THI. Überwachung von Gebäudeklima und Kontrolle bei allen klimasensitiven Produktionsprozessen in EDV-Rechenzentren, Schalschränken, in Windturbinen, Lagerräumen und Museen*. Datasheet. 2018. URL: <https://www.lufft.com/de-de/produkte/abgekuendigte-produkte-316/opus-20-thi-2242/productAction/outputAsPdf/> (visited on 06/23/2024).

- [M5S21] M5Stack, ed. *Dual-Button*. 2021. URL: docs.m5stack.com/en/unit/dual_button (visited on 05/14/2023).
- [MAB22] J. Martínez, D. Asiain, and J. R. Beltrán. “Self-Calibration Technique with Lightweight Algorithm for Thermal Drift Compensation in MEMS Accelerometers”. In: *Micromachines* 13.4 (2022). [\[pdf\]](#), p. 584. DOI: [10.3390/mi13040584](https://doi.org/10.3390/mi13040584). URL: <https://www.mdpi.com/2072-666X/13/4/584>.
- [Mae24] S. Maetschke. *APDS-9960 Gesture and Color Sensor with Arduino*. [\[pdf\]](#). 2024. URL: <https://www.makerguides.com/apds-9960-gesture-and-color-sensor-with-arduino/>.
- [Mal15] Mallon, Edward and Beddows, Patricia. *How to calibrate a compass (and accelerometer) with Arduino*. accessed date 19.05.2022. 2015. URL: <https://thecavepearlproject.org/2015/05/22/calibrating-any-compass-or-accelerometer-for-arduino/>.
- [McF+23] B. McFee et al. “librosa/librosa: 0.10.2.post1”. Version 0.10.2.post1. In: (2023). DOI: [10.5281/zenodo.11192913](https://doi.org/10.5281/zenodo.11192913).
- [Mye02] Myers, R. D. *Chromaxion: Color Information Tht Measures Up*. [letzter Zugriff 27.08.2024](#). 2002 - 2024. URL: <https://www.chromaxion.com/information/colorchecker.html>.
- [NKG13] A. Noureldin, T. B. Karamat, and J. Georgy. *Fundamentals of Inertial Navigation, Satellite-based Positioning and their Integration*. Springer Science and Business Media LLC, 2013. DOI: [10.1007/978-3-642-30466-8](https://doi.org/10.1007/978-3-642-30466-8).
- [NNK07] V. Ntouskos, I. Kalisperakis, and G. Karras. “Automatic calibration of digital cameras using planar chess-board patterns”. In: *Optical 3-D Measurement Techniques VIII* 1 (Jan. 2007). [\[pdf\]](#).
- [Nto+07] V. Ntouskos et al. “Fully automatic camera calibration using regular planar patterns”. In: *Int. Arch. Photogram. Remote Sens. Spatial Inf. Sci* 37 (Nov. 2007). [\[pdf\]](#).
- [Nto+09] V. Ntouskos et al. “FAUCCAL: an open source toolbox for fully automatic camera calibration”. In: (Jan. 2009). [\[pdf\]](#).
- [Ous18] J. Ousterhout. *A Philosophy of Software Design*. [\[pdf\]](#). Yaknyam Press, 2018. ISBN: 978-1-7321022-0-0.
- [Par16] R. Parthier. *Messtechnik: Grundlagen und Anwendungen der elektrischen Messtechnik*. 8th ed. Führt zur Sachkompetenz. Kontrollfragen mit Antworten und Übungsaufgaben mit Lösungen ermöglichen ein erfolgreiches Selbststudium. Praxisnah und leicht verständlich. Includes supplementary material: <https://sn.pub/extras>. Springer Vieweg, 2016. ISBN: 978-3-658-08120-4. URL: <https://link.springer.com/book/10.1007/978-3-658-08120-4>.
- [Pha06] H. Pham. *PyAudio*. 2006. URL: <https://people.csail.mit.edu/hubert/pyaudio/> (visited on 04/06/2024).
- [Poh17] R. O. Pohl, ed. *Pohls Einführung in die Physik: Band 1: Mechanik, Akustik und Wärmelehre*. 21st ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2017. ISBN: 9783662486634. URL: <http://nbn-resolving.org/urn:nbn:de:bsz:31-epflicht-1552851>.
- [Pro+12] A. Prokos et al. “Automatic calibration of stereo-cameras using ordinary chess-board patterns”. In: vol. XXXIX-B5. [\[pdf\]](#). Aug. 2012. DOI: [10.5194/isprsarchives-XXXIX-B5-45-2012](https://doi.org/10.5194/isprsarchives-XXXIX-B5-45-2012).
- [Qua] *Technical Data Sheet – Round LED 3mm*. [\[pdf\]](#). 2020.

- [RS17a] S. Ramalingam and P. A Sturm. "Unifying Model for Camera Calibration". In: *IEEE Trans Pattern Anal Mach Intell.* 39.7 (2017). [\[pdf\]](#), pp. 1309–1319. DOI: 10.1109/TPAMI.2016.2592904.
- [RS17b] J. Rehder and R. Siegwart. "Camera/IMU calibration revisited". In: *IEEE Sensors Journal* 17.11 (2017). [\[pdf\]](#), pp. 3257–3268. DOI: 10.1109/JSEN.2017.2674307.
- [Raj19] A. Raj. *Arduino Nano 33 BLE Sense Review - What's New and How to Get Started?* 2019. URL: <https://circuitdigest.com/microcontroller-projects/arduino-nano-33-ble-sense-board-review-and-getting-started-guide>.
- [Rei] *Batterieclip für 9-Volt-Block.* 11445. Das Datenblatt liefert allgemeine Informationen zum Batterieclip. Die Höhe des Batterieclips ist nicht aufgeführt. Reichelt Elektronik GmbH. July 2011.
- [Rei24a] Reichelt Elektronik GmbH, ed. *Arduino - Grove Universal-Kabel, 4-Pin, 5cm, fixiert (5er-Pack)*. Online; accessed 17.04.2024. 2024. URL: <https://www.reichelt.de/arduino-grove-universal-kabel-4-pin-5cm-fixiert-5er-pack--grv-cable4pin5fp191140.html>.
- [Rei24b] Reichelt Elektronik GmbH, ed. *Batterieclip für 9-Volt-Block, vertikal*. Online; accessed 17.04.2024. 2024. URL: <https://www.reichelt.de/batterieclip-fuer-9-volt-block-vertikal-clip-9v-p6665.html>.
- [Rei24c] Reichelt Elektronik GmbH, ed. *GRV 4PIN F2GRV Arduino - Grove 4-Pin Female Jumper zu Grove 4-Pin (5er-Pack)*. Online; accessed 06.06.2024. 2024. URL: <https://www.reichelt.de/arduino-grove-4-pin-female-jumper-zu-grove-4-pin-5er-pack--grv-4pin-f2grv-p191166.html>.
- [Roh12] Rohs, Michael and Kratz, Sven. *Übung Computergrafik 2. letzter Zugriff 25.08.2024.* 2012. URL: https://www.medien_ifi.lmu.de/lehre/ss12/cg2/uebung/cg2-ss2012-blatt3.pdf.
- [Run24] M. A. Rundel. *Physik Libre. Freies Physikbuch für die Sekundarstufe II.* OER (), 2024. ISBN: 978-3-8348-2605-3.
- [Rus+07] D. Russell et al. *Apparatus and Methods for 3D Printing*. United States Patent, Patent N0.: US 7,291,002 B2. 2007.
- [Ryb13] J. Rybach. *Physik für Bachelors: mit 92 durchgerechneten Beispielen, 176 Testfragen mit Antworten sowie 93 Übungsaufgaben mit kommentierten Musterlösungen*. Carl Hanser Verlag, 2013. ISBN: 978-3-446-43529-2.
- [ŠDS17] T. Štefanička, R. Ďuračiová, and C. Seres. "Development of a Web-Based Indoor Navigation System Using an Accelerometer and Gyroscope: A Case Study at The Faculty of Natural Sciences of Comenius University". In: *Slovak Journal of Civil Engineering* 25.2 (2017). [\[pdf\]](#), pp. 30–38. DOI: 10.1515/sjce-2017-0005.
- [SIB21] S. Sammito and I. Irina Böckelmann. "Nutzung der Herzschlagfrequenz und der Herzfrequenzvariabilität in der Arbeitsmedizin und der Arbeitswissenschaft". In: *AWMF Online* 2.1 (2021). [\[pdf\]](#), p. 80. URL: https://register.awmf.org/assets/guidelines/002-0421_S2k_Nutzung-Herzschlagfrequenz-Herzfrequenzvariabilität-Arbeitsmedizin-Arbeitswissenschaft_2022-03_1.pdf.
- [SO20] L. Sach and M. O'Hanlon. *Create Graphical User Interfaces with Python*. [pdf]. Raspberry Pi Press, 2020. ISBN: 978-1-912047-91-8.

- [SR14] H. Suesse and E. Rodner. *Bildverarbeitung und Objekterkennung. Computer Vision in Industrie und Medizin*. Wiesbaden: Springer Vieweg, 2014. ISBN: 978-3-8348-2605-3.
- [SS14] B. Sencer and E. Shamoto. “Curvature-continuous sharp corner smoothing scheme for Cartesian motion systems”. In: *2014 IEEE 13th International Workshop on Advanced Motion Control (AMC)*. [pdf] und [Version 2 - pdf]. Piscataway, NJ: IEEE, 2014, pp. 374–379. ISBN: 978-1-4799-2323-6. DOI: \url{10.1109/AMC.2014.6823311}.
- [STM21] STMicroelectronics International NV, ed. *MP34DT05-A – MEMS audio sensor omnidirectional digital microphone*. [pdf]. 2021. URL: <https://www.st.com/resource/en/datasheet/mp34dt05-a.pdf> (visited on 06/23/2024).
- [Sab11] A. M. Sabatini. “Estimating three-dimensional orientation of human body parts by inertial/magnetic sensing”. In: *Sensors* 11.2 (2011), pp. 1489–1525.
- [Sch05] P. Scholz. *Softwareentwicklung eingebetteter Systeme*. 1st ed. Springer, 2005.
- [Sch07] J. Schanda. *Colorimetry: Understanding the CIE System*. Wiley, 2007. ISBN: 9780470175620. URL: <https://books.google.de/books?id=uZadszSGe9MC>.
- [Sch22] T. Scherer. “Batteriemanagement”. In: *Elektor special: Stromversorgung und Batterien* (2022), pp. 6–8.
- [See12] Seeed Technology Co.,Ltd., ed. *Introduction to Grove*. [pdf]. Shenzhen Headquarters, 2012.
- [See13] Seeed Technology Co.,Ltd., ed. *Preface - Getting Started*. [pdf]. Shenzhen Headquarters, 2013.
- [See15] Seeed Technology Co.,Ltd., ed. *Grove - Ear-Clip Heart Rate Sensor*. [pdf]. 2015. URL: https://cdn-reichelt.de/documents/datenblatt/A300/101020033_01.pdf.
- [See16] Seeed Technology Co.,Ltd., ed. *Grove System*. [pdf]. Shenzhen Headquarters, 2016.
- [See24] Seeed Technology Co.,Ltd., ed. *Grove - Ear-clip Heart Rate Sensor*. 2024. URL: https://wiki.seeedstudio.com/Grove-Ear-clip_Heart_Rate_Sensor/ (visited on 09/17/2024).
- [She] *Voltage Sensor / 170640 - Data Sheet*. 170640. [pdf]. Shenzhen Global Tech Co. 2015.
- [Sima] *Joy-IT®Getriebemotor mit Rad*. [pdf]. Simac Electronics GmbH. Apr. 2018.
- [Simb] *Joy-IT®MotoDriver2*. [pdf]. Simac Electronics GmbH. Apr. 2019.
- [Simc] *Joy-IT®Ultrasonic Distance Sensor*. SEN-US01. [pdf]. Simac Electronics GmbH. Aug. 2017.
- [Simd] *KY-016 – RGB LED Modul*. [pdf]. 2020. URL: www.joy-it.net.
- [Sime] *SBC-OLED01 - Datenblatt*. [pdf]. SIMAC Electronics GmbH, 2019. URL: https://cdn-reichelt.de/documents/datenblatt/A300/DEBO_OLED_0-96_DB-DE.pdf.
- [Sim19] Simac Electronics GmbH. *COM-KY040RE-Datenblatt: Drehencoder mit Tasterfunktion*. [pdf]. 2019. URL: www.joy-it.net.

- [Smy21a] R. J. Smythe. *Advanced Arduino Techniques in Science - Refine Your Skills and Projects with PCs or Python-Tkinter*. [pdf]. Berkeley, CA: Apress, 2021. ISBN: 978-1-4842-6786-8. DOI: 10.1007/978-1-4842-6784-4. URL: <https://doi.org/10.1007/978-1-4842-6784-4>.
- [Smy21b] R. J. Smythe. *Arduino in Science - Collecting, Displaying, and Manipulation Sensor Data*. [pdf]. Berkeley, CA: Apress, 2021. ISBN: 978-1-4842-6777-6. DOI: 10.1007/978-1-4842-6777-6.
- [Stma] *LSM6DSOX Datasheet*. [pdf]. 2018. (Visited on 06/23/2023).
- [Stmb] *LSM9DS1 Data Sheets*. 2015. URL: <https://www.st.com/resource/en/datasheet/lsm9ds1.pdf> (visited on 06/11/2022).
- [Stmc] *MP34DT06J – MEMS audio sensor omnidirectional digital microphone*. [pdf]. 2021. URL: <https://www.st.com/resource/en/datasheet/mp34dt06j.pdf> (visited on 06/23/2023).
- [Sto24] P. Stoffregen. *Encoder Library*. Ed. by PJRC. [pdf]. 2024. URL: https://www.pjrc.com/teensy/td_libs_Encoder.html.
- [TM24] P. A. Tipler and G. Mosca. *Tipler Physik*. 9th ed. Springer Spektrum Berlin, May 2024, pp. 423–427. ISBN: 978-3-662-67936-4.
- [TPM14] D. Tedaldi, A. Pretto, and E. Menegatti. “A Robust and Easy to Implement Method for IMU Calibration without External Equipment”. In: *Proceedings of the IEEE International Conference on Robotics and Automation*. [pdf]. IEEE. 2014, pp. 3040–3045. DOI: 10.1109/ICRA.2014.6907275.
- [TR14] H.-R. Tränkler and L. Reindl, eds. *Sensortechnik: Handbuch für Praxis und Wissenschaft*. 2nd ed. VDI-Buch. Published: 13 January 2015. eBook Packages: Computer Science and Engineering (German Language). Berlin, Heidelberg: Springer Vieweg Berlin, Heidelberg, 2014, pp. XIV, 1596. ISBN: 978-3-642-29942-1. DOI: 10.1007/978-3-642-29942-1.
- [The22] The NumPy Community, ed. *NumPy User Guide*. [pdf]. 2022. URL: <https://numpy.org/doc/stable/user/whatisnumpy.html>.
- [Tho23] Thormaelen, Thorsten. *Grafikprogrammierung. Kameras: Perspektivische Projektion*. letzter Zugriff 29.08.2024. 2023. URL: https://www.mathematik.uni-marburg.de/~thormae/lectures/graphics1/graphics_6_1_ger_web.html#1.
- [Tri+22] H. K. Tripathy et al. “Smart COVID-shield: An IoT driven reliable and automated prototype model for COVID-19 symptoms tracking”. In: *Computing* (2022). [pdf], pp. 1–22. DOI: 10.1007/s00607-022-00975-7.
- [Vec] *Inertial Navigation Primer*. <https://www.vectornav.com/resources/inertial-navigation-primer/specifications--and--error-budgets/specs-imuspecs>. [pdf]. 2021.
- [Vis24] VisionDoktor. *Kamera Das Auge des Inspektionssystems. Bayer-Farbinterpolation*. letzter Zugriff 02.09.2024. 2024. URL: <https://www.vision-doctor.com/flaechenkameras/1chip-farbkameras/bayer-farbinterpolation.html>.
- [Voš24] A. Voš. “Volumio mit Drehgebern erweitern”. In: *Make Magazin* (2024). [pdf].

-
- [W3c] *Wire.h (I²C)*. 2023. URL: <https://html.szaktilla.de/arduino/6.html>.
- [Wah+11] W. Wahyudi et al. “Inertial Measurement Unit using multigain accelerometer sensor and gyroscope sensor”. In: *2011 International Conference on Electrical Engineering and Informatics*. [pdf]. IEEE. 2011, pp. 1–6. DOI: 10.1109/ICEEI.2011.6021711.
- [Wan+22] Y. Wang et al. “Multi-position wearable human activity recognition dataset”. In: (2022). DOI: 10.21227/z8bc-pt92. URL: <https://dx.doi.org/10.21227/z8bc-pt92>.
- [Wat17a] Waterloo Maple Inc. 2017. *Description*. letzter Zugriff 14.09.2017. 2017. URL: <https://de.maplesoft.com/support/help/Maple/view.aspx?path=module/description>.
- [Wat17b] Waterloo Maple Inc. 2017. *Export*. letzter Zugriff 14.09.2017. 2017. URL: <https://de.maplesoft.com/support/help/Maple/view.aspx?path=module/export>.
- [Wat17c] Waterloo Maple Inc. 2017. *Module*. letzter Zugriff 14.09.2017. 2017. URL: <https://de.maplesoft.com/support/help/maple/view.aspx?path=module>.
- [Wat17d] Waterloo Maple Inc. 2017. *ModuleLoad*. letzter Zugriff 14.09.2017. 2017. URL: <https://de.maplesoft.com/support/help/Maple/view.aspx?path=ModuleLoad>.
- [Wat17e] Waterloo Maple Inc. 2017. *Option*. letzter Zugriff 14.09.2017. 2017. URL: <https://de.maplesoft.com/support/help/Maple/view.aspx?path=module/option>.
- [Wat17f] Waterloo Maple Inc. 2017. *Procedures*. letzter Zugriff 14.09.2017. 2017. URL: <https://de.maplesoft.com/support/help/Maple/view.aspx?path=procedure>.
- [Wei20] S. Weinzierl. *Handbuch der Audiotechnik*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2020. ISBN: 978-3-662-60357-4. DOI: 10.1007/978-3-662-60357-4.
- [Wil03] S. Wilson. *WAVE PCM soundfile format*. 2003. URL: <https://ccrma.stanford.edu/courses/422-winter-2014/projects/WaveFormat/> (visited on 04/15/2023).
- [Wil22] W. M. Willems. *Lehrbuch der Bauphysik*. 9th ed. Springer Vieweg Wiesbaden, Nov. 2022, pp. 567–568. ISBN: 978-3-658-34093-3.
- [Zaf+20] A. Zafeiropoulos et al. “Benchmarking and Profiling 5G Verticals’ Applications: An Industrial IoT Use Case”. In: *IEEE Conference on Network Softwarization, NetSoft 2020*. 2020. DOI: 10.1109/NetSoft48620.2020.9165393.
- [Zim21] K. Zimmermann. *Fotografieren mit Kompaktkamera und Highend Smartphone*. letzter Zugriff 02.09.2024. 2021. URL: <https://www.idealo.de/magazin/technik/kompakt-und-smartphone-kameras>.

Index

- 3D printer, 563
Programmable Logic Controller, 561
File
 ./images/, 81
 .doxyfile, 76
 .ino, 77
 Adafruit-GFX.h, 314
 Adafruit-SSD1306.h, 314
 ArduCAM, 329, 351
 Arduino, 46
 Arduino LSM9DS1.h, 250
 Arduino_APDS9960, 141, 146
 Arduino_LSM9DS1.h, 249
 ArduinoBLE, 412
 ArduinoBLE.h, 410
 Blink.ino, 560
 blink.ino, 61, 62
 Blink.py, 558
 blnik.ino, 57
 BuiltinLED.cpp, 102
 BuiltinLED.h, 102
 cactus_io_BME280, 304
 cactus_io_BME280_I2C, 304
 camera, 342
 CameraVisualizerHochkant320x240x2.ino, 348
 CaptureSingleHexImage.ino, 427
 ConvertToPNG.py, 343
 datalog.txt, 447
 Datei -> SSD1206Ascii -> HelloWorld-Wire, 305
 DHT.h, 490
 doxygen-1.12.0-setup.exe, 66
 doxygen.exe, 70
 ESP8266, 330, 352
 Fade, 62
 File/Examples/01.Basics, 62
 finalpic.png, 343
 graphviz-12.2.1 (64-bit) EXE installer, 68
 Hello World, 318, 320
 HelloWorldWire, 313
 host_app, 330, 352
 imagefile, 343
 images, 81
 index.html, 79
 Kalman.h, 249, 250
 LED.cpp, 107
 LED.h, 106
 libraries, 48–50
 LinkStatus.ino, 424, 425
 LSM6DSOXSensor.h, 239
 LSM9DS1, 412
 Mag_raw.txt, 92
 memorysaver.h, 330, 352
 Mini_5MP_Plus, 330, 352
 MySketch, 46
 MySketch.ino, 46
 Nano33BLE_Led_A0.ino, 406
 Nano33BLESenseLE, 48
 Nano33BLESenseLED, 49, 50
 output.wav, 182
 PackageExample.py, 525
 PDM.h, 202
 pdm.h, 200, 201
 PowerLED.cpp, 108
 PowerLED.h, 108
 RaspberryPi, 330, 352
 RevC, 330, 352
 RGBLED.cpp, 115
 RGBLED.h, 114
 SD.h, 484
 Shield_V2, 330, 352
 SignsOfLife.cpp, 110
 SignsOfLife.h, 110
 SimpleAccelerometer, 223
 Sketch -> Include Library -> Manage Library, 408
 smoothedAudio.wav, 189
 sound1.wav, 199
 sound2.wav, 199
 sound3.wav, 199
 SSD1306Ascii.h, 225
 TestBattery.ino, 268, 270, 464
 TestCameraRawBites320x240x2.ino, 347
 TestENC.ino, 426
 TestENCheader.ino, 344
 TestENCVoidLoop.ino, 346
 TestENCVoidSetup.ino, 345
 TestHCSR04_UltraschallsensorTest.ino, 454
 TestHCSR04.ino, 456
 UTFT4ArduCAM_SPI, 329, 351

-
- Wire.h, 225, 249, 251, 314
 - wire.h, 250
 - Inertial Measurement Unit
 - see* IMU, 33, 88
 - Acoustic Overload Point
 - see* AOP, 33, 93
 - ADC, 33, 431, 432
 - Analog to Digital Converter
 - see* ADC, 33, 431
 - AOP, 33, 93
 - beats per minute
 - see* bpm, 33, 429
 - bpm, 33, 429
 - CAGD, 563
 - Splines, 563
 - CCD-Sensor, 33, 382
 - Charge-Coupled-Device
 - see* CCD, 33
 - CIE, 33, 387
 - CMOS-Sensor, 33, 382
 - CNN, 33, 561
 - Commission Internationale de l'Eclairage
 - see* CIE, 33
 - Complementary Metal-Oxide-Semiconductor
 - see* CMOS, 33
 - Computerized Numerical Control
 - see* CNN, 33, 561
 - Example, 525
 - Description, 525
 - Installation, 525
 - Introduction, 525
 - General Purpose Input Output
 - see* GPIO, 33, 322
 - GPIO, 33, 322, 453, 454
 - Heart rate
 - see* HR, 33, 429
 - HR, 33, 429
 - I²C, 33, 453, 454
 - IDE, 33, 62, 270, 456, 464, 484
 - IMU, 33, 88, 91
 - Integrated Development Environment
 - see* IDE, 33, 62
 - Inter-Integrated Circuit
 - see* I²C, 33, 453
 - Japan Solderless Terminal
 - see* JST, 33, 282
 - JST, 33, 282
 - LED, 33, 105, 106, 110, 113–115, 117, 119, 120, 285, 289–292, 297–301, 322, 431, 433
 - Brightness, 119, 299
 - Built-in LED, 101
 - Built-in RGB-LED, 113
 - Colors, 116, 300
 - Power LED, 105
 - Standard LED, 291
 - Standard RGB-LED, 297
 - Light Emitting Diode
 - see* LED, 33, 105
 - mcd, 33, 114
 - Millicandela
 - see* mcd, 33, 114
 - PCB, 33, 277
 - PDM, 33, 93, 202
 - Photoplethysmographie
 - see* PPG, 33, 430
 - Pin
 - Pin 10, 458
 - Pin 11, 123, 124, 271, 272
 - Pin 13, 101–103
 - Pin 14, 294
 - Pin 22, 114
 - Pin 22,23,24, 114
 - Pin 23, 114
 - Pin 24, 114
 - Pin 25, 106, 108
 - PLC, *see* Programmable Logic Controller
 - PPG, 33, 430
 - Printed Circuit Board
 - see* PB, 33, 277
 - Pulse Density Modulation
 - see* PDM, 33, 93
 - Pulse with Modulation
 - see* PWM Signal, 33, 106
 - Push Button
 - Built-in Push Button, 123, 271
 - PWM Signal, 33, 106, 113, 117, 291, 297, 300
 - SCL, 33, 453, 454
 - SD, 33, 483, 484
 - SDA, 33, 453, 454
 - Secure Digital
 - see* SD, 33, 483
 - Serial Clock
 - see* SCL, 33, 453
 - Serial Data
 - see* SDA, 33, 453
 - Serial Peripheral Interface
 - see* SPI, 33, 283

Speicherprogrammierbare Steuerung

see SPS, 33, 561

SPI, 33, 283, 483, 484

SPS, 33, 561

TensorFlow

see tf, 33, 483

tf, 33, 483

UART, 33, 453, 454

Universal Asynchronous Receiver Transmitter

see UART, 33, 453

VCC, 33, 268, 453, 454, 463

Voltage at the Common Collector

see VCC, 33, 268

White-point Preserving Least Square

see WPPLS, 33, 388

WPPLS, 33, 388