

Elmar Wings

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

Version: 1765
June 4, 2025

Contents

Contents	3
List of Figures	13
List of Listings	17
1. Introduction	21
1.1. Introduction	21
1.2. Challenges and Solutions	21
1.3. Structure	21
I. Arduino IDE	23
2. Arduino IDE 2.3.x	25
2.1. Arduino IDE Description	25
2.2. Installation of the Arduino IDE	26
2.2.1. Download of the Arduino IDE	26
2.2.2. Summary of Options	26
2.2.3. Installation on Windows	27
2.2.4. Installation (MacOS)	29
2.3. Overview	31
2.4. Features	32
2.4.1. Sketchbook	32
2.4.2. Boards Manager	33
2.4.3. Library Manager	33
2.5. Integrating and Using a Custom Library in Arduino IDE	34
2.5.1. Creating the Library Files	34
2.5.2. Installation and Integration of the Library	34
2.5.3. Using Symbolic Links for Library Integration	34
2.5.4. Windows Tool <code>mklink</code> to Create Symbolic Links	34
2.5.5. MacOS Command <code>ln -s</code> to Create Symbolic Links	36
2.5.6. Verifying Library Availability in the Arduino IDE	36
2.5.7. Serial Monitor	37
2.5.8. Serial Plotter	37
2.6. Examples	37
2.7. Debugging	38
2.8. Autocompletion	38
2.9. Conclusion	39
3. To-Do	41
4. Setup for the Arduino Nano 33 BLE Sense	43
4.1. Introduction	43
4.2. Configuration for the Arduino Nano 33 BLE Sense	43
4.2.1. Installation of the driver	43

4.2.2. Connection and Configuration of the Arduino Board to the Computer	44
4.2.3. Select the Appropriate Port	44
4.2.4. Upload and Verify a Sketch	46
4.3. Test the Configuration	47
4.3.1. Testing the Steps by an Example Sketch <code>blink.ino</code>	47
4.3.2. Description of Basic Sketch for Printing 'Hello'	48
5. Comments with doxygen	51
5.1. Installation	51
5.1.1. doxygen	51
5.1.2. Graphviz	53
5.2. Configuration of Graphviz and doxygen with doxywizard	56
5.2.1. Graphviz	56
5.2.2. DoxyWizard	56
5.2.3. Saving the Configuration	62
5.3. Run of doxygen	62
5.3.1. Run of doxygen using DoxyWizard	62
5.3.2. Run of doxygen using a Command Shell	62
5.4. Syntax and Keywords	63
5.4.1. Keywords for Files	64
5.4.2. Keywords for Functions	65
5.4.3. Keywords for Definitions	65
5.5. Use of Documentation	65
5.6. Add Ons	65
5.6.1. Mainpage	65
5.6.2. Use of simple HTML Commands	66
5.6.3. Inserting Enumerations	66
5.6.4. Inserting Images	67
5.6.5. Inserting HTML pages	67
II. Arduino Nano 33 BLE Sense - Onboard Sensoren	69
6. Arduino Nano 33 BLE Sense	71
6.1. Arduino Nano 33 BLE Sense	71
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	72
6.2.1. What is the difference between Rev1 and Rev2?	72
6.2.2. Changes in the Sketch	73
6.2.3. Arduino Nano 33 BLE Sense Lite	74
6.3. On-Board Sensor Description	74
6.3.1. Gesture, Proximity, and Color Detection Sensor ADPS-9960	76
6.3.2. Accelerometer, Gyroscope, and Magnetometre Sensor LSM9DS1	77
6.3.3. Pressure Sensor LPS22HB	78
6.3.4. Relative Humidity and Temperature Sensor HTS221	79
6.3.5. Digital Microphone MP34DT05-A	79
6.3.6. Bluetooth Module nRF52840	81
6.4. Bluetooth Module INA B306	82
6.5. Arduino Nano 33 BLE Pin Configuration	83
6.6. Was fehlt	83

7. Reset Button on the Arduino Nano 33 BLE Sense	85
7.1. Purpose and Functionality	85
7.1.1. Functions of the Reset Button	85
7.2. Timing and Sequence	86
7.3. Use Cases	86
7.4. Conclusion	86
8. Built-inLED	87
8.1. General Information	87
8.2. Built-in LED	87
8.3. Specification	87
8.3.1. Pin Assignment	87
8.3.2. Power Comsumption	88
8.4. Simple Code	88
8.4.1. Simple Code for the Built-in LED	88
8.4.2. Simple Code for Testing the Built-in LED	89
8.5. Tests	89
8.5.1. Simple Function Test	89
8.5.2. Test all Functions	90
8.6. Simple Application	90
8.7. Further Readings	90
9. Power LED	91
9.1. General Information	91
9.2. Power LED	91
9.3. Specification	92
9.3.1. Pin Assignment	92
9.3.2. Power Comsumption	92
9.4. Simple Code	92
9.4.1. Simple Code for LEDs	92
9.4.2. Simple Code for the Power LED	94
9.4.3. Simple Code for Testing the Power LED	94
9.4.4. Test all Functions	95
9.5. Simple Application	95
9.6. Further Readings	98
10. Built-in RGB-LED	99
10.1. General Information	99
10.2. Specific Sensor	99
10.3. Specification	100
10.3.1. Pin Assignment	100
10.3.2. Power Comsumption	100
10.4. Simple Code	100
10.5. Tests	101
10.5.1. Simple Function Test	101
10.5.2. Test all Functions	102
10.6. Simple Application	106
10.7. Further Readings	107
11. TinyML Shield: Built-in Push Button	109
11.1. General	109
11.2. Built-in Push Button	109
11.3. Specification	110
11.4. Simple Code	110
11.5. Tests	110

11.6. Interrupt Function on the Arduino Nano 33 BLE Sense	112
11.7. Simple Application	112
11.8. Further Readings	114
12. Pressure and Temperature Sensor LPS22HB	115
12.1. General	116
12.2. Specific Sensor	116
12.3. Specification	117
12.4. Library	118
12.4.1. Description	118
12.4.2. Installation	118
12.4.3. Functions	119
12.4.4. Example - Manual	120
12.4.5. Example	120
12.4.6. Example - Code	120
12.4.7. Example - Files	120
12.5. Calibration	120
12.6. Simple Code	122
12.7. Sleep Mode	122
12.8. Simple Application	124
12.9. Tests	124
12.9.1. Simple Function Test	124
12.9.2. Test all Functions	124
12.10 Simple Application	124
12.11 Further Readings	124
13. Inertial Measurement Unit	125
13.1. Introduction	125
13.2. 6-Axis IMU LSM6DSOX	126
13.3. IMU LSM6DSOX Features	126
13.4. IMU LSM6DSOX Data	128
13.4.1. Accelerometer:	128
13.4.2. Gyroscope	129
13.5. Library setup in Arduino IDE	129
13.6. Applications	130
13.7. LSM9DS1(IMU)	130
13.8. Features	130
13.9. Pin description LSM9DS1	131
13.10 Pin connections LSM9DS1	131
13.10.1. Module specifications	133
13.10.2. Block Diagram	134
13.10.3. System Requirements	136
13.10.4. Precautions to be taken	136
13.11 Bibliotheken	137
13.12 Beispiele auf dem Mikrocontroller	137
13.12.1. Testen des Sensors LSM9DS1	137
13.13 Programmierung	137
13.13.1. Programmablaufplan	137
13.13.2. Programmcode und Dokumentation	137
13.13.3. Definition Echtzeit	143
13.14 Kalibrierung	143
13.15 Probleme	144
14. Calibration	145
14.1. Introduction	145

14.2. Standard Operating Procedure	145
14.2.1. Low and high limit method	148
14.2.2. FreeIMU Calibration Application Magnetometer	148
14.2.3. Example	148
15. Errors	151
15.1. Introduction	151
15.2. Affected Parameters	151
15.2.1. Accelerometer:	152
15.2.2. Gyroscope:	152
16. IMU LSM6DSOX - Libraries and Functions	153
16.1. Libraries	153
16.1.1. Wire.h	153
16.1.2. Kalman.h	153
16.1.3. Arduino_LSM6DSOX.h	153
16.1.4. LSM6DSOXSensor.h	153
16.2. Functions	154
16.2.1. setup()	154
16.2.2. loop()	154
16.2.3. IMU.begin()	154
16.2.4. IMU.setAccelerometerRange()	154
16.2.5. IMU.setGyroscopeRange()	154
16.2.6. IMU.setAccelerometerDataRate()	155
16.2.7. IMU.setGyroscopeDataRate()	155
16.2.8. IMU.accelerationSampleRate()	155
16.2.9. IMU.gyroscopeSampleRate()	155
16.2.10. IMU.temperatureAvailable()	155
16.2.11. IMU.readTemperature()	155
16.2.12. IMU.accelerationAvailable()	155
16.2.13. IMU.readAcceleration()	155
16.2.14. IMU.gyroscopeAvailable()	156
16.2.15. IMU.readGyroscope()	156
16.2.16. randomGaussian()	156
16.2.17. kalmanX.setQ(), kalmanY.setQ(), kalmanZ.setQ()	156
16.2.18. kalmanX.setR(), kalmanY.setR(), kalmanZ.setR()	156
16.2.19. kalmanX.update(), kalmanY.update(), kalmanZ.update()	156
16.2.20. kalmanX.getSigma(), kalmanY.getSigma(), kalmanZ.getSigma()	157
16.2.21. delay()	157
16.2.22. lsm6dsosSensor.Set_X_FS()	157
16.2.23. lsm6dsosSensor.Set_G_FS()	157
16.2.24. Initialize_IMU()	158
16.2.25. Factors_Calculation()	158
16.2.26. Factors_Calculation()	158
17. Sensor Inertial Mesure Unit	159
17.1. General Description	159
17.1.1. Always On Mode	159
17.1.2. Tilt detection	159
17.1.3. Significant Motion Detection	160
17.2. Specific Sensor	160
17.2.1. LSM9DS1	160
17.2.2. Accelerometer	160
17.2.3. Gyroscope	161

17.3. Specification	162
17.3.1. Accelerometer Sensor	162
17.3.2. PIN Connction	162
17.4. Library	163
17.4.1. Library Description	163
17.4.2. Installation	164
17.5. Function	164
17.5.1. Library <code>Wire.h</code>	165
17.5.2. Function <code>IMU.begin()</code>	166
17.5.3. Function <code>IMU.end()</code>	166
17.5.4. Function <code>IMU.readAcceleration(x,y,z)</code>	166
17.5.5. Function <code>IMU.readGyroscope()</code>	167
17.5.6. Function <code>IMU.accelerationAvailable()</code>	167
17.5.7. Function <code>IMU.gyroscopeAvailable()</code>	167
17.5.8. Function <code>IMU.accelerationSampleRate()</code>	168
17.5.9. Function <code>IMU.gyroscopeSampleRate()</code>	168
18. Distance, Speed and Acceleration Detection Algorithms	169
18.1. Review of Distance Measurement Methods	169
18.2. Review of Speed and Acceleration Detection Algorithms	170
III. Arduino Nano 33 BLE Sense - External Sensors and Actors	173
19. Tiny Machine Learning Kit	175
19.1. Das Arduino Tiny Machine Learning Shield	175
19.2. Die OV7675 Kamera	176
19.3. USB-Micro-B- auf USB-A-Kabel	177
20. Powering the TinyML Shield	179
20.0.1. USB Power Delivery	179
20.1. Battery	179
20.2. Checking the Battery Voltage	180
20.3. Batterieclip	181
20.4. Spannungssensor	181
20.4.1. Durchführung	182
20.4.2. Ergebnisse	184
21. TinyML Shield: Built-in Push Button	185
21.1. General	185
21.2. Built-in Push Button	185
21.3. Specification	186
21.4. Simple Code	186
21.5. Tests	186
21.6. Interrupt Function on the Arduino Nano 33 BLE Sense	188
21.7. Simple Application	188
21.8. Further Readings	190
22. OLED-Display	191
22.1. OLED	191
22.2. Anschluss	191
22.3. Programmierung	191
22.3.1. <code>Wire.h</code>	191
22.3.2. OLED-Display	193
22.3.3. Testen des OLED-Displays	193

22.4. Software	193
22.4.1. Verwendete Bibliotheken	193
22.4.2. OLED-Display	194
22.4.3. Initialisierung und Setup	194
22.4.4. Messungssteuerung	195
22.4.5. Mikrofondatenverarbeitung	196
22.4.6. Anzeige der Ergebnisse	196
22.4.7. Umrechnung auf den Wert dB SPL	196
22.4.8. Benutzeroberfläche	196
22.5. Das OLED-Display SSD1306	196
22.6. Schaltplan des Aufbaus	197
22.7. Genutzte Bibliotheken	197
22.7.1. Bibliothek Wire.h	198
22.7.2. Bibliothek SSD1306Ascii.h für das Testprogramm	198
22.7.3. Testen des OLED-Displays	198
22.8. Beispiel eines OLED-Displays	200
22.8.1. OLED	201
23. Drehwinkel-Encoder	205
23.1. Encoder	205
23.2. Drehwinkel-Encoder	205
24. Schrittmotor NEMA 17	207
24.1. Beschreibung eines Schrittmotors	207
24.1.1. Aufbau und Funktionsweise eines Schrittmotors	207
24.1.2. Schrittmotor Bauformen	208
24.1.3. Betriebsarten unipolar und bipolar	209
24.1.4. Mikroschrittverfahren	209
24.2. Schrittverluste verhindern	209
24.2.1. Auswahl des Schrittmotors	209
24.2.2. Betriebsart	210
24.2.3. Externe Ereignisse	212
24.3. Beschreibung des verwendeten Schrittmotors	212
24.4. Spezifikation	213
24.5. Anschluss an den DRV8825	213
24.6. Weiterführende Literatur	213
25. Schrittmotortreiber DRV8825	215
25.1. Allgemeine Beschreibung eines Motortreibers	215
25.2. Spezifische Beschreibung des Schrittmotortreibers DRV8825	215
25.2.1. Anschluss des Treibers mit dem Arduino Nano 33 BLE Sense	215
25.3. Spezifikationen	215
25.4. Kalibrierung	215
25.5. Einfaches Beispiel mit Code	217
25.6. Weiterführende Literatur	217
26. Geschwindigkeitssensor LM393	219
26.1. Allgemeine Beschreibung des Sensors	219
26.2. Spezifische Beschreibung des Sensors	219
26.2.1. Funktionsweise der Lichtschranke	219
26.2.2. Signalverarbeitung mit dem LM393 Komparator	220
26.2.3. Anschluss des Sensors mit dem Arduino Nano 33 BLE Sense	220
26.3. Spezifikationen	221
26.4. Bibliothek	221
26.4.1. Installation	221

26.4.2. Code-Beispiel	221
26.5. Kalibrierung	223
26.6. Einfaches Beispiel mit Code	223
26.7. Tests	223
26.8. Weiterführende Literatur	223
27. OLED-Display	225
27.1. Allgemeine Beschreibung eines OLED-Displays	225
27.2. Spezifische Beschreibung OLED-Displays	225
27.3. Anschluss des Sensors mit dem Arduino Nano 33 BLE Sense	225
27.4. Spezifikationen	226
27.5. Bibliothek	226
27.5.1. Installation	226
27.5.2. Code-Beispiel	226
27.6. Einfaches Beispiel mit Code	226
27.7. Weiterführende Literatur	226
28. Sensor XY	229
28.1. General	229
28.2. Specific Sensor/Actor	229
28.3. Specification	229
28.4. Library	229
28.4.1. Description	229
28.4.2. Installation	229
28.4.3. Functions	229
28.5. Example	229
28.5.1. Manual	230
28.5.2. Inside the Example	230
28.5.3. Code	230
28.5.4. Files	230
28.6. Calibration	230
28.7. Simple Code	230
28.8. Simple Application	230
28.9. Tests	230
28.9.1. Simple Function Test	230
28.9.2. Test all Functions	230
28.10 Further Readings	230
29. Actor XY	231
29.1. General	231
29.2. Specific Sensor/Actor	231
29.3. Specification	231
29.4. Library	231
29.4.1. Description	231
29.4.2. Installation	231
29.4.3. Functions	231
29.5. Example	231
29.5.1. Manual	232
29.5.2. Inside the Example	232
29.5.3. Code	232
29.5.4. Files	232
29.6. Calibration	232
29.7. Simple Code	232
29.8. Simple Application	232

29.9. Tests	232
29.9.1. Simple Function Test	232
29.9.2. Test all Functions	232
29.10 Further Readings	232
30. Getting Started	233
30.1. Download and Install the NRF Connect App on Your Mobile	233
30.1.1. For Android	233
30.1.2. For iOS	233
30.2. Power ON the Arduino Nano 33 BLE Sense Lite	234
30.2.1. Connect to Arduino Nano 33 BLE Sense Lite through NRFconnect	234
30.2.2. For Android	234
30.2.3. For iOS	234
IV. Applikation <Title>	237
31. Application	239
32. Conclusion XY	241
33. To DoX Y	243
V. Templates	245
34. Sensor	247
34.1. General	247
34.2. Specific Sensor	247
34.3. Specification	247
34.4. Library	247
34.4.1. Description	247
34.4.2. Installation	247
34.4.3. Functions	247
34.4.4. Example's Manual	247
34.4.5. Inside the Example	247
34.4.6. Example's Code	247
34.4.7. Example's Files	247
34.5. Calibration	247
34.6. Simple Code	248
34.7. Simple Application	248
34.8. Tests	248
34.8.1. Simple Function Test	248
34.8.2. Test all Functions	248
34.9. Simple Application	248
34.10 Further Readings	248
35. Package Example	249
35.1. Introduction	249
35.2. Description	249
35.3. Installation	249
35.4. Example - Manual	249
35.5. Example	249
35.6. Example - Code	249
35.7. Example - Files	249

35.8. Further Reading	249
VI. Anhang	251
36. Materialliste	253
VII. Hints - Examples for LaTeX - Read and Use	255
37. Hints	257
37.1. Allgemeine mathematische Beschreibung Bézier-Kurve	258
38. Verrundung mit einem Kreisbogen	261
38.1. Gleichungen	261
38.2. Bewertung	263
38.3. Examples	264
39. Maple files	267
39.1. Geometries with Maple	267
39.2. Geometry elements used	267
39.3. Building the data structure	268
39.4. Structure of a module	268
39.4.1. General structure of a module	270
39.4.2. Saving a module	271
39.4.3. Verwendung eines Moduls	271
39.4.4. Creating a Maple module	271
39.5. Functions of the modules	272
39.5.1. Module MPoint	272
39.5.2. Module MLine	273
39.5.3. Module MArc	273
39.5.4. module MBezier	274
39.5.5. Module MPolygon	274
39.5.6. module MGeoList	274
39.5.7. Module MHermiteProblem	275
39.5.8. Module MHermiteProblemSym	275
39.5.9. Module MConstant	276
39.5.10. Module MGeneralMath	276
39.5.11. Module Biarc	277
39.5.12. Module MBiarc	277
39.6. Programme Flowchart	278
39.6.1. Overall Flow	278
39.6.2. Verrundung der Kurve	278
40. Representation of Python Programs	281
41. Representation of Programs written for Arduino Boards	283
42. First Chapter	285
43. CAGD	287
A. drawings with tikz	289
B. Criteria for a good L^AT_EX project	293

Contents	13
Literaturverzeichnis	295
Stichwortverzeichnis	301
Index	301

List of Figures

2.1. Menu Button	25
2.2. Arduino IDE 2.3.3 Download	26
2.3. Summary of DownloadOptions	27
2.4. Arduino IDE Create Agent Installation	27
2.5. Arduino Setup Installation options	28
2.6. Arduino Setup Installation Folder	28
2.7. Arduino IDE Sketch	28
2.8. Steps for Installation	29
2.9. ArduinoIDE Create Agent Installation	29
2.10. Open the Download folder	30
2.11. Copy to the Applications Folder	30
2.12. Arduino IDE Sketch	31
2.13. Overview	32
2.14. Sketchbook	32
2.15. Board Manager	33
2.16. Library Manager	33
2.17. Command Prompt with Administrator Privileges	35
2.18. Command Prompt	35
2.19. Including the Nano33BLESenseLED Library	36
2.20. Serial Monitor	37
2.21. Examples	38
2.22. Debugging Example	38
2.23. Autocomplete	39
2.24. Menu Bar Option	39
4.1. Arduino Mbed OS Nano Boards Installation	44
4.2. Select the Connected board - here Arduino Nano 33 BLE Sense	44
4.3. Arduino Nano 33 BLE Sense Reset Button	45
4.4. green and orange Builtin-LED	46
4.5. Select Available Port for Uploading Arduino Sketch	46
4.6. Upload the Program in Arduino board	47
4.7. Setting the Port	47
4.8. LED Example Sketch	48
4.9. LED-Built Example	48
5.1. Exmaple output of Graphviz	51
5.2. Start image from the website https://doxygen.nl/	52
5.3. Download area from the website https://doxygen.nl/	52
5.4. Query after the complete installation during the installation process of doxygen	53
5.5. Query after the complete installation during the installation process of doxygen	53
5.6. Start image from the website https://www.graphviz.org/	54
5.7. Download area from the website https://www.graphviz.org/download/	54
5.8. Query for adding the Graphviz path to the system variable <code>PATH</code>	55
5.9. Query for the path to Graphviz	55

5.10. Query for the start menu of Graphviz	56
5.11. DoxyWizard tab “Wizard” - Project	57
5.12. DoxyWizard tab “Wizard” - Mode	58
5.13. DoxyWizard tab “Wizard” - Diagrams	58
5.14. DoxyWizard tab “Wizard” - Output	59
5.15. DoxyWizard tab “Wizard” - Expert - Build	60
5.16. DoxyWizard tab “Wizard” - Expert - Dot	61
5.17. DoxyWizard tab “Wizard” - Run	62
6.1. Arduino Nano 33 BLE Sense Rev. 2, see Arduino Store	72
6.2. Arduino Nano 33 BLE Sense Rev. 1	73
6.3. Arduino Nano 33 BLE Sense Lite	74
6.4.	75
6.5. Pin assignment of the Arduino Nano 33 BLE Sense	76
6.6. Circuit diagram microphone	81
6.7. Connection with BLE and smartphone	82
6.8. Simple sketch to seen Arduino battery	82
6.9. Arduino Nano 33 BLE Pin Configuration	84
7.1. Arduino Nano 33 BLE Sense’s Reset Button	85
8.1. Arduino Nano 33 BLE Sense’s built-in LED with Pin 13	87
9.1. Arduino Nano 33 BLE Sense’s Power LED with Pin 25	91
10.1. Arduino Nano 33 BLE Sense’s RGB LED with Pin 22, Pin 23, and Pin 24	99
11.1. The Tiny Machine Learning Shield	109
12.1. Arduino Nano 33 BLE Sense’s pressure and temperature sensor LPS22HB115	
13.1. Examples in the library <i>Arduino_LSM9DS1</i>	125
13.2. <i>Axis Acceleration of Accelerometer Sensor</i>	128
13.3. <i>Angular Speed of Gyroscope Sensor</i>	129
13.4. Library setup in Arduino IDE	129
13.5. Pin connections LSM9DS1	131
13.6. Pin description LSM9DS1	132
13.7. Sensor Characteristics	133
13.8. Temperature Sensor Characteristics	133
13.9. Absolute Maximum Temperature	134
13.10 Accelerometer and gyroscope block diagram	134
13.11 Magnetometer block diagram	135
13.12 LSM9DS1 electrical connections	136
13.13Output des Testprogramms	137
13.14Der Programmablaufplan	138
13.15Anzeige des Arduino OLED Displays sobald der Arduino eingeschaltet ist	143
13.16Ausgabe von Messdaten	143
13.17Der Reset Button des Arduino	144
17.1. LSM9DS1 axis on this card	161
17.2. LSM9DS1 axis on this card	161
17.3. Explication of pin mode	163
17.4. Wire include	164
17.5. Board card installation	164
17.6. Library choice	164

18.1. Design of OAK-D Pro camera	170
19.1. Das Tiny Machine Learning Kit	175
19.2. Das Tiny Machine Learning Shield	176
19.3. Pin-Belegung des Tiny Machine Learning Shields	176
19.4. Das OV7675 Kameramodul	177
19.5. Ein USB-A auf Micro-USB Kabel	177
20.1. Montage des Clips	180
20.2. Komplettes System mit Batterie	180
20.3. Batterieclip für 9-Volt-Block[Rei24]	182
20.4. Voltage Sensor 170640 von dem Hersteller <i>Shenzhen Global Technology Co., Ltd</i> [She]	182
20.5. Testoutput des Spannungssensors	184
21.1. The Tiny Machine Learning Shield	185
22.1. Gesamter Schaltplan	192
22.2. Beispiele in der OLED Bibliothek	193
22.3. Testausgabe des OLED Display	194
22.4. Code Einlesen/Zählen	195
22.5. Code Messungssteuerung	195
22.6. Code	197
22.7. Pins des OLED-Displays	197
22.8. Stationärer Aufbau der Wetterstation	198
22.9. Pfad des Testprogramms	199
22.10. Erste Ausgabe Display	200
22.11. Das Display zur Ausgabe der Messwerte	201
24.1. Prinzipieller Aufbau Schrittmotor (2-phasisig) [Hagl.2021]	208
24.2. Start-Stopp Frequenz [FaulhaberDriveSystems.2020]	210
24.3. Trapezförmiges Geschwindigkeitsprofil [FaulhaberDriveSystems.2020]	211
26.1. Anschluss des Sensors ans den Arduino Nano 33 BLE Sense	220
26.2. Aufrufen der Bibliotheken in der Arduino IDE	221
26.3. Installation der TimerOne Bibliothek	222
26.4. Aufrufen der Beispiele für die TimerOne Bibliothek	222
26.5. wird noch in Code umgewandelt!	224
27.1. Installation der U8g2 Bibliothek	227
27.2. wird noch in Code umgewandelt!	227
30.1. NRF Connect Application	233
30.2. Landing page	234
30.3. List of devices	235
37.1. Bernstein polynomial of degree 3	259
37.2. Bézier curve for example 37.1	260
38.1. Smoothing a corner with the help of an arc - triangle	261
38.2. Angular change with blending arc	263
38.3. Curvature progression for a blending arc	263
38.4. Maximum range for a blending arc of a circle - $L(\varepsilon = \text{const}, \alpha)$	264
38.5. Deviation when specifying the distance L when rounding with a circular arc	264

39.1. Programme flow chart „Corner rounding“	279
39.2. Programme flowchart „Selection of the rounding strategies“	280

List of Listings

6.1. Simple example using of the builtin microphone of the Arduino Nano 33 BLE Sense	80
8.1. Header file without comments for using the Built-in LED	88
8.2. Code file without comments for using the Built-in LED	88
8.3. Simple sketch without comments to control the built-in LED	89
8.4. Simple sketch to test the built-in LED	89
8.5. A simple watch dog: A simple watchdog: the built-in LED is switched on for 1 second every 30 seconds.	90
9.1. Header file without comments for using a LED	93
9.2. Code file without comments for using a LED	93
9.3. Header file with doxygen comments for using the Power LED	94
9.4. Code file with doxygen comments for using the Power LED	94
9.5. Simple sketch to control the power LED	94
9.6. Simple sketch to check the battery state using the power LED	95
9.7. Simple code for signs of life	96
9.8. Simple code for signs of life	96
9.9. Simple sketch to check the battery state using the power LED	97
10.1. Header file without comments for using the RGB-LED	100
10.2. Code file without comments for using the RGB-LED	101
10.3. Simple sketch to test the RGB LED	101
10.4. value between 0 and 255 to write to the RGB LED	103
10.5. Different brightness levels for the RGB-LED colors	105
10.6. A simple watch dog: A simple watchdog: the built-in RGB-LED is switched on for 1 second every 30 seconds.	106
11.1. Defining the built-in button's pin as an input.	110
11.2. Read the built-in button's state	110
11.3. Simple sketch to test the push button and the built-in LED	110
11.4. Simple sketch connects the push button with an interrupt.	112
12.1. Example code for the sensor LPS22HB on the Arduino Nano 33 BLE Sense	118
12.2. Simple sketch calibrating the sensor LPS22HB	120
12.3. Sketch for the Arduino Nano 33 BLE Sense to switch the sensor LPS22HB into sleep mode	122
14.1. Example Microphone	149
14.2. Example IMU (Accelerometer and Gyroscope)	149
14.3. Example ADPS-9960	150
17.1. Simple sketch using the sensor LSM9DS1 detection	165
20.1. Beispiel-Sketch zum Testen der Batterie	183
21.1. Defining the built-in button's pin as an input.	186

21.2. Read the built-in button's state	186
21.3. Simple sketch to test the push button and the built-in LED	186
21.4. Simple sketch connects the push button with an interrupt.	188
22.1. Monitoring	195
22.2. Start and stop sampling	196
22.3. Conversion of the microphone signal	196
22.4. OLED-Aktualisierung	196
22.5. Simple program for OLED displays	200
23.1. Testprogramm für den Encoder	206
40.1. „Hello World“ in Python – Variant 1	282
41.1. „Hello World“ in Python – Variant 1	284

Acronyms

AOP Acoustic Overload Point

CNC Computerized Numerical Control

GND Ground

GPIO General Purpose Input Output

IDE Integrated Development Environment

IMU Inertial Measurement Unit

IoT Internet of Things

LED Light Emitting Diode

mcd Millicandela

PDM Pulse Density Modulation

PWM Pulse with Modulation

RGB Rot-Grün-Blau

SPS Speicherprogrammierbare Steuerung

VCC Voltage at the Common Collector

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.

WS: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 download page for Arduino IDE 2.3.3. 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." Below this, a link to "Arduino IDE 2.0 documentation" is provided. Further down, a note about "Nightly builds" is shown. At the bottom left, there's a "SOURCE CODE" link pointing to GitHub. On the right side, 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

At the very bottom of the sidebar, there's a link to "Release Notes".

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**.

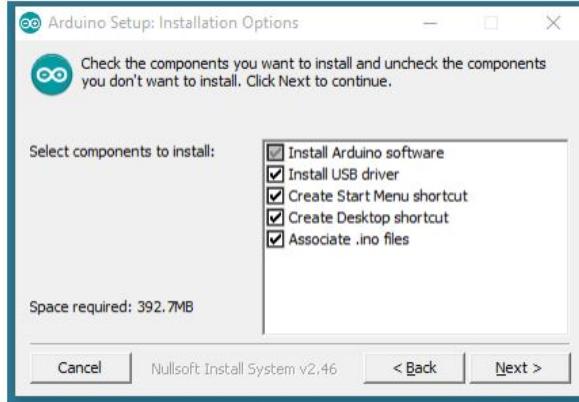


Figure 2.5.: Arduino Setup Installation options

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

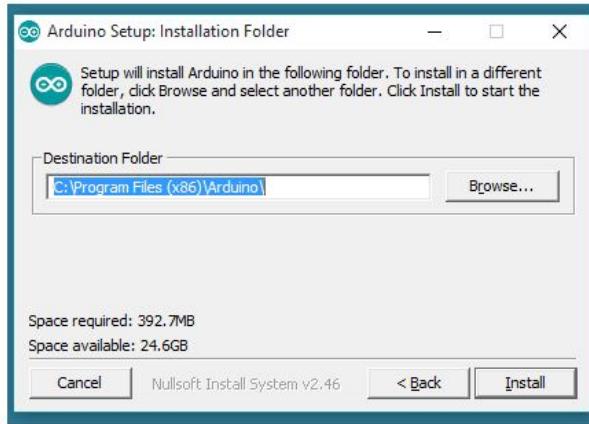


Figure 2.6.: Arduino Setup Installation Folder

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

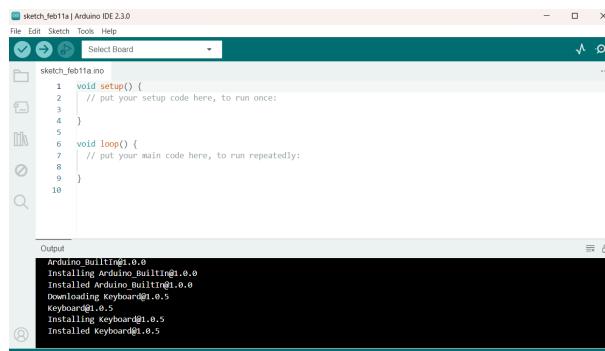


Figure 2.7.: Arduino IDE Sketch

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.

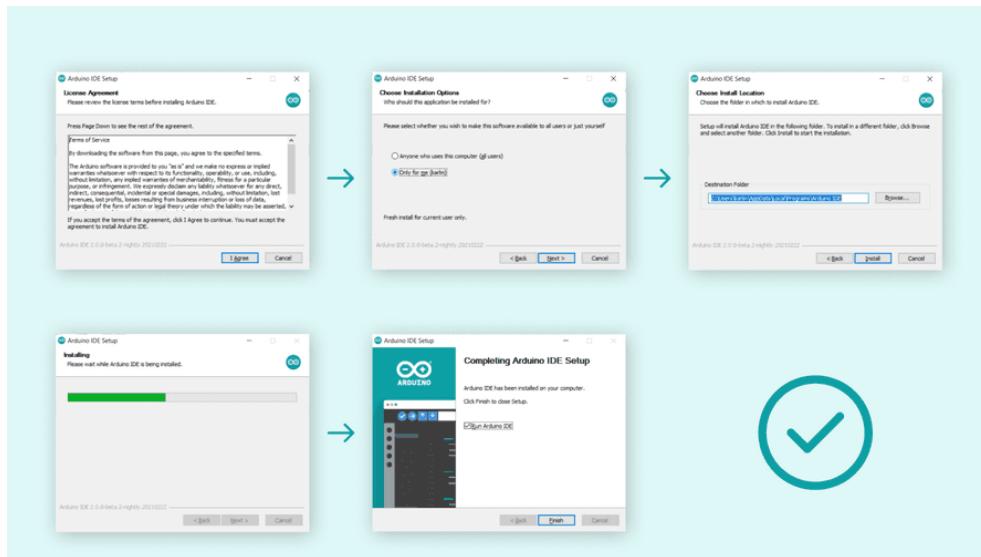


Figure 2.8.: Steps for Installation

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

WS: Die Installation muss aktualisiert werden, die folgenden Bilder müssen mit einem Mac obwohl aktualisiert

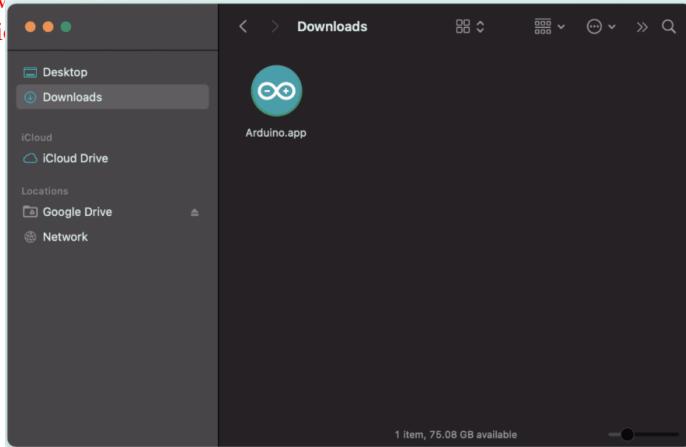


Figure 2.10.: Open the Download folder

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

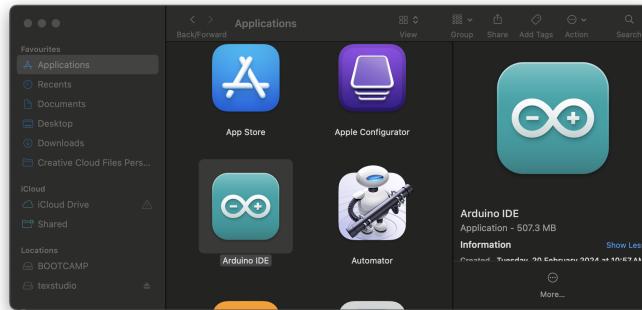


Figure 2.11.: Copy to the Applications Folder

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`.

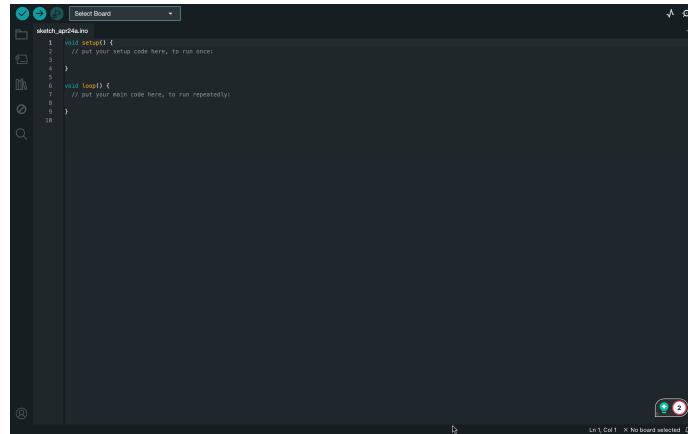


Figure 2.12.: Arduino IDE Sketch

2.3. Overview

The Arduino IDE 2 features a new sidebar, making the most commonly used tools more accessible. 2.13 [Ard24a]

- **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.

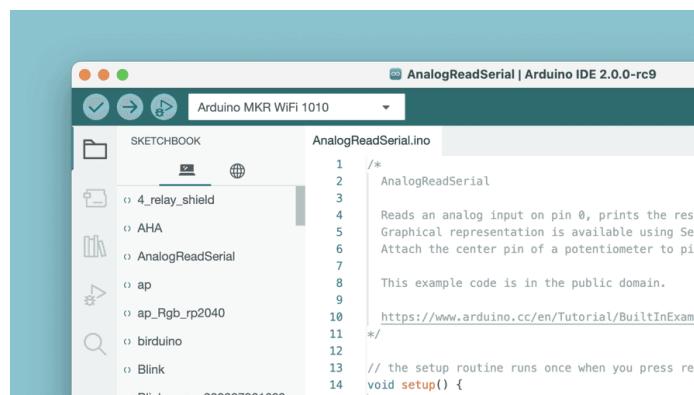


Figure 2.14.: Sketchbook

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.

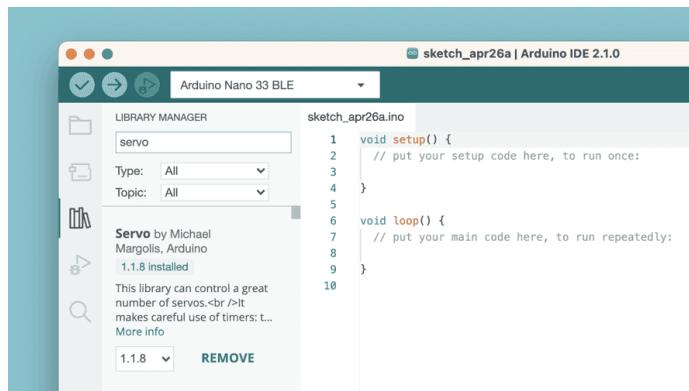


Figure 2.16.: Library Manager

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

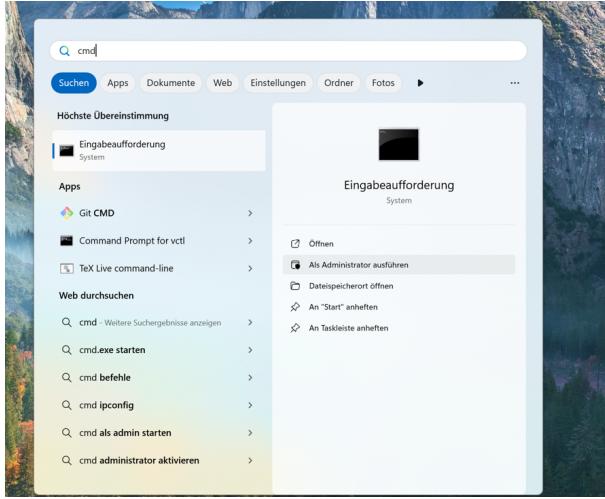


Figure 2.17.: Command Prompt with Administrator Privileges

- 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.

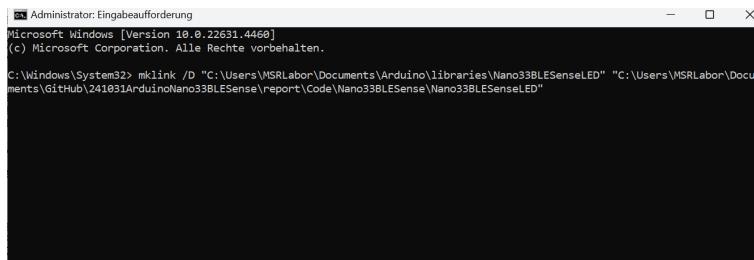


Figure 2.18.: Command Prompt

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.

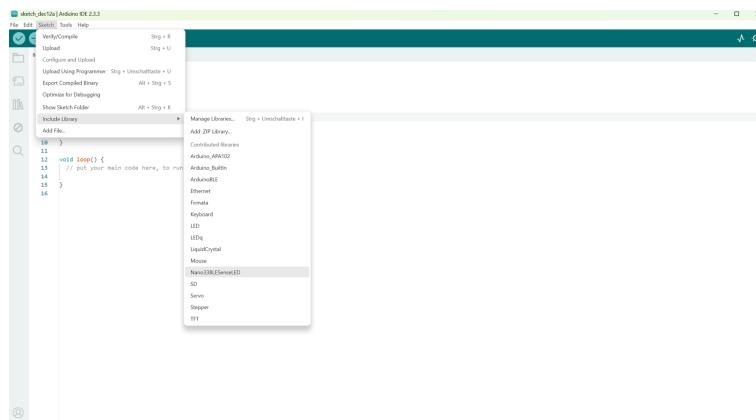


Figure 2.19.: Including the Nano33BLESenseLED Library

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.

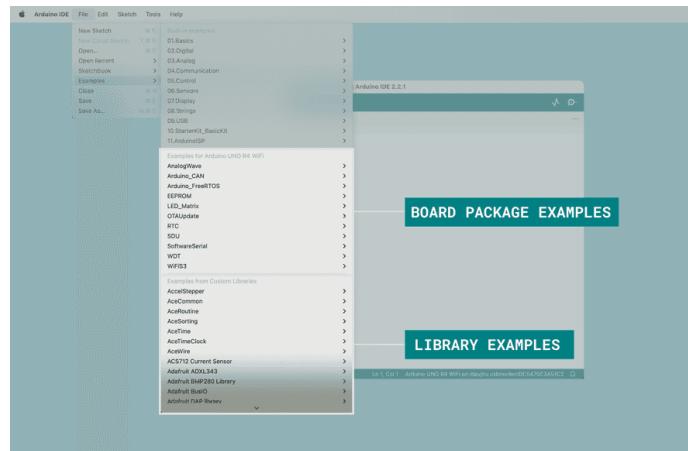


Figure 2.21.: Examples

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

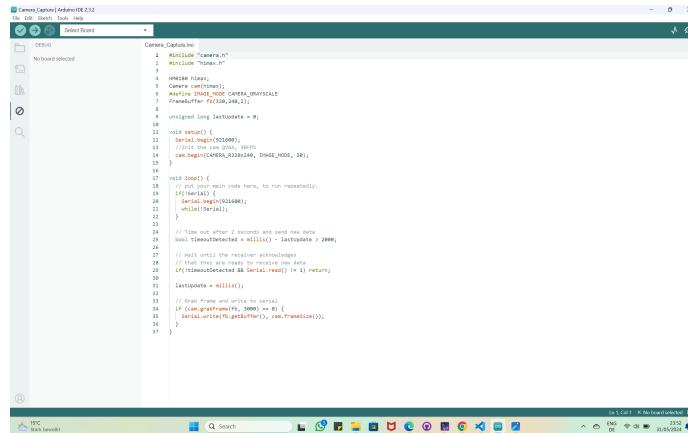


Figure 2.22.: Debugging Example

WS: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

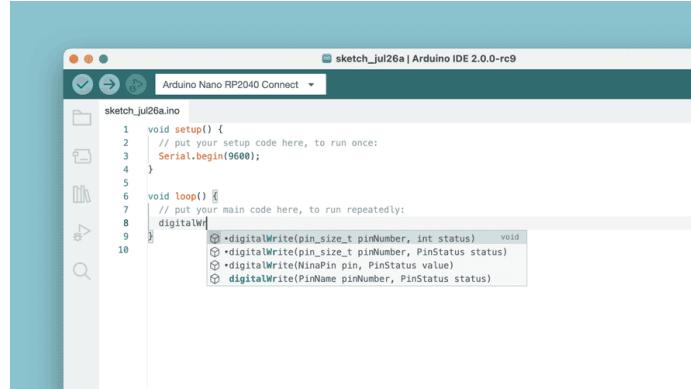


Figure 2.23.: Autocomplete

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.

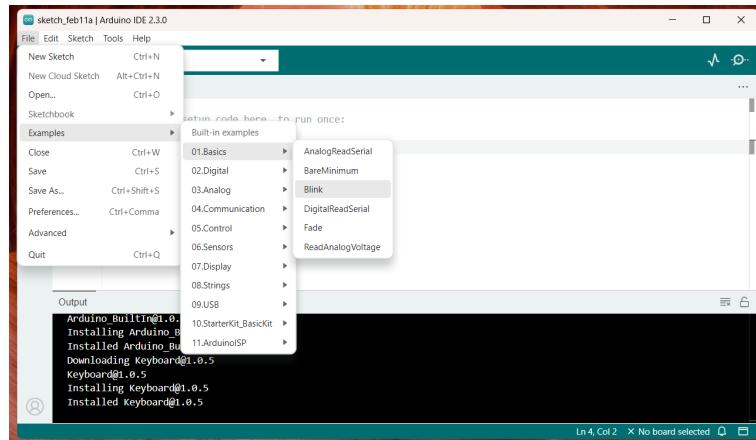


Figure 2.24.: Menu Bar Option

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.

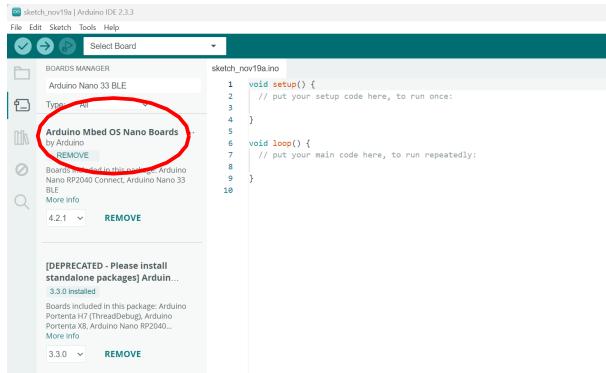


Figure 4.1.: Arduino Mbed OS Nano Boards Installation

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.

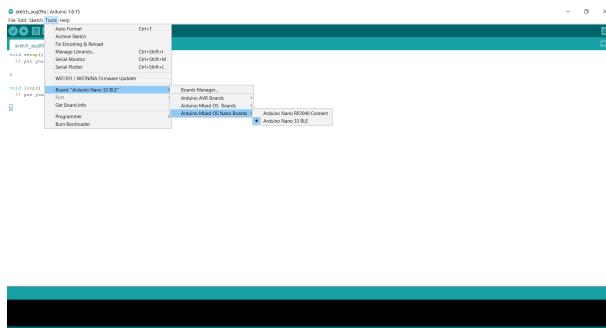


Figure 4.2.: Select the Connected board - here Arduino Nano 33 BLE Sense

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.

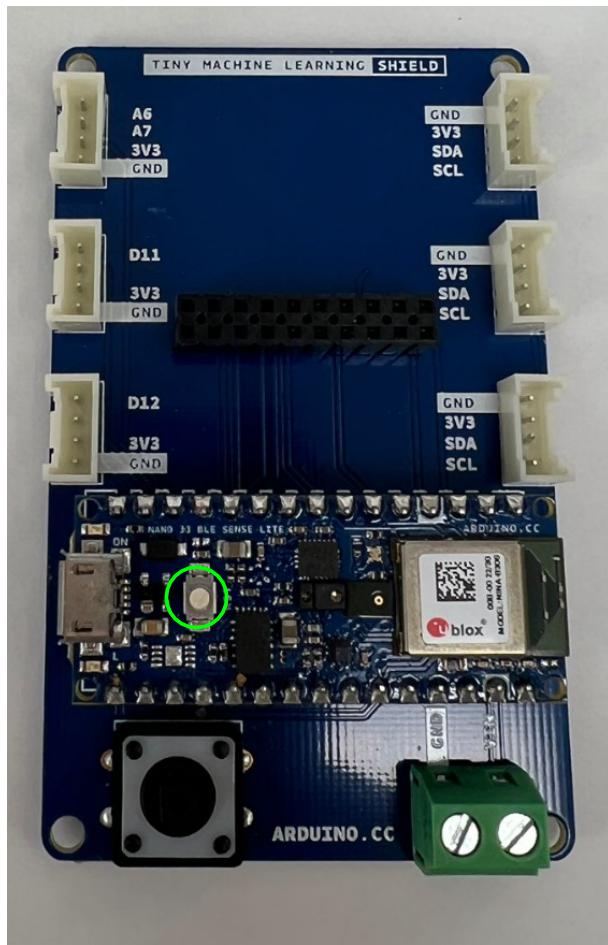


Figure 4.3.: Arduino Nano 33 BLE Sense Reset Button

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.

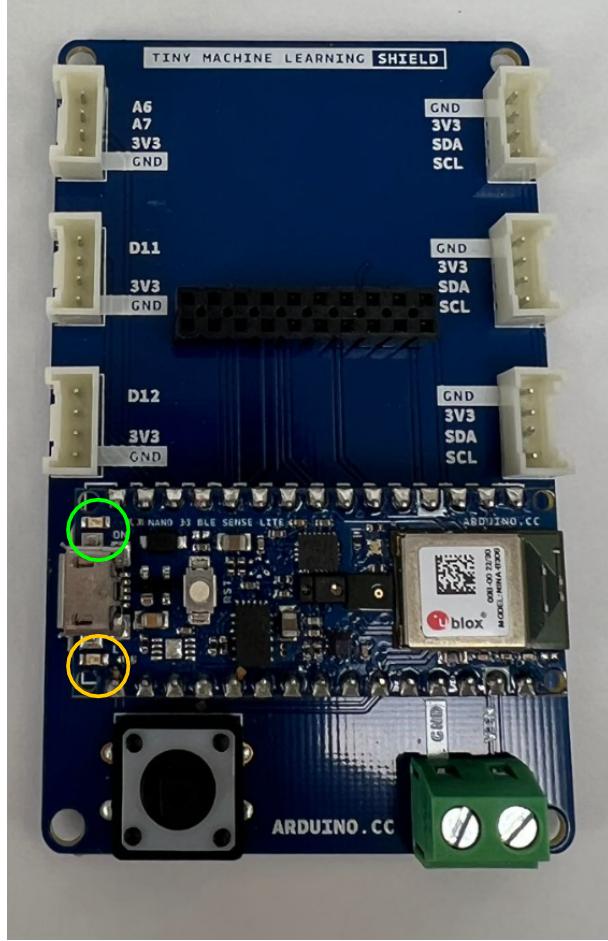


Figure 4.4.: green and orange Builtin-LED

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

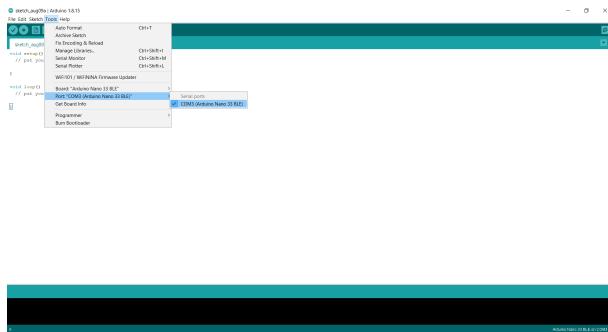


Figure 4.5.: Select Available Port for Uploading Arduino Sketch

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.

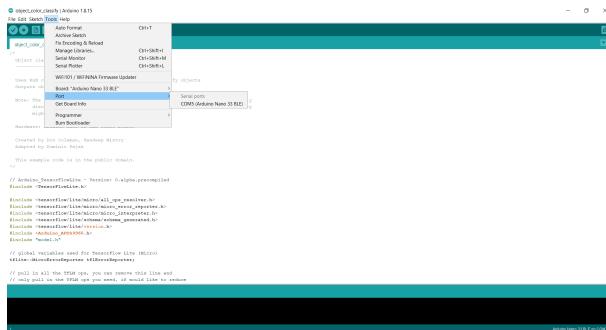


Figure 4.7.: Setting the Port

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.

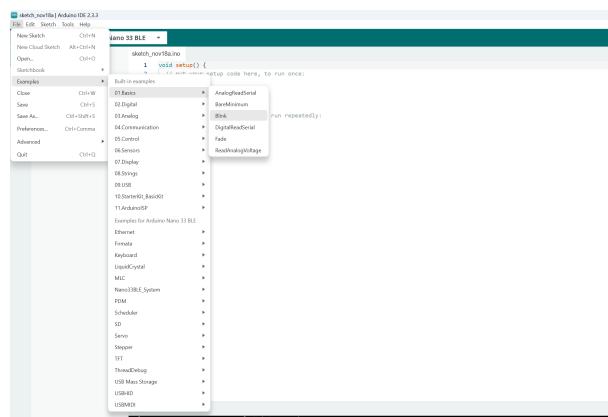


Figure 4.8.: LED Example Sketch

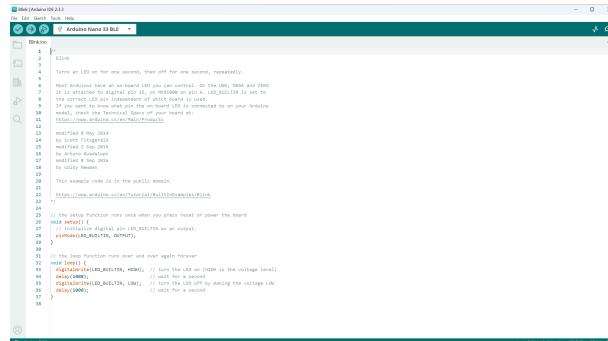


Figure 4.9.: LED-Built Example

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.

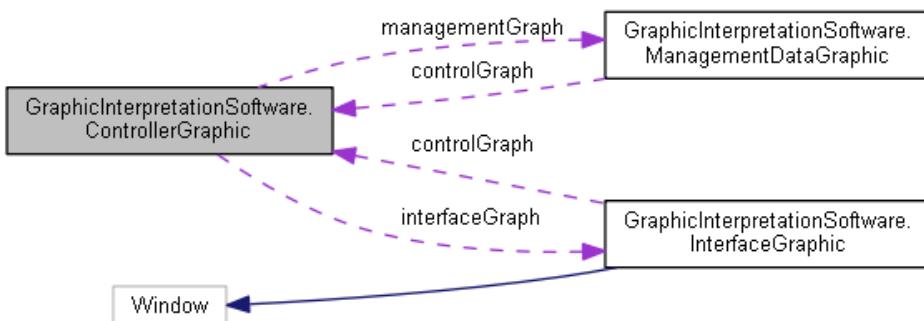


Figure 5.1.: Example output of Graphviz

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.

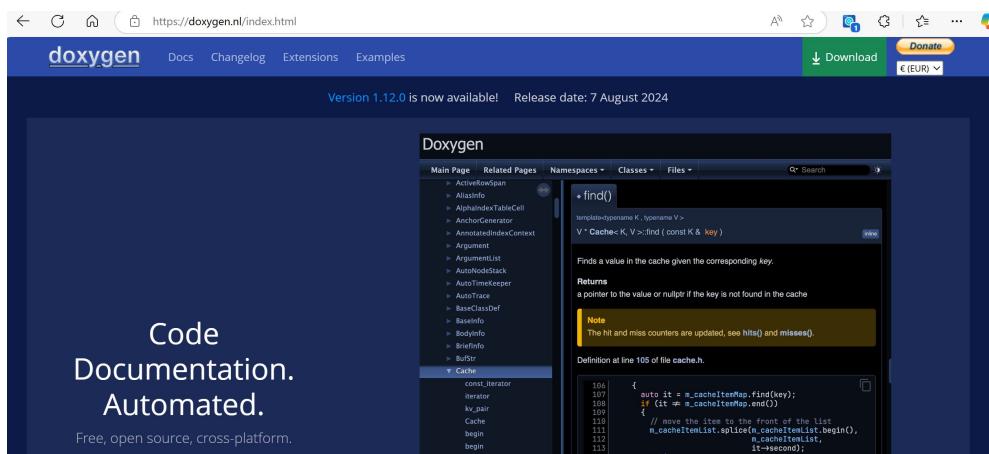


Figure 5.2.: Start image from the website <https://doxygen.nl/>

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.

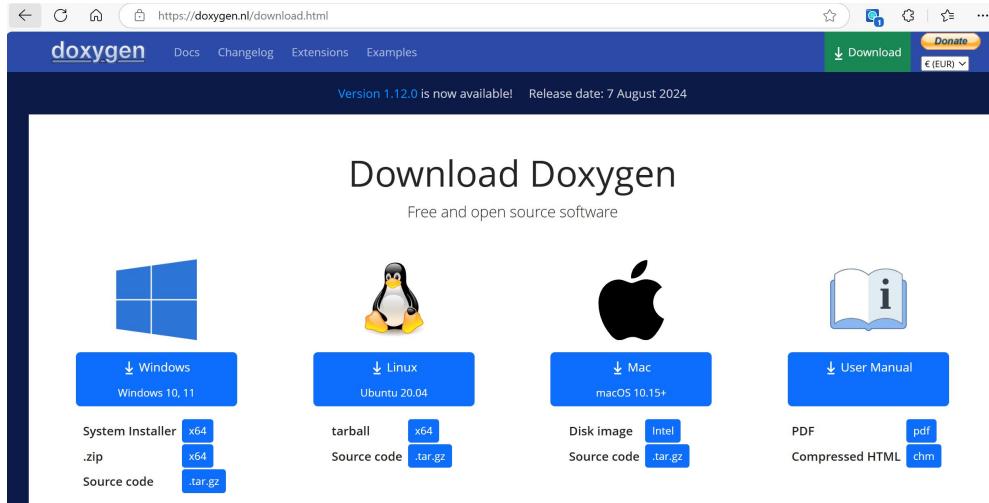


Figure 5.3.: Download area from the website <https://doxygen.nl/>

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.

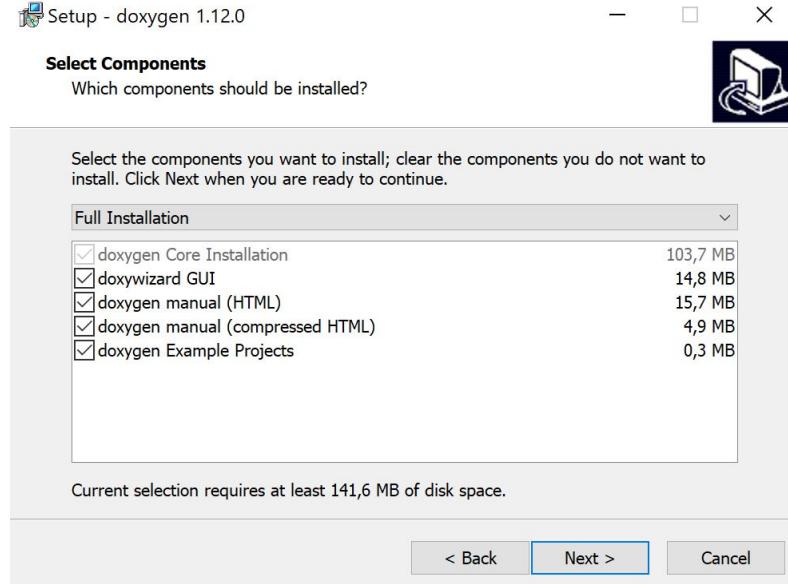


Figure 5.4.: Query after the complete installation during the installation process of doxygen

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

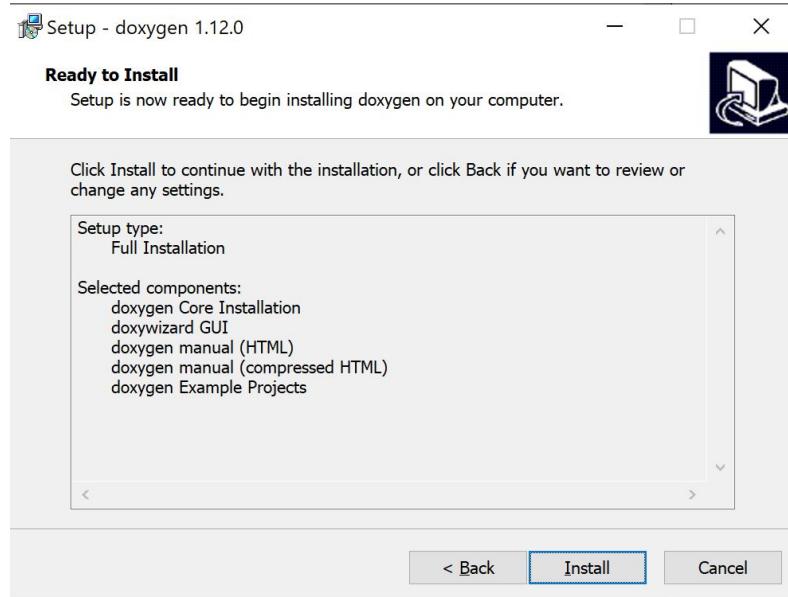


Figure 5.5.: Query after the complete installation during the installation process of doxygen

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.

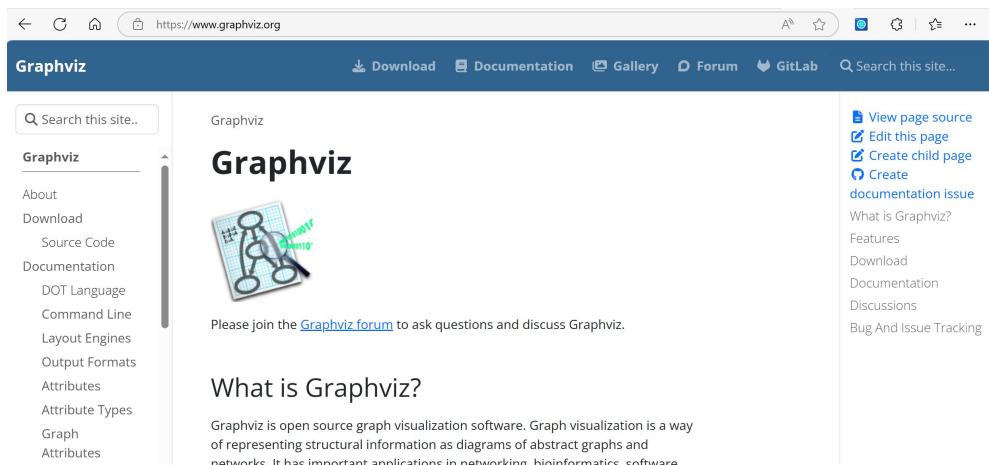


Figure 5.6.: Start image from the website <https://www.graphviz.org/>

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.

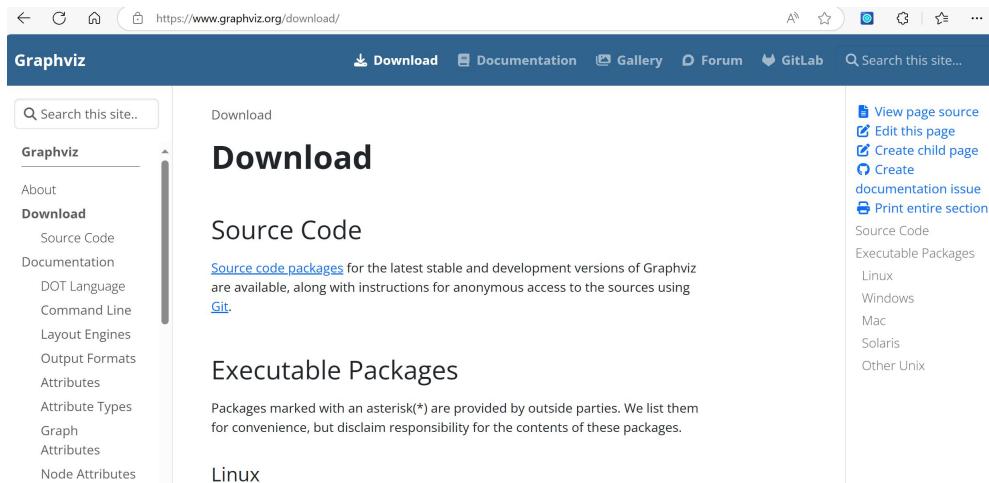


Figure 5.7.: Download area from the website <https://www.graphviz.org/download/>

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.

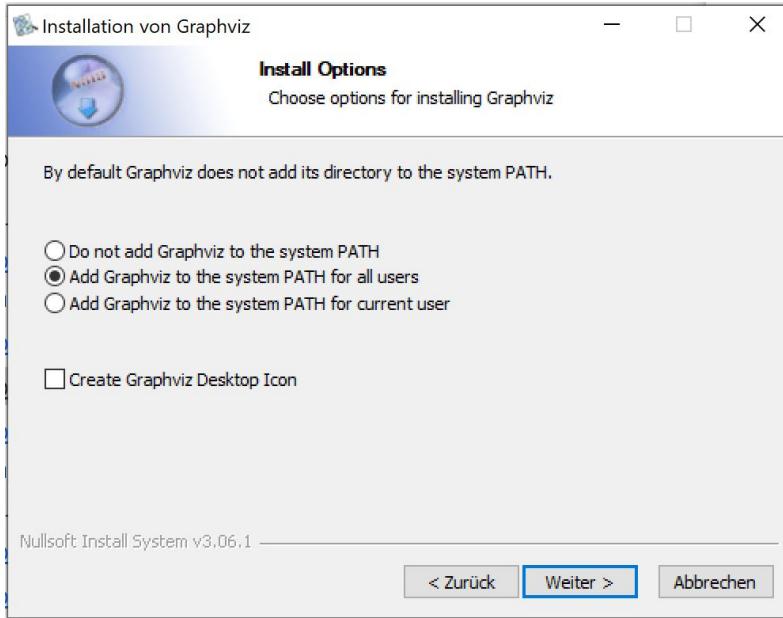


Figure 5.8.: Query for adding the Graphviz path to the system variable **PATH**

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.

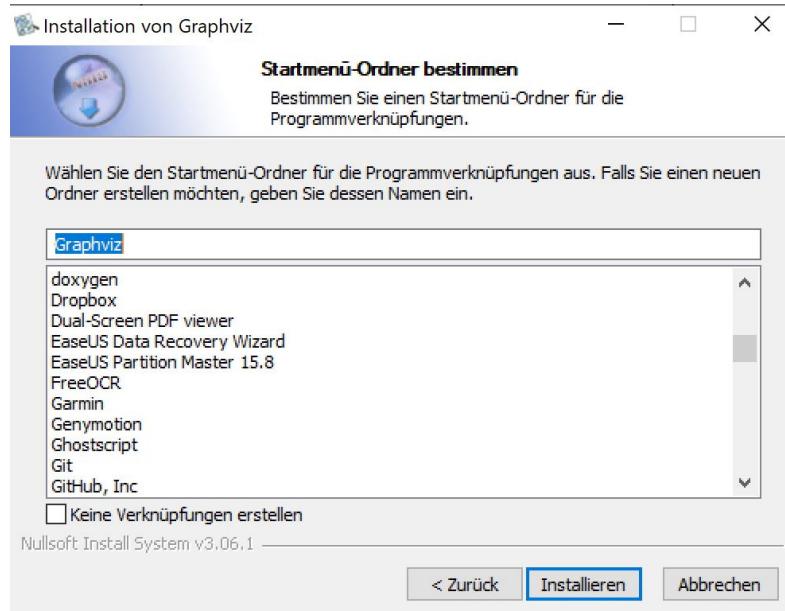


Figure 5.10.: Query for the start menu of Graphviz

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.

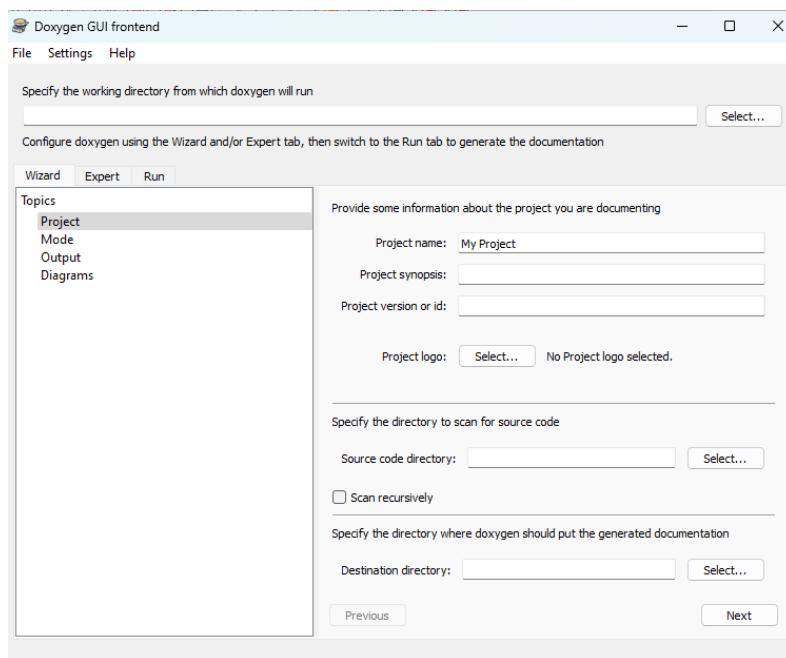


Figure 5.11.: DoxyWizard tab “Wizard” - Project

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.

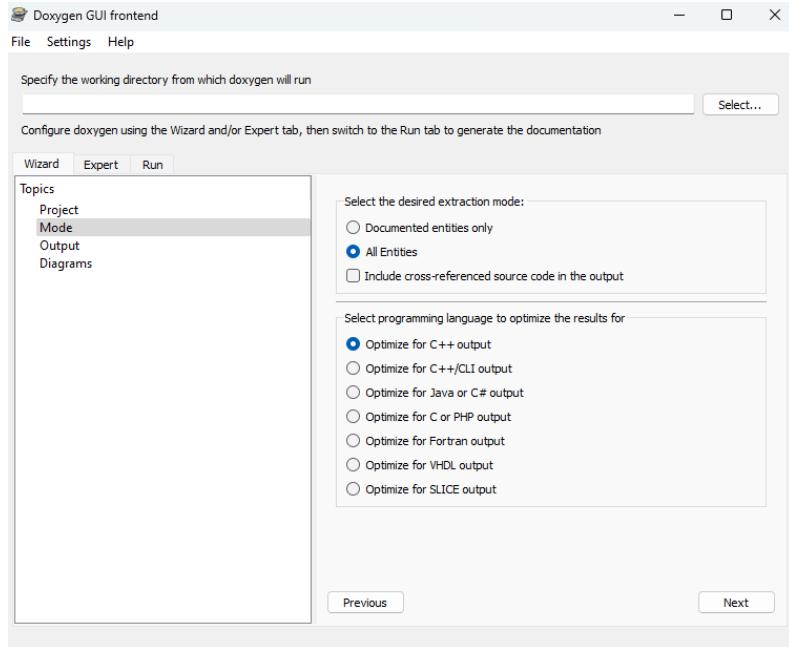


Figure 5.12.: DoxyWizard tab “Wizard” - Mode

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.

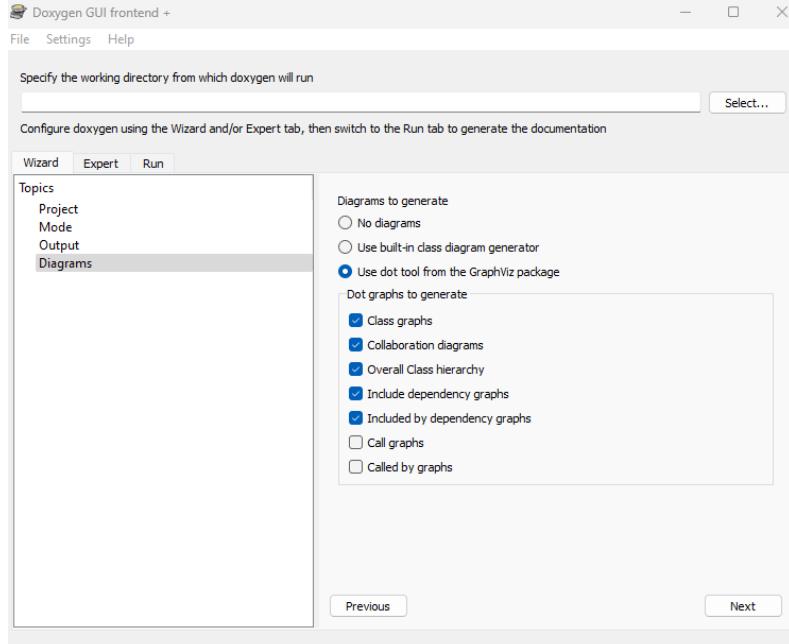


Figure 5.13.: DoxyWizard tab “Wizard” - Diagrams

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.

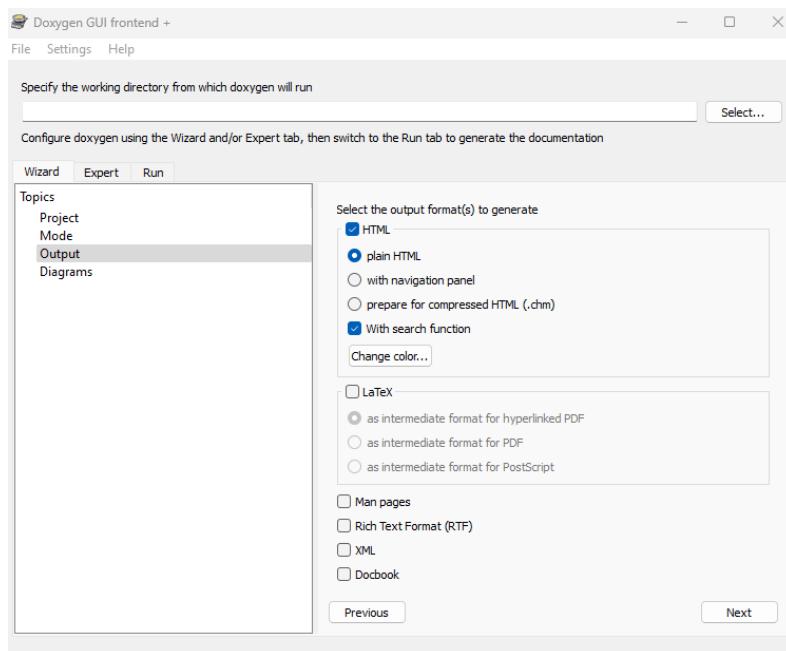


Figure 5.14.: DoxyWizard tab “Wizard” - Output

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.

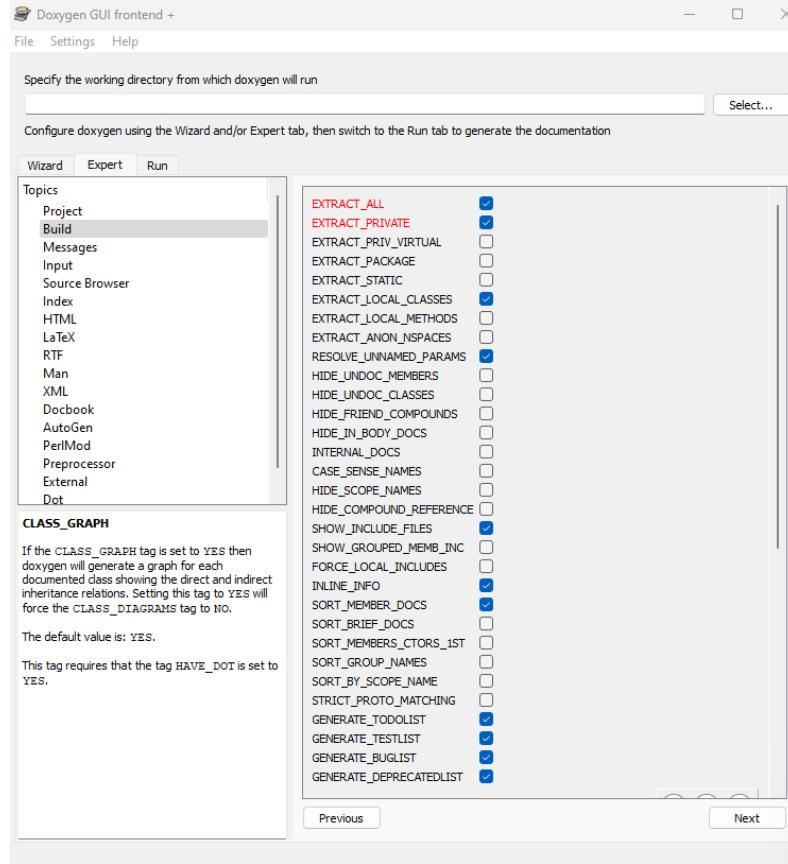


Figure 5.15.: DoxyWizard tab “Wizard” - Expert - Build

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.

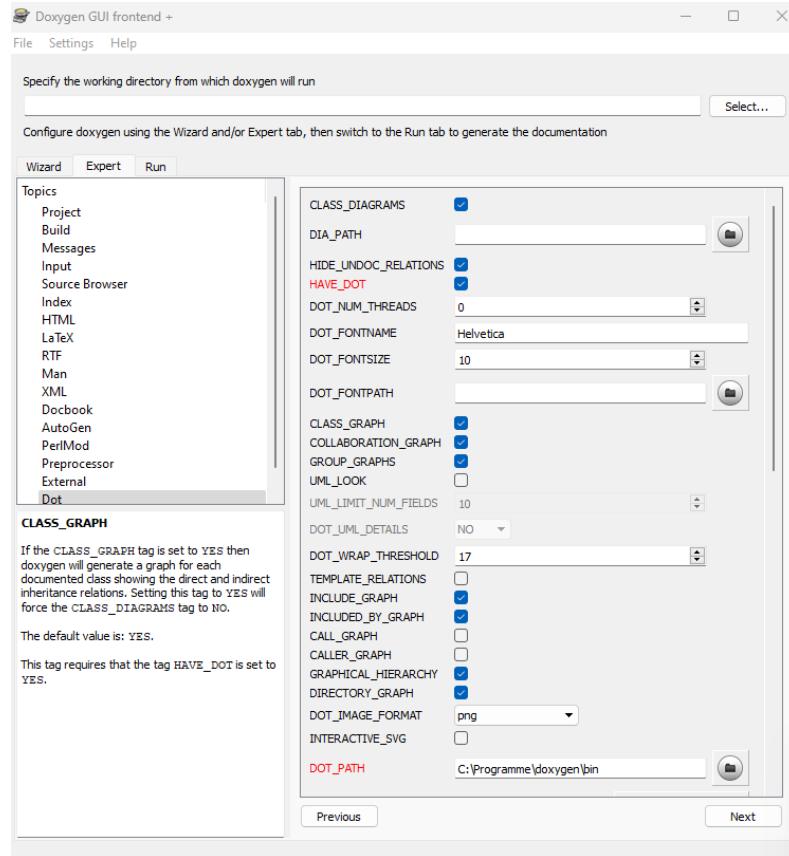


Figure 5.16.: DoxyWizard tab “Wizard” - Expert - Dot

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.

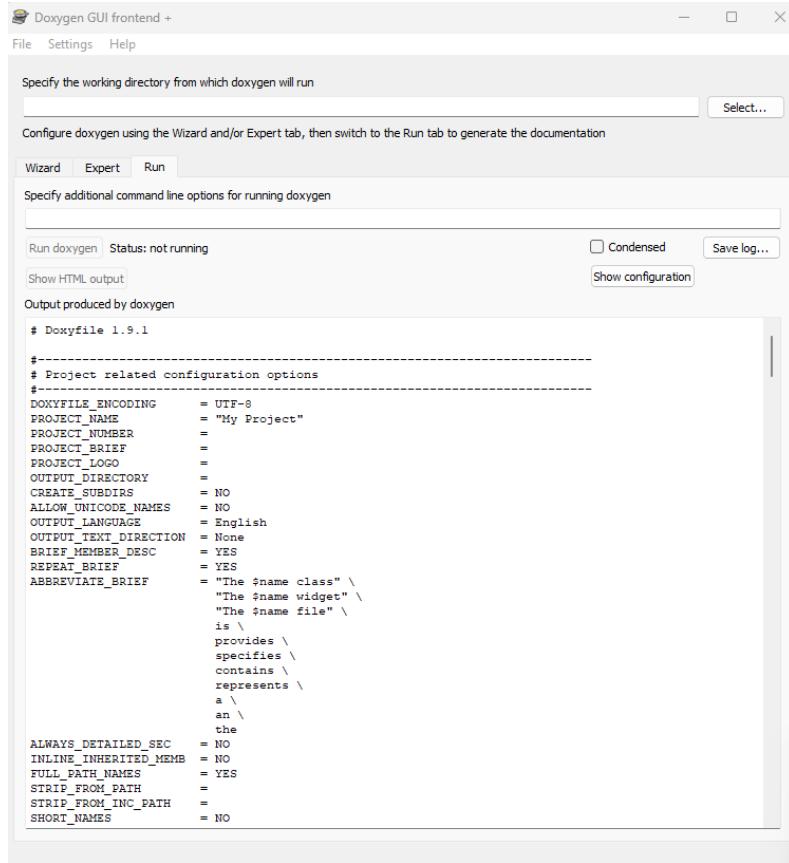


Figure 5.17.: DoxyWizard tab “Wizard” - Run

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 **.doxygenfile**.

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 doxygen file 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 and robotics.

</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 same hardware as the Arduino Uno, but with a smaller form factor and a built-in BLE module. It is a fully open-source hardware platform. The board was designed to be compatible with the Arduino IDE and other popular development environments.
*/
```


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]

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

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.

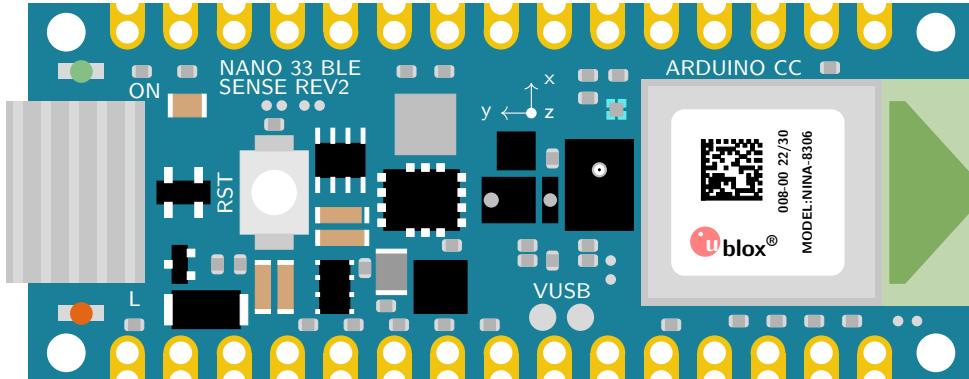


Figure 6.1.: Arduino Nano 33 BLE Sense Rev. 2, see Arduino Store

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 a 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. [Ard21]

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,

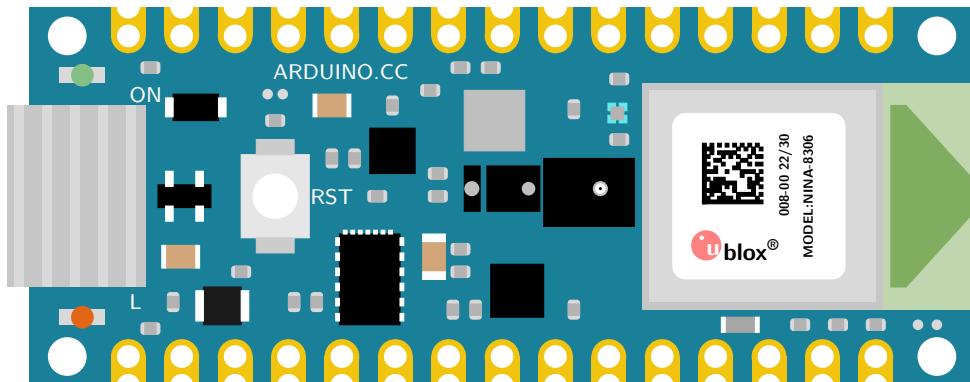


Figure 6.2.: Arduino Nano 33 BLE Sense Rev. 1

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

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,

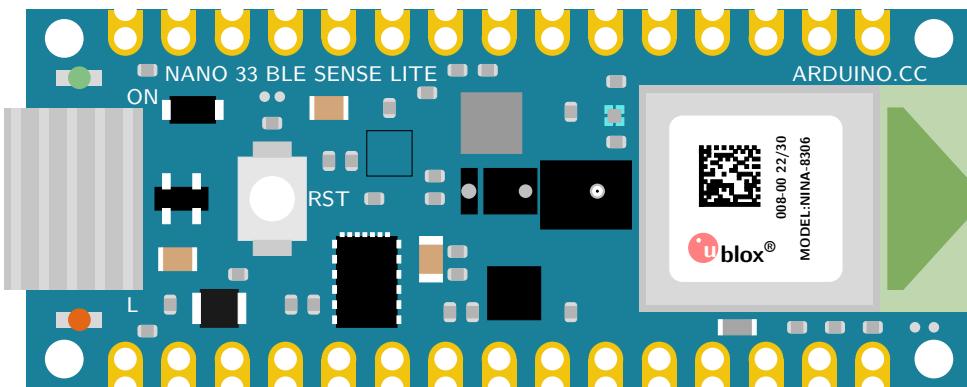


Figure 6.3.: Arduino Nano 33 BLE Sense Lite; the empty blank square marks the position of the missing sensor HTS221

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 [Ard21]. 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.

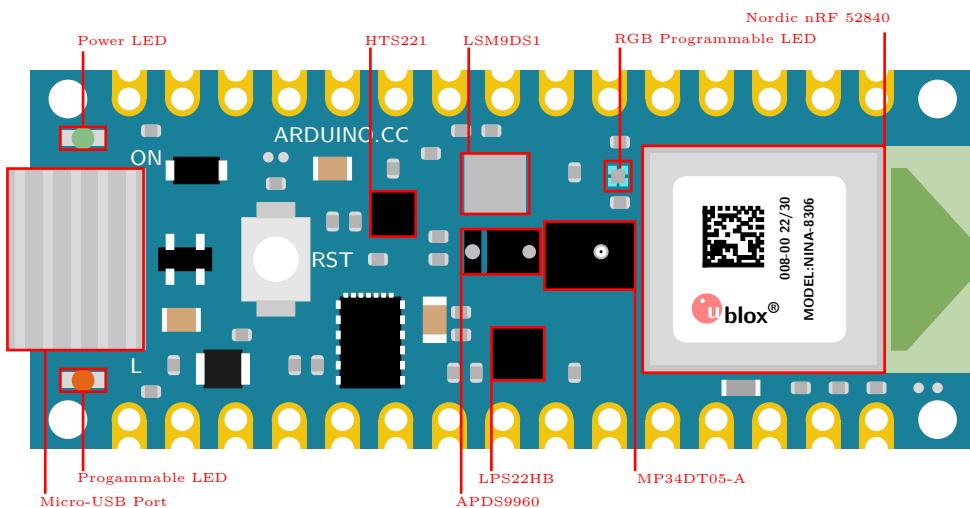


Figure 6.4.

WS:Auch mit dem Rev 2. 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.

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 [Ard21]

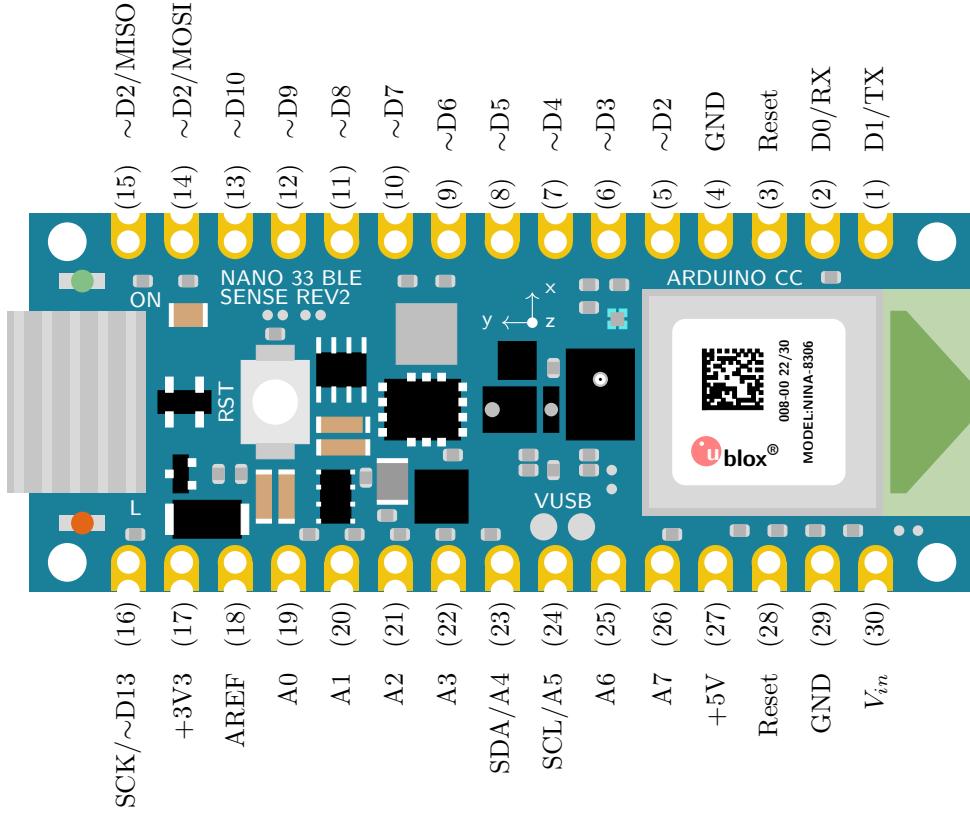


Figure 6.5.: Pin assignment of the Arduino Nano 33 BLE Sense and for the Arduino Nano 33 BLE Sense Rev 2; note that the orange built-in LED is connected to pin D13 and the power LED to pin D1. The built-in RGB LED occupies pins D18, D19 and D20.

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). [Ard21] 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]

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:

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]

WS:Here, we need more information because we

want to use it:

6.3.3. Pressure Sensor LPS22HB

- General infomration

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:
trix with example!

- Altimeters and barometers for portable devices

- drW Weather station equipment

- Sports watchs

- specials for this imu

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. [STMI17]
parameters for coding

- Absolutdruckbereich: 260 bis 1260 hPa

see Code/-

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

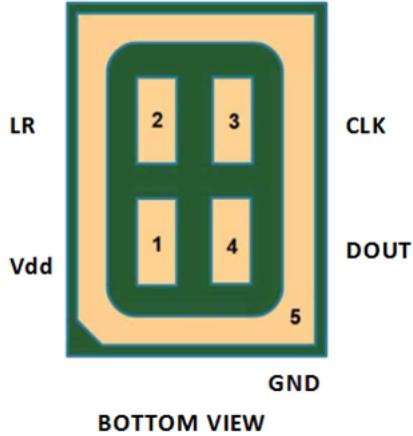


Table 1. Pin description

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.

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.

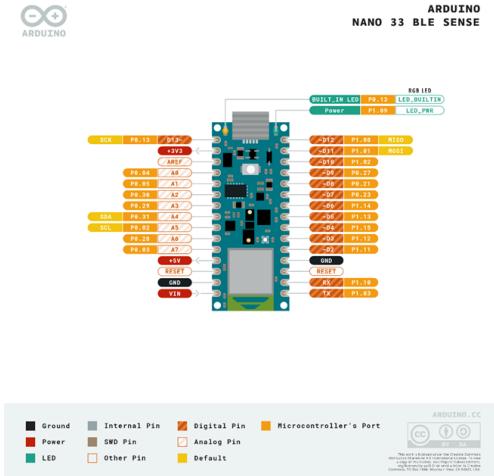
UART pin The board features a UART communication protocol that is used for serial communication and carries two pins Rx and Tx. The Rx is a receiving pin used to receive the serial data while Tx is a transmission pin used to transmit the serial data.

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



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.

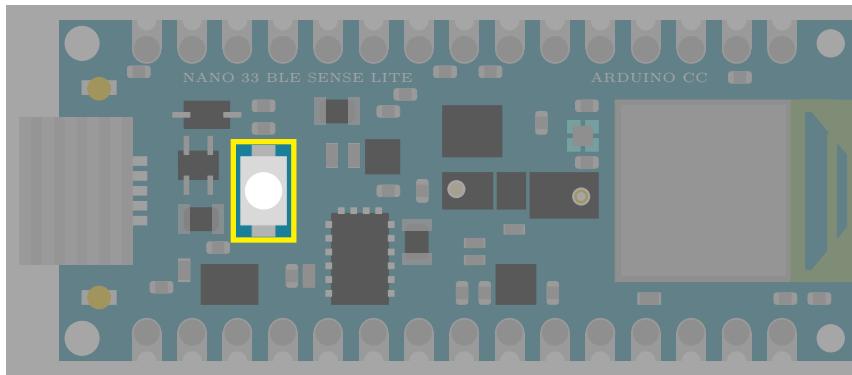


Figure 7.1.: Arduino Nano 33 BLE Sense's Reset Button

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. [Ard24b; 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]

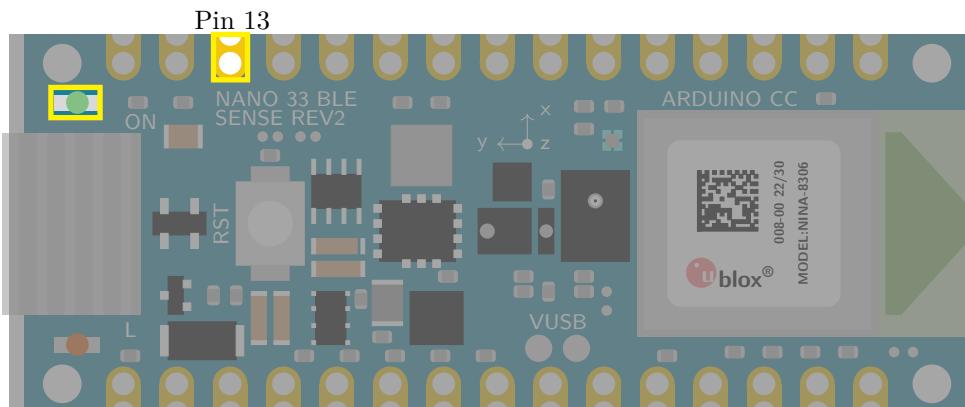


Figure 8.1.: Arduino Nano 33 BLE Sense's built-in LED with Pin 13

8.3. Specification

8.3.1. Pin Assignment

The built-in LED is an orange LED and connected to pin 13. The brightness cannot 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
1004 /**
1005 /**
1006 /**
1008 #endif

```

..../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 {

```

..../Code/Nano33BLESense/LEDs/LEDBuiltin.cpp

8.4.2. Simple Code for Testing the Built-in LED

In the sketch 8.3, 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

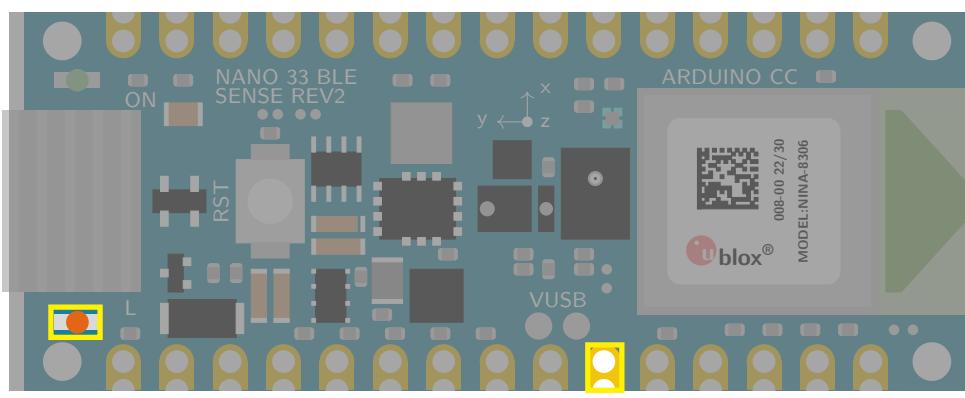
Arduino boards have a power LED on board. Normally, the power LED indicates the status of the board.

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. [Ard24b; 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. [Ard24b; 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
1001 #define LEDPower_h
1002
1003 /**
1004 */
1005 /**
1006 */
1007 #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>
1001 #include <LEDPower.h>
1002
1003 #define LED_PWR 25 /*< Define the pin for the power LED */
1004 * @return 1 - error
1005 */
1006 int LEDPowersetup()
1007 {
1008     // initialize digital pin LED_BUILTIN as an output.
1009     LEDSetup(LED_PWR);
1010 * @return 0 - success
1011 * @return 1 - error
1012 */
1013 int LEDPower(bool On)
1014 {
    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. [Ard24c]

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   BatteryState(false);
1054
1055   // Delay between measurements
1056   delay(5000);
1057 }
```

..../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. [Ard24b; Ard23a; Ard23b]

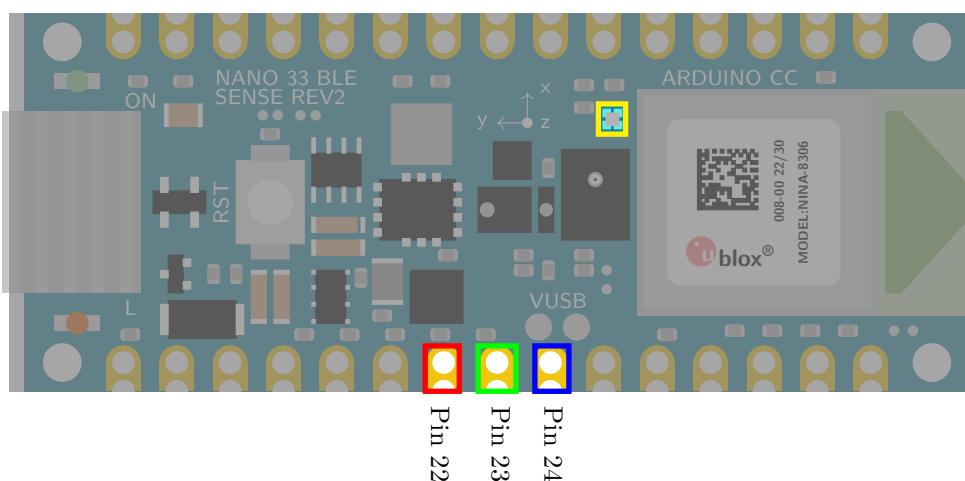


Figure 10.1.: Arduino Nano 33 BLE Sense's RGB LED with Pin 22, Pin 23, and Pin 24

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
*
* @brief Simple program for testing the RGB-LED
*
* Turns the RGB-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 22,
* 23, 24
*
*/
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
*
* Turns the RGB-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 22,
* 23, 24
*/
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 /**
* @brief the setup function runs once when you press reset or power the
* board
*
* Standard function of Arduino sketches
*
* Initialization of the pin 22, pin 23, and 24 as output
*
* @param —
*
* @return void
*/
1038
1039 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() {
1037     // Initialize the function SignsOfLife
1038     SignsOfLifeInit(LED_RED, CycleTimeOn, CycleTimeOff)

1039     // Application
1040     // ...
1041 }

1042

1043 /**
1044 * @brief the loop function runs over and over again forever
1045 *
1046 * standard function of Arduino sketches
1047 *
1048 * switching the led on / off for the signs of life
1049 *
1050 * @param —
1051 *
1052 * @return void
1053 */
1054 void loop() {
1055     // Switch red part of the RGB LED on/off
1056     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]

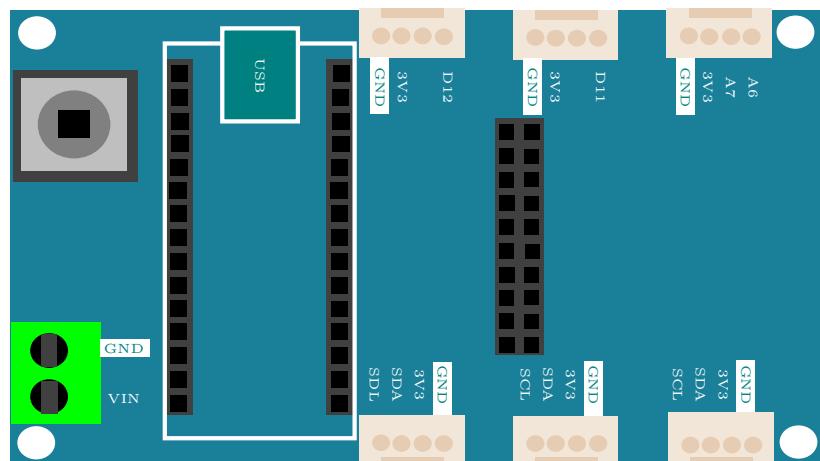


Figure 11.1.: Tiny Machine Learning Shield [Gua21]

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 21.3.

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
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 #define SET_ON true /*< Define flag for switching on */
1025 #define SET_OFF false /*< Define flag for switching off */
1026
1027 int buttonState = 0; /*< state of the built-in push button */
1028
1029 /**
1030 * @brief the setup function runs once when you press reset or power the
1031 * board
1032 *
1033 * standard function of Arduino sketches
1034 *
1035 * Initialization of the built-in LED and the push button
1036 *
1037 * @param —
1038 * @return void
1039 */
1040
1041 void setup() {
1042     // Initialize the pin as an output
1043     LEDBuiltinsetup();
1044     // Initialize the pin as an input
1045     pinMode(BUTTON_PIN, INPUT_PULLUP);
1046 }
1047
1048 /**
1049 * @brief the loop function runs over and over again forever
1050 *
1051 * standard function of Arduino sketches
1052 *
1053 * switching the built-in led on for 2 sec, if the push button is
1054 * pressed
1055 *
1056 * @param —
1057 *
1058 * @return void
1059 */
1060 void loop()
1061 {
1062     buttonState = digitalRead(BUTTON_PIN);
1063     if (buttonState == HIGH)
1064     {
1065         // Turn the LED on
1066         LEDBuiltin(SET_ON);
1067         // Wait for two second
1068         delay(2000);
1069         // Turn the LED off
1070         LEDBuiltin(SET_OFF);
1071         // Wait for one second
1072         delay(1000);
1073         buttonState = LOW;
1074     }
1075 }
```

..../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 21.4 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. [Arde]

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  
*/  
1054 void setup() {  
    // Initialize the pin as an output  
1056     LEDBuiltinsetup();  
    // Initialize the pin as an input  
1058     pinMode(BUTTON_PIN, INPUT_PULLUP);  
    // Initialize the interrupt function  
1060     attachInterrupt(digitalPinToInterrupt(BUTTON_PIN), buttonPressed,  
                      CHANGE);  
1061 }  
  
1063 /**  
1064 * @brief Interrupt service function  
*  
1066 * Attention: as short as possible  
*  
1068 * The function changes just one flag.  
*/  
1070 void buttonPressed()  
1071 {  
    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.

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

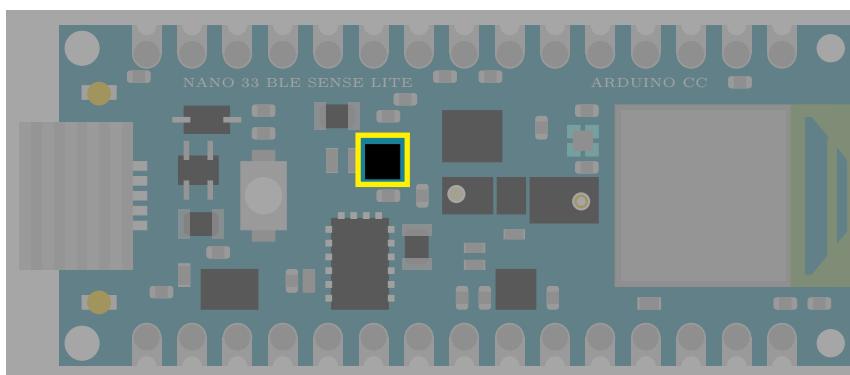


Figure 12.1.: Arduino Nano 33 BLE Sense's pressure and temperature sensor LPS22HB

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.

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
1042   // Read pressure value from the sensor
1043   int32_t pressure = lps22hb.readPressure();
1044
1045   // Print temperature and pressure values to the serial console
1046   Serial.print("Temperature: ");
1047   Serial.print(temp);
1048   Serial.println(" C");
1049   Serial.print("Pressure: ");
1050   Serial.print(pressure);
1051   Serial.println(" mbar");
1052
1053   // Wait for 1 second before taking the next reading
1054   delay(1000);
1055 }
```

..../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 in Listing 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

```
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. Inertial Measurement Unit

13.1. 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.

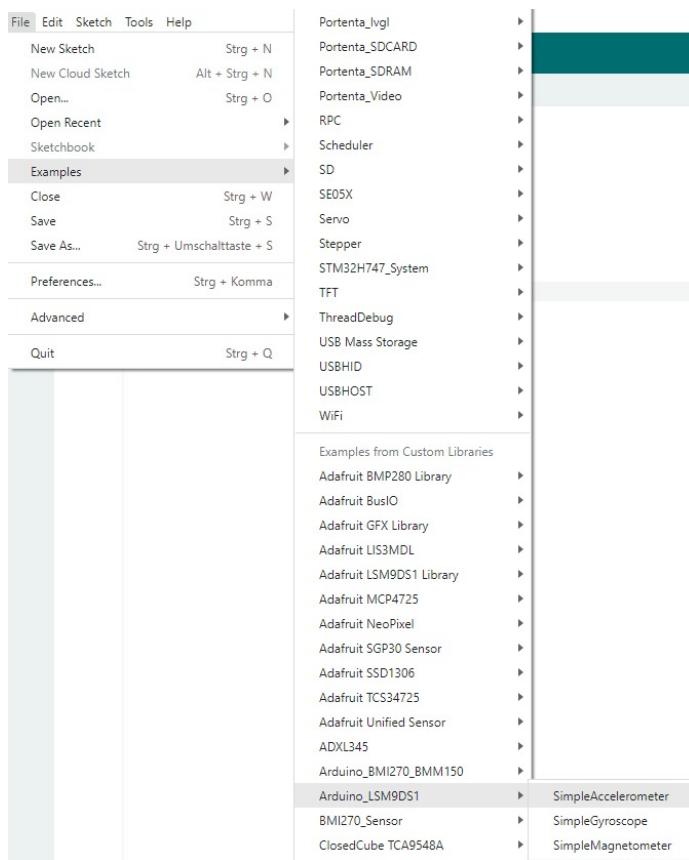


Figure 13.1.: Examples in the library [Arduino_LSM9DS1](#)

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]

13.2. 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.

13.3. 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.

13.4. 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 13.2

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 13.3

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.

13.4.1. Accelerometer:

[Chi+06]

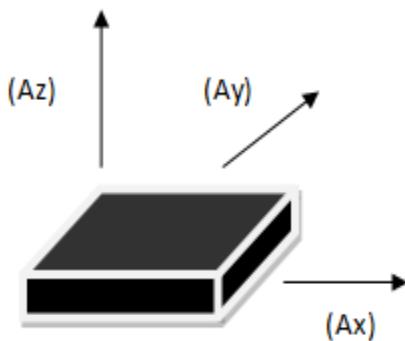


Figure 13.2.: Axis Acceleration of Accelerometer Sensor
WS:tikz!

- 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)

13.4.2. Gyroscope

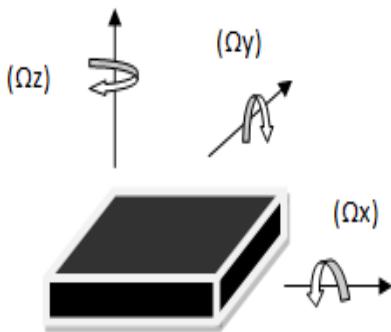


Figure 13.3.: Angular Speed of Gyroscope Sensor

- 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 ($^{\circ}/s$)

13.5. 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.

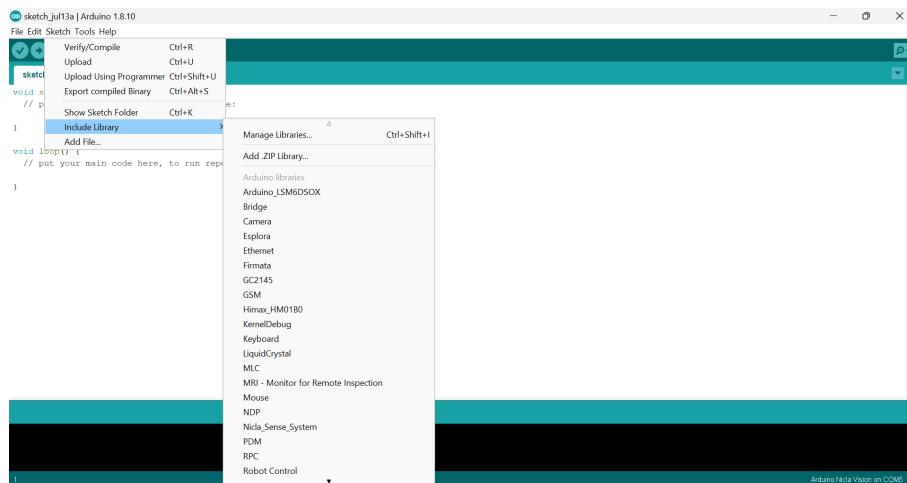


Figure 13.4.: Library setup in Arduino IDE

13.6. 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.

13.7. LSM9DS1(IMU)

13.8. 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.

13.9. Pin description LSM9DS1

13.10. Pin connections LSM9DS1

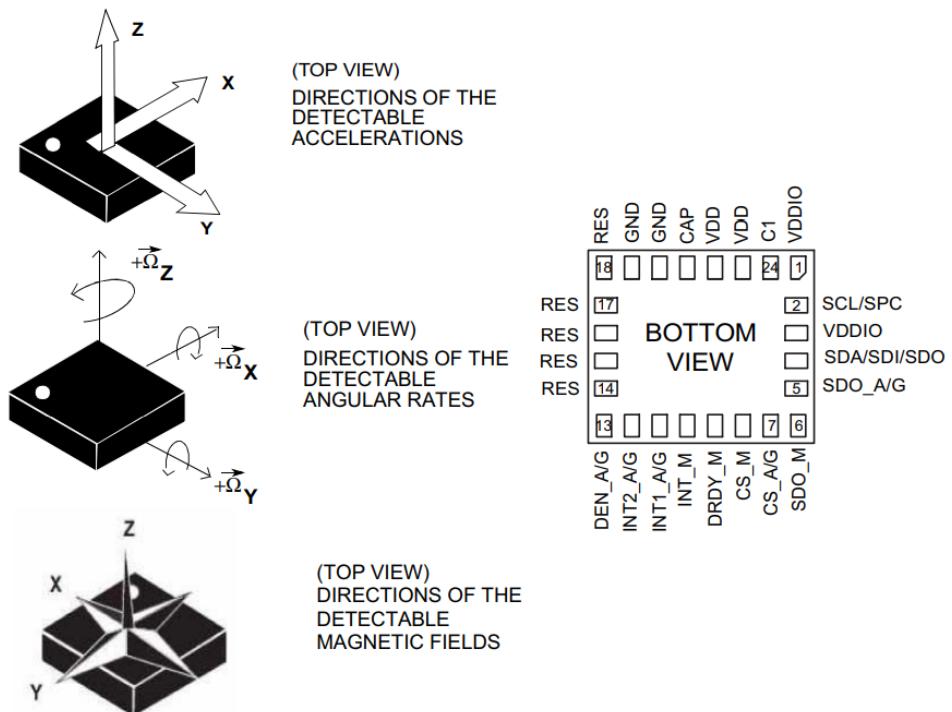


Figure 13.5.: Pin connections LSM9DS1
[Stmb]

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 13.6.: 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 13.7.: Sensor Characteristics
[Stmb]

Pin description LSM9DS1

13.10.1. 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 13.8.: 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 13.9.: Absolute Maximum Temperature
[Stmb]

13.10.2. Block Diagram

Accelerometer and gyroscope digital block diagram

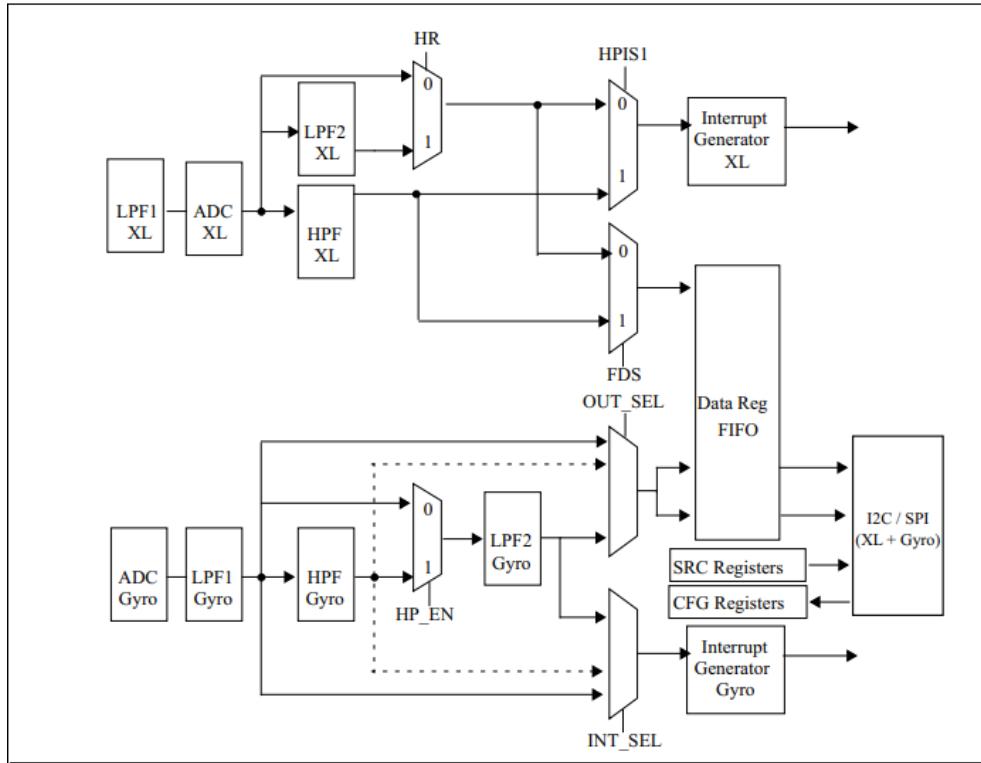


Figure 13.10.: Accelerometer and gyroscope block diagram
[Stmb]

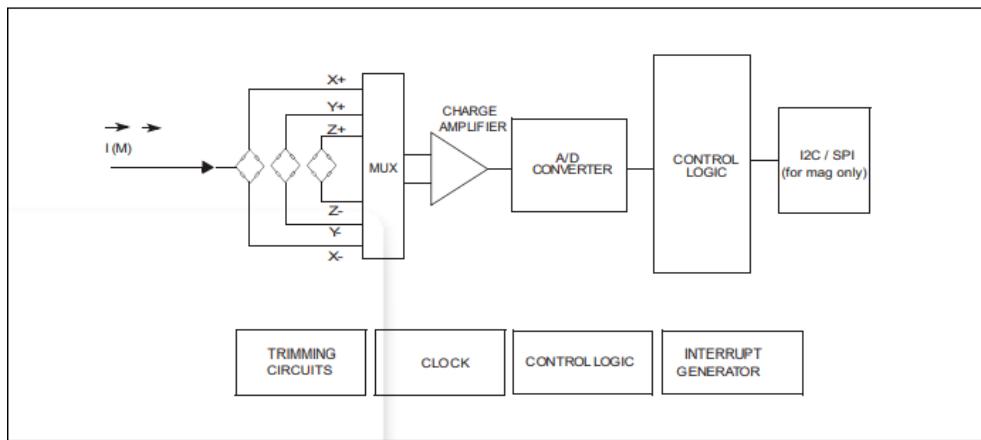
Magnetometer digital block diagram

Figure 13.11.: Magnetometer block diagram
[Stmb]

LSM9DS1 electrical connections

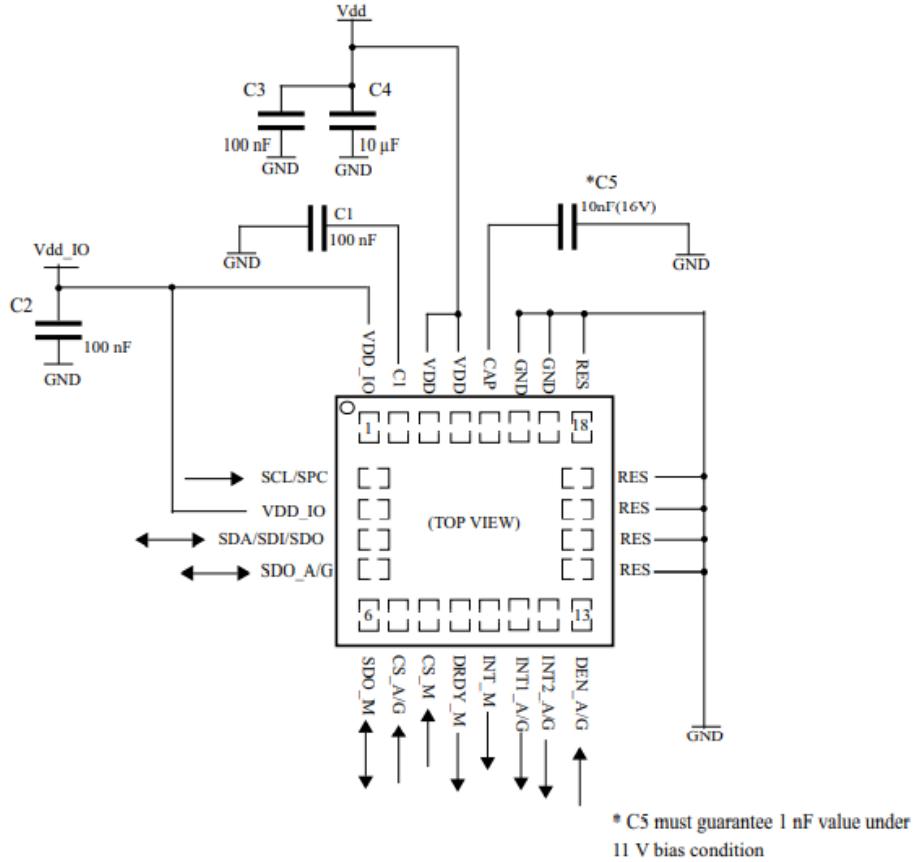


Figure 13.12.: LSM9DS1 electrical connections
[Stmb]

13.10.3. 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

13.10.4. 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.

13.11. 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.

13.12. Beispiele auf dem Mikrocontroller

13.12.1. 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 13.13.: Output des Testprogramms

13.13. Programmierung

13.13.1. Programmablaufplan

13.13.2. Programmcode und Dokumentation

```

1000 #include <Arduino_LSM9DS1.h>
1001 #include <Wire.h>
1002 #include "SSD1306Ascii.h"
1003 #include "SSD1306AsciiWire.h"

```

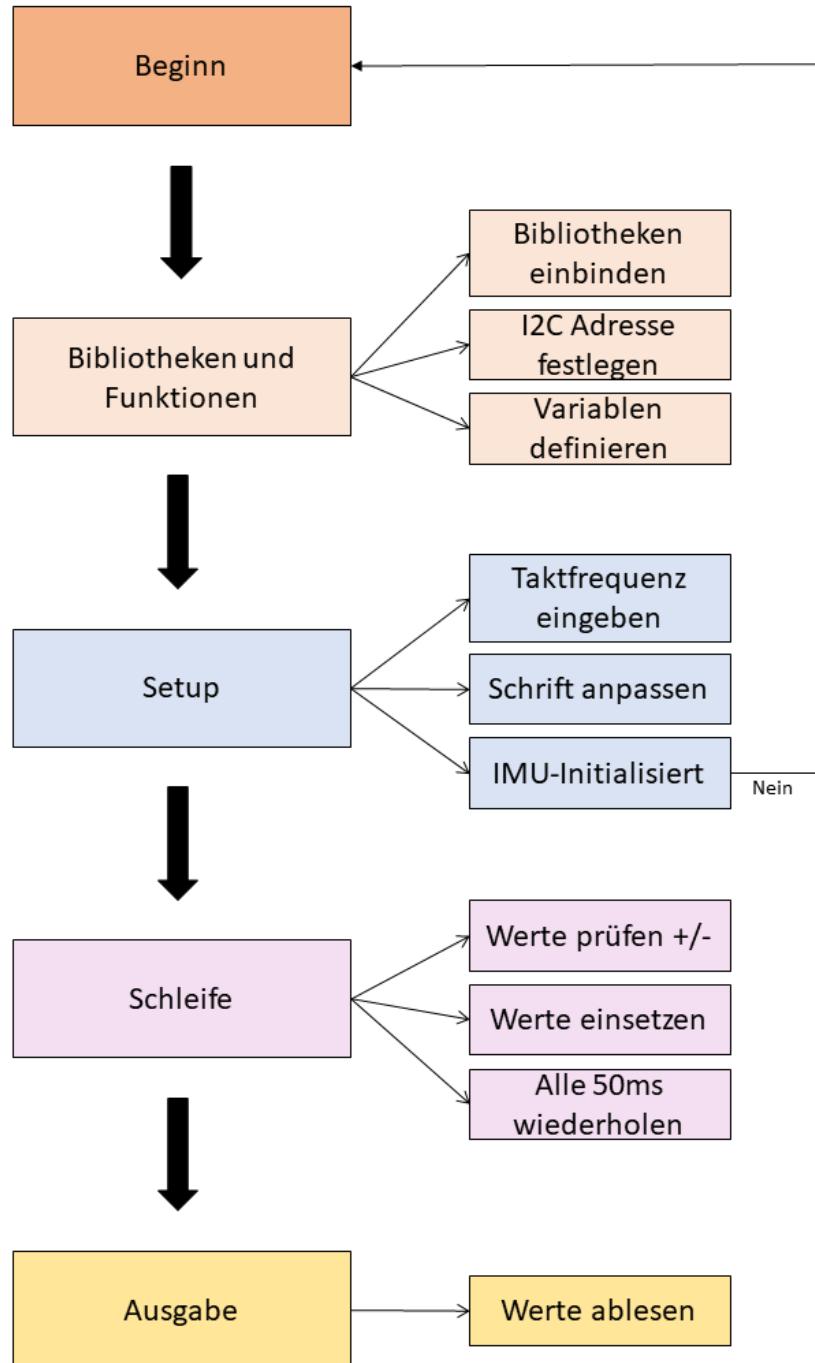


Figure 13.14.: Der Programmablaufplan

```

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.

WS:Der Begriff Echtzeit
 ist nicht bekannt. Hier
 muss gemessen werden!

13.13.3. 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.

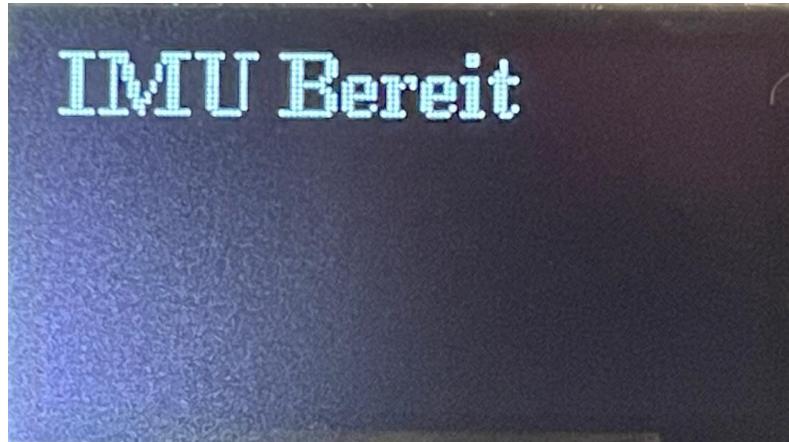


Figure 13.15.: Anzeige des Arduino OLED Displays sobald der Arduino eingeschaltet ist

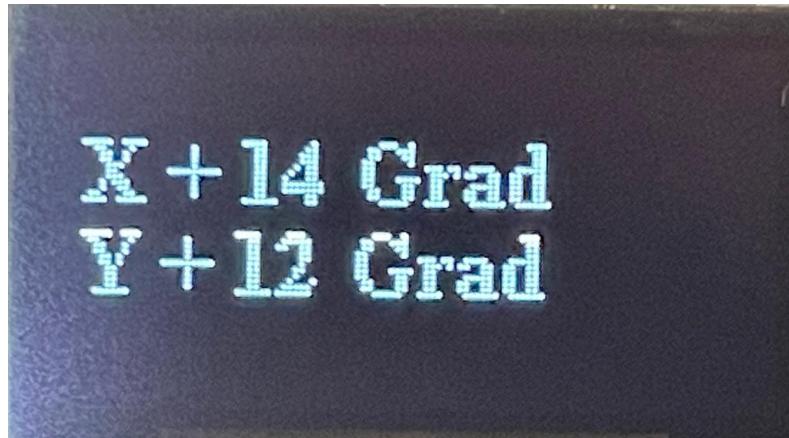


Figure 13.16.: Ausgabe von Messdaten

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.

13.14. 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.

13.15. 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.

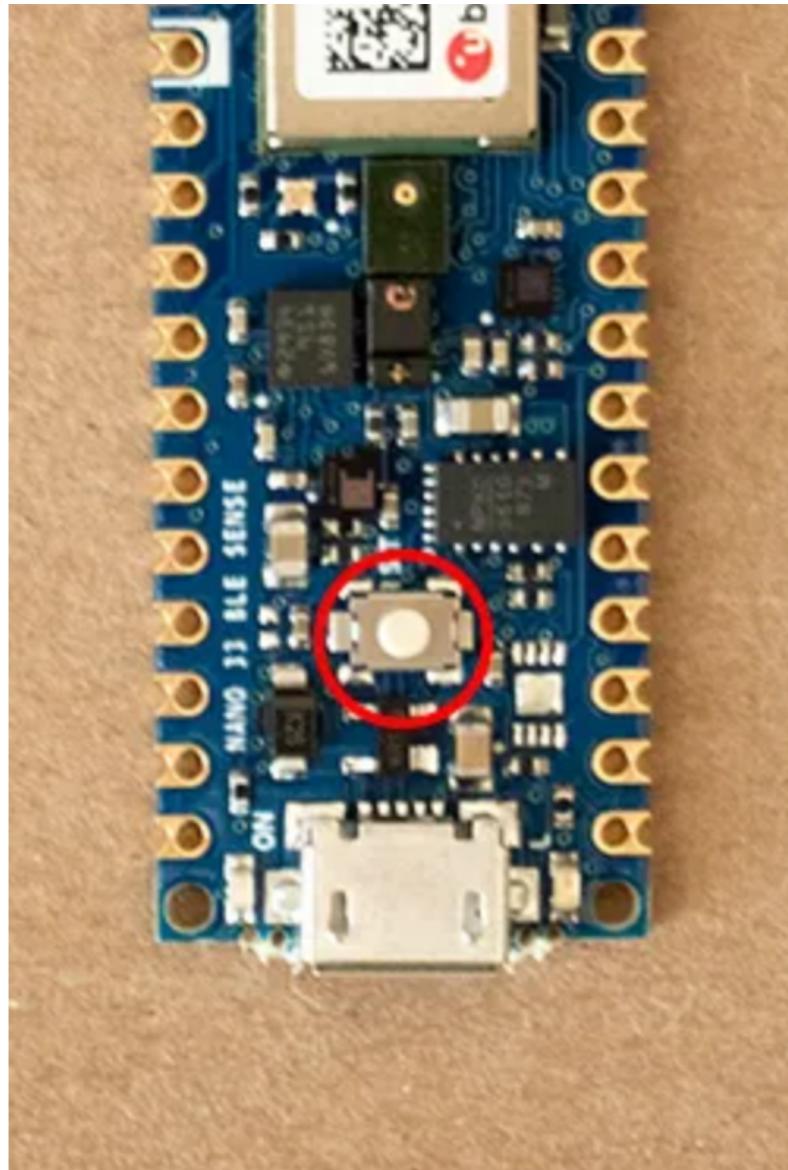


Figure 13.17.: Der Reset Button des Arduino

14. Calibration

14.1. 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]

14.2. 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.

14.2.1. 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[RS17]. 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[RS17]. 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[RS17]. 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[RS17].

14.2.2. 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 [RS17]:

- $Xm\text{-nanoTesla} = \text{rawCompass.m.x} * (100000.0 / 1100.0);$
- 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 [RS17]. 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 [RS17].

14.2.3. 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 14.1.: 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 14.2.: 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 14.3.: Example ADPS-9960

15. Errors

15.1. 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]

15.2. Affected Parameters

Bias, drift, and noise errors will affect all the accelerometers and gyroscopes parameters. Here's how they impact each parameter:

15.2.1. 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.

15.2.2. 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]

16. IMU LSM6DSOX - Libraries and Functions

16.1. 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.

16.1.1. 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]

16.1.2. 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]

16.1.3. 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]

16.1.4. 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.

16.2. 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.

16.2.1. **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.[Ardi]

16.2.2. **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.[Ardh]

16.2.3. **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.

16.2.4. **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.

16.2.5. **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.

16.2.6. 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.

16.2.7. 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.

16.2.8. 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.

16.2.9. 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.

16.2.10. 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.

16.2.11. 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.

16.2.12. 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.

16.2.13. 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.

16.2.14. 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.

16.2.15. 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.

16.2.16. 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.

16.2.17. 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.

16.2.18. 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.

16.2.19. 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.

16.2.20. `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.

16.2.21. `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.

16.2.22. `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.

16.2.23. `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.

16.2.24. 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.

16.2.25. 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.

16.2.26. 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.

17. 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]

17.1. 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.

17.1.1. 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.

17.1.2. 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.

17.1.3. 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.

17.2. Specific Sensor

17.2.1. 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.

17.2.2. 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.

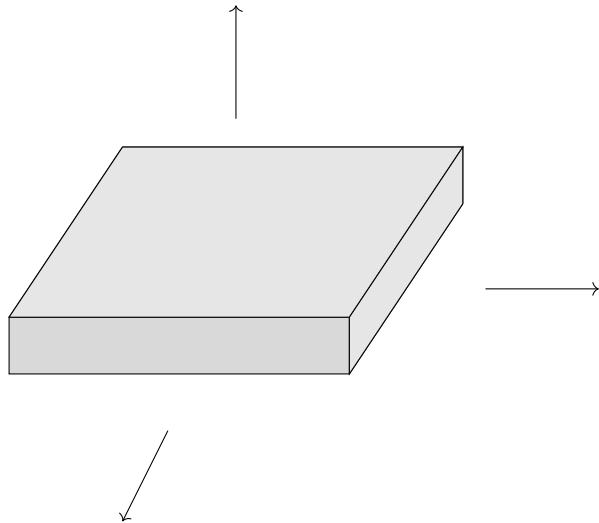


Figure 17.1.: LSM9DS1 axis on this card

17.2.3. 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.

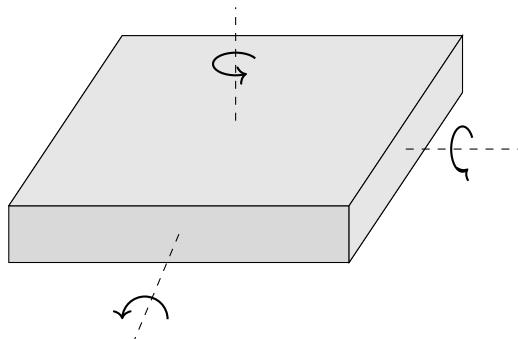


Figure 17.2.: LSM9DS1 axis on this card

17.3. Specification

17.3.1. 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.

17.3.2. 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.

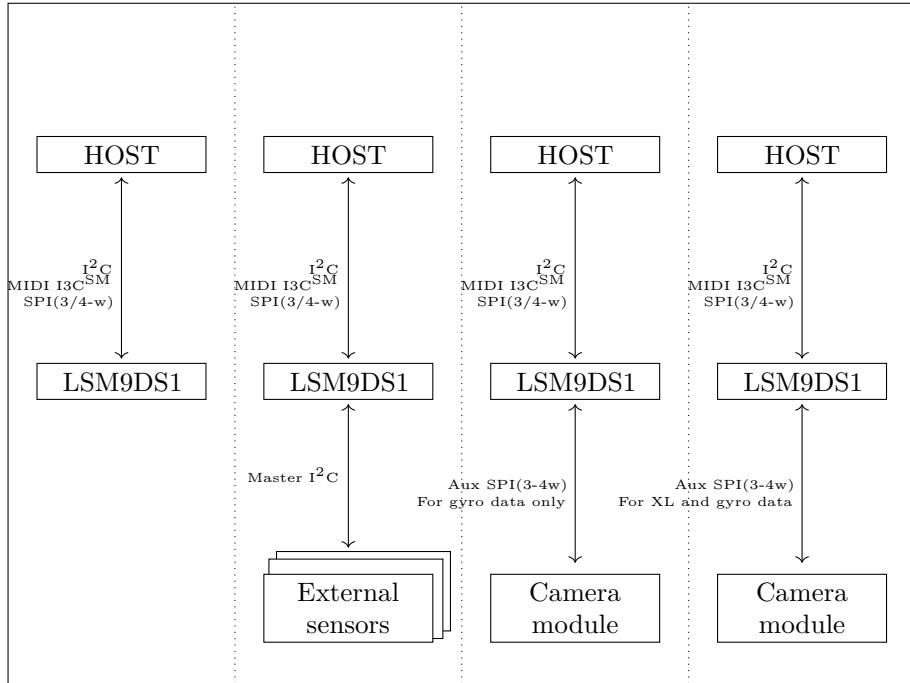


Figure 17.3.: Explication of pin mode

17.4. Library

17.4.1. 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.

17.4.2. 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 17.4.: Wire include

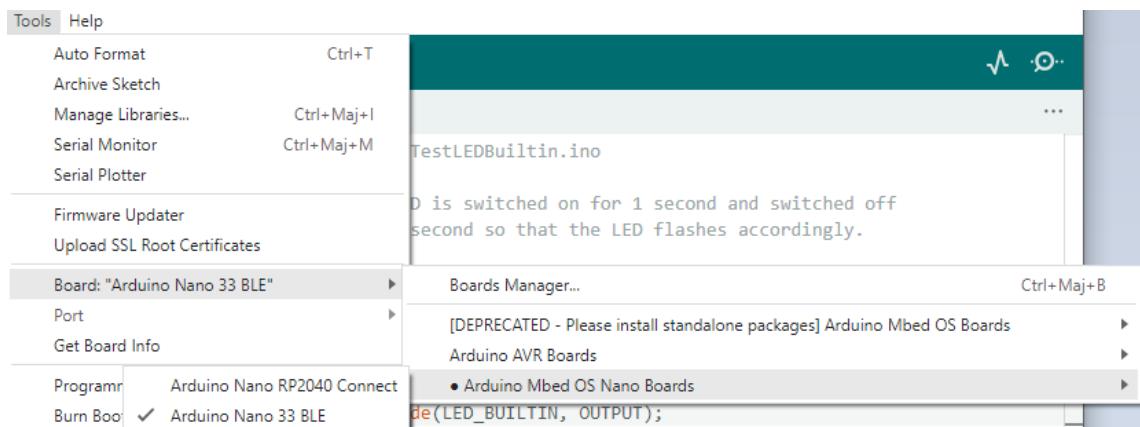


Figure 17.5.: Board card installation

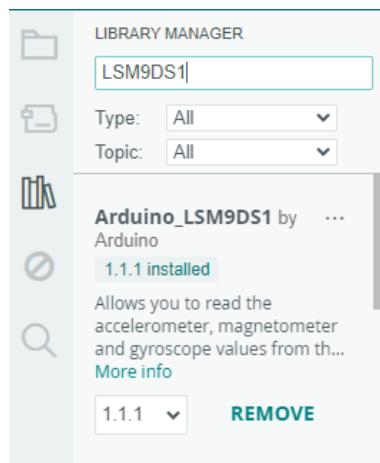


Figure 17.6.: Library choice

17.5. 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.

17.5.1. 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 17.1.: 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

17.5.2. 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 }
```

17.5.3. 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 }
```

17.5.4. 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 if (IMU.accelerationAvailable()) {
1005     IMU.readAcceleration(x, y, z);
1006
1008     Serial.print(x);
1009     Serial.print('\t');
1010     Serial.print(y);
1011     Serial.print('\t');
1012     Serial.println(z); }
```

17.5.5. 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); }
```

17.5.6. 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); }
```

17.5.7. 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); }
```

17.5.8. 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");
```

17.5.9. 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");
```

18. Distance, Speed and Acceleration Detection Algorithms

18.1. 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 18.1.: 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

$$\text{depth} = \frac{f \times \text{baseline}}{\text{disparity}} \quad (18.1)$$

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>

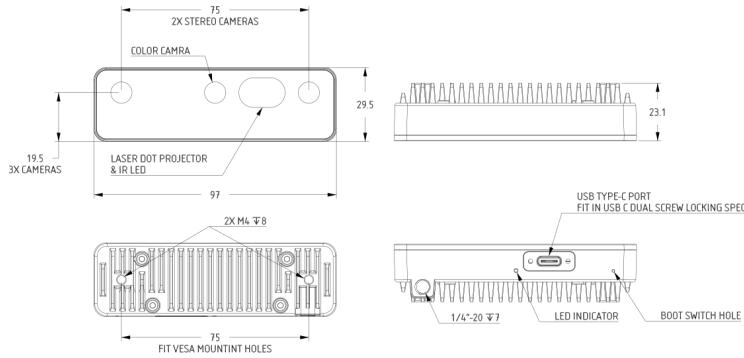


Figure 18.1.: Design of OAK-D Pro camera

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.

18.2. 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 18.2.: Comparison of distance measurement technologies: LiDAR, Millimeter-Wave Radar, and Binocular Cameras

Part III.

Arduino Nano 33 BLE Sense - External Sensors and Actors

19. Tiny Machine Learning Kit

The Tiny Machine Learning Kit will equip you with all the tools which are needed to start with machine learning. [Ardk]

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 19.1.: Das Tiny Machine Learning Kit [Ardk]

19.1. 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.

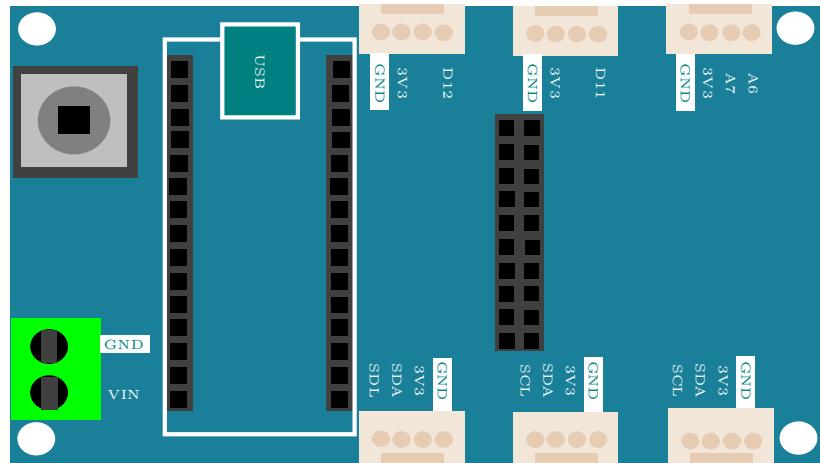


Figure 19.2.: Das Tiny Machine Learning Shield [Gua21]

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 19.3 dokumentiert.

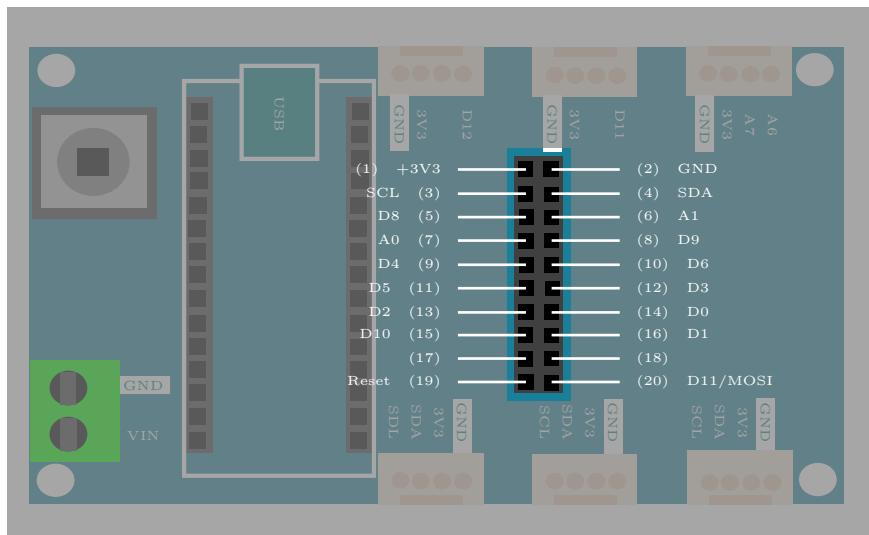


Figure 19.3.: Pin-Belegung des Tiny Machine Learning Shields [Gua21]

19.2. 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.

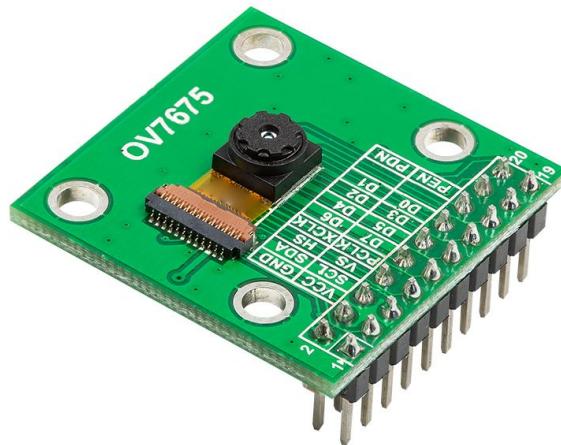


Figure 19.4.: Das OV7675 Kameramodul [Ardk]

19.3. 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 19.5.: Ein USB-A auf Micro-USB Kabel

20. 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.

20.0.1. 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.

20.1. 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.

WS:tikz-Bild des Shields

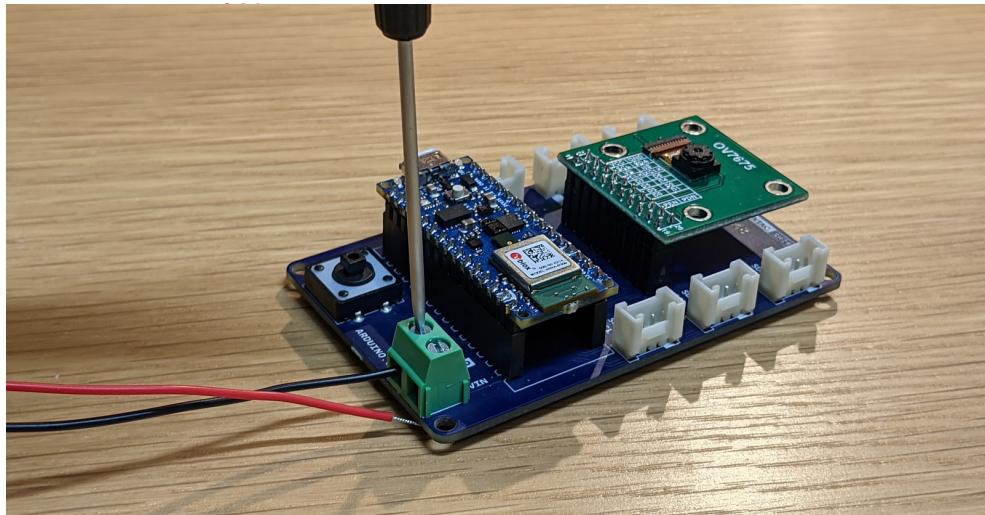


Figure 20.1.: Montage des Clips

An image of screwing down the black negative terminal to attach a wire.

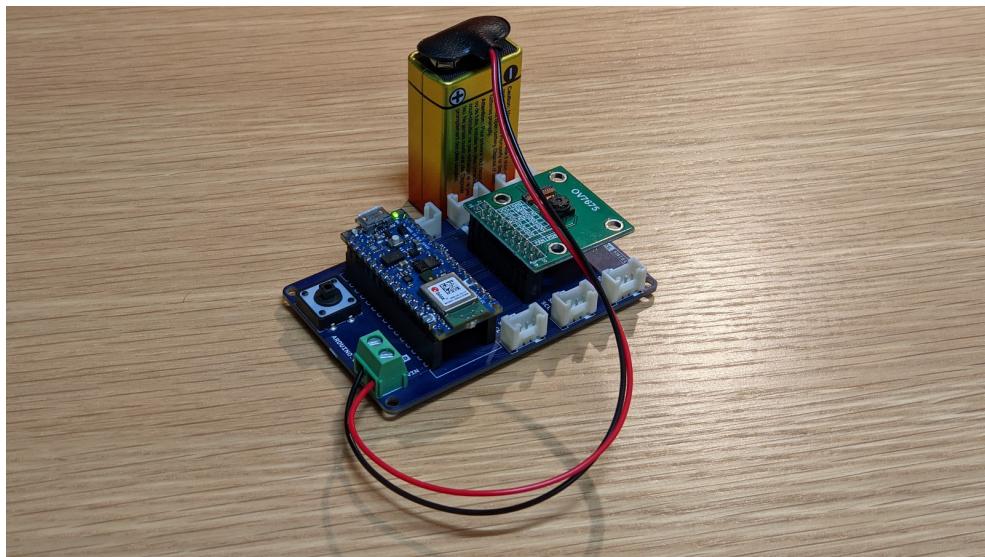


Figure 20.2.: Komplettes System mit Batterie

An image of the attached battery powering the device.

20.2. 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

20.3. Batterieclip

Der Batterieclip in Abb. 20.3, 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].

20.4. 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 20.3.: Batterieclip für 9-Volt-Block[Rei24]

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 20.4.: Voltage Sensor 170640 von dem Hersteller *Shenzhen Global Technology Co., Ltd*[She]

Mit dem Sketch [TestBattery.ino](#), siehe ??TestBattery, soll der Spannungssensor getestet werden.

20.4.1. 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 20.1.: 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 /**
1010 * Pin A7 is referenced as the voltage pin for the voltage sensor. */
1011 #define VoltagePin A7
1012
1013 /**
1014 * The parameter SignalEingang is later assigned the value read in from
1015 *          the VoltagePin pin. */
1016 long SignalEingang;
1017
1018 /**
1019 * @brief Starting serial communication
1020 *
1021 * @details The function is executed when the programme is started. The
1022 *          serial interface to the computer is initialised at 9600 baud
1023 *          There is no return value.
1024 */
1025 void setup() {
1026     Serial.begin(9600);
1027 }
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 }

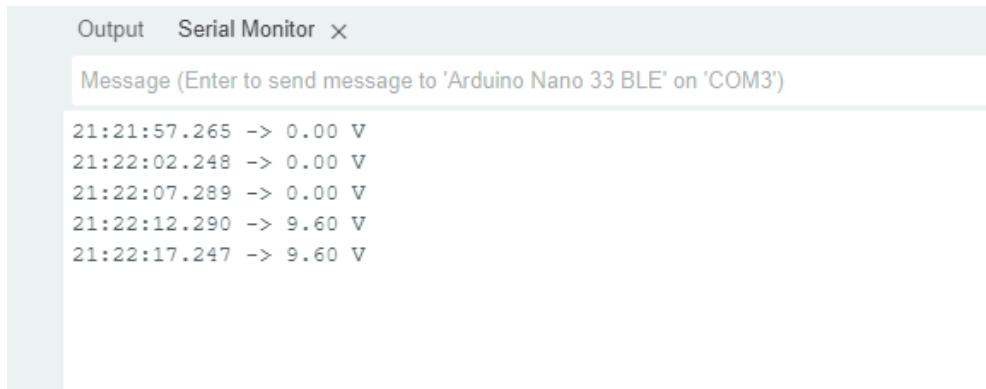
```

..../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.

20.4.2. 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 20.5 zeigt den gemessenen Spannungsverlauf. Die angezeigte Spannung liegt in dem erwarteten Bereich einer 9 V Batterie.



The screenshot shows the Arduino Serial Monitor interface. The title bar says "Output Serial Monitor X". Below it is a message input field with the placeholder "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 20.5.: Testoutput des Spannungssensors

21. TinyML Shield: Built-in Push Button

21.1. 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]

21.2. 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]

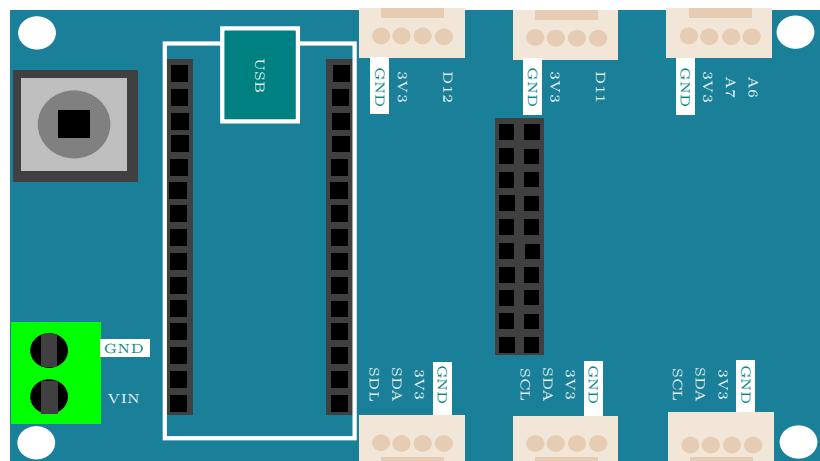


Figure 21.1.: Tiny Machine Learning Shield [Gua21]

21.3. 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

21.4. 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 21.1.: 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 21.2.: Read the built-in button's state

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

21.5. Tests

The simplest test is the flashing of an LED for 2 seconds, if the button is pressed, see sketch 21.3.

Listing 21.3.: 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 #define SET_ON true /*< Define flag for switching on */
1025 #define SET_OFF false /*< Define flag for switching off */
1026
1027 int buttonState = 0; /*< state of the built-in push button */
1028
1029 /**
1030 * @brief the setup function runs once when you press reset or power the
1031 * board
1032 *
1033 * standard function of Arduino sketches
1034 *
1035 * Initialization of the built-in LED and the push button
1036 *
1037 * @param —
1038 * @return void
1039 */
1040
1041 void setup() {
1042     // Initialize the pin as an output
1043     LEDBuiltinsetup();
1044     // Initialize the pin as an input
1045     pinMode(BUTTON_PIN, INPUT_PULLUP);
1046 }
1047
1048 /**
1049 * @brief the loop function runs over and over again forever
1050 *
1051 * standard function of Arduino sketches
1052 *
1053 * switching the built-in led on for 2 sec, if the push button is
1054 * pressed
1055 *
1056 * @param —
1057 *
1058 * @return void
1059 */
1060 void loop()
1061 {
1062     buttonState = digitalRead(BUTTON_PIN);
1063     if (buttonState == HIGH)
1064     {
1065         // Turn the LED on
1066         LEDBuiltin(SET_ON);
1067         // Wait for two second
1068         delay(2000);
1069         // Turn the LED off
1070         LEDBuiltin(SET_OFF);
1071         // Wait for one second
1072         delay(1000);
1073         buttonState = LOW;
1074     }
1075 }
```

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

21.6. 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.

21.7. Simple Application

This code sketch 21.4 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. [Arde]

WS:andere Überschrift

für simple Application

Listing 21.4.: 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 *
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.

21.8. 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]

22. OLED-Display

22.1. 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.[Sim]

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

22.2. Anschluss

In der Abbildung 22.1 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.

22.3. Programmierung

22.3.1. [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.

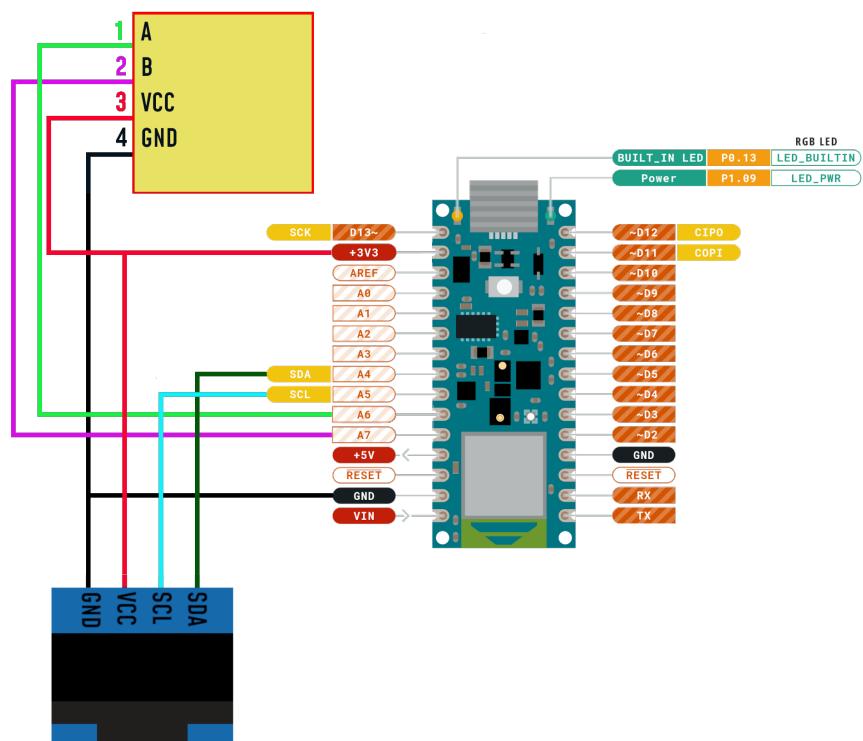


Figure 22.1.: Gesamter Schaltplan

22.3.2. 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.

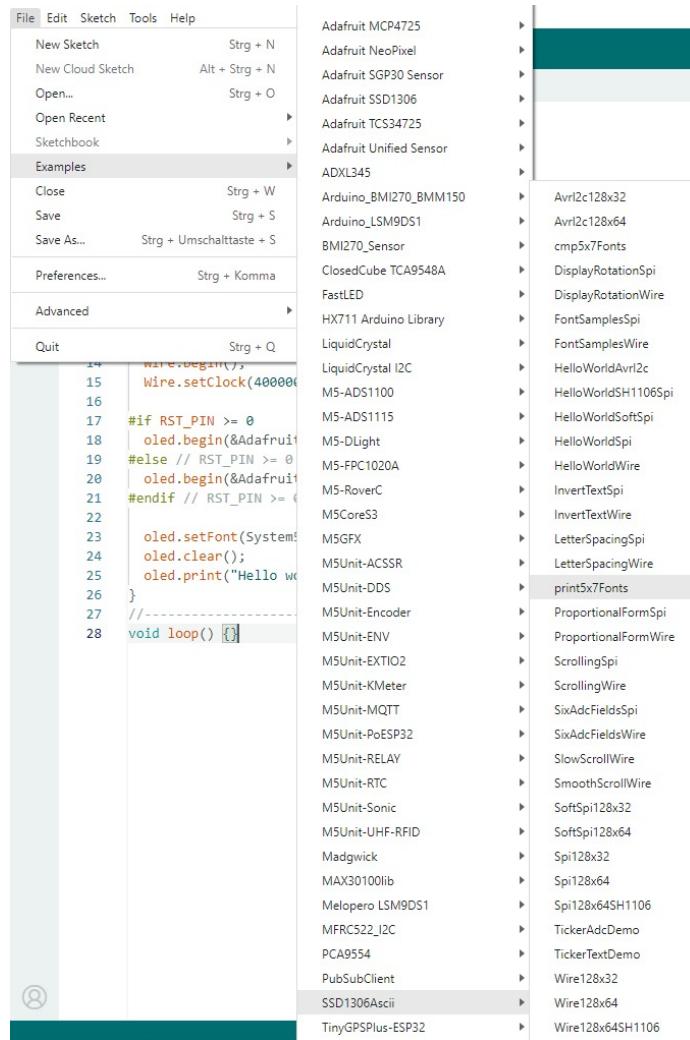


Figure 22.2.: Beispiele in der OLED Bibliothek

22.3.3. 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.

22.4. Software

22.4.1. Verwendete Bibliotheken

Zur Ansteuerung des Displays werden folgende Bibliotheken verwendet:

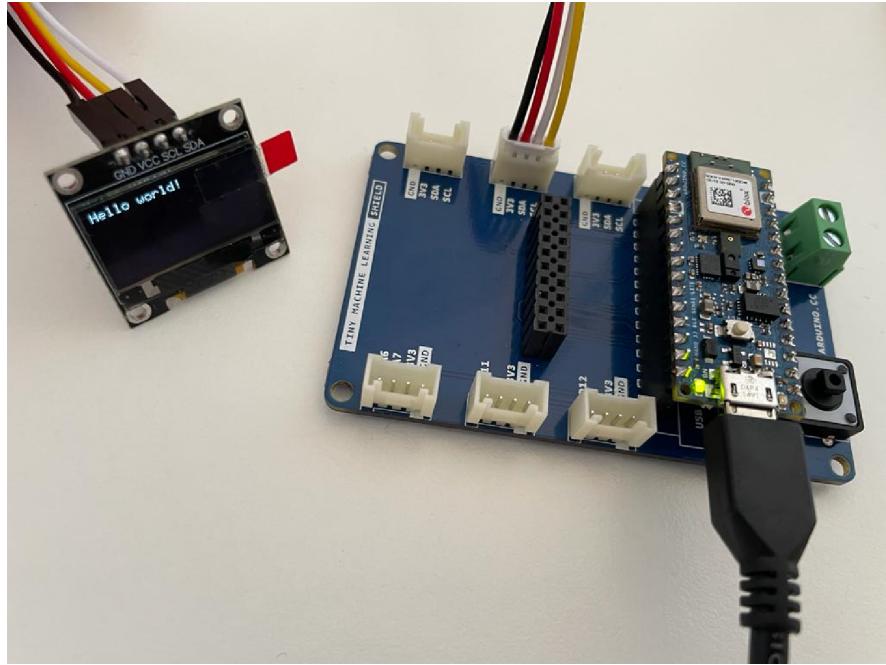


Figure 22.3.: Testausgabe des OLED Display

- [Wire.h](#): Diese Bibliothek ermöglicht die Kommunikation über den I2C-Bus, der für die Ansteuerung des OLED-Displays verwendet wird [Ardf]
- [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]

[WS: Ist dies kompatibel?
Was ist damit möglich?]

22.4.2. 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
```

22.4.3. 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

22.4.4. 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 22.4.: Code Einlesen/Zählen

Die Hauptfunktion `loop()` enthält zwei Funktionen für die Messungssteuerung und die Benutzeroberfläche:

```
67 void loop() {
68     HandleInput();
69     HandleUI();
70 }
```

Figure 22.5.: Code Messungssteuerung

Die Funktion `HandleInput()` überwacht den Zustand der beiden Taster und steuert somit den Messungsablauf:

Listing 22.1.: 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.

22.4.5. Mikrofondatenverarbeitung

Die Funktion `StartSampling()` initialisiert die Datenverarbeitung:

Listing 22.2.: Start and stop sampling

`PDM.begin(1, 16000)`: Initialisiert eine Abtastrate von 16.000 Hz. Die Funktion `StopSampling()` beendet die Aufnahme von Audiodaten.

22.4.6. 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.

22.4.7. Umrechnung auf den Wert dB SPL

Die Umrechnung des Mikrofonsignals auf den dB SPL-Wert erfolgt in der Funktion `getDbValueFromPMC()`:

Listing 22.3.: 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.

22.4.8. Benutzeroberfläche

Die Funktion `HandleUI()` aktualisiert OLED-Display Anzeige:

Listing 22.4.: 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.

22.5. 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 22.6.: 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

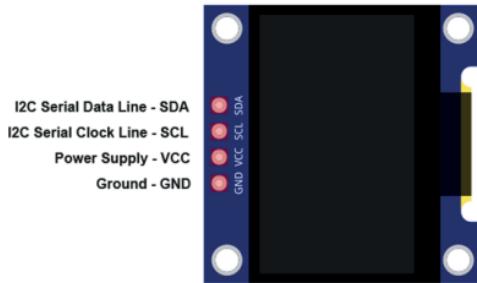


Figure 22.7.: Pins des OLED-Displays.

Das Display verfügt über vier Pins, welche in der Abbildung 22.7 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 ; Funb]

22.6. 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 22.8 abgebildet.

22.7. 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 leichter lösen zu können und ein neu schreiben von komplexen Funktionen zu vermeiden.

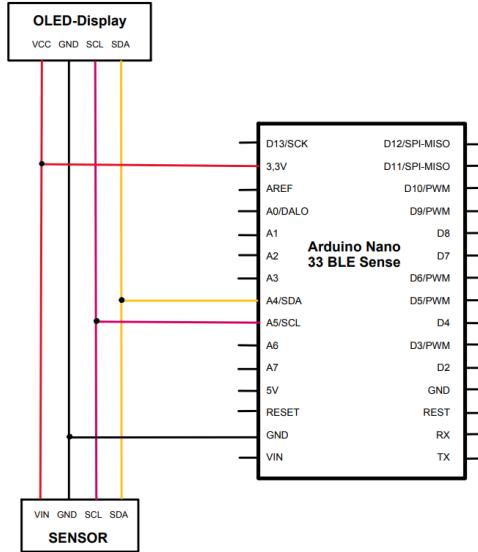


Figure 22.8.: Stationärer Aufbau der Wetterstation
[Arda]

In unserem Fall verwenden wir Bibliotheken für den Arduino, den Sensor BME280 und das OLED-Display.

22.7.1. 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<....h>` [W3c]

22.7.2. 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 den 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. [Fun]

22.7.3. 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 Abbildung22.9).

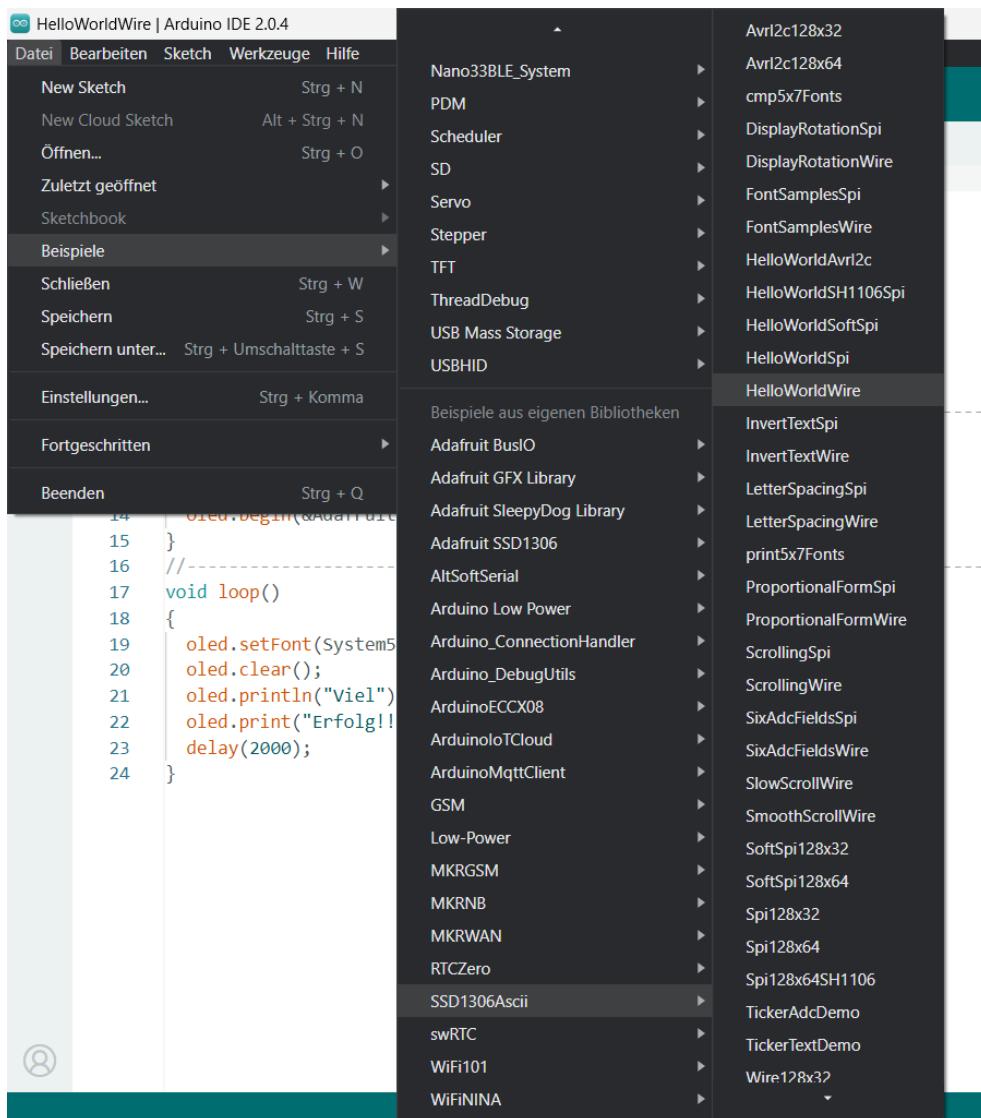


Figure 22.9.: Pfad des Testprogramms.



Figure 22.10.: Erste Ausgabe Display
[Funb]

Das Testprogramm [Hello World](#) (siehe Seite 200) wird dafür benutzt, um den Text Hello World auf dem Display auszugeben (siehe Abbildung 22.10). 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 22.5.: 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 22.10). Hiermit ist sichergestellt worden, dass die Entwicklungsumgebung und der Compiler korrekt funktionieren. [Funa]

22.8. 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.

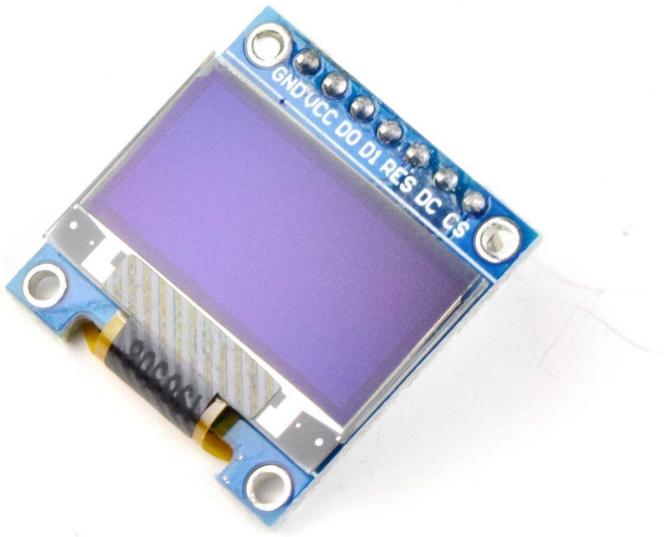


Figure 22.11.: Das Display zur Ausgabe der Messwerte

[Quelle](#) [Qualität](#), [Artikel](#), [Display 0,96"](#), Grove OLED-Display, SSD

...

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	

22.8.1. 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;
1001 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);
1001 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 [Ardj].

Anschließend wird das I²C-Protokoll mit dem Objekt *Wire* und der Funktion *begin()* gestartet. Dabei wird der Arduino als Master im I²C-Protokoll angemeldet[Ardf]. Anschließend wird die Taktfrequenz mit 400kHz festgelegt [Ardg].

```
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();
```


23. 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]

23.1. 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]

23.2. Drehwinkel-Encoder

Um die Funktion und korrekte Ansteuerung des Drehwinkel-Encoders zu testen, wird das Testprogramm 23.1 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 23.1.: Testprogramm für den Encoder

24. Schrittmotor NEMA 17

In diesem Kapitel folgt eine grundsätzliche Beschreibung eines Schrittmotors. Darauf folgt die Erläuterungen zum Aufbau und Funktionsweise eines Schrittmotors sowie die Aufzählung verschiedener Grundbauarten. Nachfolgend werden Ursachen und Lösungen zum Verhindern von Schrittfehlern und der verwendete Schrittmotor genauer erläutert.

24.1. Beschreibung eines Schrittmotors

Ein Schrittmotor ist ein Elektromotor, der sich für präzise Positionierungsaufgaben eignet. Im Gegenteil zu anderen Elektromotoren wird bei einem Schrittmotor keine Positionsmeßung oder Positionsregelung benötigt. Andere mechanische Antriebssysteme benötigen einen geschlossenen Regelkreis und mechanische Bremsen um Drehzahl und Position einzuhalten. Der Motor führt durch die Rotation des Rotors diskrete Schritte aus, wobei jeder Steuerimpuls eine Verschiebung um einen konstanten Winkel bewirkt. Mit diesem Motor ist eine erreichbare Positioniergenauigkeit von $0,1^\circ$ möglich. Diese Art von Elektromotor wird beispielsweise in Druckern oder Scanern, aber auch im Kraftfahrzeugbereich verwendet. Im Kraftfahrzeugbereich werden die Schrittmotoren zur Spiegelverstellung sowie der Sitzverstellung verwendet. Das maximal erreichbare Drehmoment eines Schrittmotors liegt bei zwei Newtonmeter und die maximal erreichbare Drehzahl bei ca. 2000 Umdrehung pro Minute. Der Vorteil eines Schrittmotors ist die Wartungsfreiheit, da der Rotor keine Wicklungen hat. [Babiel.2023][Hagl.2021][Bernstein.2018][Schroder.2021]

24.1.1. Aufbau und Funktionsweise eines Schrittmotors

Der Schrittmotor besteht aus einem Stator, einem Rotor ohne Wicklungen und einer Steuerelektronik. Diese Steuerelektronik setzt sich zusammen aus einer Treiberstufe und der eigentlichen Steuerung. Bei dem Stator handelt es sich um den feststehenden äußeren Teil und bei dem Rotor um den beweglichen inneren Teil, wie in Abbildung 24.1 zu erkennen ist. Im Stator sind Spulen verbaut, die von einem Strom durchflossen werden. Hierdurch entsteht ein magnetisches Feld. Da der Rotor magnetisch ist, folgt er dem Magnetfeld des Stators. Soll eine Bewegung hervorgerufen werden, werden einzelne Wicklungsstränge ein- aus- oder umgeschaltet. Durch diesen Vorgang wird ein rotierendes Magnetfeld erzeugt. Das erzeugte Magnetfeld zieht den Rotor an. Der Rotor wird bei jedem Puls (Takt) um einen Winkelschritt weitergeschaltet. Die Anzahl der Polpaare im Stator geben die Anzahl der Schritte vor. Die Drehzahl und die Drehrichtung hängt von der Reihenfolge und der Häufigkeit der Stromimpulse ab. Es gibt drei verschiedene Betriebsarten, die abhängig von der Genauigkeit und der Drehzahl sind. Im Vollschrittbetrieb werden alle Polpaare bestromt. In dem Halbschrittbetrieb wird die Schrittzahl des Motors verdoppelt, wodurch sich die Positionsauflösung im Gegensatz zum Vollschrittbetrieb verdoppelt. Allerdings wird in diesem Betrieb das Drehmoment reduziert. In dem Mikroschrittbetrieb bewegt sich der Rotor in sehr kleinen Schritten. Hierdurch wird eine hohe Positioniergenauigkeit und ein ruhiger Lauf erreicht, da die Ströme und das Drehmoment in kleineren Schritten verändert werden. Bei einer zu hohen Schrittfrequenz kann ein Schrittverlust entstehen. Bei einem Schrittverlust überspringt der Motor einzelne Winkelschritte und landet in

einer vorherigen oder nächsten Position gleicher Phase. Durch die offene Steuerkette werden die Verluste nicht erkannt.[**Hagl.2021**][**Bernstein.2018**][**Schroder.2021**]

Figure 24.1.: Prinzipieller Aufbau Schrittmotor (2-phasisig) [**Hagl.2021**]

24.1.2. Schrittmotor Bauformen

Wie bei anderen Elektromotoren gibt es auch bei dem Schrittmotor verschiedene Grundbauarten.

- Permanentmagneterreger-Schrittmotor (PM-Schrittmotor)
- Reluktanzschrittmotor (VR-Schrittmotor)
- Hybridschrittmotor

Der **permanentmagnetische Schrittmotor** hat einen Permanentmagneten in dem Rotor verbaut. Hierbei stellt sich der permanentmagnetische Rotor immer so, dass der Nordpol des Rotors dem Nordpol des Statorfeldes gegenüber liegt und der Südpol des Rotors dem Südpol des Statorfeldes. In dieser Ausrichtung ziehen sich die Pole gegenseitig an. Die Drehrichtung des Rotors hängt von der Fließrichtung des Stromes ab. Der permanentmagnetische Schrittmotor entwickelt im ausgeschalteten Zustand ein Drehmoment zur Selbsthaltung. Dies ist aufgrund des permanentmagnetischen Rotors möglich. Bei diesem Drehmoment handelt es sich um das höchste Drehmoment, das auf die Welle des Motors übertragen werden kann, ohne dass diese sich in eine rotierende Bewegung versetzt. Zu dieser Art von Schrittmotoren gehören beispielsweise der Klauenpol-Schrittmotor und der Scheibenmagnet-Schrittmotor. Bei der zweiten Bauart handelt es sich um den **Reluktanzschrittmotor**. Bei dieser Bauart besteht der Rotor aus einem weichmagnetischen Material und besitzt eine gezahnte Form. So lange der Schrittmotor von keinem Strom durchflossen wird, entsteht kein Magnetfeld. Sobald der Motor in Betrieb genommen wird, entsteht ein magnetischer Fluss innerhalb des Rotors. Wird nun eine Wicklung erregt, wird der nächste Zahn des Rotors angezogen. Dadurch, dass die Rotorzähne ungleich der Polteilung sind, kann das System unendlich lange fortgesetzt werden. Die Anzahl der Schritte und die Genauigkeit des Reluktanzschrittmotors ist abhängig von der Anzahl der Zähne auf dem Rotor. Aus technischer Sicht sind mit dieser Bauart Schrittwinkel unter 1° möglich. Damit die Drehrichtung verändert werden kann, sind mindestens zwei Strangwicklungen nötig. Bei den **Hybridschrittmotoren**, auch bekannt als Hybrid-schrittmotoren handelt es sich um eine Kombination aus dem Reluktanzschrittmotor und dem Permanentmagneterreger-Schrittmotor. Durch diese Kombination aus den beiden Schrittmotoren werden die Vorteile aus der kleinen Schrittweite, dem hohen Drehmoment und dem Selbstthaltemoment genutzt. Bei dem Hybridschrittmotor besteht der Rotor aus zwei um eine halbe Zahnteilung versetzten weichmagnetischen Polrädern, die eine zahnförmige Form haben. Bei den beiden Polrädern bildet das eine Polrad den Nordpol und das zweite Polrad den Südpol. Zwischen den beiden Polrädern befindet sich ein Permanentmagnet. Anders als bei anderen Schrittmotoren wird der Rotor bei dieser Bauart axial magnetisiert. Damit ein kleiner Schrittewinkel möglich ist, haben die Statorpole ebenfalls eine zahnförmige Form. Die Ausrichtung des Rotors ist abhängig von der Stromrichtung und wird durch den minimalen Widerstand bestimmt, der sich aus dem Stromfluss durch die einzelnen Stränge ergibt. Wird ein besonders kleiner Schrittewinkel benötigt, kann dies durch Erhöhung der Zähnezahl erreicht werden. [**Schroder.2021**] [**Hagl.2021**] [**Babiel.2023**]

24.1.3. Betriebsarten unipolar und bipolar

Neben den oben bereits genannten Betriebsarten, kann außerdem zwischen dem Unipolarbetrieb und dem Bipolarbetrieb unterschieden werden. Der große Unterschied zwischen den beiden Betriebsarten besteht darin, dass in dem Unipolarbetrieb der Strom in eine Richtung fließt. Bei dem Bipolarbetrieb hingegen fließt der Strom in beide Richtungen. Dies ist möglich, da jeder Wicklungsstrang über eine Vollbrücke gespeist wird. Ein weiterer Unterschied besteht in der Schaltung der Zweige, durch die ein Gleichstrom fließt. In dem Unipolarbetrieb werden die beiden Zweige in Reihe geschaltet. Jeder Wicklungsstrang wird mit zwei Drähten parallel gewickelt. Sind die beiden Wicklungsstränge in dem Bipolarbetrieb parallel gewickelt, müssen die Zweige parallelgeschaltet werden. Im Bipolarbetrieb kann ein höherer Wirkungsgrad erzielt werden, wo hingegen der Unipolarbetrieb eine deutlich einfachere Schaltung aufweist. [Schroder.2021]

24.1.4. Mikroschrittverfahren

Mit dem Mikroschrittverfahren können Zwischenschritte und Mikroschritte erzeugt werden. Es bietet die Fähigkeit, Geräusche und Vibrationen zu minimieren sowie die Auflösung der Umdrehung zu erhöhen. Um Zwischenschritte zu erreichen, müssen beide Wicklungen eines Schrittmotors gleichzeitig mit Strom versorgt werden. Mit dem Sinus/Cosinus-Mikroschrittverfahren können die Mikroschrittverfahren realisiert werden, indem der Strom in jeder Phase des Motors sinusförmig variiert. Durch die Anwendung sinusförmiger Ströme auf die Motorwicklungen wird eine feinere und gleichmäßige Bewegung erzielt, was zur einer präziseren Steuerung und reduzierten mechanischen Vibrationen führt. [MarcMcComb.2024]

Ein Mikroschritt-Treiber unterteilt die Motorschrittewinkel in vier oder mehr Teil. Es handelt sich um einen Controller, der die Methode der Stromreglung verwendet, um die Mikroschritte zu erzeugen. Der Controller bietet außerdem den Vorteil einer erhöhten Flexibilität bei der Integration der Strombegrenzung und der Anpassung der Antriebscharakteristik als Reaktion auf Änderungen der Systemdynamik. Die Anzahl der Unterteilungen ist einstellbar, was eine präzise Steuerung und Anpassung an spezifische Anforderungen ermöglicht. [Babiel.2023]

24.2. Schrittverluste verhindern

Um mögliche Schrittverlusten einzugrenzen und oder sie zu verhindern, wird in diesem Kapitel beschrieben, wie die Ursachen für Schrittverluste oder Stillstand methodisch zu ermitteln sind. [FaulhaberDriveSystems.2020]

24.2.1. Auswahl des Schrittmotors

Zunächst muss ein passender Motor für die Anwendung gewählt werden. Dabei sollten folgende grundlegende Regeln erfüllt sein:

- Motorauswahl durch Höchstwerte für Drehmoment und Drehzahl (Worst-Case-Szenario)
- Verwendung eines Sicherheitsaufschlag von 30 % auf die Drehmoment-Drehzahl-Kennlinie (Kippmoment)
- Sicherstellen, dass externe Ereignisse die Anwendung nicht blockieren können

Sind die geforderten Drehmomente bei den jeweiligen Drehzahlen, den Motorspezifikation entsprechend, dann sind keine Probleme zu erwarten. Ist der Motor zu schwach gewählt und die Anwendung fordert mehr Leistung als der Motor abgeben kann, so bleibt der Motor stehen. Der nächste Schritt ist die Durchführung von Testdurchläufen.

Es soll im Betrieb überprüft werden, ob Schrittverluste auftreten. Schrittmotoren verlieren konstruktionsbedingt nicht nur einen einzigen Schritt. Bei geringen Drehzahlen verliert der Motor ein Vielfaches von vier Schritten.[**FaulhaberDriveSystems.2020**]

24.2.2. Betriebsart

In diesem Abschnitt werden je nach Betriebsart mögliche Ursachen erläutert, falls der Schrittmotor bei den Tests versagt.

Start-Stopp-Betrieb

Der Motor ist mit der Last fest verbunden und wird mit konstanter Drehzahl betrieben. Innerhalb des ersten Schrittes muss der Motor auf die vorgegebene Frequenz beschleunigen wie in Abbildung 24.2 zu sehen ist.[**FaulhaberDriveSystems.2020**]

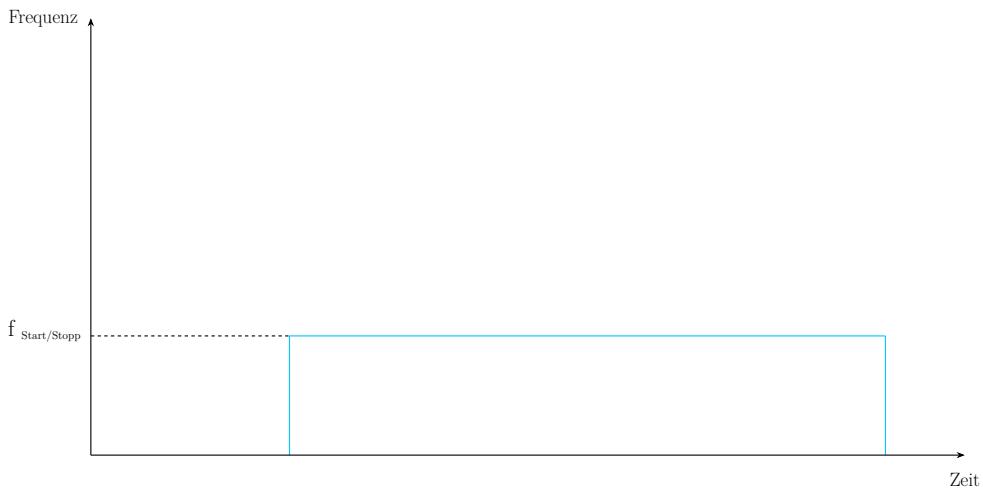


Figure 24.2.: Start-Stopp Frequenz [**FaulhaberDriveSystems.2020**]

Fehlerbild: Motor läuft nicht an (siehe Ursachen und Lösungen aus Tabelle 24.1)

Beschleunigung und Rampenprofil (Trapezförmig)

Der Motor kann mit einer im Controller vorgegebenen Beschleunigungsrate bis auf die Maximalfrequenz beschleunigen wie in Abbildung 24.3 zu sehen ist.[**FaulhaberDriveSystems.2020**]

Fehlerbild: Motor beendet die Beschleunigungsrampe nicht (siehe Ursachen und Lösungen aus Tabelle 24.2)

Ursachen	Lösungen
Last zu Hoch	Falscher Motor, größeren Motor wählen
Frequenz zu hoch	Frequenz reduzieren
Pendelt der Motor von Links nach Rechts	Es könnte eine Phase unterbrochen oder nicht angeschlossen sein, dies muss repariert werden
Phasenstrom passt nicht	Phasenstrom erhöhen

Table 24.1.: Ursachen und Lösungen: Motor läuft nicht an
[**FaulhaberDriveSystems.2020**]

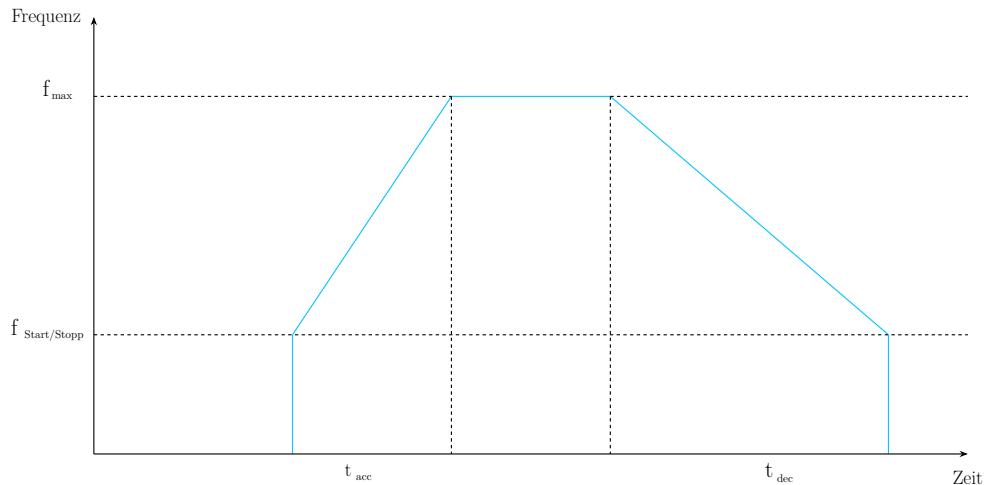


Figure 24.3.: Trapezförmiges Geschwindigkeitsprofil [FaulhaberDriveSystems.2020]

Ursachen	Lösungen
Motor bleibt bei der Resonanzfrequenz hängen	<ul style="list-style-type: none"> Beschleunigung erhöhen, um die Resonanzfrequenz schneller zu durchlaufen Start-Stopp Frequenz über dem Resonanzpunkt wählen Halbschritt- oder Mikroschrittbetrieb verwenden Mechanische Dämpfung vorsehen
Falsche Einstellung von Versorgungsspannung oder Strom zu gering	<ul style="list-style-type: none"> Spannung oder Strom erhöhen Motor mit geringerer Impedanz testen Stromregelung verwenden
Maximaldrehzahl zu hoch	<ul style="list-style-type: none"> Maximaldrehzahl reduzieren Beschleunigungsrampe abflachen
Schlechte Vorgabe der Beschleunigungsrampe durch Elektronik	<ul style="list-style-type: none"> Anderen Controller verwenden

Table 24.2.: Ursachen und Lösungen: Motor beendet die Beschleunigungsrampe nicht [FaulhaberDriveSystems.2020]

Fehlerbild: Motor beschleunigt bis zur Enddrehzahl und bleibt stehen, sobald eine konstante Drehzahl erreicht ist (siehe Ursachen und Lösungen aus Tabelle 24.3)

Ursachen	Lösungen
Motor wird an Leistungsgrenze betrieben und bleibt stehen aufgrund zu hoher Beschleunigung	<ul style="list-style-type: none"> Ruckeln verringern durch geringere Beschleunigungsrate oder durch unterschiedliche Beschleunigungsrampen, erst steil dann flacher Drehmoment erhöhen Motor im Mikroschrittbetrieb betreiben Mechanische Dämpfung vorsehen

Table 24.3.: Ursachen und Lösungen: Motor beschleunigt bis zur Enddrehzahl und bleibt stehen, sobald eine konstante Drehzahl erreicht ist. [FaulhaberDriveSystems.2020]

24.2.3. Externe Ereignisse

Lastrückkopplung

„Manchmal wird der vom Motor angetriebene Mechanismus/Last während der Bewegung „aufgezogen“ und gibt diese Energie wieder an den Motor zurück, wenn die Ströme ausgeschaltet werden. Der Mechanismus könnte z.B. ein Unterstellungsgetriebe sein.“[FaulhaberDriveSystems.2020] Wird diese Energie an den Motor zurück geleitet, kann es passieren, dass der Motor sich um einen Winkel verdreht, der mehr als einem Schritt entspricht. Dabei kann es passieren, dass der Motor nicht ausreichend Drehmoment entwickelt und nicht oder erst nach 4 Vollschriften anläuft. [FaulhaberDriveSystems.2020]

Lösungen:

- Die Kommutierung so programmieren, dass Wert und Polarität vor dem Abschalten gespeichert und beim Wiedereinschalten verwenden werden kann.
- Nicht vollständig den Strom abschalten, sondern bei Motorstillstand einen reduzierten Stand-By Strom aufrechterhalten

Erhöhung der Nutzlast mit der Zeit

„Manchmal läuft der Motor für eine lange Zeit störungsfrei und viel später treten die ersten Schrittverluste auf. In diesem Fall ist es sehr wahrscheinlich, dass die Last, die der Motor „sieht“, sich geändert hat. Das kann auf Verschleiß der Motorlager oder ein externes Ereignis zurückzuführen sein.“[FaulhaberDriveSystems.2020]

Lösungen:

- Prüfen ob ein externes Ereignis durch Veränderung des Mechanismus vorliegt.
- Prüfen ob Lagerverschleiß vorhanden ist. Verwendung von Kugellager erhöhen die Lebensdauer des Motors.
- Verwendung von Schmiermittel um Reibung zu verhindern.

24.3. Beschreibung des verwendeten Schrittmotors

Der hier verwendete NEMA 17 Schrittmotor ist ein bipolarer Schrittmotor. "NEMA" steht hier für die Norm des US-amerikanischen "National Electrical Manufacturers Association". Die 17 in der Modellbezeichnung steht für die Flanschgröße, welche 1,7 x 1,7 Zoll beträgt. Der NEMA 17 besitzt keinen integrierten Treiber, sondern muss

über einen externen Treiber wie den DRV8825 (Kapitel xx) angesteuert werden. In Verbindung mit dem Treiber unterstützt der Schrittmotor bis zu 1/32 Mikroschritte. Der NEMA 17 Schrittmotor findet vor allem in 3D-Druckern, CNC-Fräsen, Kameraschlitten oder Robotikachsen Anwendung.

24.4. Spezifikation

In Tabelle xx sind die Spezifikationen des NEMA 17 aufgelistet.

Spezifikation	Wert
Wellenmaße	4,5 x 22 mm (Einzelwelle)
Anschluss	4-Pol Buchse
Schritte pro Umdrehung	200
Abmessungen	42 x 42 x 42 mm
Haltemoment	0,45 Nm
Nennspannung	3,3 V
Nennstrom	1,5 A
Schrittwinkel	1,8°
Anzahl der Phasen	2
Phasenwiderstand	2,2 Ohm
Phaseninduktivität	5 mH
s Isolationswiderstand	500 V DC
Isolationsklasse	B (130°)
Trägheitsmoment	57 g · cm ²
Rastmoment	0,015 Nm
zuläss. Temperaturbereich	-10 °C bis 50 °C

Table 24.4.: Spezifikationen des NEMA 17 Schrittmotors

24.5. Anschluss an den DRV8825

Der Schrittmotor wird mit dem Schrittmotortreiber DRV8825 verbunden. Die Verbindung kann Tabelle xx entnommen werden.

DRV8825	Schrittmotor
AOUT1	Spule A, Ende 1
AOUT2	Spule A, Ende 2
BOUT1	Spule B, Ende 1
BOUT2	Spule B, Ende 2

Table 24.5.: Pin-Belegung für den Anschluss des Treibers an den Schrittmotor

24.6. Weiterführende Literatur

einfügen

25. Schrittmotortreiber DRV8825

In diesem Kapitel wird beschrieben, was ein Motortreiber ist, welche Aufgaben er erfüllt und wie man den DRV8825 mit dem Arduino Nano 33 BLE Sense verwendet.

25.1. Allgemeine Beschreibung eines Motortreibers

Ein Motortreiber ist ein Elektronikbaustein, der Mikrocontrollern wie dem Arduino Nano 33 BLE Sense ermöglicht elektrische Motoren anzusteuern. Dabei kann es sich zum Beispiel um Gleichstrommotoren oder Schrittmotoren handeln.

Motortreiber haben mehrere Funktionen. Zum einen verstärken sie die Spannung und den Strom, da Mikrocontroller meist zu wenig Leistung liefern, um den Motor direkt zu betreiben. Zum anderen sorgen Motortreiber dafür, dass man die Drehrichtung und die Drehzahl regeln kann. Außerdem haben viele Motortreiber auch eine Schutzfunktion, die vor Überstrom, Überhitzung oder Kurzschluss schützen.

25.2. Spezifische Beschreibung des Schrittmotortreibers DRV8825

Der DRV8825 ist ein Motortreiber, der vor für bipolare Schrittmotoren wie hier der NEMA 17-04 eingesetzt wird. Er wird in Druckern, CNC-Maschinen oder in der Robotik verwendet.

Der Motor lässt sich über den DRV8825 von Vollschritt bis 1/32-Schritt einstellen. Er verfügt über einen konfigurierbaren Abklingmodus (tritt bei Erreichen des Stromlimits ein), sodass langsames, schnelles oder gemischtes Abklingen verwendet werden kann. Außerdem besitzt er einen stromsparenden Schlafmodus, der die interne Schaltung abschaltet und den Ruhestromverbrauch minimiert.

Der DRV8825 verfügt über Schutzfunktionen für Überstrom, Kurzschluss, Unterspannung und Übertemperatur. Außerdem werden Fehler über den nFAULT-Pin (siehe Abbildung xx) signalisiert.

25.2.1. Anschluss des Treibers mit dem Arduino Nano 33 BLE Sense

Tabelle xx kann entnommen werden, wie der DRV8825 an den Arduino Nano 33 BLE Sense angeschlossen wird. Tabelle xx zeigt, wie der DRV8825 an den Schrittmotor NEMA 17-04 angeschlossen wird.

25.3. Spezifikationen

Die Spezifikationen sind in Tabelle xx und xx zu sehen.

25.4. Kalibrierung

Da der Treiber den Strom durch den Schrittmotor regelt ist die Kalibrierung des DRV8825 erforderlich, um zu vermeiden, dass der Schrittmotor überhitzt oder der Treiber überlastet.

DRV8825	Arduino Nano 33 BLE Sense
STEP	z.B. D3
DIR	z.B. D4
EN	GND
nRESET	HIGH (z.B. an 5V)
nSLEEP	HIGH (z.B. an 5V)
GND	GND
VDD	5V

Table 25.1.: Pin-Belegung für den Anschluss des Treibers an den Arduino

DRV8825	Schrittmotor
AOUT1	Spule A, Ende 1
AOUT2	Spule A, Ende 2
BOUT1	Spule B, Ende 1
BOUT2	Spule B, Ende 2

Table 25.2.: Pin-Belegung für den Anschluss des Treibers an den Schrittmotor

Spezifikation	Wert
Versorgungsspannung	-0,3 V bis 47 V
Digitale Eingangsspannung	-0,5 V bis 7 V
Referenzspannung	-0,3 V bis 4 V
Motorstrom	bis zu 2,5 A
Temperaturbereich	-40 °C bis 150 °C

Table 25.3.: Absolute Maximalwerte

Spezifikation	Wert
Versorgungsspannung	8,2 V - 45 V
Referenzspannung	1 V bis 3,5 V
Motorstrom	0 A bis 2,5 A

Table 25.4.: Empfohlene Betriebsbedingungen

Für die Kalibrierung wird ein Multimeter, ein Schraubendreher für das Potentiometer, der GND-Pin des Arduino und der VREF-Messpunkt (Metallplättchen am Potentiometer) benötigt.

Im ersten Schritt ist es wichtig, dass der Schrittmotor nicht angeschlossen ist. Als nächstes wird die Stromversorgung eingeschaltet. Nun wird das Multimeter verwendet, dabei wird der Pluspol am VREF-Messpunkt verbunden und der Minuspol am GND-Anschluss des Arduino. Aus dem abgelesenen Wert kann nun die Zielspannung für VREF berechnet werden und dann am Potentiometer mit dem Schraubendreher eingestellt werden. Dabei ist darauf zu achten das Potentiometer nicht zu überdrehen, da es empfindlich ist.

Nach dem Kalibrieren des Treibers kann der Schrittmotor angeschlossen werden.

25.5. Einfaches Beispiel mit Code

```
1 // Pin-Zuweisung
2 const int stepPin = 3;    // STEP
3 const int dirPin = 4;    // DIR
4 const int enablePin = 5; // optional: nENBL (LOW = aktiv)
5
6 void setup() {
7     pinMode(stepPin, OUTPUT);
8     pinMode(dirPin, OUTPUT);
9     pinMode(enablePin, OUTPUT);
10
11    digitalWrite(enablePin, LOW);   // Treiber aktivieren
12    digitalWrite(dirPin, HIGH);    // Drehrichtung setzen (HIGH oder LOW)
13 }
14
15 void loop() {
16     // Einen Schritt erzeugen:
17     digitalWrite(stepPin, HIGH);
18     delayMicroseconds(1000); // Schrittempulsdauer (kleiner = schneller)
19     digitalWrite(stepPin, LOW);
20     delayMicroseconds(1000); // Pausenzeit (je nach gewünschter Drehzahl)
21 }
```

25.6. Weiterführende Literatur

einfügen

26. Geschwindigkeitssensor LM393

26.1. 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.

26.2. 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.

26.2.1. 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.

26.2.2. 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 Fototransistor verbunden. Wenn Infrarotlicht auf dem Empfänger auftrifft (durch Loch der Lochscheibe) ändert sich die Spannung am Eingang B. Wenn die Spannung im Fototransistor 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 Fototransistor 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.

26.2.3. 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:

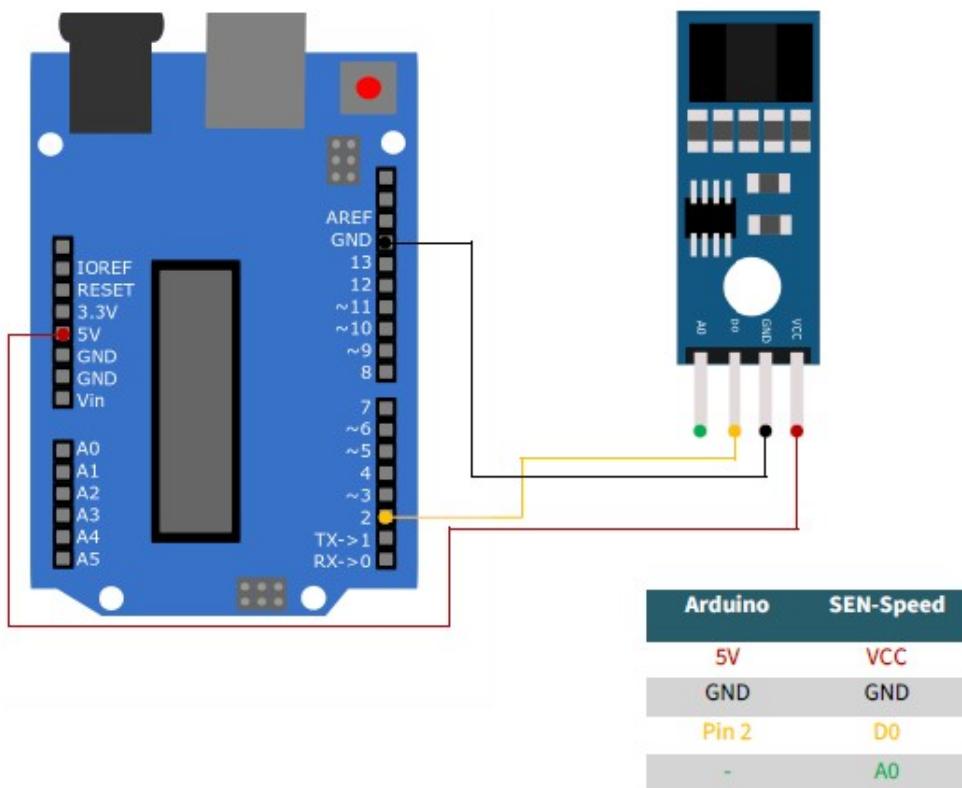


Figure 26.1.: Anschluss des Sensors ans den Arduino Nano 33 BLE Sense

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 26.1.: Spezifikationen des Speedsensor LM393

26.3. Spezifikationen

26.4. 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

26.4.1. 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)

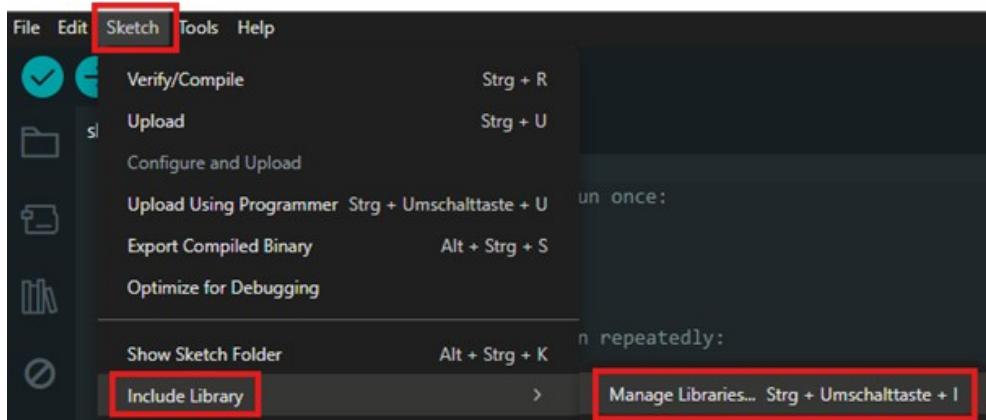


Figure 26.2.: Aufrufen der Bibliotheken in der Arduino IDE

26.4.2. 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 26.3.: Installation der TimerOne Bibliothek

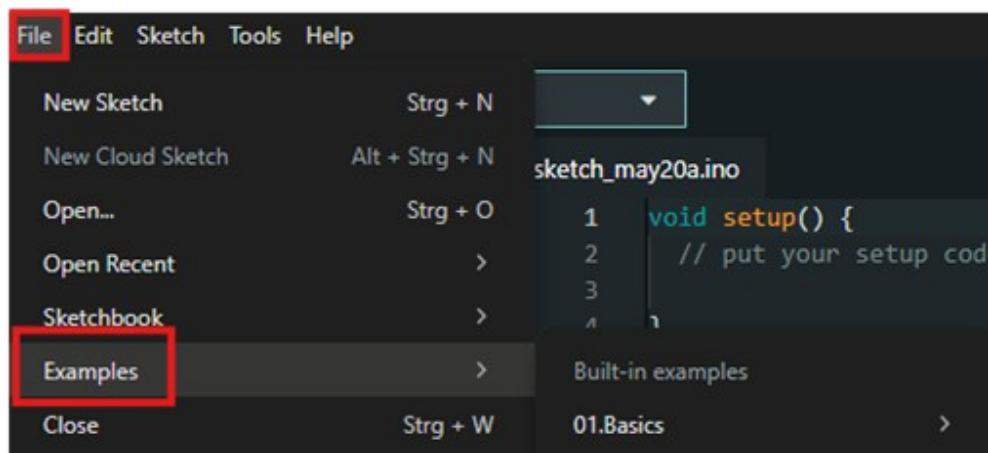


Figure 26.4.: Aufrufen der Beispiele für die TimerOne Bibliothek

26.5. 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.

26.6. 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.

26.7. 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.

26.8. Weiterführende Literatur

einfügen

```

// 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
}

// Zaehlt 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 26.5.: wird noch in Code umgewandelt!

27. OLED-Display

In diesem Kapitel wird erklärt wie was ein OLED-Display ist, wie es funktioniert und wie es in diesem Projekt eingebunden wird.

27.1. Allgemeine Beschreibung eines OLED-Displays

Ein OLED-Display ist eine Art von Bildschirm, bei der organische Materialien Licht emittieren, wenn Strom durch sie hindurchfließt. "OLED" steht dabei für "Organic Light Emitting Diode".

Im Gegensatz zu LCD-Displays benötigen OLED-Displays keine Hintergrundbeleuchtung, da jede einzelne Pixelzelle selbst leuchtet. Außerdem sind OLED-Displays oft dünner, leichter, bieten eine besseren Kontrast und einen breiteren Betrachtungswinkel. Durch die schnelle Reaktionszeit von OLED-Displays werden sie vor allem in Smartphones, Fernsehern, Smartwatches oder High-End-Monitoren verbaut.

Nachteile von OLED-Displays liegen darin, dass sie teurer sind als LCD-Displays. Außerdem haben vor allem blaue OLED-Displays oft eine kürzere Lebensdauer und im Allgemeinen besteht die Gefahr des Einbrennens (Burn-in). Das bedeutet, dass es bei statischen Bildern dazu kommen kann, dass Reste des Bildes auf dem Display sichtbar bleiben.

27.2. Spezifische Beschreibung OLED-Displays

Es wird ein 2,42 Zoll OLED-Display von Joy-IT verwendet. Dabei handelt es sich um ein Display mit 128 x 64 Pixeln, welches gelbe Schrift auf schwarzem Hintergrund darstellt. Es besitzt eine SPI- und eine I2C-Schnittstelle, sodass es mit dem verwendeten Arduino Nano 33 BLE Sense kompatibel ist.

27.3. Anschluss des Sensors mit dem Arduino Nano 33 BLE Sense

In Tabelle xx ist gezeigt wie das OLED-Display korrekt an den Arduino Nano 33 BLE Sense angeschlossen wird:

OLED-Display	Arudino Nano 33 BLE Sense
1	GND
2	3V3
4	D9
5	D8
7	A5
8	A4

Table 27.1.: Pinbelegung für I2C-Verbindung zwischen OLED-Display und Arduino Nano 33 BLE Sense

27.4. Spezifikationen

Spezifikation	Wert
Typ	Negativ OLED
Auflösung	128 x 64 Pixel
Abmessungen	71 x 50 x 7 mm
Versorgungsspannung	3 - 5 V
Versorgungsstrom	90 mA
Logik Level	3 V
High Level Eingang	mind. 2,4 V
Low Level Eingang	max. 0,6 V
Helligkeit	100 - 120 CD/cm ²
Kontrast	> 2000:1
Blickwinkel	> 160°
zuläss. Betriebstemperatur	-40 °C bis 85 °C
Lagerungstemperatur	-45 °C bis 90 °C

Table 27.2.: Spezifikationen des OLED-Displays

27.5. Bibliothek

Für die Verwendung des OLED-Displays mit dem Arduino Nano 33 BLE Sense wird die U8g2 by oliver Bibliothek benötigt.

[Welche Funktionen werden verwendet + Erklärung](#)

27.5.1. Installation

Im Folgenden wird erklärt, wie die U8g2 Bibliothek installiert wird.

1. Öffnen der Arduino IDE
2. Klick auf Sketch, Bibliothek einbinden, Bibliotheken verwalten
3. In der Suchleiste U8g2 eingeben
4. Klick auf Installieren (Version aus Abbildung xx entnehmen)

27.5.2. Code-Beispiel

Auf Code-Beispiele für die U8g2 Bibliothek kann man über Datei, Beispiele, U8g2 zugreifen.

27.6. Einfaches Beispiel mit Code

Mit dem Code-Beispiel kann getestet werden, ob das Display korrekt angeschlossen ist. Wenn die Verbindung korrekt ist, dann erscheint auf dem Display "Hello World".

27.7. Weiterführende Literatur

[einfügen](#)

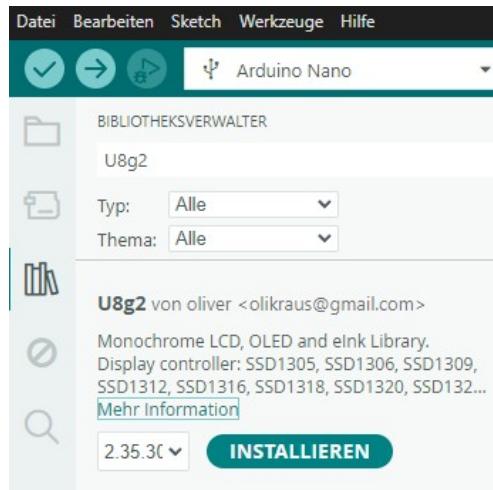


Figure 27.1.: Installation der U8g2 Bibliothek

```
1 #include <U8g2lib.h>
2 #include <Wire.h>
3
4 // Konstruktor für SSD1309 im I2C-Modus, Hardware I2C mit default Pins (A4=SDA, A5=SCL)
5 U8G2_SSD1309_128X64_NONAME2_1_HW_I2C u8g2(U8G2_R0, /* reset= */ U8X8_PIN_NONE);
6
7 void setup() {
8     u8g2.begin(); // Initialisiert das Display
9 }
10
11 void loop() {
12     u8g2.clearBuffer(); // Bildspeicher löschen
13     u8g2.setFont(u8g2_font_ncenB08_tr); // Schriftart wählen
14     u8g2.drawString(0, 24, "Hello World!"); // Text auf Puffer schreiben
15     u8g2.sendBuffer(); // Puffer an Display senden
16     delay(1000); // 1 Sekunde warten
17 }
```

Figure 27.2.: wird noch in Code umgewandelt!

28. Sensor XY

Introduction

WS:cite books,
applications, board

28.1. General

General description

cite books

28.2. Specific Sensor/Actor

cite board

28.3. Specification

- Cite the data sheet
- Extract te information from the data sheet
- Circuit Diagram

28.4. Library

28.4.1. Description

28.4.2. Installation

- data types
- data structure
- data quantity
- data quality

28.4.3. Functions

28.5. Example

In this section, a simple example is described in detail, which demonstrates how the library supports the sensor/actor.

28.5.1. Manual**28.5.2. Inside the Example****28.5.3. Code****28.5.4. Files****28.6. Calibration**

cite method

28.7. Simple Code**28.8. Simple Application****28.9. Tests****28.9.1. Simple Function Test****28.9.2. Test all Functions****28.10. Further Readings**

29. Actor XY

Introduction

29.1. General

General description

cite books

29.2. Specific Sensor/Actor

cite board

29.3. Specification

- Cite the data sheet
- Extract te information from the data sheet
- Circuit Diagram

29.4. Library

29.4.1. Description

29.4.2. Installation

- data types
- data structure
- data quantity
- data quality

29.4.3. Functions

29.5. Example

In this section, a simple example is described in detail, which demonstrates how the library supports the sensor/actor.

29.5.1. Manual**29.5.2. Inside the Example****29.5.3. Code****29.5.4. Files****29.6. Calibration**

cite method

29.7. Simple Code**29.8. Simple Application****29.9. Tests****29.9.1. Simple Function Test****29.9.2. Test all Functions****29.10. Further Readings**

30. Getting Started

30.1. 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:

30.1.1. 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.

30.1.2. 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.

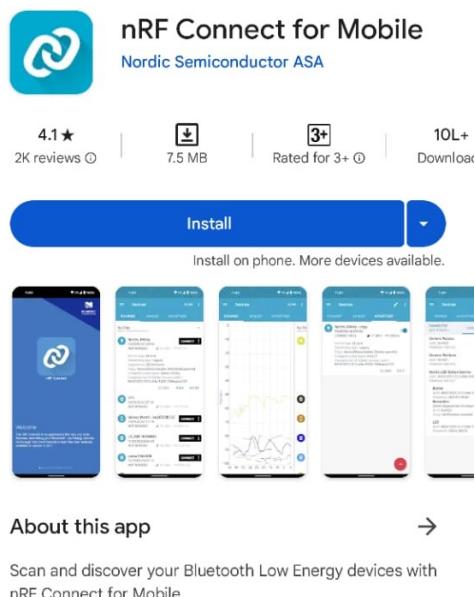


Figure 30.1.: NRF Connect Application

30.2. 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.

30.2.1. Connect to Arduino Nano 33 BLE Sense Lite through NRFconnect

30.2.2. 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.

30.2.3. 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.

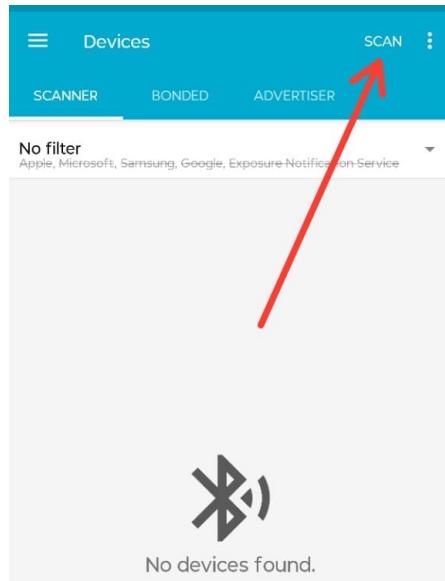


Figure 30.2.: Landing page

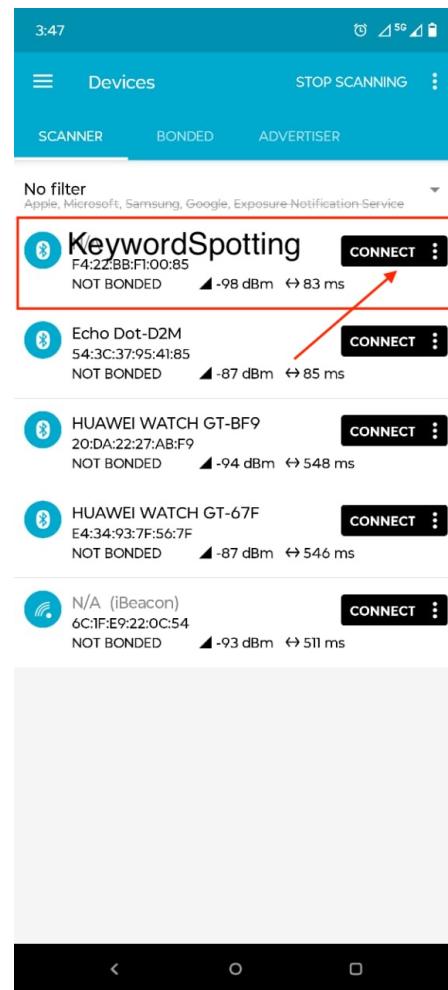


Figure 30.3.: List of devices

Part IV.

Applikation <Title>

31. Application

32. Conclusion XY

33. To DoX Y

Part V.

Templates

34. Sensor

Introduction

34.1. General

General description

cite books

34.2. Specific Sensor

cite board

34.3. Specification

- cite data sheet

- Circuit Diagram

34.4. Library

34.4.1. Description

34.4.2. Installation

34.4.3. Functions

34.4.4. Example's Manual

34.4.5. Inside the Example

34.4.6. Example's Code

34.4.7. Example's Files

34.5. Calibration

cite method

34.6. Simple Code

34.7. Simple Application

34.8. Tests

34.8.1. Simple Function Test

34.8.2. Test all Functions

34.9. Simple Application

34.10. Further Readings

35. Package Example

35.1. Introduction

35.2. Description

35.3. Installation

`pip packageExample`

35.4. Example - Manual

35.5. Example

35.6. Example - Code

35.7. Example - Files

`PackageExample.py`

35.8. Further Reading

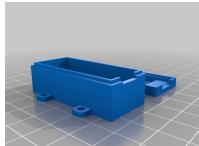
References

- [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

36. 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

37. Hints

Please, use **biber.exe**.

If you want to create a symbol directory, call makeindex:

makeindex %.nlo -s nomencl.ist

37.1. 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:

$$b_{i,n}(\lambda) = \binom{n}{i} (1-\lambda)^{n-i} \lambda^i, \quad \lambda \in [0, 1], \quad i = 0, \dots, n \quad (37.1)$$

Definition. The binomial coefficient is defined by:

$$\binom{n}{i} = \frac{n!}{i!(n-i)!}, \quad i = 0, \dots, n \quad (37.2)$$

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

$$C(\lambda) = \sum_{i=0}^n Q_i \cdot b_{i,n}(\lambda) \quad (37.3)$$

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.

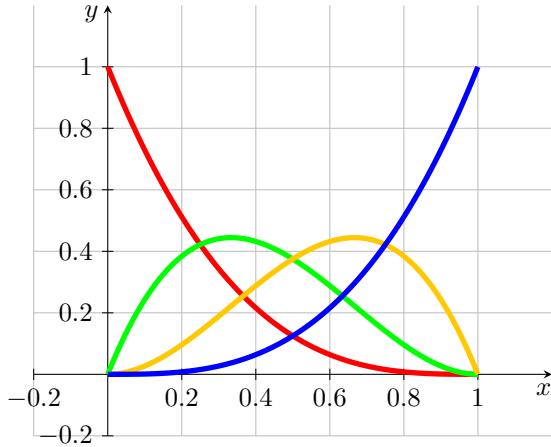


Figure 37.1.: Bernstein polynomial of degree 3

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:

$$Q_0 = P_0, \quad Q_1 = P_0 + \lambda_0 \vec{t}_0, \quad Q_2 = P_1 - \lambda_1 \vec{t}_1, \quad Q_3 = P_1 \quad (37.4)$$

[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 ??.

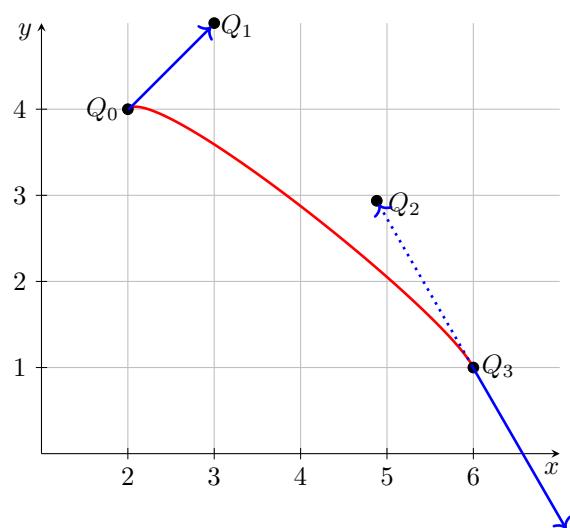


Figure 37.2.: Bézier curve for example 37.1

38. Verrundung mit einem Kreisbogen

38.1. 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 $HP(L, \alpha)$.

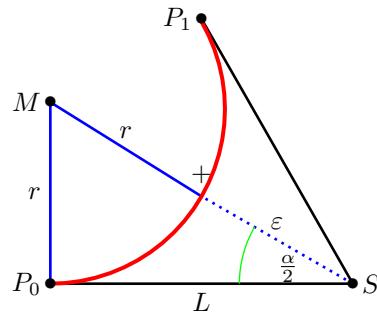


Figure 38.1.: Smoothing a corner with the help of an arc - triangle

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:

$$L(\varepsilon, \alpha) = \varepsilon \cdot \frac{P_{1,x}}{\sqrt{P_{1,x}} - P_{1,y}} = \varepsilon \cdot \frac{L + L \cdot \cos(\alpha)}{\sqrt{L + L \cdot \cos(\alpha)} - L + L \cdot \cos(\alpha)}$$

- b) When the distance L is specified, the following maximum error results:

$$\varepsilon(L, \alpha) = L \cdot \frac{\sqrt{P_{1,x}} - P_{1,y}}{P_{1,x}} = L \cdot \frac{\sqrt{L + L \cdot \cos(\alpha)} - L + L \cdot \cos(\alpha)}{L + L \cdot \cos(\alpha)}$$

These considerations result in limitations for the use of this strategy, which are summarised in the following comment.

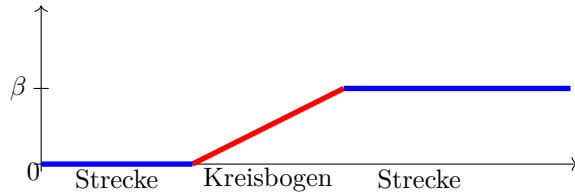


Figure 38.2.: Angular change with blending arc

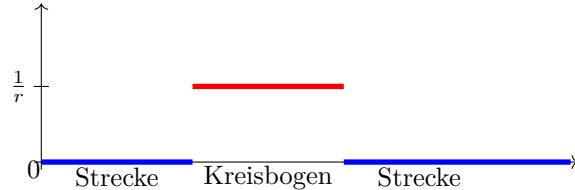


Figure 38.3.: Curvature progression for a blending arc

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)

$$\vec{t}_0 = \vec{t}_1 \Leftrightarrow \alpha = \pi$$

38.2. Bewertung

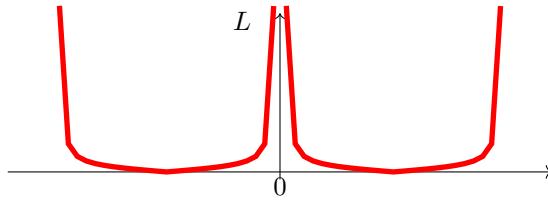
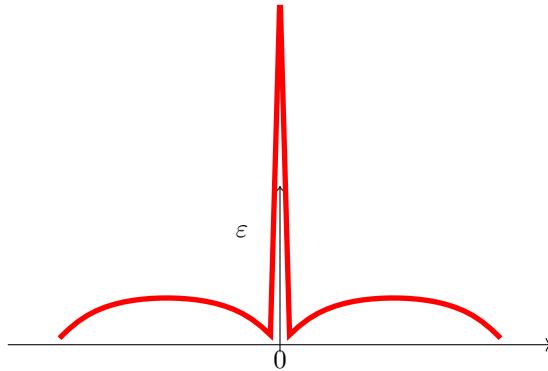
Rounding by means of a circular arc leads to a continuous course of the angle change. The figure ?? illustrates this.

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 38.4.: Maximum range for a blending arc of a circle - $L(\varepsilon = \text{const}, \alpha)$ Figure 38.5.: 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.

38.3. Examples

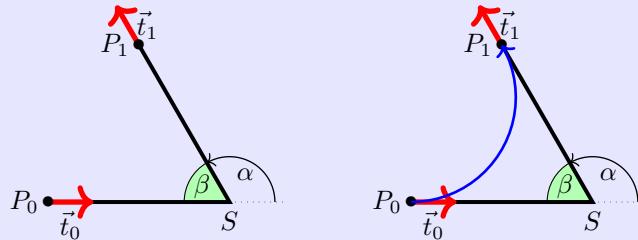
Beispiel. Let it be the symmetric Hermite problem

$$\left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 6.0 + 6.0 \cdot \cos\left(\frac{2}{3}\pi\right) \\ 6.0 \cdot \sin\left(\frac{2}{3}\pi\right) \end{pmatrix}, \begin{pmatrix} \cos\left(\frac{2}{3}\pi\right) \\ \sin\left(\frac{2}{3}\pi\right) \end{pmatrix}, \begin{pmatrix} 6.0 \\ 0 \end{pmatrix}, 6.0 \right\}$$

with $L = 6$ and $\alpha = \frac{2}{3}\pi$.

For the blending arc follows:

$$M = \begin{pmatrix} 0 \\ 3,46412 \end{pmatrix}; \quad r = 3,46412; \quad \phi_0 = -\frac{\pi}{2}; \quad \alpha = \frac{2}{3}\pi$$



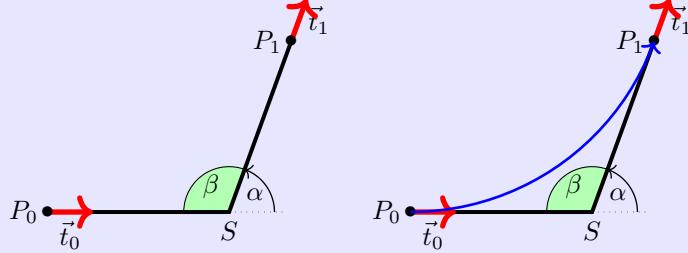
Beispiel. Let it be the symmetric Hermite problem

$$\left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 6.0 + 6.0 \cdot \cos(\frac{7}{18}\pi) \\ 6.0 \cdot \sin(\frac{7}{18}\pi) \end{pmatrix}, \begin{pmatrix} \cos(\frac{7}{18}\pi) \\ \sin(\frac{7}{18}\pi) \end{pmatrix}, \begin{pmatrix} 6.0 \\ 0 \end{pmatrix}, 6.0 \right\}$$

with $L = 6$ and $\alpha = \frac{7}{18}\pi$.

For the blending arc follows:

$$M = \begin{pmatrix} 0 \\ 8,5689 \end{pmatrix}; \quad r = 8,5689; \quad \phi_0 = -\frac{\pi}{2}; \quad \alpha = \frac{7}{18}\pi$$



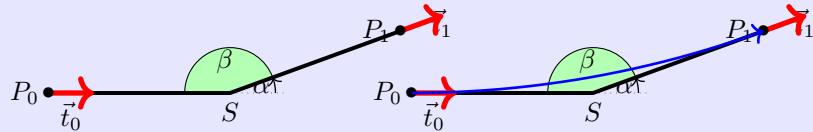
Beispiel. Let it be the symmetric Hermite problem

$$\left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 6.0 + 6.0 \cdot \cos(\frac{1}{9}\pi) \\ 6.0 \cdot \sin(\frac{1}{9}\pi) \end{pmatrix}, \begin{pmatrix} \cos(\frac{1}{9}\pi) \\ \sin(\frac{1}{9}\pi) \end{pmatrix}, \begin{pmatrix} 6.0 \\ 0 \end{pmatrix}, 6.0 \right\}$$

with $L = 6$ and $\alpha = \frac{1}{9}\pi$.

For the blending arc follows:

$$M = \begin{pmatrix} 0 \\ 34,0277 \end{pmatrix}; \quad r = 34,0277; \quad \phi_0 = -\frac{\pi}{2}; \quad \alpha = \frac{1}{9}\pi$$

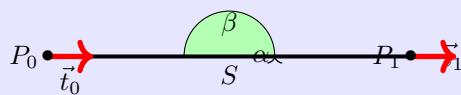


Beispiel. Let it be the symmetric Hermite problem

$$\left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 12.0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 6.0 \\ 0 \end{pmatrix}, 6.0 \right\}$$

with $L = 6$ and $\alpha = 0$.

There is no blending arc here.



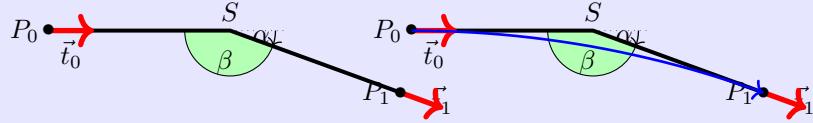
Beispiel. Let it be the symmetric Hermite problem

$$\left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 6.0 + 6.0 \cdot \cos(-\frac{1}{9}\pi) \\ 6.0 \cdot \sin(-\frac{1}{9}\pi) \end{pmatrix}, \begin{pmatrix} \cos(-\frac{1}{9}\pi) \\ \sin(-\frac{1}{9}\pi) \end{pmatrix}, \begin{pmatrix} 6.0 \\ 0 \end{pmatrix}, 6.0 \right\}$$

with $L = 6$ and $\alpha = -\frac{1}{9}\pi$.

For the blending arc follows:

$$M = \begin{pmatrix} 0 \\ -34,0277 \end{pmatrix}; \quad r = 34,0277; \quad \phi_0 = \frac{\pi}{2}; \quad \alpha = \frac{1}{9}\pi$$



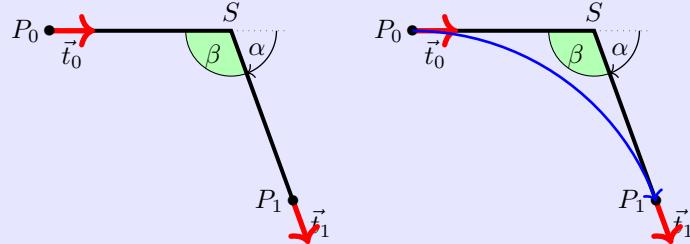
Beispiel. Let it be the symmetric Hermite problem

$$\left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 6.0 + 6.0 \cdot \cos(-\frac{7}{18}\pi) \\ 6.0 \cdot \sin(-\frac{7}{18}\pi) \end{pmatrix}, \begin{pmatrix} \cos(-\frac{7}{18}\pi) \\ \sin(-\frac{7}{18}\pi) \end{pmatrix}, \begin{pmatrix} 6.0 \\ 0 \end{pmatrix}, 6.0 \right\}$$

with $L = 6$ and $\alpha = -\frac{7}{18}\pi$.

For the blending arc follows:

$$M = \begin{pmatrix} 0 \\ -8,5689 \end{pmatrix}; \quad r = 8,5689; \quad \phi_0 = \frac{\pi}{2}; \quad \alpha = -\frac{7}{18}\pi$$



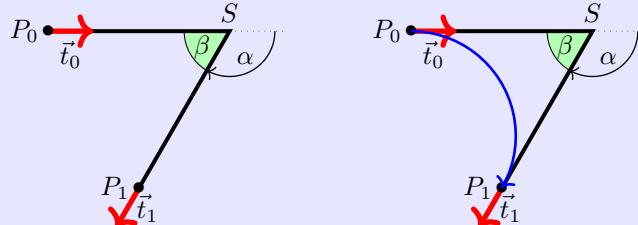
Beispiel. Let it be the symmetric Hermite problem

$$\left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 6.0 + 6.0 \cdot \cos(-\frac{2}{3}\pi) \\ 6.0 \cdot \sin(-\frac{2}{3}\pi) \end{pmatrix}, \begin{pmatrix} \cos(-\frac{2}{3}\pi) \\ \sin(-\frac{2}{3}\pi) \end{pmatrix}, \begin{pmatrix} 6.0 \\ 0 \end{pmatrix}, 6.0 \right\}$$

with $L = 6$ and $\alpha = -\frac{2}{3}\pi$.

For the blending arc follows:

$$M = \begin{pmatrix} 0 \\ -3,4641 \end{pmatrix}; \quad r = 3,46412; \quad \phi_0 = \frac{\pi}{2}; \quad \alpha = -\frac{2}{3}\pi$$



39. Maple files

[Wat17c; Wat17d; Wat17b; Wat17e; Wat17a; Wat17f]

39.1. 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.

39.2. 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.

39.3. 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“.

39.4. 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;
```

39.4.1. 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`

39.4.2. 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);`

39.4.3. 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.

39.4.4. 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.

39.5. 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.

39.5.1. 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

39.5.2. 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

39.5.3. 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

39.5.4. 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

39.5.5. 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

39.5.6. 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

39.5.7. 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

39.5.8. 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

39.5.9. 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

39.5.10. 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

39.5.11. 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.

39.5.12. 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:

[MVBIAARC, [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

39.6. Programme Flowchart

39.6.1. Overall Flow

39.6.2. Verrundung der Kurve

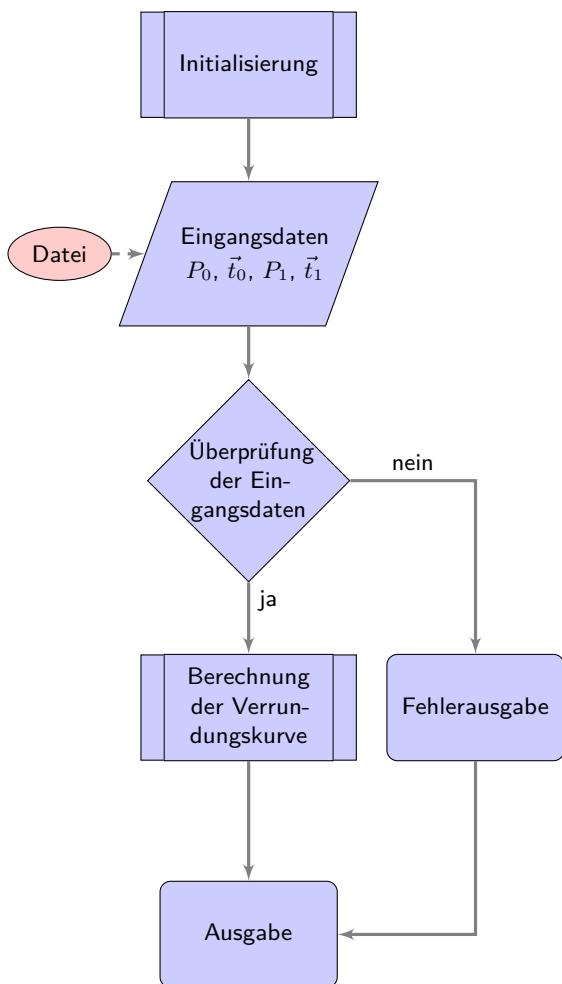


Figure 39.1.: Programme flow chart „Corner rounding“

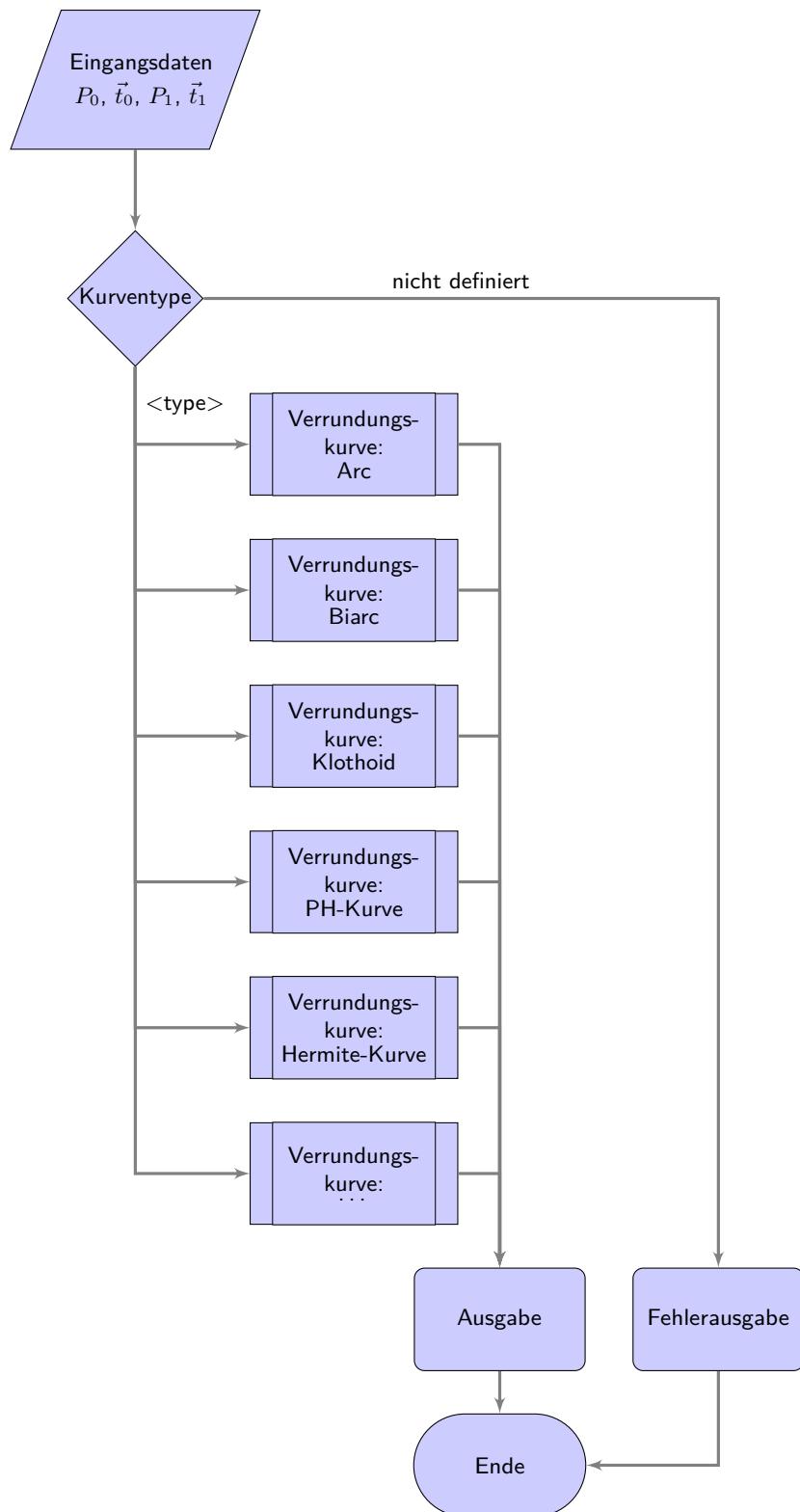


Figure 39.2.: Programme flowchart „Selection of the rounding strategies“

40. Representation of Python Programs

It is possible to integrate a part into the document, see 40.1. 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 40.1.: 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`

41. Representation of Programs written for Arduino Boards

It is possible to integrate a part into the document, see 41.1. 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 41.1.: 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 }
```

42. First Chapter

...

A Speicherprogrammierbare Steuerung (SPS) is ...

A Computerized Numerical Control (CNC) needs a SPS (PLC) to ...

43. 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

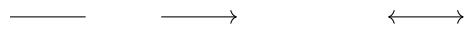
²Zaf+20.

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}
```



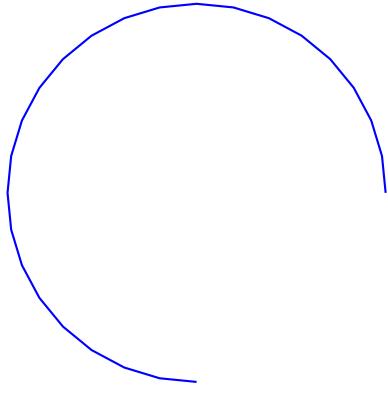
Drawing a thick blue line

```
\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)}); 

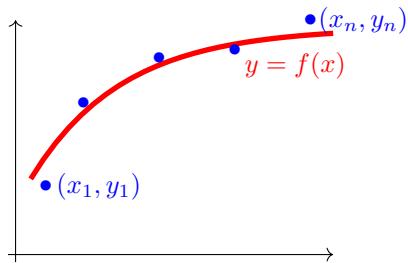
% Bezeichnung
\node [red] (O) at (3.5,2.5) {$y=f(x)$};

% 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) {$\bullet$};

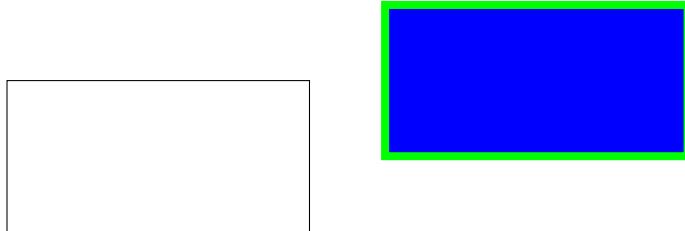
% Koordinatensystem
\draw [color=black,->] (-0.2,-0.1) -- (-0.2,3.1);
\draw [color=black,->] (-0.3,0) -- (4,0);
\end{tikzpicture}
```



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}
```

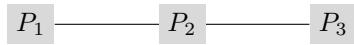


Use of points

```
%\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) { $P_3$ };

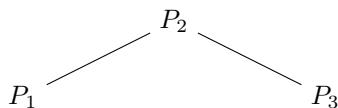
  \begin{scope}[on background layer]
    \draw (P1) -- (P3);
  \end{scope}
\end{tikzpicture}
```



Use of nodes

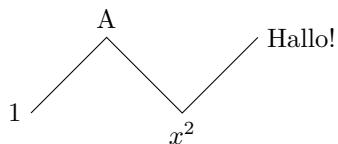
```
\begin{tikzpicture}
  \node (P1) at (0,0){$P_1$};
  \node (P2) at (2,1){$P_2$};
  \node (P3) at (4,0){$P_3$};

  \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).
- [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 Reference – Interrupt*. 2019. URL: <https://www.arduino.cc/reference/de/language/functions/external-interrupts/attachinterrupt/> (visited on 06/26/2024).
- [Ardf] *Arduino Reference – Wire*. 2022. URL: <https://www.arduino.cc/reference/en/language/functions/communication/wire/> (visited on 06/26/2023).
- [Ardg] *Arduino Reference – Wire.setClock()*. 2022. URL: <https://www.arduino.cc/reference/en/language/functions/communication/wire/setclock/> (visited on 06/26/2023).
- [Ardh] *Arduino Reference – loop()*. 2019. URL: <https://www.arduino.cc/reference/en/language/structure/sketch/loop/> (visited on 06/26/2023).
- [Ardi] *Arduino Reference – setup()*. 2019. URL: <https://www.arduino.cc/reference/en/language/structure/sketch/setup/> (visited on 06/26/2023).
- [Ardj] *Arduino Referenz – pinMode()*. 2019. URL: <https://reference.arduino.cc/reference/de/language/functions/digital-io/pinmode/> (visited on 06/26/2023).
- [Ardk] *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).

-
- [Ard21] 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] 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] 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. *Getting started with the Arduino Nano 33 BLE Sense*. 2024. URL: <https://wiki-content.arduino.cc/en/Guide/NANO33BLESense>.
- [Ard24c] Arduino, ed. *Language Reference*. 2024. URL: <https://docs.arduino.cc/language-reference/>.
- [Ari21] D. Aristi. *LCD_I2C library on GitHub*. 2021. URL: https://github.com/blackhack/LCD_I2C (visited on 07/09/2023).
- [Az] 0,96 Zoll OLED Display - Datenblatt. [pdf]. AZ-Delivery, 2023. URL: <https://www.az-delivery.de/products/0-96zolldisplay>.
- [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.
- [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.
- [Box21] J. Boxall. *Arduino Workshop - A Hands-On Introduction with 65 Projects*. 2. No Starch Press, 2021. ISBN: 9781593274481.
- [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>.
- [DIN66025-2] DIN Deutsches Institut für Normung e.V. *DIN 66025-2:1988-09: Programmierung für numerisch gesteuerte Arbeitsmaschinen: Wegbedingungen und Zusatzfunktionen*. Norm. Berlin, Sept. 1988.

- [Dev24] S. Developers. *Sphinx*. 2024. URL: <https://www.sphinx-doc.org/en/master/> (visited on 12/09/2024).
- [Din] *DIN EN ISO 13850: Sicherheit von Maschinen - Not-Halt-Funktion - Gestaltungsleitsätze (ISO 13850:2015)*. 2016. URL: <https://www.din.de/de/mitwirken/normenausschusse/nam/veroeffentlichungen/wdc-beuth:din21:233572513>.
- [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.
- [Fun] *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).
- [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.
- [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.
- [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).
- [Hil22] G. Hiller. *Color measurement - the CIE color space*. [pdf]. Datacolor, 2022.
- [Jak+10] G. Jaklic et al. “On Interpolation by Planar Cubic G^2 Pythagorean-Hodograph Spline Curves”. In: *Mathematics of Computation* (2010). [pdf].
- [KKV14] K. Karvinen, T. Karvinen, and V. Valtokari. *Sensoren – messen und experimentieren mit Arduino und Raspberry Pi*. dpunkt, 2014. ISBN: 97833864901607.

-
- [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>.
- [Lib21] A. Libraries, ed. *Arduino_LSM6DSOX Library for Arduino*. 2021. URL: https://github.com/arduino-libraries/Arduino_LSM6DSOX (visited on 07/09/2023).
- [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. URL: <https://www.mdpi.com/2072-666X/13/4/584>.
- [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/>.
- [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.
- [Ous18] J. Ousterhout. *A Philosophy of Software Design*. [pdf]. Yaknyam Press, 2018. ISBN: 978-1-7321022-0-0.
- [RS17] 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.

- [Rei24] 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>.
- [Rus+07] D. Russell et al. *Apparatus and Methods for 3D Printing*. United States Patent, Patent N0.: US 7,291,002 B2. 2007.
- [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.
- [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}.
- [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.
- [She] *Voltage Sensor / 170640 - Data Sheet*. 170640. [pdf]. Shenzhen Global Tech Co. 2015.
- [Sim] *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.

-
- [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.
- [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.
- [Voš24] A. Voš. “Volumio mit Drehgebern erweitern”. In: *Make Magazin* (2024). [pdf].
- [W3c] *Wire.h (I^2C)*. 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](https://de.maplesoft.com/support/help/Maple/view.aspx?path=module/description). 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](https://de.maplesoft.com/support/help/Maple/view.aspx?path=module/export). 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](https://de.maplesoft.com/support/help/maple/view.aspx?path=module). 2017. URL: <https://de.maplesoft.com/support/help/maple/view.aspx?path=module>.
- [Wat17d] Waterloo Maple Inc. 2017. *ModuleLoad*. [letzter Zugriff 14.09.2017](https://de.maplesoft.com/support/help/Maple/view.aspx?path=ModuleLoad). 2017. URL: <https://de.maplesoft.com/support/help/Maple/view.aspx?path=ModuleLoad>.
- [Wat17e] Waterloo Maple Inc. 2017. *Option*. [letzter Zugriff 14.09.2017](https://de.maplesoft.com/support/help/Maple/view.aspx?path=module/option). 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](https://de.maplesoft.com/support/help/Maple/view.aspx?path=procedure). 2017. URL: <https://de.maplesoft.com/support/help/Maple/view.aspx?path=procedure>.
- [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.

Index

- 3D printer, 289
- Programmable Logic Controller, 287
- File
 -/images/, 69
 - .doxyfile, 64
 - .ino, 65
 - Adafruit-GFX.h, 196
 - Adafruit-SSD1306.h, 196
 - Arduino, 34
 - Arduino LSM9DS1.h, 166
 - Arduino_LSM9DS1.h, 165
 - Blink.ino, 286
 - blink.ino, 49, 50
 - Blink.py, 284
 - blnik.ino, 45
 - BuiltinLED.cpp, 90
 - BuiltinLED.h, 90
 - doxygen-1.12.0-setup.exe, 54
 - doxygen.exe, 58
 - Fade, 50
 - File/Examples/01.Basics, 50
 - graphviz-12.2.1 (64-bit) EXE installer, 56
 - Hello World, 200, 202
 - HelloWorldWire, 195
 - images, 69
 - index.html, 67
 - Kalman.h, 165, 166
 - LED.cpp, 95
 - LED.h, 94
 - libraries, 36–38
 - LSM6DSOXSensor.h, 155
 - Mag_raw.txt, 80
 - MySketch, 34
 - MySketch.ino, 34
 - Nano33BLESenseLE, 36
 - Nano33BLESenseLED, 37, 38
 - PackageExample.py, 251
 - PowerLED.cpp, 96
 - PowerLED.h, 96
 - RGBLED.cpp, 103
 - RGBLED.h, 102
 - SignsOfLife.cpp, 98
 - SignsOfLife.h, 98
 - SimpleAccelerometer, 139
 - SSD1306Ascii.h, 141
 - TestBattery.ino, 184, 186
 - Wire.h, 141, 165, 167, 196
 - wire.h, 166
- Inertial Measurement Unit
 - see* IMU, 21, 76
- Acoustic Overload Point
 - see* AOP, 21, 81
- AOP, 21, 81
- CAGD, 289
 - Splines, 289
- CNN, 21, 287
- Computerized Numerical Control
 - see* CNN, 21, 287
- Example, 251
 - Description, 251
 - Installation, 251
 - Introduction, 251
- General Purpose Input Output
 - see* GPIO, 21, 204
- GPIO, 21, 204
- IDE, 21, 50, 186
- IMU, 21, 76, 79
- Integrated Development Environment
 - see* IDE, 21, 50
- LED, 21, 93, 94, 98, 101–103, 105, 107, 108, 204
 - Brightness, 107
 - Built-in LED, 89
 - Built-in RGB-LED, 101
 - Colors, 104
 - Power LED, 93
- Light Emitting Diode
 - see* LED, 21, 93
- mcd, 21, 102
- Millicandela
 - see* mcd, 21, 102
- PDM, 21, 81
- Pin
 - Pin 11, 111, 112, 187, 188
 - Pin 13, 89–91

Pin 22, 102
Pin 22,23,24, 102
Pin 23, 102
Pin 24, 102
Pin 25, 94, 96
PLC, *see* Programmable Logic Controller
Pulse Density Modulation
 see PDM, 21, 81
Pulse with Modulation
 see PWM Signal, 21, 94
Push Button
 Built-in Push Button, 111, 187
PWM Signal, 21, 94, 101, 105

Speicherprogrammierbare Steuerung
 see SPS, 21, 287
SPS, 21, 287

VCC, 21, 184
Voltage at the Common Collector
 see VCC, 21, 184