

Elmar Wings

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

Version: 1765
April 18, 2024

Contents

Contents	3
List of Figures	15
List of Listings	21
I. Arduino IDE	25
1. Arduino IDE 1.8.x	27
1.1. Introduction	27
1.2. Arduino IDE Description	27
1.3. Installation	27
2. Arduino IDE 1.8.x - Configuration for Arduino Nano	31
2.1. Configuration for the Arduino Nano 33 BLE Sense	31
2.2. Select the Appropriate Board	31
2.3. Select the Appropriate Port	31
3. Arduino IDE 1.8.x - First Steps	35
3.1. Simple Example “Hello World”	35
3.2. Upload Code in Arduino Board	35
3.3. Output Window (Serial Monitor)	36
4. Arduino IDE 2.3.x	39
4.1. Arduino IDE Description	39
4.1.1. Installation	39
4.1.2. Configuration for the Arduino Nano 33 BLE Sense	41
4.1.3. Setup	42
4.1.4. Constraints	42
4.1.5. Select the Appropriate Port	43
4.1.6. Upload Code in Arduino Board	44
4.1.7. Data Quality	45
4.1.8. Data Quantity	45
4.1.9. Data Types	46
4.1.10. Data Structure	47
4.1.11. Conclusions	47
4.2. example code for IMU on the Arduino Nano 33 BLE Sense.	48
4.3. example code for sensor calibration on the Arduino Nano 33 BLE Sense.	49
5. Arduino Web Editor	51
6. Kammandozeileninterpreter (CLI)	53
6.1. Installation und Verwendung des CLI	53
6.2. Konfiguration des Arduino CLI	54
6.3. Funktionsübersicht	54
6.3.1. Grundfunktionen	56

6.4. Erste Schritte mit dem Arduino Nano 33 BLE Sense Lite mit Hilfe des CLI	57
6.4.1. Erkennung der angeschlossenen Boards	57
6.4.2. Erstellung eines neuen Sketches	57
6.4.3. Kompilieren eines Sketches	58
6.4.4. Hochladen eines Sketches	58
6.4.5. Installation von Bibliotheken	59
6.4.6. Beispiel „Hello World“	59
6.5. Beschreibung der Software auf dem PC	60
7. Arduino Lab for MicroPython	65
II. Arduino Nano 33 BLE Sense - Onboard Sensoren	67
8. Arduino Nano 33 BLE Sense	69
8.1. Arduino Nano 33 BLE Sense	69
8.2. Was ist der Unterschied zwischen Rev1 und Rev2?	71
8.3. Unterschied zwischen dem Arduino Nano 33 BLE Sense Rev1, dem Arduino Nano 33 BLE Sense Rev2 und dem Arduino Nano 33 BLE Sense Lite	71
8.4. On-Board Sensor Description	71
8.4.1. Gesture, Proximity, and Color Detection Sensor ADPS-9960 . .	73
8.4.2. Accelerometer, Gyroscope, and Magnetometre Sensor LSM9DS1	73
8.4.3. Pressure Sensor LPS22HB	74
8.4.4. Relative Humidity and Temperature Sensor HTS221	75
8.4.5. Digital Microphone MP34DT05-A	75
8.4.6. Bluetooth Module nRF52840	76
8.5. Arduino Nano 33 BLE Pin Configuration	78
8.6. Was fehlt	79
9. Tiny Machine Learning Kit	81
9.1. Das Arduino Tiny Machine Learning Shield	81
9.2. Die OV7675 Kamera	82
9.3. USB-Micro-B- auf USB-A-Kabel	83
10. Power LED	85
10.1. General Information	85
10.2. Specific Sensor	85
10.3. Specification	86
10.4. Simple Code	86
10.5. Tests	86
10.5.1. Simple Function Test	86
10.5.2. Test all Functions	87
10.6. Simple Application	88
10.7. Further Readings	89
11. Built-inLED	91
11.1. General Information	91
11.2. Specific Sensor	91
11.3. Specification	91
11.4. Simple Code	92
11.5. Tests	92
11.5.1. Simple Function Test	92
11.5.2. Test all Functions	93

11.6. Simple Application	93
11.7. Further Readings	94
12. Built-in RGB-LED	95
12.1. General Information	95
12.2. Specific Sensor	95
12.3. Specification	96
12.4. Simple Code	97
12.5. Tests	98
12.5.1. Simple Function Test	98
12.5.2. Test all Functions	98
12.6. Simple Application	101
12.7. Further Readings	102
12.8. to do	102
13. Built-in Push Button	103
13.1. General	103
13.2. Specific Sensor	103
13.3. Specification	103
13.4. Bibliothek	103
13.5. Simple Code	104
13.6. Tests	104
13.7. Simple Application	105
13.8. Further Readings	106
14. Sensormodul APDS9960 for Gesture, Proximity, and Color Detection	107
14.1. Sensormodul APDS-9960	107
14.2. General	107
14.3. Specific Sensor	107
14.4. Specification	107
14.5. Bibliothek	107
14.5.1. Description	108
14.5.2. Installation	108
14.5.3. Functions	109
14.5.4. Example - Manual	113
14.5.5. Example	113
14.5.6. Example - Code	113
14.5.7. Example - Files	115
14.6. Calibration	115
14.7. Simple Code	115
14.8. Simple Application	115
14.9. Tests	115
14.9.1. Simple Function Test	115
14.9.2. Test all Functions	115
14.10. Simple Application	115
14.10.1. Color Detection	118
14.10.2. Gestenerkennung	119
14.11. Further Readings	119
15. Mikrophon MP34DT05-A	121
15.1. Puls-Dichtemodulation	121
15.2. Bibliothek <code>pdm.h</code>	122
15.3. General Information	123
15.4. Decibels	123
15.5. Specific Sensor	123

15.6. Specification	123
15.7. Simple Code	123
15.8. Tests	124
15.8.1. Simple Function Test	124
15.8.2. Test all Functions	124
15.9. Simple Application	124
15.9.1. Pin-Konfiguration	124
15.9.2. Initialisierung und Setup	124
15.9.3. Messungssteuerung	124
15.9.4. Mikrofondatenverarbeitung	125
15.9.5. Anzeige der Ergebnisse	125
15.9.6. Umrechnung auf den Wert dB SPL	126
15.9.7. Benutzeroberfläche	126
15.10 Further Readings	126
16. Test of the Hardware	129
16.1. General Tests	129
16.2. Testing the On-Board Sensor	129
16.2.1. LSM9DS1 (Accelerometer, Gyroscope, and Magnetometre Sensor)	130
16.2.2. LPS22HB (Pressure Sensor)	130
16.2.3. HTS221 (Humidity and Temperature Sensor)	133
16.2.4. MP34DT05 (Digital Microphone)	134
16.3. Test with Bluetooth Module Connection	134
16.4. Reset of the Arduino	135
17. Testspezifikation	137
17.1. Sketch "Hello World"	137
17.2. Test der Sensoren eines Arduino Nano 33 BLE Sense	137
18. Serial Communication between PC and Arduino	141
18.1. Installation of a library on the PC	141
18.2. Development of the Python program on the PC	141
18.3. Development of the Arduino Sketch and Preparation of the Arduino	142
18.4. Identical configuration of the serial port and the Arduino sketch	143
18.5. Further Readings	143
18.6. Example: Transfer of an Image using the Serial Port	143
19. Protokoll I²C	145
19.1. Pin-Belegung für I ² C beim Arduino Nano 33 BLE Sense	146
20. Serial Peripheral Interface	147
21. Inertial Measurement Unit	149
21.1. Introduction	149
21.2. 6-Axis IMU LSM6DSOX	150
21.3. IMU LSM6DSOX Features	150
21.4. IMU LSM6DSOX Data	152
21.4.1. Accelerometer:	152
21.4.2. Gyroscope	153
21.5. Library setup in Arduino IDE	153
21.6. Applications	154
21.7. Bibliotheken	154
21.8. Beispiele auf dem Mikrocontroller	154
21.8.1. Testen des Sensors LSM9DS1	154

21.9. Programmierung	155
21.9.1. Programmablaufplan	155
21.9.2. Programmcode und Dokumentation	155
21.9.3. Definition Echtzeit	160
21.10 Kalibrierung	161
21.11 Probleme	161
22. Calibration	163
22.1. Introduction	163
22.2. Standard Operating Procedure	163
22.2.1. Low and high limit method	166
22.2.2. FreeIMU Calibration Application Magnetometer	166
22.2.3. Example	166
23. Errors	169
23.1. Introduction	169
23.2. Affected Parameters	169
23.2.1. Accelerometer:	170
23.2.2. Gyroscope:	170
24. IMU LSM6DSOX - Libraries and Functions	171
24.1. Libraries	171
24.1.1. Wire.h	171
24.1.2. Kalman.h	171
24.1.3. Arduino_LSM6DSOX.h	171
24.1.4. LSM6DSOXSensor.h	171
24.2. Functions	172
24.2.1. setup()	172
24.2.2. loop()	172
24.2.3. IMU.begin()	172
24.2.4. IMU.setAccelerometerRange()	172
24.2.5. IMU.setGyroscopeRange()	172
24.2.6. IMU.setAccelerometerDataRate()	173
24.2.7. IMU.setGyroscopeDataRate()	173
24.2.8. IMU.accelerationSampleRate()	173
24.2.9. IMU.gyroscopeSampleRate()	173
24.2.10. IMU.temperatureAvailable()	173
24.2.11. IMU.readTemperature()	173
24.2.12. IMU.accelerationAvailable()	173
24.2.13. IMU.readAcceleration()	173
24.2.14. IMU.gyroscopeAvailable()	174
24.2.15. IMU.readGyroscope()	174
24.2.16. randomGaussian()	174
24.2.17. kalmanX.setQ(), kalmanY.setQ(), kalmanZ.setQ()	174
24.2.18. kalmanX.setR(), kalmanY.setR(), kalmanZ.setR()	174
24.2.19. kalmanX.update(), kalmanY.update(), kalmanZ.update()	174
24.2.20. kalmanX.getSigma(), kalmanY.getSigma(), kalmanZ.getSigma()	175
24.2.21. delay()	175
24.2.22. lsm6dsosSensor.Set_X_FS()	175
24.2.23. lsm6dsosSensor.Set_G_FS()	175
24.2.24. Initialize_IMU()	176
24.2.25. Factors_Calculation()	176
24.2.26. Factors_Calculation()	176

III. Arduino Nano 33 BLE Sense - Externe Sensoren und Aktoren **177**

25. External Standard LED	179
25.1. General	179
25.2. Specification	180
25.3. Using Standard External LED	181
25.3.1. Connecting a LED to the Arduino Board	181
25.3.2. Pins	181
25.4. Bibliothek	181
25.5. Simple Code	182
25.6. Tests	182
25.6.1. Simple Function Test	182
25.6.2. Test all Functions	183
25.7. Simple Application	183
25.8. Further Readings	184
26. External RGB LED	185
26.1. Standard RGB-LED	185
26.2. Specific Sensor	186
26.2.1. Pins	186
26.3. Specification	186
26.4. Calibration	186
26.5. Simple Code	186
26.6. Simple Application	186
26.7. Tests	186
26.7.1. Simple Function Test	186
26.7.2. Test all Functions	187
26.7.3. Brightness of the RGB-LED	187
26.8. Simple Application	189
26.9. Further Readings	189
27. Sensor BME280 für Temperatur, Luftfeuchtigkeit und den Luftdruck	191
27.1. Beschreibung der Hardware	191
27.2. Schaltplan des Aufbaus	192
27.2.1. Bibliothek cactus_io_BME280 für den Sensor BME280	192
27.2.2. Testen des OLED-Displays	192
28. Servomotor JAMRA 033212	195
28.1. Datenblatt JAMRA 033212	196
28.2. Schaltplan	196
29. OLED-Display	199
29.1. OLED	199
29.2. Anschluss	199
29.3. Programmierung	199
29.3.1. Wire.h	199
29.3.2. OLED-Display	201
29.3.3. Testen des OLED-Displays	201
29.4. Software	201
29.4.1. Verwendete Bibliotheken	201
29.4.2. OLED-Display	202
29.4.3. Initialisierung und Setup	202
29.4.4. Messungssteuerung	203
29.4.5. Mikrofondatenverarbeitung	204
29.4.6. Anzeige der Ergebnisse	205

29.4.7. Umrechnung auf den Wert dB SPL	205
29.4.8. Benutzeroberfläche	205
29.5. Das OLED-Display SSD1306	206
29.6. Schaltplan des Aufbaus	207
29.7. Genutzte Bibliotheken	207
29.7.1. Bibliothek Wire.h	207
29.7.2. Bibliothek SSD1306Ascii.h für das Testprogramm	208
29.7.3. Testen des OLED-Displays	208
29.8. Beispiel eines OLED-Displays	208
29.8.1. OLED	211
30. Kamera-Modul ArduCAM OV2640	213
30.1. Indroduction	213
30.2. Produktbeschreibung	213
30.2.1. Pin Configuration of Arducam OV2640 2MP Mini	214
30.3. ArduCAM Interface with Arduino	215
30.4. ArduCAM Library Introduction	215
30.5. Libraries Structure	217
30.6. How to use	217
30.6.1. Edit <code>memoriesaver.h</code> file	217
30.6.2. Choose correct CS pin for your camera	217
30.6.3. Upload the examples	217
30.6.4. How To Connect Bluetooth Module	218
31. Lens Calibration Tool	219
31.1. Übersicht	219
31.2. Applications	219
31.3. Package Contents	220
32. BlueTooth	221
32.1. Quick Start	221
32.2. Supplies	221
32.3. Step 1: Introduction	221
32.4. How pfodApp is optimised for short BLE style messages	223
32.5. Step 2: Creating the Custom Android Menus and Generating the Code	223
32.6. BLE Mobile App “Nordic Semiconductors nRF Connect”	224
32.7. Arduino BLE Example – Explained Step by Step	226
32.7.1. Arduino BLE Example Code Explained	226
32.7.2. Arduino BLE – Bluetooth Low Energy Introduction	226
32.7.3. Arduino BLE Example 1 – Battery Level Indicator	226
32.7.4. Arduino BLE Tutorial Battery Level Indicator Code	228
32.7.5. Arduino Bluetooth Battery Level Indicator Code Explained	228
32.7.6. Installing the App for Android	229
32.7.7. Example 2 – Arduino BLE Accelerometer Tutorial	229
32.8. Arduino Library <code>BLEAK</code>	235
IV. Projects	237
33. TinyML and Edge Computing	239
33.1. TinyML	239
33.2. Real Time	239
33.3. Edge computing	239

34. TensorFlow Lite	241
34.1. TensorFlow Lite	241
34.2. What is TensorFlow Lite?	241
34.3. Procedure	241
34.4. TensorFlow Lite Converter	242
34.5. TensorFlow Lite Interpreter	243
35. Introduction	245
36. Project Magic Wand	247
36.1. Development	247
36.1.1. Database	247
36.1.2. Data Preparation	247
36.1.3. Data Transformation & Mining	249
36.1.4. Model	250
36.1.5. Evaluation and Verification	254
36.2. Deployment	255
36.2.1. Software	255
36.2.2. Program Flow Code	262
37. Magic Wand Program Workflow	265
37.1. Development	267
37.1.1. Introduction	267
37.1.2. System Requirements	267
37.1.3. Overview of the Program	267
37.2. Code Structure	267
37.2.1. Cases - ePendingMovement, eRecordingGesture	271
37.2.2. arduino_acceleometer_handler	272
37.2.3. Arduino_Output_Handler	273
37.2.4. main_functions	273
37.2.5. constants.h	274
37.2.6. Magic Wand Program Workflow	276
37.2.7. Frequently Asked Questions	277
37.3. Getting to Know Arduino Nano Magic Wand	277
37.3.1. Key Features	277
37.3.2. Functional Parts	278
37.4. Board Operation	278
37.5. LSM9DS1(IMU)	280
37.6. Features	280
37.7. Pin description LSM9DS1	282
37.8. Pin connections LSM9DS1	282
37.8.1. Module specifications	282
37.8.2. Block Diagram	282
37.8.3. Using Magic Wand	284
37.9. Connect with Computer	284
37.10Open Arduino IDE	284
37.11Select Port	285
37.12Run the application	285
37.13Open Serial Monitor	286
37.14Start waving magic wand	287
37.14.1System Requirements	287
37.14.2Precautions to be taken	287
37.14.3FAQs	287

38. Project Implementation Steps	289
38.1. Edge Impulse	289
38.2. Gesture Detection	290
38.2.1. First Approach for Gesture Detection	290
38.3. Gesture Detection Test Using MediaPipe	290
38.3.1. Successful Approach of Gesture Detection	291
38.3.2. Gesture Detection on Arduino Nano 33 BLE Sense	292
38.3.3. Gesture Detection Results On Arduino Nano 33 BLE Sense	292
38.4. Gesture Detection Using LSTM Layer and Tensor Flow	293
38.4.1. Gesture Detection Results Using LSTM and TensorFlow using MediaPipe	293
38.5. Object Detection	294
38.5.1. Yolov3 and Yolov4 Object Detection Model	294
38.5.2. OpenCV Gpu support using Darknet YOLOV4	294
38.5.3. Deep Sort Object Tracking Model	295
39. Open Question and Possible Solution	297
39.1. Checkpoint 1 (Arduino IDE and Python supportive Packages)	297
39.1.1. Possible Solution	297
39.2. Checkpoint 2 (Arducam Mini 2MP OV2640 Replacement)	298
39.2.1. Possible solution	298
39.3. Checkpoint 3 (TensorFlow lite not supported complex object)	298
39.3.1. Possible solution	299
39.4. Object Detection Part Solution	299
39.5. Possible operating system options for Machine Learning Model	299
39.5.1. Nvidia GPU Installation requirement	300
40. Source Codes	301
40.0.1. Hand Landmark Tracking	301
40.0.2. Test Example for Counting figure Using Landmark detection	301
40.0.3. HandTracking Module	303
40.0.4. Detecting 6 Different Gesture Code Using Hand Landmarks	304
40.0.5. Module for Running code on Arduino	306
41. Magic Wand	309
41.1. Attention	309
41.2. Gehäuse	310
42. Development	313
42.1. Why KDD for the magic wand?	313
42.2. Database	313
42.3. Data Collection and Creation	313
42.4. Data Characteristics	316
42.5. Data Transformation and Data Mining	318
42.5.1. Model	319
42.6. Evaluation and Verification	324
42.7. Development Environment	324
42.8. Versions	325
42.9. Description	325
42.10. Installation	326
42.10.1. Arduino IDE on PC	328
42.11. Configuration	329
42.11.1. Configuration for the Arduino Nano 33 BLE Sense	329
42.12. First Steps	329

42.13 Program "Hello World"	330
42.13.1 Description of Basic Sketch for Printing 'Hello'	331
V. Add Ons	333
43. Bootloader	335
43.1. Kopieren eines Arduinos	335
44. Arduino's Science Journal	337
45. Edge Impulse with the Arduino Nano 33 BLE Sense	339
VI. To Do	341
46. To Do	343
46.1. Task: TensorFlow Lite	343
46.1.1. Description	343
46.1.2. Installation	343
46.1.3. Usage	343
46.1.4. Conversiong/Optimization	343
46.1.5. Filetypes	344
46.1.6. Restrictions	344
46.1.7. Tensorflow lite Micro	344
46.2. Arduino Nano 33 BLE Sense	344
46.2.1. Arduino Web Editor	345
46.2.2. Arduino IDE 2.x.x	345
46.2.3. Command Line Interpreter CLI	345
46.3. IMU	345
46.4. Camera module OV7675	346
46.5. Light/Color Sensor	347
46.5.1. Light	347
46.5.2. Color	347
46.5.3. Color Models	347
46.5.4. Description of the light/color sensor	347
46.6. Speech Recognition with the Package <code>DSpotterSDK_Maker_33BLE</code>	347
46.6.1. Description of Speech Recognition	347
46.6.2. Description of the library and its handling	347
46.6.3. Description of the functionality	347
46.6.4. Description of the algorithm used	347
46.6.5. Description of use and evaluation	347
46.6.6. Example programs	347
46.6.7. Description of the microphone	347
46.7. KDD	347
46.8. Creating a Template	348
46.9. Tasks	348
46.9.1. Task "Jetson Orin"	348
46.9.2. Task "DepthAI OAK-D-Pro"	348
46.9.3. DepthAI OAK-D-Pro auto focus	348
46.9.4. Task "Google Coral Micro"	350
46.9.5. Task "Google Dev Board"	350
46.9.6. Task "Raspberry Pi 400"	350
46.9.7. Task "ESP32"	350
46.9.8. Task "Arduino Nano 33 BLE Sense"	351

Contents	13
-----------------	-----------

46.10IndIn Student	351
46.11Radio	351
46.12Yahboom	351
46.13ESP32CAM	352
46.14Power BI	352
46.15TinyML	352
46.16Magic Wand	352
46.17LaTeX	352
46.18To Do	353
46.19French Student I	353
46.20French Student II	354
46.21Pixy CAM	354
46.22Potenzielle Aufgaben	354
46.232.Teil	356
47. First Steps with the Nano 33 BLE Sense	359
48. First Steps with the ArduCAM	361
49. Training a Custom Machine Learning Model for Nano 33 BLE Sense	363
50. Use TensorFlow Lite with Nano 33 BLE Sense	365
50.1. GitHub	365
50.2. Checking your Installation	365
50.3. Compatibility	365
50.4. License	366
50.5. Contributing	366
50.6. Gesture Recognition Model	366
50.7. Goals	367
50.8. Hardware & Software Needed	367
50.9. Microcontrollers and TinyML	367
50.10TensorFlow Lite for Microcontrollers Examples	368
50.11How to Run the Examples Using Arduino Create Web Editor	369
50.12Focus On The Speech Recognition Example	369
50.13How To Run The Examples Using the Arduino IDE	369
50.14Training a TensorFlow Lite Micro Model For Arduino	371
50.15IDE Setup	371
50.16Streaming Sensor Data From the Arduino Board	371
50.17Visualizing Live Sensor Data Log From the Arduino Board	372
50.18Capturing Gesture Training Data	374
50.19Training in TensorFlow	375
50.20Classifying IMU Data	378
50.21Conclusion	380
50.22Nicht zu Rev2 kompatibel	380
50.22.1.Use the version of the library the tutorials were written for	380
50.22.2.Update the tutorial sketches to work with the current version of the library	380
50.22.3.Comment	380
51. Links	381
51.1. Arduino Projects	381
52. Sensor	383
52.1. General	383
52.2. Specific Sensor	383

52.3. Specification	383
52.4. Bibliothek	383
52.4.1. Description	383
52.4.2. Installation	383
52.4.3. Functions	383
52.4.4. Example - Manual	383
52.4.5. Example	383
52.4.6. Example - Code	383
52.4.7. Example - Files	383
52.5. Calibration	383
52.6. Simple Code	384
52.7. Simple Application	384
52.8. Tests	384
52.8.1. Simple Function Test	384
52.8.2. Test all Functions	384
52.9. Simple Application	384
52.10 Further Readings	384
53. Package Example	385
53.1. Introduction	385
53.2. Description	385
53.3. Installation	385
53.4. Example - Manual	385
53.5. Example	385
53.6. Example - Code	385
53.7. Example - Files	385
53.8. Further Reading	385
VII. Anhang	387
54. Materialliste	389
55. Datenblätter	391
55.1. Datenübersicht	391
56. Datenblätter Arduino Vision Shield	397
Literaturverzeichnis	413
Stichwortverzeichnis	419
Index	419

List of Figures

1.1.	Menu bar options	28
1.2.	Menu buttons	28
1.3.	Arduino Creat Agent Installation	29
1.4.	Arduino Setup Installation options	29
1.5.	Arduino Setup: Installation Folder	29
1.6.	Arduino Sketch	30
2.1.	Arduino Mbed OS Nano Boards Installation	31
2.2.	Select the Connected board -here: Arduino Nano 33 BLE Sense	32
2.3.	Arduino Nano 33 BLE Sense: Reset Button	33
2.4.	Arduino Nano 33 BLE Sense: Orange LED Glow	33
2.5.	Select Available Port for Uploading Arduino Sketch	34
3.1.	LED-Example Test	35
3.2.	Upload the Program in Arduino board	36
3.3.	Setting the Port	37
3.4.	Serial Monitor Icon	37
3.5.	Output Window	38
4.1.	Menu Button.	39
4.2.	Arduino IDE Creat Agent Installation.	40
4.3.	Menu Bar Option.	40
4.4.	Arduino Setup Installation options.	40
4.5.	Arduino Setup Installation Folder.	41
4.6.	Arduino IDE Sketch.	41
4.7.	Arduino Mbed OS Nano Boards Installation.	42
4.8.	LED-Example Test.	42
4.9.	Select the Connected board -here Arduino Nano 33 BLE Sense.	43
4.10.	Arduino Nano 33 BLE Sense Reset Button.	43
4.11.	Arduino Nano 33 BLE Sense Orange LED Glow.	43
4.12.	Select Available Port for Uploading Arduino Sketch.	44
4.13.	Upload the Program in Arduino board.	44
4.14.	Setting the Port.	44
4.15.	Serial Monitor Icon.	48
4.16.	Output Window.	48
6.1.	Installation von Arduino-CLI	54
6.2.	Standardeinstellungen der Konfigurationsdatei	55
6.3.	Rückgabewerte der „board list“-Funktion	57
6.4.	Installieren des Kerns für den Arduino Nano 33 BLE Sense Lite	57
6.5.	Kompilieren eines Sketches mit Command Line Interface (CLI)	58
6.6.	Für das Kompilieren des Blink-Sketches wird der Dateipfad zu dem Sketch mit angegeben (rot unterstrichen). Außerdem sollte der FQBN des Arduinos, für den der Sketch kompiliert wird, genannt werden.	59
6.7.	Hochladen des kompilierten Sketches aus dem Temp-Ordner	59
6.8.	Die orange LED des Arduinos beginnt zu blinken	60

8.1.	Arduino Nano 33 BLE Sense, see Arduino Store	70
8.2.	Components in Arduino Nano 33 BLE Sense [Raj19]	72
8.3.	Circuit diagram microphone	76
8.4.	Arduino Nano 33 BLE Pin Configuration	79
9.1.	Das Tiny Machine Learning Kit	81
9.2.	Das Tiny Machine Learning Shield	82
9.3.	Das OV7675 Kameramodul	83
9.4.	Ein USB-A auf Micro-USB Kabel	83
10.1.	Arduino Nano 33 BLE Sense's Power LED	85
11.1.	Arduino Nano 33 BLE Sense's built-in LED	91
12.1.	Arduino Nano 33 BLE Sense's RGB LED	96
14.1.	APDS-9960 Gesture, Proximity, Color Sensor	108
14.2.	APDS-9960 Gesture, Proximity, Color Sensor	109
14.3.	APDS-9960 Gesture, Proximity, Color Sensor	109
14.4.	Gesture, Proximity, Color Sensor Output Window	115
15.1.	Code Einlesen/Zählen	125
15.2.	Code Messungssteuerung	125
15.3.	Code Überwachung	127
15.4.	Code Initialisierung der Datenverarbeitung	127
15.5.	Code Umrechnung Mikrofonsignal	128
15.6.	Code OLED-Aktualisierung	128
16.1.	Power On Arduino Nano 33 BLE Sense	129
16.2.	9-Axis IMU Sensor	130
16.3.	Results of IMU-Tests	132
16.4.	LPS22HB, Pressure and Temperature sensor	132
16.5.	LPS22HB, Output	133
16.6.	HTS221, Humidity and Temperature Sensor	133
16.7.	HTS221, Output Window	133
16.8.	MP34DT05, Digital Microphone	134
16.9.	MP34DT05, Serial Plotter	134
16.10.	Bluetooth Connection	135
16.11.	The Reset Button of the Arduino	135
19.1.	Aufbau des Protokolls I ² C	145
19.2.	I ² C-Protokoll Quelle: [GW22]	146
21.1.	Examples in the library <i>Arduino_LSM9DS1</i>	149
21.2.	<i>Axis Acceleration of Accelerometer Sensor</i>	152
21.3.	<i>Angular Speed of Gyroscope Sensor</i>	153
21.4.	Library setup in Arduino IDE	153
21.5.	Output des Testprogramms	155
21.6.	Der Programmablaufplan	156
21.7.	Anzeige des Arduino OLED Displays sobald der Arduino eingeschaltet ist	161
21.8.	Ausgabe von Messdaten	161
21.9.	Der Reset Button des Arduino	162
25.1.	Resistor with 220 Ohm for an external LED.	180
25.2.	Parts of a LED.	181
25.3.	Schematic Symbol for a LED.	181

26.1. External RGB LED with Resistors [Sima]	185
26.2. Connectors of a RGB LED.	186
27.1. Sensor BME280	191
27.2. Stationärer Aufbau der Wetterstation	192
27.3. Pfad des Testprogramms.	193
27.4. Erste Ausgabe Display	194
28.1. Aufbau eines Servomotors Quelle	196
28.2. PWM am Servomotor Quelle	197
28.3. Auszug aus Gebrauchsanleitung JAMRA 033212, Seite 1 [Jam]	197
28.4. Schaltplan zum Anschluss eines Servomotors JAMRA 033212	198
29.1. Gesamter Schaltplan	200
29.2. Beispiele in der OLED Bibliothek	201
29.3. Testausgabe des OLED Display	202
29.4. Code Einlesen/Zählen	203
29.5. Code Messungssteuerung	203
29.6. Code	206
29.7. Pins des OLED-Displays.	206
29.8. Stationärer Aufbau der Wetterstation	207
29.9. Pfad des Testprogramms.	209
29.10. Erste Ausgabe Display	210
29.11. Das Display zur Ausgabe der Messwerte	210
30.1. Kamera IMX477 der Firma Arducam; [Ard21]	214
30.2. ArduCAM Pin Config	215
30.3. ArduCAM Interface with Arduin Mega 2560	215
31.1. Kalibrierungswerkzeug der Firma Arducam; [Ard21]	219
32.1. BLE Devices – Peripheral and Central Devices	222
32.2. Step 2: Creating the Custom Android Menus and Generating the Code	223
32.3. App nRF is searching the Arduino Nano BLE Sense	224
32.4. App nRF is searching the Arduino Nano BLE Sense	225
32.5. App nRF is getting data the Arduino Nano BLE Sense	225
32.6. BLE Devices – Peripheral and Central Devices	228
32.7. Battery app “nRF Connect”	230
32.8. nRF Connect Screen	233
32.9. Accelerometer Values Read from Arduino Using BLE	234
34.1. Tensorflow Lite workflow [Kha20]	242
34.2. The basic process for creating a model for an edge computer[Goo19a] .	242
35.1. Gestures used namely Slope, W and ring	245
36.1. Wing Gesture [WS20]	248
36.2. Ring Gesture [WS20]	248
36.3. Slope Gesture [WS20]	248
36.4. Convolutional Neural Network (CNN) sequence to classify gestures. Source : [WS20]	250
36.5. Inertial Measurement Unit (IMU) signals [Xu+22]	251
36.6. Inertial Measurement Unit (IMU) Accelerometer Graph [WS20]	252
36.7. A convolution window overlaid on the data. [WS20]	252
36.8. Max Pooling. [WS20]	253

36.9. Convolutional Neural Network (CNN) sequence to classify Wing,Ring and Slope.	254
36.10. Sample Confusion Matrix for the dataset [WS20]	255
36.11 Arduino Nano website downloads page	255
36.12 Arduino Nano software version number	256
36.13 Arduino IDE initial software window	257
36.14 Arduino IDE Boards Manager menu	257
36.15 Arduino IDE Boards Manager list	258
36.16 Selection of correct board for the program	258
36.17 Selection of correct port for uploading the program	258
36.18 Arduino Board Info	259
36.19 Managing libraries in the Arduino IDE	259
36.20 Function to Recognize gestures	260
36.21 Code snippet that shows processing of the waved gestures	260
36.22 Switch case statement, Pending Movement	260
36.23 Code snippet that explains to train or to capture gestures	261
36.24 Code snippet displaying a Wing gesture	261
36.25 The value of constants and constant thresholds in the program	261
36.26 The displayed Wing gesture	262
36.27 The displayed slope gesture	262
36.28 The displayed ring gesture	262
36.29 The unknown gesture	263
 37.1. Code Snippet function Main header files()	268
37.2. Code snippet for isMoving() function	268
37.3. Code snippet that shows processing of the waved gestures	269
37.4. The pending switch case that checks if wand is moving or not	270
37.5. Code snippet checking if the wand moves or not	270
37.6. Code snippet to check if the wand is still	271
37.7. Code snippet to check if the wand is still	271
37.8. Code snippet showing capturing gesture function for training	272
37.9. Code snippet showing the loop() function in the program	273
37.10 The accelerometer handler function	274
37.11 Gesture Predictor function	275
37.12 The code handling the output	275
37.13 Code snippet showing the loop() function in the program	276
37.14 Code displaying threshold values	276
37.15 Magic Wand Program Workflow	276
37.16 Components in Arduino Nano 33 BLE Sense [Raj19]	278
37.17 How to connect with computer	279
37.18 Pinout	280
37.19 Schematic connection diagram	281
37.20 Technical Specification	281
37.21 Pin connections LSM9DS1	282
37.22 Pin description LSM9DS1	283
37.23 Sensor Characteristics	284
37.24 Temperature Sensor Characteristics	284
37.25 Absolute Maximum Temperature	285
37.26 Accelerometer and gyroscope block diagram	285
37.27 Magnetometer block diagram	286
37.28 LSM9DS1 electrical connections	286
 38.1. Edge Impulse Flow Chart	289
38.2. Counting Finger Using MediaPipe and OpenCV	290
38.3. Hand Landmarks using MediaPipe	291

38.4. Hand Landmarks using MediaPipe Function	291
38.5. Gesture Detection on Arduino	292
38.6. Gesture Detection Using LSTM, TensorFlow, and MediaPipe	293
38.7. Object detection Model Comparison	294
39.1. Pyfirmata Communication for Python on Arduino	298
39.2. Logitech Camera Replacement	299
42.1. Connecting the Arduino Nano BLE 33 Sensor through Bluetooth to create our own Dataset	314
42.2. Wing Gesture [WS20]	314
42.3. Ring Gesture [WS20]	315
42.4. Slope Gesture [WS20]	315
42.5. Creating and collecting the data by moving the sensor to get correct gesture	315
42.6. Inertial Measurement Unit (IMU) signals [Xu+22]	320
42.7. Inertial Measurement Unit (IMU) Accelerometer Graph	320
42.8. A convolution window overlaid on the data	321
42.9. Max Pooling	322
42.10Convolutional Neural Network (CNN) sequence to classify Wing,Ring and Slope	323
42.11 Menu button.	326
42.12 Menu bar options.	326
42.13 Arduino Setup Installation options.	327
42.14Arduino Setup Installation Folder.	327
42.15Arduino Sketch.	328
42.16Arduino Create Agent Installation.	329
42.17 Arduino Mbed OS Nano Boards Installation.	329
50.1. Hello-World-Programm für TensorFlow Lite	366
50.2. Arduino Nano 33 BLE Sense Rev2 board is smaller than a stick of gum.	368
50.3. Create lib	369
50.4. LED red “No” – LED green “yes”	370
50.5. upload	371
50.6. Install Nano BLE board	372
50.7. Install the necessary libraries	372
50.8. Serial Plotter	374
50.9. Data recorded by your movements	375
50.10Arduino gesture recognition training colab.	376
50.11Gesture classifier	376
50.12Opening a new tab in your sketch	378
50.13Guessing the gesture with a confidence score	378

List of Listings

8.1. Simple example using of the builtin microphone of the Arduino Nano 33 BLE Sense	77
10.1. Simple sketch to control the power LED	86
10.2. Simple sketch to test the power LED	86
10.3. Simple sketch to check the battery state using the power LED	87
10.4. Simple sketch to check the battery state using the power LED	88
11.1. Simple sketch to control the built-in LED	92
11.2. Simple sketch to test the built-in LED	92
11.3. A simple watch dog: A simple watchdog: the built-in LED is switched on for 1 second every 30 seconds.	93
12.1. Simple sketch to test the RGB LED	97
12.2. Simple sketch to test the RGB LED	98
12.3. value between 255 - 0 to write to the RGB LED	99
12.4. Different brightness levels for the RGB LED colors	100
12.5. A simple watch dog: A simple watchdog: the built-in LED is switched on for 1 second every 30 seconds.	101
13.1. Defining the built-in button's pin as an input.	104
13.2. Read the built-in button's state	104
13.3. Simple sketch to test the built-in LED	104
13.4. Simple sketch to control an external LED. Here, pushing the built-in button is handled by an interrupt. Then the LED switch on for 2 sec.	105
14.1. Simple sketch using the sensor APDS9960 for colors	113
14.2. Simple sketch using the sensor APDS9960 for measuring the proximity	113
14.3. Simple sketch using the sensor APDS9960 for gesture detection	114
14.4. Simple sketch using the sensor APDS9960	115
15.1. Simple sketch to control the built-in LED	124
16.1. Simple example using of the builtin IMU of the Arduino Nano 33 BLE Sense	131
18.1. Python program for the Windows PC for communication with an Arduino via the serial interface.	141
18.2. Sketch of an Arduino for communication with a Windows PC via the serial interface.	142
22.1. Example Microphone	167
22.2. Example IMU (Accelerometer and Gyroscope)	167
22.3. Example ADPS-9960	168
25.1. Defining the pin for an external LED	182
25.2. Defining the pin for an external LED	182
25.3. Swichting On the LED	182

25.4. Simple sketch to test a LED	182
25.5. Simple sketch to test a LED with pulse width modulation	183
25.6. Simple sketch to control an external LED. Here, pushing the built-in button is handled by an interrupt. Then the LED switch on for 2 sec.	183
26.1. Swichting On the LEDs	187
26.2. Swichting Off the LEDs	187
26.3. value between 255 - 0 to write to the RGB LED	187
26.4. Different brightness levels for the RGB LED colors	188
27.1. Testprogramm für ein OLED-Display	194
29.1. Monitoring	203
29.2. Start and stop sampling	204
29.3. Conversion of the microphone signal	205
29.4. OLED-Aktualisierung	205
29.5. Simple program for OLED displays	208
32.1. Arduino BLE Tutorial Battery Level Indicator Code	227
41.1. Einschalten der Farben der RGB-Led	310
41.2. Farbwechsel der RGB-Led	311
50.1. IMU Capture	373
50.2. Arduino TinyML Workshop-Programm	377
50.3. Verwendung des Modells	379

Acronyms

AOP Acoustic Overload Point

CLI Command Line Interface

CNN Convolutional Neural Network

CPU Central Processing Unit

GPIO General Purpose Input Output

GPU Graphics Processing Unit

I²C Inter-Integrated Circuit

IDE Integrated Development Environment

IMU Inertial Measurement Unit

IoT Internet of Things

IR Infrarot

KDD Knowledge Discovery in Databases

LED Light Emitting Diode

mcd Millicandela

PDM Pulse Density Modulation

PWM Pulse with Modulation

RGB Rot-Grün-Blau

SCL Serial Clock

SDA Serial Data

Part I.

Arduino IDE

1. Arduino IDE 1.8.x

1.1. Introduction

In this chapter we will be going through the description of the Arduino IDE and describe the features of it and what are all the options present in the IDE and how can we use it. Further we will be describing about the installation procedure of the Arduino IDE and how to get started with it.

1.2. Arduino IDE Description

It is an open source official Arduino software which used for editing, uploading and compiling codes in to the Arduino module. It is a cross-platform software which is available for Operating Systems like Windows, Linux, macOS. It runs on Java platform and supports a range of Arduino modules. It supports C and C++ languages. The microcontrollers present on the Arduino boards are programmed which accepts the information in the form of code. The program written in the IDE is called a sketch which will generate a Hex file which is then transferred and uploaded in the controller. The IDE environment is made up of two parts: an editor and a compiler. The editor is used to write the required code, while the compiler is used to compile and upload the code to the Arduino Module.[FAD18]

The Menu bar has options such as File in which there are many options including Opening a new file or existing, Examples-in which we can find sketches for different applications like Blink, Fade etc. There is an error console at the bottom of the screen for displaying errors.

The 6 buttons are present on top of the screen are presented in figure 1.2.

- The check mark is used to verify your code. Click this once you have written your code.
- The arrow uploads your code to the Arduino to run.
- The dotted paper will create a new file.
- The upward arrow is used to open an existing Arduino project.
- The downward arrow is used to save the current file.
- The far right button is a serial monitor, which is useful for sending data from the Arduino to the PC for debugging purposes.

1.3. Installation

To install the Arduino IDE, we need to download the latest version from the Arduino webpage <https://www.arduino.cc/en/software>. We can select the version based on the operating system we are using. Here we are installing Arduino 1.8.15 for a Windows 10 operating system. The set up file name is arduino-1.8.15-windows.exe and the size of it is 1,17,470 KB. we can specify the path according to our needs. Here the path is set as `C:/Program Files (x86)/Arduino`. The most recent offline

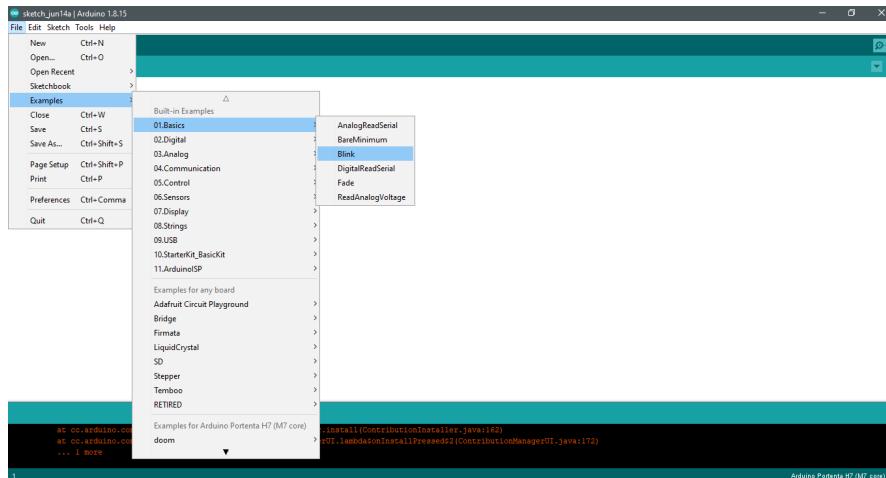


Figure 1.1.: Menu bar options



Figure 1.2.: Menu buttons

arduino IDE 1.8.15 can be seen in Figure, 1.3 it is also supportive for all operating systems.

After the download is done, open the setup file and proceed to install. Select all the components in the dialog box and click Next.

Select the destination folder and click Install

Once the installation is done, open the Arduino IDE and a default sketch appears on the screen as shown.

It can be seen from the figure 1.6 that the basic arduino sketch has two parts. The first part is the function `void setup()` which returns void and we do the intiliazation 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.



Figure 1.3.: Arduino Creat Agent Installation

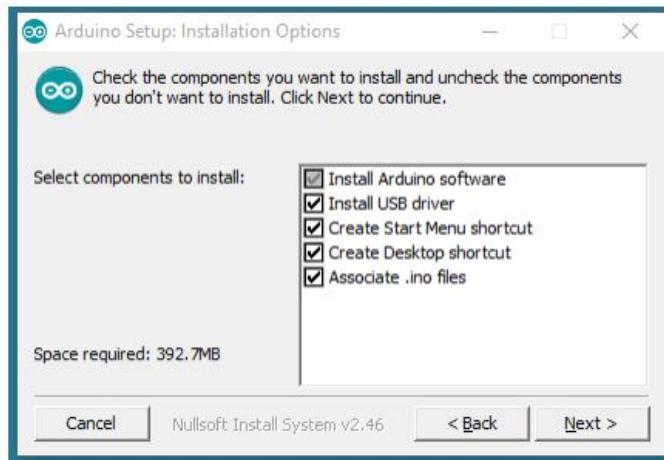


Figure 1.4.: Arduino Setup Installation options

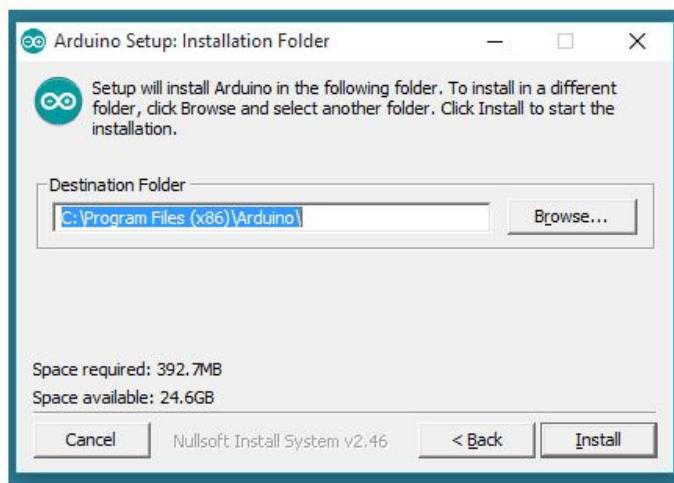


Figure 1.5.: Arduino Setup: Installation Folder

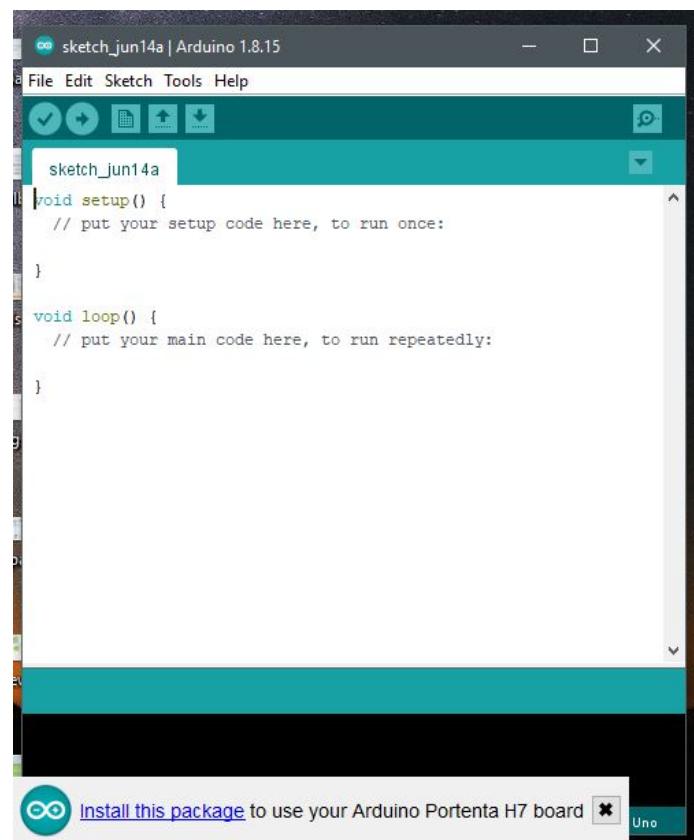


Figure 1.6.: Arduino Sketch

2. Arduino IDE 1.8.x - Configuration for Arduino Nano

2.1. Configuration for the Arduino Nano 33 BLE Sense

To program the Arduino Nano 33 BLE Sense in offline state, we need to install one of the latest arduino IDE on our desktop. After installation, for getting access to the Arduino nano 33 ble sense board, we need to make configuration in our IDE. By opening the IDE, go to tool which can be seen on the uper left corner in IDE, in the tool there is an option for managed board. At this point we need to write our board name in the search which is Arduino Nano 33 BLE Sense as shown in figure, 2.1 select the Arduino Mbed OS Boards and install it. The Mbed OS nano board supports also other nano family boards including Arduino nano 33 ble sense, after installing simply connect the Arduino Nano 33 BLE Sense to the computer via USB cable.

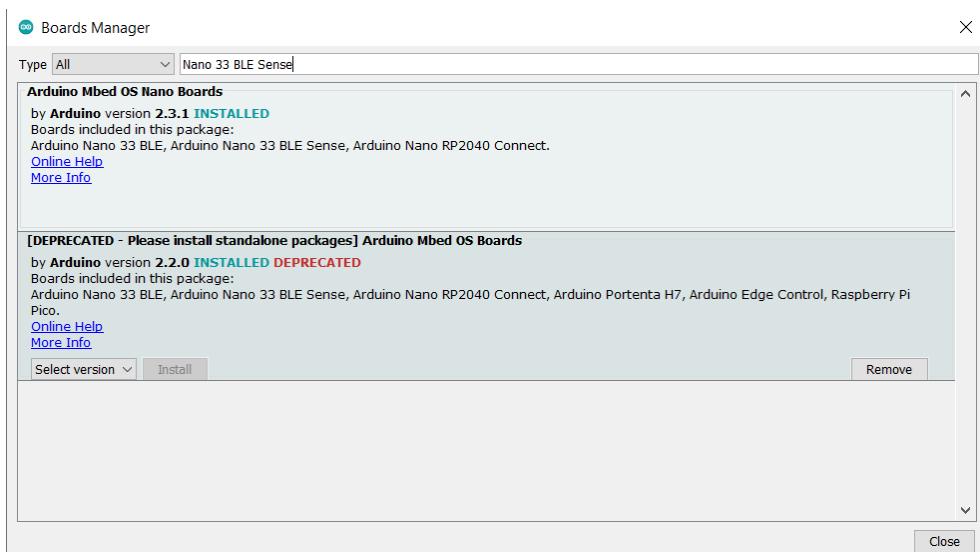


Figure 2.1.: Arduino Mbed OS Nano Boards Installation

2.2. Select the Appropriate Board

At this step we need to go to the tool → Arduino board and select the connected board which is Arduino Nano 33 Ble Sense as shown in the figure 2.2

2.3. Select the Appropriate Port

By selecting the Arduino nano 33 BLE sense board, next we need to check the connected port. For doing this, we need to set our arduino borad in boot setup by clicking the white reset button on arduino as show in figure2.3.

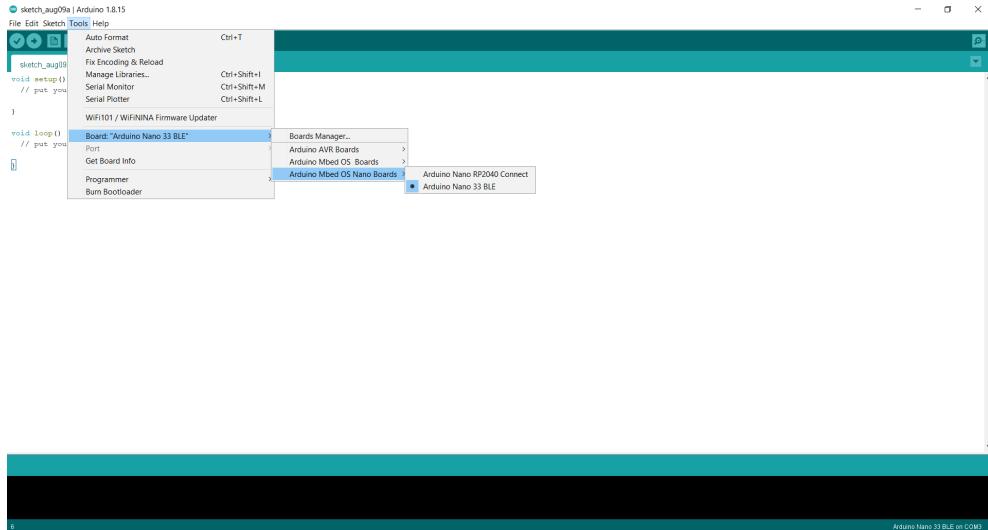


Figure 2.2.: Select the Connected board -here: Arduino Nano 33 BLE Sense

By clicking the white reset button, the arduino board will be in boot setup and make sure to check the orange LED glows as shown in the figure 2.4.
After successfully applying the above mention step, next we need to select the connected port before upload the program. For this, go to tool select arduino port and make sure to check it available port for uploading the program as shown in figure 2.5

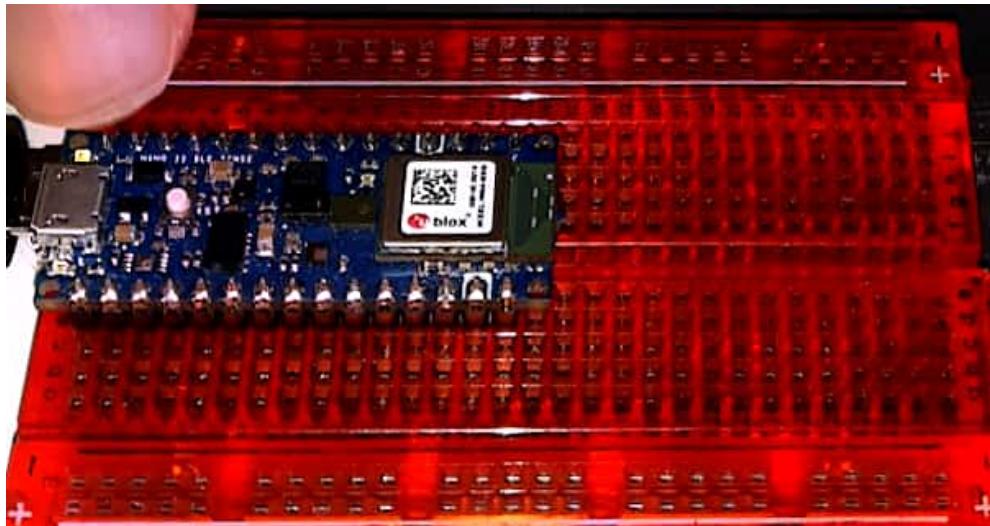


Figure 2.3.: Arduino Nano 33 BLE Sense: Reset Button

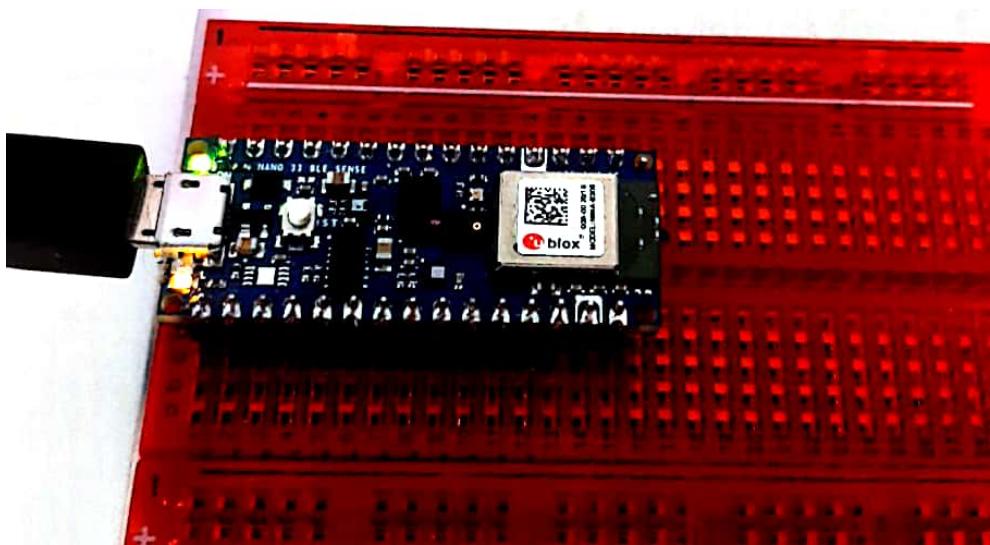


Figure 2.4.: Arduino Nano 33 BLE Sense: Orange LED Glow

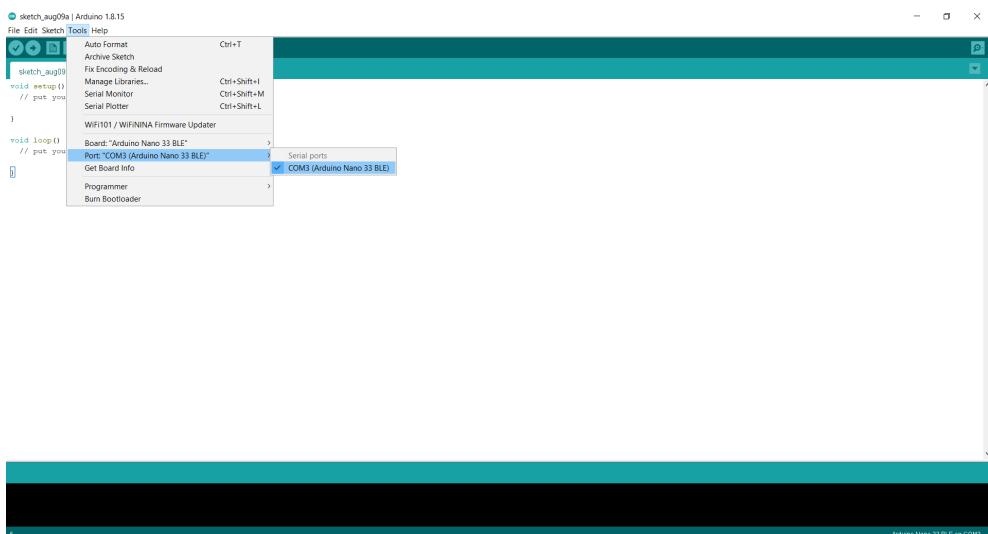


Figure 2.5.: Select Available Port for Uploading Arduino Sketch

3. Arduino IDE 1.8.x - First Steps

3.1. Simple Example “Hello World”

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 LED blink example first as shown in the figure. 3.1.

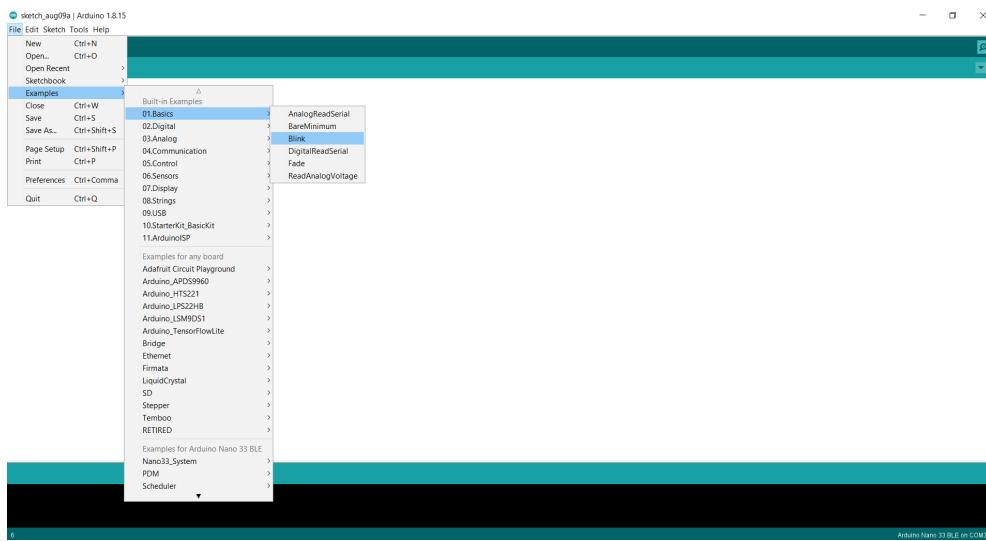


Figure 3.1.: LED-Example Test

This LED-blink example support all the arduino boards, for the checking purposes just need to run this basic example on any arduino embed board and it will blink the LED on our Arduino board after pre-set miliseconds. In the same example folder, there are also number of build in usefull example written in Arduino IDE for embedded boards. These examples are very usefull for getting the basic knowledge about the board and programming.

WS:Where is the code?

3.2. Upload Code in Arduino Board

It's time to upload the Arduino program. There are five icons (verify, upload, new, open, save) below the file section, before uploading the program the best practice is to verify the program first, it show us if there are any error or warning in the program exist or not. By successfully verifying the program we can safely upload the program by click the upload button in the top below the file section as shown in figure. 3.2. After uploading, the code will compile and if there is any issue in our program it will pop up in the bottom black window as well.

After successfully uploading and compiling the code in Arduino board, it also require to change the port again as we did it previously. Go to tool select arduino port and make sure to check the port again as shown in figure 3.3 by getting output in the serial monitor.

```

object_color.classify | Arduino 1.8.15
File Edit Sketch Topics Help
Upload
object_color.classify.ino.ino
Object classifier by color
-----
Uses RGB color sensor input to Neural Network to classify objects
Outputs object class to serial using unicode emojis

Note: The direct use of C++ pointers, namespaces, and dynamic memory is generally
discouraged in Arduino examples, and in the future the TensorFlowLite library
might change to make the sketch simpler.

Hardware: Arduino Nano 33 BLE Sense board.
Created by Don Coleman, Sandeep Mistry
Adapted by Dominic Palak

This example code is in the public domain.

// Arduino_TensorFlowLite - Version: 0.alpha.precompiled
#include "TensorFlowLite.h"

#include "tensorflow/lite/micro/all_ops_resolver.h"
#include "tensorflow/lite/micro/micro_error_reporter.h"
#include "tensorflow/lite/micro/micro_interpreter.h"
#include "tensorflow/lite/schema/schema_generated.h"
#include "tensorflow/lite/version.h"
#include "Arduino_APDS9960.h"
#include "model.h"

// global variables used for TensorFlow Lite (Micro)
tflite::MicroErrorReporter tfliteErrorReporter;

// pull in all the VTM ops, you can remove this line and
// only pull in the VTM ops you need, if would like to reduce

```

Arduino Nano 33 BLE on COM3

Figure 3.2.: Upload the Program in Arduino board

WS:The Arduino Portenta H7 is on the images.

3.3. Output Window (Serial Monitor)

Serial Monitor is the another window on the Arduino IDE, which shows the Input/Output of our program and results appear on it as per the required output. For getting access to Serial Monitor, we need to go extreme right in the Arduino IDE, the small circle pop up when we reach it is the serial monitor as show in the figure. 3.4

WS:More details

The Final results, all the variables, input, sensor values are shown in the serial monitor the (Output Window) as shown in the figure 4.16 by clicking the serial monitor button.

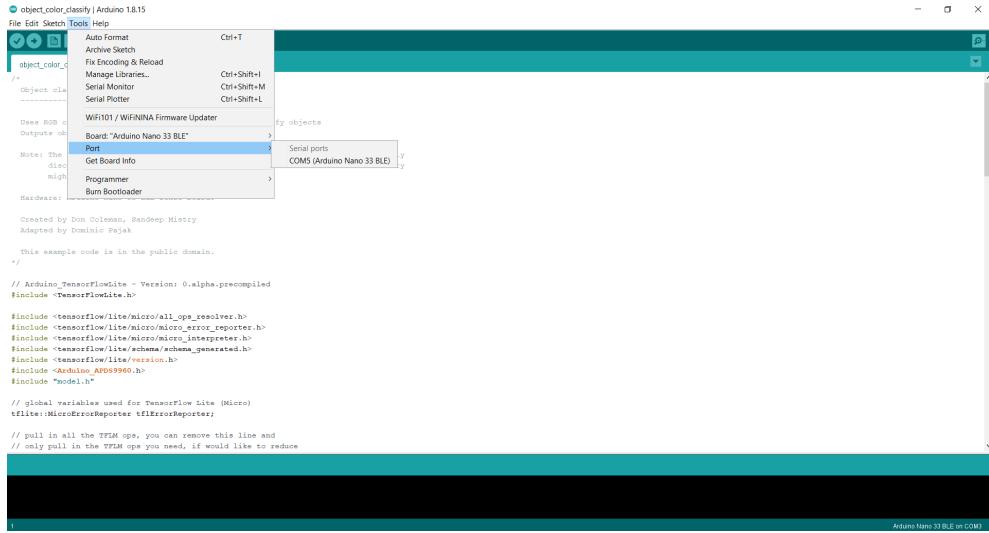


Figure 3.3.: Setting the Port



Figure 3.4.: Serial Monitor Icon

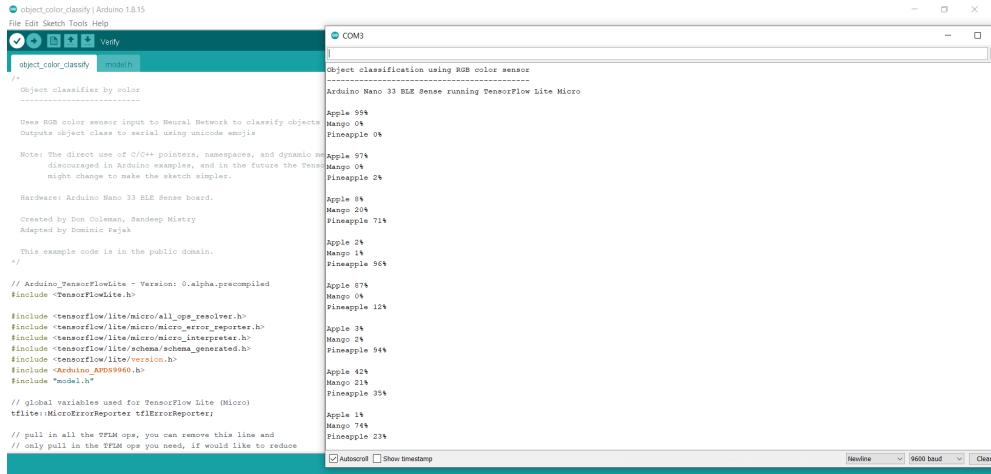


Figure 3.5.: Output Window

4. Arduino IDE 2.3.x

4.1. Arduino IDE Description

It is an open source official Arduino software which used for editing, uploading and compiling codes in to the Arduino module. It is a cross-platform software which is available for Operating Systems like Windows, Linux, macOS. It runs on Java platform and supports a range of Arduino modules. It supports C and C++ languages. The microcontrollers present on the Arduino boards are programmed which accepts the information in the form of code. The program written in the IDE is called a sketch which will generate a Hex file which is then transferred and uploaded in the controller. The IDE environment is made up of two parts: an editor and a compiler. The editor is used to write the required code, while the compiler is used to compile and upload the code to the Arduino Module.[FAD18] The Menu bar has options such as File in which there are many options including Opening a new file or existing, Examples-in which we can find sketches for different applications like Blink, Fade etc. There is an error console at the bottom of the screen for displaying errors.

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



Figure 4.1.: Menu Button.

- The check mark is used to verify your code. Click this once you have written your code.
- The arrow uploads your code to the Arduino to run.
- The dotted paper will create a new file.
- The upward arrow is used to open an existing Arduino project.
- The downward arrow is used to save the current file.
- The far right button is a serial monitor, which is useful for sending data from the Arduino to the PC for debugging purposes.

4.1.1. Installation

Arduino Nano 33 BLE Sense uses the Arduino software integrated development environment (IDE) for programming, which is the most widely used and common (IDE) for all arduino boards that can be run online and offline. This is a open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. There are various version of software which is supported for each operating system (OS) e.g: mac, linux, and windows. Arduino community also provide us to start coding online and save our sketches in the cloud, this online arduino editor is most up-to-date version of the IDE includes all libraries and also supports new Arduino boards. For getting access to these software packages go to the following link <https://www.arduino.cc/en/software> and get more up to date information, because every single day there are some updates occurs which is available on the link mention above. These software can be used with any Arduino board, the most recent offline

WS:citations instead of
links, why 1.8.15?

arduino IDE 1.8.15 can be seen in Figure,42.16. it is also supportive for all operating systems.



Figure 4.2.: Arduino IDE Creat Agent Installation.

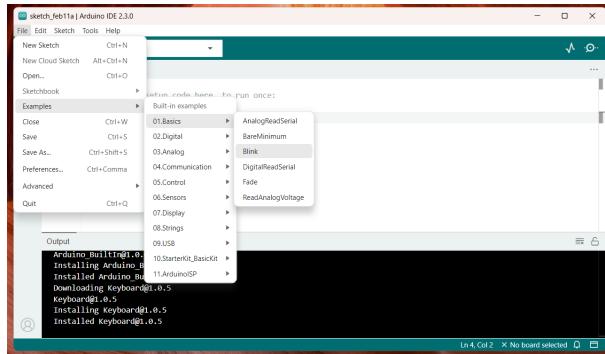


Figure 4.3.: Menu Bar Option.

After the download is done, open the setup file and proceed to install. Select all the components in the dialog box and click Next.

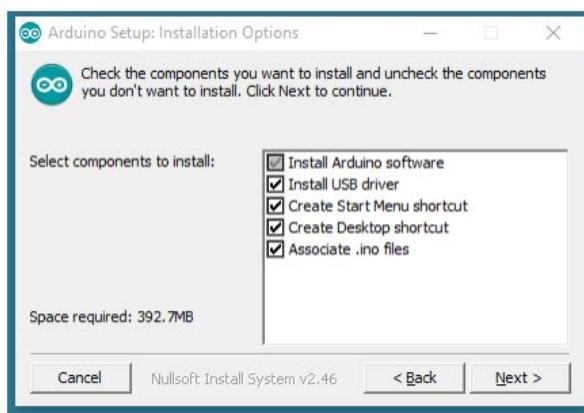


Figure 4.4.: Arduino Setup Installation options.

Select the destination folder and click Install

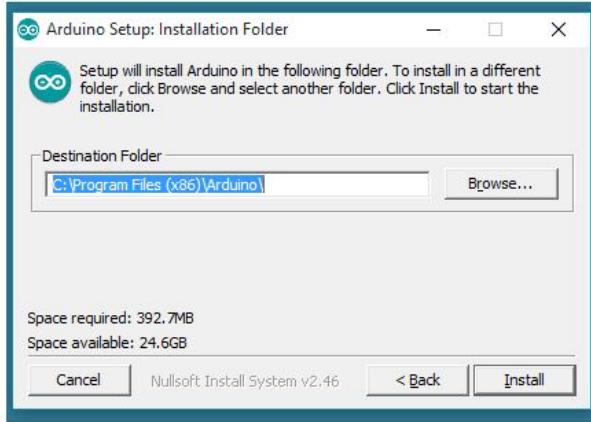


Figure 4.5.: Arduino Setup Installation Folder.

Once the installation is done, open the Arduino IDE and a default sketch appears on the screen as shows.

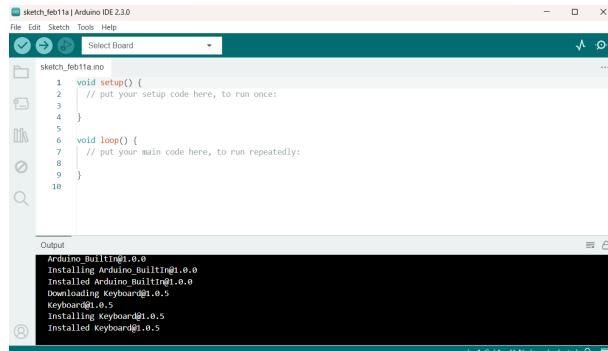


Figure 4.6.: Arduino IDE Sketch.

It can be seen from the above figure that the basic arduino sketch has two parts. The first part is the function `void setup()` which returns void and we do the intiliaztion 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.

4.1.2. Configuration for the Arduino Nano 33 BLE Sense

To program the Arduino Nano 33 BLE Sense in offline state, we need to install one of the latest arduino IDE on our desktop. After installation, for getting access to the Arduino nano 33 ble sense board, we need to make configuration in our IDE. By opening the IDE, go to tool which can be seen on the uper left corner in IDE, in the tool there is an option for managed board. At this point we need to write our board name in the search which is Arduino Nano 33 BLE Sense as shown in figure,42.17.Select the Arduino Mbed OS Boards and install it. The Mbed OS nano board supports also other nano family boards including Arduino nano 33 ble sense, after installing simply connect the Arduino Nano 33 BLE Sense to the computer via USB cable.

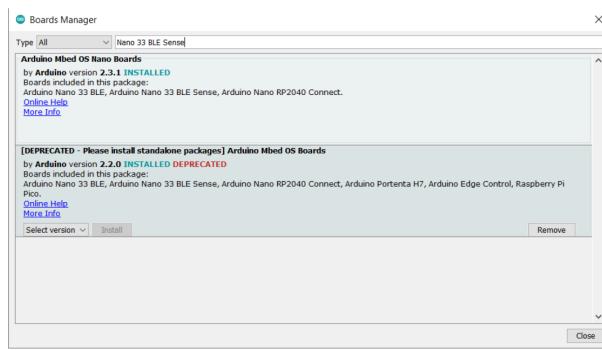


Figure 4.7.: Arduino Mbed OS Nano Boards Installation.

4.1.3. Setup

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 LED blink example first as shown in the figure. 4.8.

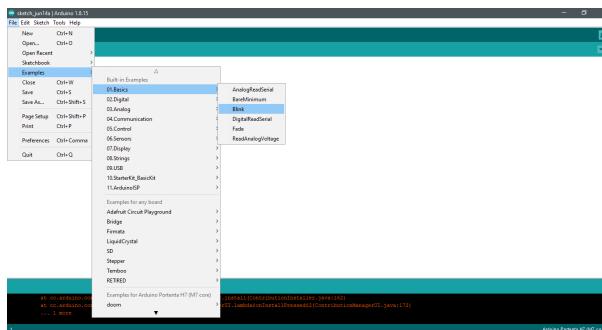


Figure 4.8.: LED-Example Test.

This LED-blink example support all the arduino boards, for the checking purposes just need to run this basic example on any arduino embed board and it will blink the LED on our Arduino board after pre-set miliseconds. In the same example folder, there are also number of build in usefull example written in Arduino IDE for embedded boards. These examples are very usefull for getting the basic knowledge about the board and programming.

4.1.4. Constraints

There are some pre-requisite steps need to follow either we need to run the build in example or run by our own written program. By operating the Arduino board with Laptop with the help of USB connection, need to open the Arduino IDE on desktop, it appears a blank arduino environment page just a Void setup and void loop written on it. At this step we need to go to the tool-Arduino board and select the connected board which is Arduino Nano 33 Ble Sense as shown in the figure 4.9

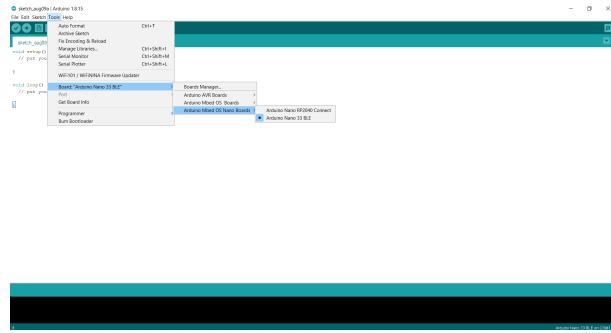


Figure 4.9.: Select the Connected board -here Arduino Nano 33 BLE Sense.

4.1.5. Select the Appropriate Port

By selecting the Arduino nano 33 BLE sense board, next we need to check the connected port. For doing this, we need to set our arduino borad in Boot setup by clicking the white reset button on arduino as show in figure 4.10.

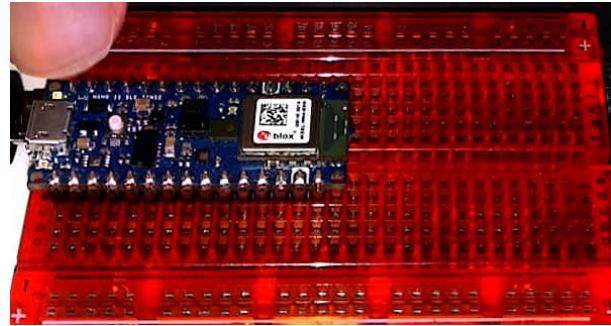


Figure 4.10.: Arduino Nano 33 BLE Sense Reset Button.

By clicking the white reset button, the arduino borad will be in boot setup and make sure to check the orange LED glows as shown in the figure 4.11



Figure 4.11.: Arduino Nano 33 BLE Sense Orange LED Glow.

After successfully applying the above mention step, next we need to select the connectedport before upload the program. For this, go to tool select arduino port and make sure to check it available port for uploading the program as shown in figure 4.12

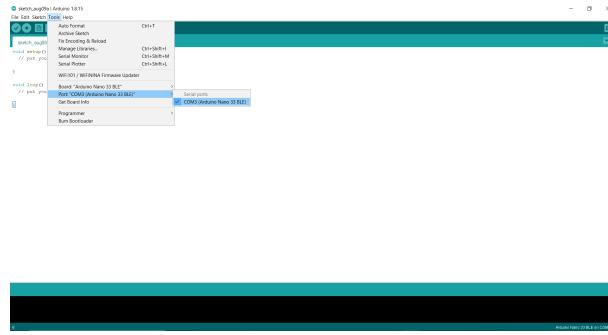


Figure 4.12.: Select Available Port for Uploading Arduino Sketch.

4.1.6. Upload Code in Arduino Board

By making sure to select the appropriate port, it's time to upload the Arduino program. There are five icons (verify, upload, new, open, save) below the file section, before uploading the program the best practice is to verify the program first, it show us if there are any error or warning in the program exist or not. By successfully verifying the program we can safely upload the program by click the upload button in the top below the file section as shown in figure. 4.13



Figure 4.13.: Upload the Program in Arduino board.

After uploading, the code will compile and if there is any issue in our program it will pop up in the bottom black window as well. After successfully uploading and compiling the code in Arduino board, it also require to change the port again as we did it previously. Go to tool select arduino port and make sure to check the port again as shown in figure 4.14 by getting output in the serial monitor.

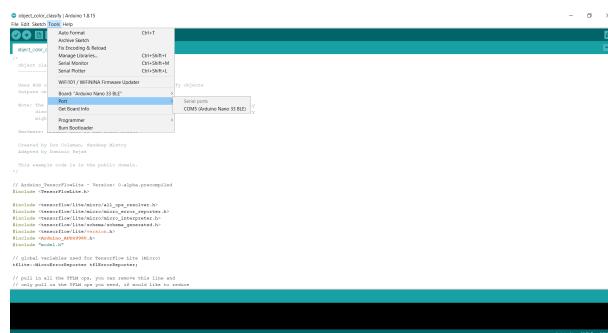


Figure 4.14.: Setting the Port.

4.1.7. Data Quality

The Arduino Integrated Development Environment (IDE) is a key component in the Magic Wand project with Arduino Nano 33 BLE Sense. Here's a detailed explanation of its quality aspects. [FAD18]

1. **Ease of Use:** The Arduino IDE is user-friendly and designed to be easy to use for beginners, while also providing advanced features for experienced users.
2. **Cross-Platform Compatibility:** The Arduino IDE is compatible with Windows, macOS, and Linux, making it accessible to a wide range of users.
3. **Open-Source:** The Arduino IDE is open-source, which means its source code is freely available. This allows for community contributions, leading to continuous improvements and updates.
4. **Support for Multiple Boards:** The Arduino IDE supports not only the Arduino Nano 33 BLE Sense but also a wide range of other Arduino boards.
5. **Built-In Code Editor:** The Arduino IDE comes with a built-in code editor that provides features like syntax highlighting and automatic indentation, making it easier to write and debug code.
6. **Library Manager:** The Arduino IDE includes a Library Manager that makes it easy to install and manage libraries. This is particularly useful for the Magic Wand project, which requires libraries like TensorFlow Lite for Microcontrollers.
7. **Serial Monitor:** The Arduino IDE includes a Serial Monitor that allows you to monitor data sent from the Arduino Nano 33 BLE Sense in real-time. This is crucial for the Magic Wand project, as it allows you to monitor the gesture recognition process.
8. **Community Support:** The Arduino IDE has a large and active community. This means you can find a wealth of tutorials, guides, and troubleshooting advice online.

4.1.8. Data Quantity

Magic Wand project with Arduino Nano 33 BLE Sense, “Data Quantity” in the software description of the Arduino IDE refers to the amount of data that the software can handle or process. Here's a detailed explanation:

1. **Code Size:** The Arduino IDE needs to handle the code for the Magic Wand project. This includes the code for collecting and processing sensor data, running the machine learning model, and any additional functionality.
2. **Library Size:** The Arduino IDE also needs to handle various libraries that are used in the project. This includes the TensorFlow Lite for Microcontrollers library, which is used to run the machine learning model on the Arduino Nano 33 BLE Sense.
3. **Sensor Data:** The Arduino IDE needs to handle the sensor data that is collected by the Arduino Nano 33 BLE Sense. In one experiment, a window size of 2 seconds was used, which means 200 rows of accelerometer data or 600 values of x, y, and z acceleration axis were fed into the model.[FAD18]
4. **Model Data:** The Arduino IDE needs to handle the data of the machine learning model. This includes the model parameters and the model output.

5. **Data Transmission:** The Arduino IDE also handles data transmission between the Arduino Nano 33 BLE Sense and the computer. This includes uploading the program to the board and transmitting sensor data and model output to the computer for monitoring.

4.1.9. Data Types

- **Magic Wand Project with Arduino Nano 33 BLE Sense:** This project involves using TensorFlow Lite for Microcontrollers to run a deep learning model on the Arduino Nano 33 BLE. The microcontroller is turned into a digital "magic wand" that can recognize various gestures.[FAD18]

- **Software Components:**

- **TensorFlow Lite For Microcontrollers:** This is an optimized version of TensorFlow, targeted to run TensorFlow models on tiny, low-powered hardware such as microcontrollers. It doesn't require operating system support, any standard C or C++ libraries, or dynamic memory allocation.
- **Arduino IDE:** This is the software used to write and upload computer code to the physical board.

- **Data Types:** You'll likely be working with arrays to store the sensor data, and you'll use TensorFlow Lite's data types for the model's input and output. The exact data types will depend on your specific implementation and the requirements of the TensorFlow Lite model.

1. **Numeric Data Types:**

- Identify the numeric data types used, such as integers, floating-point numbers, or doubles.
- Describe the range and precision of each numeric data type.

2. **Text Data Types:**

- Specify if the software handles text data, such as strings or characters.
- Discuss any character encoding or formatting requirements.

3. **Sensor Data Types:**

- If interacting with sensors, specify the types of sensor data used.
- Describe the format and units of measurement for each sensor data type.

4. **Custom Data Types:**

- Discuss any custom data types defined in the software, such as structs or classes.
- Explain the purpose and structure of each custom data type.

5. **Data Conversion and Casting:**

- If data conversion or casting operations are performed, describe how different data types are converted or casted.

6. **Data Representation:**

- Explain how data is represented, including binary, hexadecimal, or other representations.

4.1.10. Data Structure

- **Magic Wand Project with Arduino Nano 33 BLE Sense:** project involves using TensorFlow Lite for Microcontrollers to run a deep learning model on the Arduino Nano 33 BLE. The microcontroller is turned into a digital "magic wand" that can recognize various gestures.[Har23]

- **Data Structures:**

- **Sensor Data:** The Arduino Nano 33 BLE Sense collects sensor data, which is multidimensional and complex. This data is typically stored in arrays or similar data structures for processing.
- **Model Input and Output:** The sensor data is passed to the TensorFlow Lite model as input, and the model outputs a simple classification. The exact data structures for these will depend on the requirements of your TensorFlow Lite model.
- **Gesture Recognition:** The project involves recognizing gestures, which are classified into different categories. You might use data structures like enumerations or similar to represent these different categories.

1. **Arrays:** Used to store a collection of data items of the same type. Example: storing sensor readings over time, storing LED brightness levels.
2. **Structures (`struct`):** Allows grouping multiple variables of different types under a single name. Example: defining a struct to hold sensor data (e.g., temperature, humidity).
3. **Classes:** Used for creating user-defined data types with properties and methods. Example: defining a class for managing gestures detected by the wand.
4. **Linked Lists:** Data structures consisting of a sequence of elements where each element points to the next element. Example: implementing a linked list to manage a queue of actions to perform.
5. **Stacks:** Last in, first out (LIFO) data structures where elements are inserted and removed from the same end. Example: using a stack to manage function calls or nested operations.
6. **Queues:** First in, first out (FIFO) data structures where elements are inserted at the end and removed from the front. Example: implementing a queue for handling incoming commands or tasks.
7. **Trees:** Hierarchical data structures consisting of nodes connected by edges. Example: representing hierarchical data such as menu structures or organizational charts.

4.1.11. Conclusions

Output Window (Serial Monitor)

Serial Monitor is the another window on the Arduino IDE, which shows the Input/Output of our program and results appear on it as per the required output. For getting access to Serial monitor, we need to go extreme right in the Arduino IDE, the small circle pop up when we reach it is the serial monitor as show in the figure.4.15



Figure 4.15.: Serial Monitor Icon.

The Final results, all the variables, input, sensor values are shown in the serial monitor the (Output Window) as shown in the figure4.16 by clicking the serial monitor button.

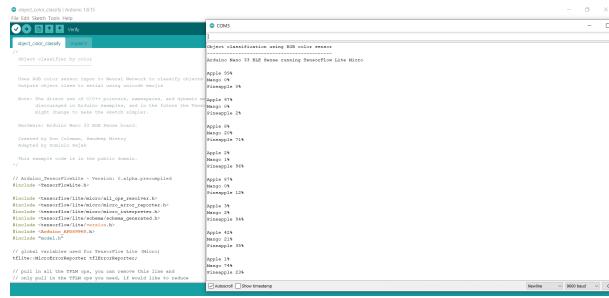


Figure 4.16.: Output Window.

4.2. example code for IMU on the Arduino Nano 33 BLE Sense.

```
#include <Arduino_LSM9DS1.h>

void setup() {
    Serial.begin(9600);
    while (!Serial);

    if (!IMU.begin()) {
        Serial.println("Failed to initialize IMU!");
        while (1);
    }

    Serial.print("Accelerometer sample rate = ");
    Serial.print(IMU.accelerationSampleRate());
    Serial.println(" Hz");
}

void loop() {
    float x, y, z;

    if (IMU.accelerationAvailable()) {
        IMU.readAcceleration(x, y, z);
        Serial.print("Acceleration X: ");
        Serial.print(x);
        Serial.print(" Acceleration Y: ");
        Serial.print(y);
        Serial.print(" Acceleration Z: ");
        Serial.print(z);
        Serial.println();
    }
}
```

```

        Serial.print(x);
        Serial.print(" Y: ");
        Serial.print(y);
        Serial.print(" Z: ");
        Serial.println(z);
    }
}

```

This code snippet utilizes the Adafruit CircuitPython library for interacting with the accelerometer sensor on the Arduino Nano 33 BLE Sense. It continuously scans for BLE devices advertising the Adafruit accelerometer service, connects to the first one found, and then reads and prints accelerometer data in the main loop. Make sure to install the required CircuitPython libraries and dependencies before running this code on your Arduino Nano 33 BLE Sense.[Har23]

4.3. example code for sensor calibration on the Arduino Nano 33 BLE Sense.

```

#include <Arduino_LSM9DS1.h> // Include the LSM9DS1 library for sensor calibration

void setup() {
    Serial.begin(9600); // Initialize serial communication
    while (!Serial); // Wait for Serial to be ready

    // Initialize the LSM9DS1 sensor
    if (!IMU.begin()) {
        Serial.println("Failed to initialize IMU!");
        while (1); // End program if initialization fails
    }

    // Perform sensor calibration
    Serial.println("Starting sensor calibration ...");
    if (!IMU.calibrate()) {
        Serial.println("Calibration failed!");
        while (1); // End program if calibration fails
    }
    Serial.println("Calibration successful!");
}

void loop() {
    // Your main program loop goes here
}

```

Listing 4.1: Sensor Calibration Example Code

You can add additional code to the loop() function to perform other tasks once sensor calibration is complete.[RS17]

5. Arduino Web Editor

6. Kommandozeileninterpreter (CLI)

Quellen für die Bilder fehlen!

Seit 2018 bietet die Firma Arduino auch einen Kommandozeileninterpreter an. [Ardb] Die Web-Version der Entwicklungsumgebung verwendet diese Schnittstelle. Das heißt, alle Funktionalitäten, die die Entwicklungsumgebung zur Verfügung stellt, stehen hier auch zur Verfügung. So kann unter Windows, MacOS und Linux [Ard22c]. Die Entwicklung mit Hilfe des Kommandozeileninterpreters ermöglicht es, wenn konsequent mit Hilfe von Batch-Dateien gearbeitet wird, qualitätssichernde Maßnahmen gezielt durchzuführen. So kann die Konfiguration der Entwicklungsumgebung gesichert und nachvollziehbar wiederholt werden.

WS:Erklärung von
Kommandos in den
Anhang

Der Quellcode kann frei genutzt werden, allerdings muss man bei einer gewerblichen Nutzung eine Lizenz bei der Firma anfragen [Ard22a]. Eine ausführliche Dokumentation wird zur Verfügung gestellt. [Ard20; Ard22b]

Bei der Entwicklung der Command Line Interface (CLI) stehen drei Aspekte im Fokus.

1. Einerseits ermöglicht die Schnittstelle die Einbettung der Entwicklung in die gewohnte Entwicklungsumgebung. So kann die Schnittstelle auch mit Edge Impulse genutzt werden.
2. Zweitens ermöglicht sie die Integration der Entwicklung in die Prozesse Continuous Development and Continuous Integration. Sie ermöglicht damit die Automatisierung der typischen Aktivitäten im Bereich der Softwareentwicklung.
3. Des Weiteren ermöglicht es nun die vereinfachte Kommunikation mit Edge Computern, zum Beispiel mit einem Raspberry PI.

6.1. Installation und Verwendung des CLI

Da CLI ständig weiterentwickelt wird, muss zunächst die aktuelle Version aus dem GitHub-Projekt <https://arduino.github.io/arduino-cli/0.32/installation/#latest-release>, siehe [Ard22a], geladen werden. Die hier verwendete Version ist 0.33. [Ardb]

Nach dem erfolgreichen Download der Datei [arduino-cli_0.33.0_Windows_64bit.zip](#) kann der Ordnerinhalt an einen selbst definierten Speicherpfad extrahiert werden. Die gepackte Datei enthält die Lizenzbestimmungen und die ausführbare Datei [arduino-cli.exe](#). Um die Funktion des Kommandozeileninterpreters in verschiedenen Pfaden nutzen zu können, sollte der Pfad zum Speicherort der Datei [arduino-cli.exe](#) der Systemvariablen [PATH](#) hinzugefügt werden. Anschließend lässt sich das Interface durch die Eingabe [arduino-cli](#) in die Kommandozeile nutzen.

Nachdem die Installation abgeschlossen ist, kann [arduino-cli board list](#) in der Eingabeaufforderung eingegeben werden. Das angeschlossene Gerät wird als Arduino Portenta H7 mit Portnummer und Typ angezeigt, wie in Abbildung 6.1 dargestellt. Es ist zu beachten, dass die Versionsnummer, hier 0.33.1, gegebenenfalls anzupassen ist. Wenn ein Arduino Nicla Vision oder ein Arduino Nano 33 BLE sense angeschlossen ist, werden entsprechende Meldungen ausgegeben.

Das Programm [arduino-cli-0.33.1.windows.exe](#) wird nun in [arduino-cli.exe](#) umbenannt. Daher kann im Folgenden im der Programmname [arduino-cli.exe](#) verwendet werden.

```
C:\Users\Shoeb>arduino-cli board list
Port Type      Board Name          FQBN                Core
COM6 Serial Port (USB) Arduino Portenta H7 (M7 core)  arduino:mbed:envie_m7    arduino:mbed
                                         Arduino Portenta H7 (M7 core)  arduino:mbed_portenta:envie_m7  arduino:mbed_portenta
                                         Arduino Portenta H7 (ThreadDebug) arduino:mbed:envie_m7_thread_debug  arduino:mbed

C:\Users\Shoeb>
```

Figure 6.1.: Installation von Arduino-CLI

6.2. Konfiguration des Arduino CLI

<https://learn.sparkfun.com/tutorials/efficient-arduino-programming-with-arduino-introduction-to-the-arduino-cli>

Damit CLI dir Arduino-Installation finden kann, hilft es, eine Konfigurationsdatei für CLI zu erstellen. Diese Konfigurationsdatei ist in einem Format YAML definiert. Zur Erstellung einer Basis-Konfigurationsdatei kann CLI verwendet werden, in dem in einer Kommandozeile

```
arduino-cli.exe config init
```

eingegeben wird. Dieser Befehl erstellt eine neue Datei `.cli-config.yml`. Dort sind alle notwendigen Parameter deklariert. Die Standardeinstellungen der Konfigurationsdatei sind in Abbildung 6.2 dargestellt. Folgende Parameter müssen in der Regel geändert werden:

- `sketchbook_path`: Verzeichnis des Arduino-Sketche. Hier werden alle Bibliotheken und Hardware-Definitionen installiert.
- `arduino_data`: Installationsort des Arduino-Boards und des Bibliotheksmangers. In den meisten Fällen sollte dies nicht geändert werden müssen.

Die anderen Optionen können in der Regel auf ihren Standardwerten bleiben.

S: Welche Möglichkeiten gibt es in der Konfigurationsdatei?

6.3. Funktionsübersicht

Die Datei `README.md` im GitHub-Repository “Arduino CLI” enthält eine hervorragende Übersicht über die Funktionen und Möglichkeiten.[Ard22a]
Folgende Funktionen stehen zur Verfügung [Ardb]:

- `arduino-cli`
- `board`
 - `board attach`
 - `board details`
 - `board list`
 - `board listall`
 - `board search`

```
board_manager:
  additional_urls: []
build_cache:
  compilations_before_purge: 10
  ttl: 720h0m0s
daemon:
  port: "50051"
directories:
  data: C:\Users\denni\AppData\Local\Arduino15
  downloads: C:\Users\denni\AppData\Local\Arduino15\staging
  user: C:\Users\denni\Documents\Arduino
library:
  enable_unsafe_install: false
logging:
  file: ""
  format: text
  level: info
metrics:
  addr: :9090
  enabled: true
output:
  no_color: false
sketch:
  always_export_binaries: false
updater:
  enable_notification: true
```

Figure 6.2.: Standardeinstellungen der Konfigurationsdatei

- `burn-bootloader`
- `cache`
- `cache clean`
- `compile`
- `completion`
- `config`
 - `config dump`
 - `config init`
 - `config add`
 - `config delete`
 - `config remove`
 - `config set`
- `core`
 - `core download`
 - `core install`
 - `core list`
 - `core search`
 - `core uninstall`
 - `core update-index`
 - `core upgrade`
- `daemon`
- `debug`

- `lib`
 - `lib deps`
 - `lib download`
 - `lib examples`
 - `lib install`
 - `lib list`
 - `lib search`
 - `lib uninstall`
 - `lib update-index`
 - `lib upgrade`
- `monitor`
- `outdated`
- `sketch`
 - `sketch archive`
 - `sketch new`
- `update`
- `upgrade`
- `upload`
- `version`

6.3.1. Grundfunktionen

Als Grundfunktionen werden die Funktionen definiert, die notwendig sind, um einen fertigen Sketch auf einen Arduino zu laden. Die Kommunikation mit einem angeschlossenen Arduino wird später mit Hilfe des seriellen Monitors umgesetzt.

Die erste Grundfunktion ist der Befehl `board list`. Mit diesem lassen sich alle am PC angeschlossenen Arduino Boards mit zusätzlichen Informationen – wie beispielsweise dem verwendeten Port oder dem Kern des Boards – anzeigen. Diese Informationen sind für die Verwendung des korrekten Boards notwendig.

Die Befehle `core list` und `core install` sind ebenfalls notwendig. Mit dem Befehl `core list` können die bereits installierten Kerne aufgelistet werden. Falls der unter `board list` angegebene Kern nicht installiert sein sollte, lässt sich dieser mit Befehl `core install` hinzufügen.

Die dritte Grundfunktion ist der Befehl `lib` mit den Erweiterungen `lib list` und `lib install`. Mit dem Befehl `lib list` lässt sich prüfen, welche Bibliotheken auf dem System bereits installiert sind. Würde für einen Sketch eine zusätzliche Bibliothek benötigt, könnte diese mit dem Befehl `lib install` installiert werden.

Mit diesen Grundfunktionen können die Vorbereitungen für das Kompilieren und Hochladen eines Sketches getroffen werden. Mit der Ausführung des Befehls `compile` wird ein Sketch in für ein ausgewähltes Board lesbaren Code übersetzt. Das anschließende Hochladen eines kompilierten Sketches auf ein Board erfolgt mit dem Befehl `upload`. Dabei wird der aus dem Befehl `board list` bekannte Port des Boards als Ziel für den Upload genannt. Die Grundfunktionen sind in der Tabelle 6.1 zusammengefasst.

Table 6.1.: Liste der Grundfunktionen

Nr.	Grundfunktion	Beschreibung
1	<code>board list</code>	Auflisten der angeschlossenen Arduinos mit Zusatzinfo
2	<code>core list/install</code>	Auflisten und Installieren von Kernen
3	<code>lib list/install</code>	Auflisten und Installieren von Bibliotheken
4	<code>compile</code>	Kompilieren eines Sketches für ein spezielles Board
5	<code>upload</code>	Hochladen eines Sketches auf einen Arduino

6.4. Erste Schritte mit dem Arduino Nano 33 BLE Sense Lite mit Hilfe des CLI

6.4.1. Erkennung der angeschlossenen Boards

Mit Hilfe von CLI kann abgefragt werden, welche Boards installiert sind:

```
arduino-cli board listall
```

Falls das angeschlossene Board vermisst wird, muss es installiert werden:

```
arduino-cli core install arduino:avr
```

Mit Hilfe von CLI kann abgefragt werden, welche Boards angeschlossen sind:

```
arduino-cli board list
```

So kann nach dem Anschließen des Arduinos an einen PC mit dem Befehl „arduino-cli board list“ geprüft werden, ob das Board von dem Computer erkannt wird. Die Funktion „board list“ besitzt mehrere Rückgabewerte. So wird der Port, über den das Board mit dem PC verbunden ist, das Protokoll, der Typ, der Platinenname und der FQBN sowie Kern, wie in Abb. 6.3 dargestellt, zurückgegeben.

```
C:\Users\denni>arduino-cli board list
Port Protokoll Typ Platinenname FQBN Kern
COM4 serial Serial Port (USB) Arduino Nano 33 BLE arduino:mbed_nano:nano33ble arduino:mbed_nano
```

Figure 6.3.: Rückgabewerte der „board list“-Funktion

```
C:\Users\denni>arduino-cli core install arduino:mbed_nano|
```

Figure 6.4.: Installieren des Kerns für den Arduino Nano 33 BLE Sense Lite

Zu Beginn sollte der richtige Kern für den Arduino Nano 33 BLE Sense Lite, wie in Abb. 6.4 gezeigt, installiert werden. Dies erfolgt mit dem „core install“-Befehl und dem richtigen Kernnamen, der aus Abbildung 6.3 entnommen werden kann.

6.4.2. Erstellung eines neuen Sketches

Der erste Schritt ist die Erstellung eines neuen Sketches:

```
arduino-cli sketch new cli_test
```

Dieser Befehl erstellt ein Verzeichnis mit dem Namen `cli_test`, die eine Datei mit dem gleichen Namen enthält.

```

Select C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.17134.345]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\[REDACTED] >arduino-cli-0.2.2-alpha.preview-windows.exe compile --fqbn arduino:avr:uno C:\Users\[REDACTED]\Downloads\Arduino\cli_test
Sketch uses 444 bytes (1%) of program storage space. Maximum is 32256 bytes.
Global variables use 9 bytes (0%) of dynamic memory, leaving 2039 bytes for local variables. Maximum is 2048 bytes.

C:\Users\[REDACTED]\Downloads>

```

Figure 6.5.: Kompilieren eines Sketches mit CLI

6.4.3. Kompilieren eines Sketches

Die Funktion „compile“ von CLI kann verwendet werden, um einen Sketch für jedes unterstützte Board zu kompilieren. Die entscheidende Option, die diese Funktion benötigt, ist der Board-Typ, der mit der Option `-fqbn` angegeben werden kann. Die Abkürzung `fqbn` bedeutet „fully-qualified board name“, was „vollqualifizierter Board-Name“ bedeutet. Beispielsweise stehen folgende Boards zur Verfügung:

- Arduino Uno: `arduino:avr:uno`
- Arduino Mega: `arduino:avr:mega`
- SparkFun RedBoard oder SparkFun BlackBoard: `SparkFun:avr:RedBoard`; dies erfordert die zusätzliche Installation der Definition „SparkFun avr board“.
- SparkFun SAMD21 Mini: `SparkFun:samd:samd21_mini`; dies erfordert die zusätzliche Installation der Definition „SparkFun samd board“.

Die möglichen Boards sind wie folgt aufgebaut: `manufacturer:architecture:board`.

Mit dem folgenden Kommando kann der Beispiel-Sketch, der sich im Ordner `C:/Users/user.name/Documents` befindet, für einen Arduino UNO kompiliert werden.:

`arduino-cli compile -fqbn arduino:avr:uno C:/Users/user.name/Documents/Arduino/cli_test`

In der Abbildung 6.5 sieht man die Meldungen von CLI.

Es können folgende Flags hinzugefügt werden:

- Ausführlich `-v`: Nützlich, wenn alle Optionen und Dateien angezeigt werden sollen, die in Ihrem Sketch kompiliert werden.
- Build Pfad `-build-path [string]`: Nützlich, wenn die kompilierten Objekt- und Hex-Dateien zu speichern sind. Auf meinem Windows-System muss der Wert dieses Parameters ein vollständiger Pfad sein.

6.4.4. Hochladen eines Sketches

Ein kompilierter Sketch kann hochgeladen werden. Wie der Kompilierbefehl erfordert auch der Upload-Befehl eine `-fqbn`. Außerdem wird eine serielle Schnittstelle zum Hochladen benötigt, die mit der Option `-p` festgelegt wird. Der folgende Befehl lädt den Beispiel-Sketch an einen Windows-COM-Anschluss auf COM18 hoch:

`arduino-cli upload -p COM18 -fqbn -v arduino:avr:uno C:/Users/user.name/Documents/Arduino/cli_test`

Falls es ordnungsgemäß durchgeführt wurde, sollte die RX/TX-LEDs des Arduinos zu blinken beginnen und kurz darauf einen leeren Sketch auszuführen.

WS:Portenta H7 und
Nicla Vision fehlen

6.4.5. Installation von Bibliotheken

Falls eine Bibliothek benötigt wird, so kann zunächst mit dem Kommando

```
arduino-cli lib search ethernet
```

on die Bibliothek, hier `ethernet`, installiert ist. Mit dem Befehl

```
arduino-cli lib install "UIPEthernet"
```

wird sie dann installiert.

6.4.6. Beispiel „Hello World“

Um das Kompilieren eines Sketches sowie das anschließende Hochladen mit Hilfe des Arduino-CLI zu demonstrieren, soll das „Hello World!“-Pendant für Mikrocontroller verwendet werden. Dafür wird ein einfacher Sketch, der die LED des Arduinos zum Blinken bringt, benutzt.

```
void setup(){
    pinMode(LED_BUILTIN, OUTPUT);
}
void loop(){
    digitalWrite(LED_BUILTIN, HIGH);
    delay(1000);
    digitalWrite(LED_BUILTIN, LOW);
    delay(1000);
}
```

In der zu Beginn einmalig ausgeführten `setup()`-Funktion wird die eingebaute LED als Output definiert `pinMode(LED_BUILTIN, OUTPUT)`. Dadurch lässt sich die LED später ansteuern. Innerhalb der `loop()`-Funktion wird die LED zunächst eingeschaltet `digitalWrite(LED_BUILTIN, HIGH)` und mit einer Verzögerung von einer Sekunde `delay(1000)` wieder abgeschaltet `digitalWrite(LED_BUILTIN, LOW)`. Vor dem erneuten Einschalten der LED zu Beginn der Schleife wird am Schleifenende nochmal eine Sekunde gewartet um den blinkenden Effekt zu erzielen.
Um den Sketch zu kompilieren, kann der zuvor als Grundfunktion definierte Befehl `compile`, wie in Abb. 6.6 dargestellt, verwendet werden.

```
C:\Users\denni>arduino-cli compile -b arduino:mbed_nano:nano33ble C:\Users\denni\A\Ablage\ArduinoProjekte\blink
```

Figure 6.6.: Für das Kompilieren des Blink-Sketches wird der Dateipfad zu dem Sketch mit angegeben (rot unterstrichen). Außerdem sollte der FQBN des Arduinos, für den der Sketch kompiliert wird, genannt werden.

Anschließend kann der kompilierte Sketch mit Hilfe der „upload“-Funktion auf den Arduino hochgeladen werden. Dafür wird, wie in Abb. 6.7 dargestellt, der Speicherpfad zu dem Sketch angegeben. Dadurch erfolgt das Hochladen der kompilierten Daten für den angegebenen Sketch.

```
C:\Users\denni>arduino-cli upload -p COM5 C:\Users\denni\A\Ablage\ArduinoProjekte\blink
```

Figure 6.7.: Hochladen des kompilierten Sketches aus dem Temp-Ordner

Nach dem erfolgreichen Upload beginnt die in Abb. 6.8 gezeigte LED des Arduino Nano 33 BLE Sense Lite zu blinken.

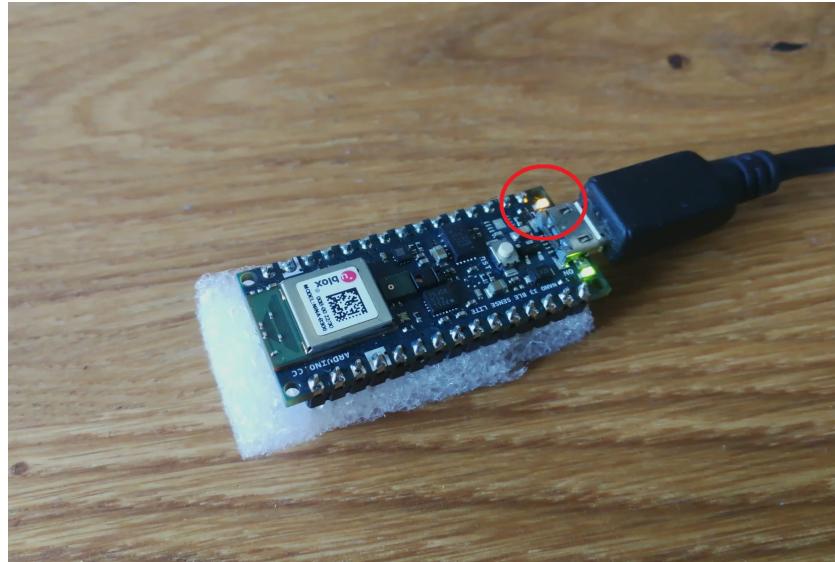


Figure 6.8.: Die orange LED des Arduinos beginnt zu blinken

6.5. Beschreibung der Software auf dem PC

Um einen erhöhten Automatisierungsgrad zu erreichen und die Potentiale des Arduino-CLI zu demonstrieren, wird ein Batch-Skript zum Testen der Sensoren verwendet. Dabei werden die Ergebnisse in einer Log-Datei dokumentiert. In diesem Abschnitt soll die Funktionsweise des Batch-Skriptes beschrieben werden.

```
@echo off
echo Logdatei Arduino-Setup, %time% Uhr, %date% > setup-log.txt
echo. >> setup-log.txt
echo. >> setup-log.txt
```

Mit dem Befehl `echo off` wird die Ausgabe des Skript-Inhaltes bei Ausführung verhindert. Dadurch wird eine ansprechendere Frontend-Programmierung ermöglicht. In der zweiten Zeile wird die Log-Datei erstellt. Beim Erstellen wird mit der Überschrift `Logdatei Arduino-Setup` sowie dem Auslesen der Systemzeit `%time%` und des Datums `%date%` die erste Zeile des Protokolls versehen. Für die spätere Übersichtlichkeit der Log-Datei folgen zwei leere Zeilen `echo. >> setup-log.txt`. Mit dem einmaligen Anwenden des größer als Operators `> setup-log.txt` wird eine neue Datei erstellt oder eine Bestehende mit dem gleichen Namen überschrieben. Bei doppelter Anwendung des größer als Operators `>> setup-log.txt` wird der linksseitige Inhalt in einer neuen Zeile an eine bestehende Datei angehängt. So wird die Log-Datei Zeile für Zeile beschrieben und nicht einmalig komplett am Ende des Batch-Skriptes.

```
:: Gegebenenfalls wird der Kern installiert
echo Kernstatus: >> setup-log.txt
arduino-cli core install arduino:mbed_nano >> setup-log.txt
```

Kommentare in dem Batch-Skript werden durch zwei aufeinanderfolgende Doppelpunkte gekennzeichnet `:: Gegebenenfalls wird der Kern installiert` und sollen die Nachvollziehbarkeit innerhalb des Skriptes erhöhen. Mit dem Befehl `echo Kernstatus: >> setup-log.txt` wird in der Log-Datei gekennzeichnet, dass in der nächsten Zeile Informationen über den zu installierenden Kern für den Arduino Nano 33 BLE Sense Lite folgen. Das Arduino-CLI bringt bereits die Intelligenz

mit zu prüfen, ob der angegebene Kern schon installiert ist. Sollte dies der Fall sein, erfolgt die entsprechende Dokumentation in der Log-Datei. Das Installieren des Kerns wird mit der Zeile `arduino-cli core install arduino:mbed_nano » setup-log.txt` initialisiert. Dabei ist der für den Arduino Nano 33 BLE Sense Lite notwendige Kern mit dem Namen `arduino:mbed_nano` angegeben. Der Rückgabewert der Funktion zum Kerninstallation wird in der Log-Datei gespeichert.

```
echo Es wird nach angeschlossenen Boards gesucht ...
:: Auflisten der angeschlossenen Arduinos
echo Folgende Boards sind am PC angeschlossen: >> setup-log.txt
echo .
echo. >> setup-log.txt
arduino-cli board list >> setup-log.txt
arduino-cli board list
echo .
```

Zu Beginn dieses Skript-Abschnittes wird zunächst der Nutzer der Software über die Suche nach den angeschlossenen Arduino Platinen informiert `echo Es wird nach angeschlossenen Boards gesucht....`. Folgend wird mit dem Befehl `echo Folgende Boards sind am PC angeschlossen: » setup-log.txt` über den Inhalt der nächsten Zeile in der Log-Datei informiert. Die Information darüber welche Arduinos an dem PC angeschlossen sind, kann mit dem Befehl `arduino-cli board list`, der hier doppelt ausgeführt wird, gewonnen werden. Bei der ersten Ausführung wird die Rückgabe des Arduino-CLI in der Log-Datei gespeichert `» setup-log.txt`, die zweite Ausführung dient zur Anzeige in dem aktuell ausgeführten Skript. Die Information über die angeschlossenen Arduinos benötigt der Nutzer im nächsten Schritt für eine Eingabe.

```
:: Abfragen des Portnamens fuer spaeteren Upload
:: der Sensorentestdatei
set /p port="Bitte den Portnamen des Sense-Lite eingeben und bestaetigen: "
echo Der Port %port% wurde gewaehlt. >> setup-log.txt
```

In der Zeile `set /p port="Bitte den Portnamen des Sense-Lite eingeben und bestaetigen: "` erfolgt eine Nutzerabfrage. Der Befehl `set` ermöglicht das Setzen der Variable `port` auf einen bestimmten Wert. Mit dem Zusatz `/p` wird dieser bestimmte Wert auf die folgende Nutzereingabe gesetzt. Die Ausführung des Skriptes wird bis zur erfolgreichen Eingabe des Nutzers gestoppt. Die Nutzereingabe beinhaltet den Port des angeschlossenen Arduino Nano 33 BLE Sense Lite. Für die spätere Nachvollziehbarkeit wird der gewählte Port in der Log-Datei mit der Zeile `echo Der Port %port% wurde gewaehlt. » setup-log.txt` dokumentiert.

```
:: Anlegen des Ordners fuer den kompilierten Sensorentest
set ordner=SensorentestCompiledData
mkdir %ordner%
```

Die Variable `ordner` bekommt den Wert `SensorentestCompiledData` zugewiesen. In der nächsten Zeile `mkdir %ordner%` wird der Ordner mit dem Namen `SensorentestCompiledData` für das spätere Speichern des kompilierten Sketches angelegt.

```
:: Gegebenenfalls werden die noetigen Bibliotheken fuer
:: den Sensorentest installiert
:: Bib fuer IMU
echo Bib fuer IMU >> setup-log.txt
arduino-cli lib install Arduino_LSM9DS1 >> setup-log.txt
echo. >> setup-log.txt
```

```
:: Bib fuer Farbsensor
echo Bib fuer Farbsensor >> setup-log.txt
arduino-cli lib install Arduino_APDS9960 >> setup-log.txt
echo . >> setup-log.txt

:: Bib fuer Druck- und Temperatursensor
echo Bib fuer Druck- und Temperatursensor >> setup-log.txt
arduino-cli lib install Arduino_LPS22HB >> setup-log.txt
echo . >> setup-log.txt
```

In der Log-Datei wird jeweils dokumentiert um welche Bibliothek es sich in der folgenden Zeile handelt `SensorentestCompiledData`. Anschließend erfolgt mit dem Befehl `arduino-cli lib install Arduino_LSM9DS1 » setup-log.txt` das Installieren der Bibliothek sowie das Speichern des Rückgabewertes der Installation in der Log-Datei. Eine bereits installierte Bibliothek wird von dem Arduino-CLI erkannt und es folgt eine entsprechende Rückgabe in die Log-Datei. Das Vorgehen ist für alle drei zu installierenden Bibliotheken identisch.

```
:: Kompilieren des Sensorentest-Sketches
echo Kompilieren des Sensorentest-Sketches: >> setup-log.txt
echo . >> setup-log.txt
arduino-cli compile -b arduino:mbed_nano:nano33ble %cd%\SensortestLite
--build-path %cd%\%ordner% >> setup-log.txt
```

Nachdem die notwendigen Bibliotheken installiert sind, kann der zuvor erstellte Sketch zum Testen der Sensoren kompiliert werden. Das Kompilieren des Sketches für den Arduino Nano 33 BLE Sense Lite erfolgt mit dem Befehl `arduino-cli compile -b arduino:mbed_nano:nano33ble %cd%/SensortestLite`. Der hintere Teil des Befehls gibt den Speicherpfad des zu kompilierenden Sketches an. Weiterhin lässt sich der Zielordner für den kompilierten Sketch mit dem Zusatz `--build-path %cd%\%ordner% » setup-log.txt` definieren. Der Rückgabewert des Kompilierens mit dem Arduino-CLI wird in der Log-Datei gespeichert.

```
:: Hochladen des kompilierten Sketches auf den Arduino
echo Hochladen des kompilierten Sketches auf den Arduino >> setup-log.txt
arduino-cli upload -p %port% --input-dir %cd%\%ordner% >> setup-log.txt
```

Der kompilierte Sketch wird in der Zeile `arduino-cli upload -p %port% -input-dir %cd%\%ordner% » setup-log.txt` auf den durch `port` angeschlossenen Arduino Nano 33 BLE Sense Lite hochgeladen. Der Speicherpfad der kompilierten Daten wird mit dem Zusatz `-input-dir %cd%\%ordner%` angegeben.

```
:: Oeffnen des seriellen Monitors
start monitor_log
:: Automatisches Schlieszen des seriellen Monitors nach 4 Sekunden
:: (n=1 Sekunden, mit n=5)
ping 127.0.0.1 -n 5 > nul
taskkill /im serial-monitor.exe /F
```

Mit dem Befehl `start monitor_log` wird ein weiteres Batch-Skript zum Auslesen des seriellen Monitors aufgerufen. Die Zeile `ping 127.0.0.1 -n 5 > nul` stoppt die Ausführung der Software für vier Sekunden. Dabei werden fünf Anfragen an den lokalen Rechner gesendet und jeweils eine Sekunde zwischen zwei Anfragen gewartet. Mit `> nul` wird die Ausgabe der Antworten ins Leere geleitet und nicht angezeigt. Ist die Zeitspanne von vier Sekunden abgelaufen wird der serielle Monitor zum Auslesen der Sensordaten automatisch mit dem Befehl `taskkill /im serial-monitor.exe /F` geschlossen. Über den Zusatz `/im` kann das zu schließende Fenster `serial-monitor.exe` benannt werden. Das erzwungene Schließen des seriellen Monitors

erfolgt mit dem Anhang [/F](#). Nachdem der serielle Monitor ausgelesen und geschlossen wurde, kann die Software beendet werden. Die Ergebnisse des Sensoren-Tests lassen sich im Anschluss der Log-Datei entnehmen.

Das Öffnen des seriellen Monitors erfolgt in dem Skript [monitor_log.bat](#).

```
echo Sensordaten: >> setup-log.txt  
echo. >> setup-log.txt
```

echo Der serielle Monitor wird geoeffnet , die Sensordaten werden ausgelesen und i

```
arduino-cli monitor -p %port% >> setup-log.txt
```

Der serielle Monitor kann mittels des Arduino-CLI Befehls `arduino-cli monitor -p %port% » setup-log.txt` geöffnet werden. Dabei wird mit `-p %port%` die Verbindung für den angegebenen Port hergestellt. Mit dem Zusatz `» setup-log.txt` werden die empfangenen Daten in die Log-Datei geschrieben.

7. Arduino Lab for MicroPython

Part II.

Arduino Nano 33 BLE Sense - Onboard Sensoren

8. Arduino Nano 33 BLE Sense

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

WS:Advertisement!
Rewrite more as an engineer!

8.1. 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 8.1 shows a Arduino Nano 33 BLE Sense, shows how compact it is and small in size.

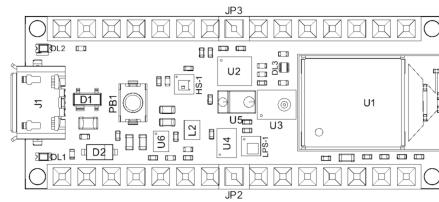
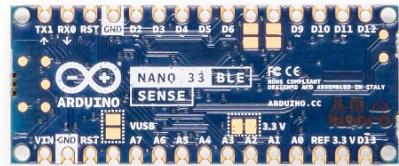
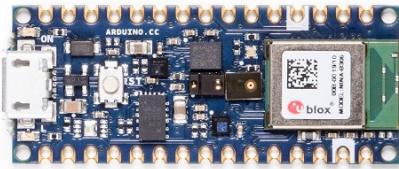


Figure 8.1.: Arduino Nano 33 BLE Sense, see Arduino Store

WS:tikz?

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 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.

8.2. Was ist der Unterschied zwischen Rev1 und Rev2?

Es gab einige Änderungen an den Sensoren zwischen beiden Revisionen:

Austausch des IMU von LSM9DS1 (9 Achsen) durch eine Kombination aus zwei IMUs (BMI270 - 6 Achsen IMU und BMM150 - 3 Achsen IMU). Austausch des Temperatur- und Feuchtigkeitssensors von HTS221 zu HS3003. Austausch des Mikrofons von MP34DT05 zu MP34DT06JTR. Zusätzlich wurden einige Komponenten und Änderungen vorgenommen, um die Benutzererfahrung zu verbessern:

Austausch der Stromversorgung von MPM3610 zu MP2322. Hinzufügen eines VUS-B-Lötpads auf der Oberseite der Platine. Neue Testpunkte für USB, SWDIO und SWCLK.

Muss ich meinen Sketch für die vorherige Revision ändern?

Für Sketches, die mit Bibliotheken wie LSM9DS1 für das IMU oder HTS221 für den Temperatur- und Feuchtigkeitssensor erstellt wurden, müssen für die neue Revision diese Bibliotheken zu folgenden geändert werden: Arduino_BMI270_BMM150 für das neue kombinierte IMU und Arduino_HS300x für den neuen Temperatur- und Feuchtigkeitssensor.

Rev 1 [TensorFlow Lite lib](#)

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

Das Tiny Machine Learning Kit enthält nur den Arduino Nano 33 BLE Sense Lite. Im Folgenden werden hier die Unterschiede der drei verschiedenen Versionen erläutert.

Die Unterschiede zwischen dem Arduino Nano 33 BLE Sense und dem Arduino Nano 33 BLE Sense Rev2 betreffen die IMU, den Temperatur- und Feuchtigkeitssensor, das Mikrofon, den Crypto-Chip und den Spannungswandler. In der zweiten Revision wurde die IMU LSM9DS1 durch zwei Module ersetzt: einerseits durch einen Beschleunigungs- und Drehratensensor BMI270 und das Magnetometer BMI150. Bei den Feuchtigkeitssensoren wurde der HTS221 durch den HS3003 ausgetauscht, wobei letzterer eine höhere Genauigkeit verspricht. Die Werte der Mikrofone sind trotz des Wechsels von dem MP34DT05 zum MP34DT06JTR gleichgeblieben. Ebenso unterscheiden sich nur die Bauteilbezeichnungen der Spannungswandler. Bei der ersten Generation wird der MPM3610 verwendet, der Rev2 hat den MP2322 verbaut. Bei dem Rev2 ist allerdings der Crypto-Chip nicht mehr vorhanden.

Der Arduino Nano 33 BLE Sense Lite unterscheidet sich nur geringfügig von dem Arduino Nano 33 BLE Sense Rev2. Der einzige Unterschied zwischen den beiden Modellen ist, dass die Lite Version nicht über den HTS221, sondern über den Sensor LPS22HB verfügt. Hierbei handelt es sich um einen Drucksensor, mit dem es allerdings auch möglich ist, die Temperatur zu messen. Feuchtigkeit lässt sich also nicht mit dem Lite messen. Um relative Luftfeuchtigkeit zu messen, muss ein separater Sensor angeschlossen werden. Der Grund für diese Entscheidung ist, dass die Firma Arduino Schwierigkeiten hat, den aktuellen Lagerbestand aufrechtzuerhalten. [FD22]

8.4. 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 used for measuring both the analog and digital values around the surrounding. Arduino Nano 33 BLE sense is very small 45mm × 18mm in size, which makes it very useful 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 usefull 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 usefull 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 below figure 8.2 show the embed sensors on the board, with 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 follow.

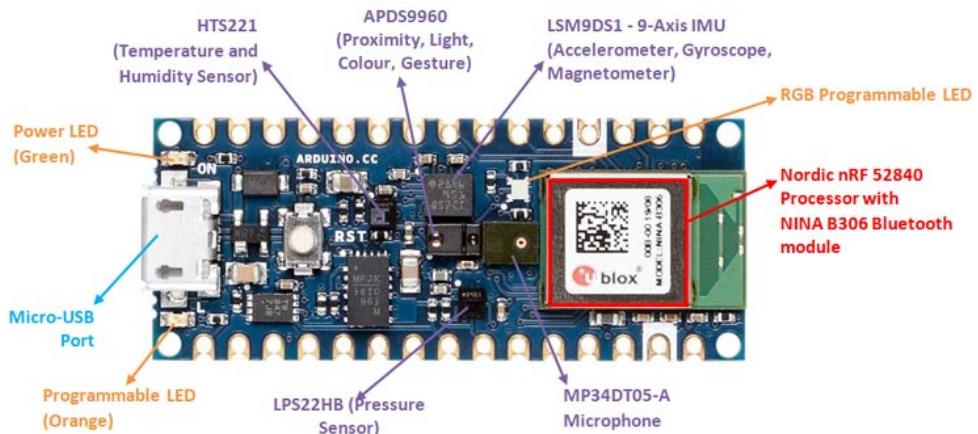


Figure 8.2.: Components in Arduino Nano 33 BLE Sense [Raj19]

WS:Overpic

Another point to bear in mind is the overall 'trueness' of sensor readings based on where do 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 8.1.: Technical Specifications of Arduino Nano 33 BLE Sense [Ard21]

8.4.1. 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

8.4.2. Accelerometer, Gyroscope, and Magnetometre Sensor LSM9DS1

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

For the following Applications the IMU is in use:

- Indoor navigation
- Advanced gesture recognition
- Gaming and virtual reality input devices
- Display/map orientation and browsing
- Consumer electronics; Smartphones, tablets, fitness trackers for motion sensing and orientation.
- Compact transportation solutions like Segway.

- Sports Technology - helping athletes to know how they can improve their movements.

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

WS:The explanation of drift is to small; citations

Calibration of an IMU

WS:What is calibration? citations!

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]

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

- General infomration about imus
- angle to roation matrix with example!
- calibration
- drift
- ...
- specials for this imu
- description of the parameters for coding
- example code

8.4.3. Pressure Sensor LPS22HB

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

For the following Applications the sensor is in use:

- Altimeters and barometers for portable devices
- Weather station equipment

- Sports watches

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 $+85^{\circ}\text{C}$. [STM17]

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

8.4.4. 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°C bis $+120^{\circ}\text{C}$. Zur Spannungsversorgung werden 1,7 bis 3,3 Volt benötigt. Eine Temperaturnmessung erfolgt mit einer Genauigkeit von $\pm 5^{\circ}\text{C}$. [STM23]

- GND - Ground
- DRDY - Data ready output signal
- SCL/SPC - I2C serial clock (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

8.4.5. 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 \pm 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

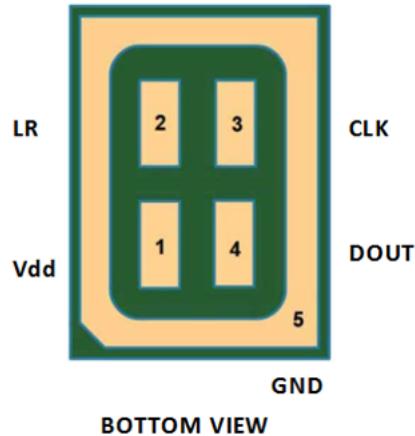


Table 1. Pin description

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 8.3.: Circuit diagram microphone [Stmc]

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 8.1 will print the sound level of the microphone to the serial monitor every 100 milliseconds.

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

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

8.4.6. 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

```
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 8.1.: Simple example using of the builtin microphone of the Arduino Nano 33 BLE Sense

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

8.5. 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 8.4 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.

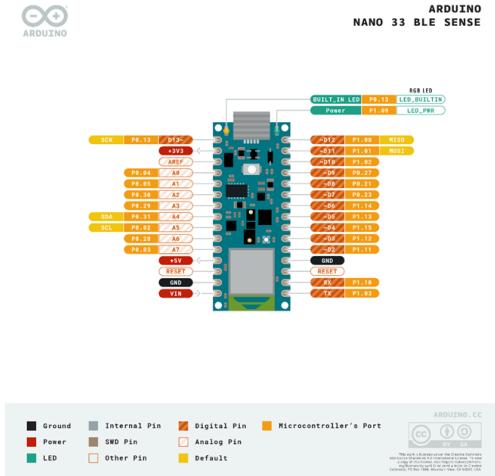


Figure 8.4.: Arduino Nano 33 BLE Pin Configuration

8.6. Was fehlt

- Beschreibung der Sensoren
 - Hintergrundwissen
 - Beschreibung
 - Eigenschaften
 - Kalibrierung
- Beschreibung der Bibliothek
 - Beschreibung
 - Installation
 - Funktionen
- Laufzeit, Speicherplatz, ...
- Software-Dokumentation
- Verwendung von Werkzeugen, z.B. Doxygen
 - Beschreibung, inklusive im Quellcode; z.B. Header
 - Handbuch
 - Ablaufdiagramm
 - ...
- Interpretation der Ergebnisse
- Literatur, Bilder
- ...

9. 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 9.1.: Das Tiny Machine Learning Kit [Ardk]

9.1. 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.

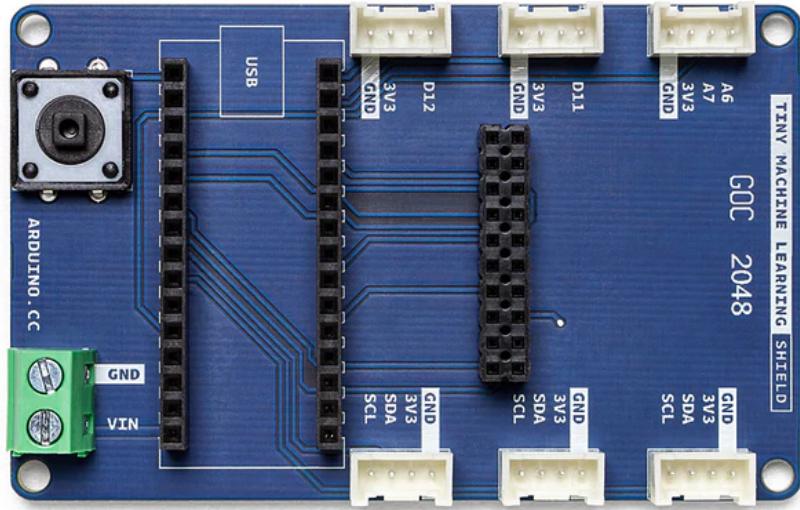


Figure 9.2.: Das Tiny Machine Learning Shield [Ardk]

9.2. 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.

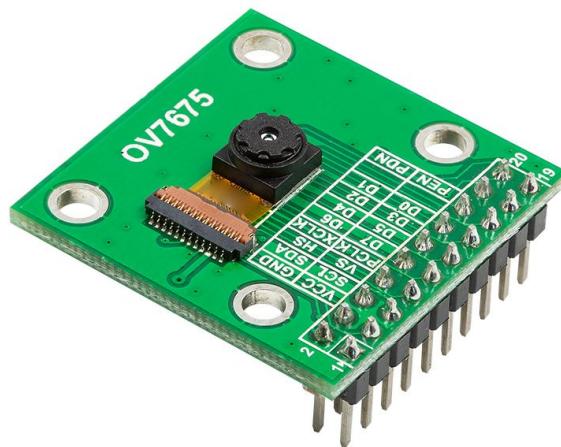


Figure 9.3.: Das OV7675 Kameramodul [Ardk]

9.3. 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 9.4.: Ein USB-A auf Micro-USB Kabel

10. Power LED

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

10.1. General Information

The power LED on an Arduino board is a small, usually red or green, 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.

WS:cite books

10.2. Specific Sensor

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.

WS:cite Arduino

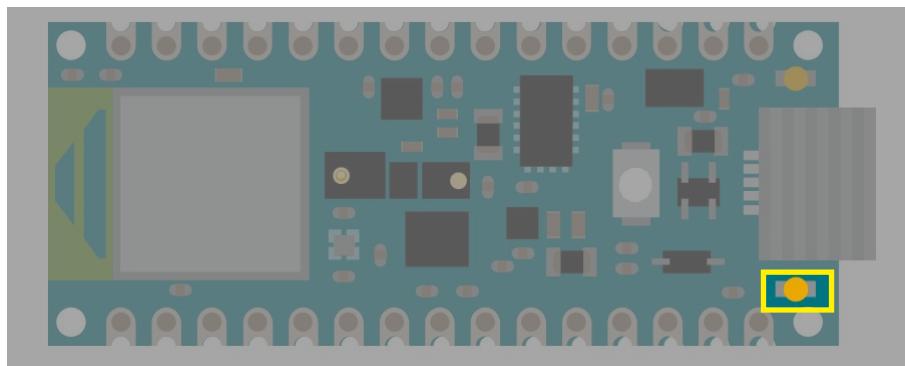


Figure 10.1.: Arduino Nano 33 BLE Sense's Power LED

WS:tikz picture with power LED

10.3. Specification

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

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.

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

S:cite data sheet, power consumption?

WS:to be tested!

S:What happen if there is another LED at the pin? Both in use?

10.4. Simple Code

In the sketch 10.2, a variable is connected to pin 25. The pin 25 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 10.1.: Simple sketch to control the power LED

```

1000 // How to control the power LED of the Arduino Nano 33 BLE Sense
1002 //
1003 // file: TestLEDPower.ino
1004 //
1005 // The LED is switched on for 1 second and switched off
1006 // for 1 second so that the LED flashes accordingly.
1007
1008 #define LED_PWR 25 // Define the pin for the power LED
1009
1010 void setup() {
1011     // Initialize the pin as an output
1012     pinMode(LED_PWR, OUTPUT);
1013 }
1014
1015 void loop() {
1016     // Turn the LED on
1017     digitalWrite(LED_PWR, HIGH);
1018     // Wait for one second
1019     delay(1000);
1020     // Turn the LED off
1021     digitalWrite(LED_PWR, LOW);
1022     // Wait for one second
1023     delay(1000);
1024 }
```

..../Code/Nano33BLESense/Test/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.

WS:Arduino Referenz

<https://github.com/pfeerick/elapsedMillis/wiki>

10.5. Tests

10.5.1. Simple Function Test

The simplest test is the flashing of the LED at 2 Hz.

Listing 10.2.: Simple sketch to test the power LED

```

1000 // How to control the power LED of the Arduino Nano 33 BLE Sense
1002 //
1003 // file: TestLEDPower.ino
1004 //
1005 // The LED is switched on for 1 second and switched off
1006 // for 1 second so that the LED flashes accordingly.
1007
1008 #define LED_PWR 25 // Define the pin for the power LED
1009
1010 void setup() {
1011     // Initialize the pin as an output
1012     pinMode(LED_PWR, OUTPUT);
1013 }
1014
1015 void loop() {
1016     // Turn the LED on
1017     digitalWrite(LED_PWR, HIGH);
1018     // Wait for one second
1019     delay(1000);
1020     // Turn the LED off
1021     digitalWrite(LED_PWR, LOW);
1022     // Wait for one second
1023     delay(1000);
1024 }
```

..../Code/Nano33BLESense/Test/TestLEDPower.ino

10.5.2. Test all Functions

The brightness of the power LED can be controlled. This is demonstrated in the example sketch 10.3.

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

Listing 10.3.: Simple sketch to check the battery state using the power LED

```

1000 // How to control the power LED of the Arduino Nano 33 BLE Sense
1002 //
1003 // file: TestLEDPowerBrightness.ino
1004 //
1005 // Using PWM, the LED's brightness is increasing
1006 // until full and reverse.
1007
1008 #define LED_PWR 25
1009
1010 // Define the initial brightness values (0–255)
1011 int Brightness = 0;
1012
1013 // Define the increment/decrement value
1014 int redStep = 5;
1015
1016 void setup() {
1017     // Initialize the pin as an output
1018     pinMode(LED_PWR, OUTPUT);
1019 }
1020
1021 void loop() {
1022     // Write the PWM values to the LED pin
1023     analogWrite(LED_PWR, redBrightness);
1024
1025     // Update the brightness values
1026     Brightness += Step;
```

```

1028 // Check if the brightness values are out of range and reverse the
1029 // direction
1030 if (Brightness <= 0 || Brightness >= 255) {
1031     Step = -Step;
1032 }
1033 // Wait for 10 milliseconds
1034 delay(10);
}

```

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

10.6. 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.

A simple application is to check the condition of the battery. The sktech 10.4 demonstrates, if the voltage drops too low, the power LED flashes.

Listing 10.4.: Simple sketch to check the battery state using the power LED

```

1000 // Test of the Power LED
1001 //
1002 // file: TestLEDPowerBattery.ino
1003 //
1004 // Check the battery state.
1005 // if the battery state less 20%, the power LED will blink.
1006 #include <Arduino.h>

1008 // LED pin for visual indication (optional)
1009 #define LED_PWR 25 // Define the pin for the power LED
1010
1011 // Battery measurement pin
1012 const int BATTERY_PIN = A0;

1014 // Reference voltage for 3.3V (adjust if using external power)
1015 const float REFERENCE_VOLTAGE = 3.3;
1016
1017 const int LED_PWR = 25;

1018 void setup() {
1019     // set LED pin as output
1020     pinMode(LED_PWR, OUTPUT);
1021 }

1023 // Input:
1024 // SendMessages false - no messages
1025 //                  true - with messages
1026 // Return: 0 - okay
1027 //          1 - < 20%
1028 int BatteryState(bool SendMessages)
1029 {
1030     int ret;

```

```
1032 // Read battery voltage from analog pin
1033 float rawVoltage = analogRead(BATTERY_PIN) * (REFERENCE_VOLTAGE /
1023.0);
1034
1035 // Calculate percentage based on reference voltage (adjust based on
1036 // battery specs)
1037 float batteryPercentage = (rawVoltage / 4.2) * 100.0;
1038
1039 if (SendMessages)
1040 {
1041     // Print battery voltage and percentage (for debugging)
1042     Serial.print("Battery voltage: ");
1043     Serial.print(rawVoltage);
1044     Serial.println(" V");
1045     Serial.print("Battery percentage: ");
1046     Serial.print(batteryPercentage);
1047     Serial.println("%");
1048 }
1049
1050 // Optional: blink LED based on battery level (adjust thresholds)
1051 if (LED_PWR >= 0) {
1052     if (batteryPercentage < 20) {
1053         digitalWrite(LED_PWR, HIGH);
1054         delay(500);
1055         digitalWrite(LED_PWR, LOW);
1056         delay(500);
1057         ret = 1;
1058     } else {
1059         digitalWrite(LED_PWR, HIGH);
1060         ret = 0;
1061     }
1062 }
1063 else {
1064     ret = 0;
1065 }
1066 return ret;
1067 }
1068
1069 void loop() {
1070     BatteryState(false);
1071     // Delay between measurements
1072     delay(5000);
1073 }
```

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

10.7. Further Readings

WS:citations

11. Built-inLED

Some Arduino boards have a LED on board which can use for the application.

11.1. 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.

WS:cite books

11.2. Specific Sensor

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.

tikz picture with built-in LED

WS:cite Arduino

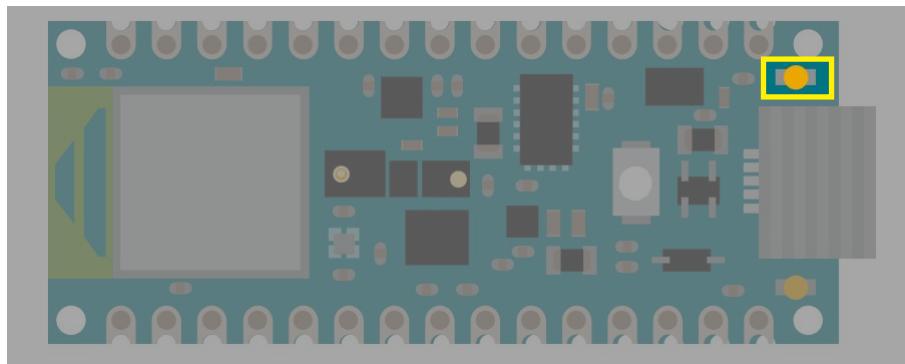


Figure 11.1.: Arduino Nano 33 BLE Sense's built-in LED

WS:tikz picture with
built-in LED

11.3. Specification

The built-in LED is a orange LED and connected to pin 13. The brightness can not be controlled.

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`.

S:cite data sheet, power consumption?

S:What happen if there is another LED at the pin? Both in use?

11.4. Simple Code

In the sketch 15.1, a variable is connected to pin 13. The pin is defined as an output in the function `setup`. In the function `loop`, the LED is switched on for 1 second and switched off for 1 second so that the LED flashes accordingly.

Listing 11.1.: Simple sketch to control the built-in LED

```

1000 // How to control the built-in LED of the Arduino Nano 33 BLE Sense
1001 //
1002 // file: TestLEDBuiltin.ino
1003 //
1004 // The LED is switched on for 1 second and switched off
1005 // for 1 second so that the LED flashes accordingly.
1006 //
1007 // Define the pin for the built-in LED
1008 #define LED_BUILTIN 13

1009 void setup() {
1010     // Initialize the pin as an output
1011     pinMode(LED_BUILTIN, OUTPUT);
1012 }

1013 void loop() {
1014     // Turn the LED on
1015     digitalWrite(LED_BUILTIN, HIGH);
1016     // Wait for one second
1017     delay(1000);
1018     // Turn the LED off
1019     digitalWrite(LED_BUILTIN, LOW);
1020     // Wait for one second
1021     delay(1000);
1022 }

```

..//Code/Nano33BLESense/Test/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.

WS:Arduino Referenz
<https://github.com/pfeerick/elapsedMillis/wiki>

11.5. Tests

11.5.1. Simple Function Test

The simplest test is the flashing of the LED at 2 Hz, see sketch 11.2.

Listing 11.2.: Simple sketch to test the built-in LED

```

1000 // How to control the built-in LED of the Arduino Nano 33 BLE Sense
1001 //
1002 // file: TestLEDBuiltin.ino
1003 //
1004 // The LED is switched on for 1 second and switched off
1005 // for 1 second so that the LED flashes accordingly.
1006

```

```

1008 // Define the pin for the built-in LED
#define LED_BUILTIN 13

1010 void setup() {
    // Initialize the pin as an output
1012     pinMode(LED_BUILTIN, OUTPUT);
}

1014 void loop() {
    // Turn the LED on
1016     digitalWrite(LED_BUILTIN, HIGH);
    // Wait for one second
1018     delay(1000);
    // Turn the LED off
1020     digitalWrite(LED_BUILTIN, LOW);
    // Wait for one second
1022     delay(1000);
}

```

..../Code/Nano33BLESense/Test/TestLEDBuiltin.ino

11.5.2. Test all Functions

As the built-in does not allow any special manipulations, all functions are covered with the sketch 11.2.

11.6. 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 12.5.

Listing 11.3.: A simple watch dog: A simple watchdog: the built-in LED is switched on for 1 second every 30 seconds.

```

1000 // How to control the built-in LED of the Arduino Nano 33 BLE Sense
1001 /**
1002 // file: TestLEDBuiltinApplication.ino
1003 /**
1004 // The LED is switched on for 1 second and switched off
1005 // for 29 second so that the LED flashes accordingly.
1006 /**
1007 // Define the pin for the built-in LED
#define LED_BUILTIN 13

1010 // Stores the last time the LED was turned on
1011 unsigned long currentMillis = 0;
1012 // Stores the last time the LED was turned on
1013 unsigned long previousMillis = 0;
1014 // Duty cycle
#define CycleTimeOn 1000
1016 // Switch-off time
#define CycleTimeOff 29000
1018 // state: true - LED on
// false - LED off
1020 bool SignsOfLifeState = false;

1022

1024 void setup() {
    // Initialize the pin as an output
1026     pinMode(LED_BUILTIN, OUTPUT);

```

```

1028 }
1029 // This function checks the elapsed time.
1030 // According the elapsed time the LED is switched on or off
1031 void SignsOfLife() {
1032     // Check if CycleTimeOff have passed since the last LED turn on
1033     currentMillis = millis();
1034     if (SignsOfLifeState == true){
1035         if (currentMillis - previousMillis >= CycleTimeOn) {
1036             digitalWrite(ledPin, LOW);    // Turn LED on
1037             previousMillis = currentMillis;
1038             SignsOfLifeState = false;
1039         }
1040     }
1041     else {
1042         if (currentMillis - previousMillis >= CycleTimeOff) {
1043             digitalWrite(ledPin, HIGH);   // Turn LED on
1044             previousMillis = currentMillis;
1045             SignsOfLifeState = true;
1046         }
1047     }
1048 }
1049
1050
1051 void loop() {
1052     // Switch builtin LED on/off
1053     SignsOfLife();
1054
1055     // Application
1056 }

```

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

11.7. Further Readings

WS:citations

12. Built-in RGB-LED

Some Arduino boards have a RGB-LED on board which can use for the application.

12.1. 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.

WS:cite books

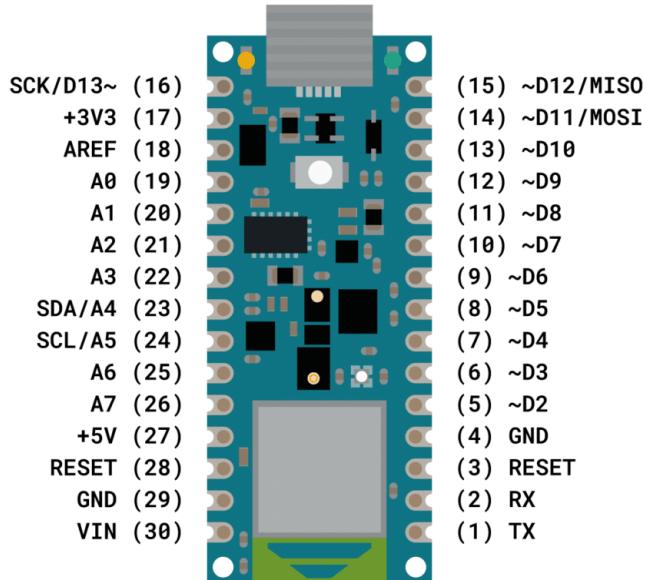
12.2. Specific Sensor

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.

tikz picture with RGB-LED



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.

WS:cite Arduino

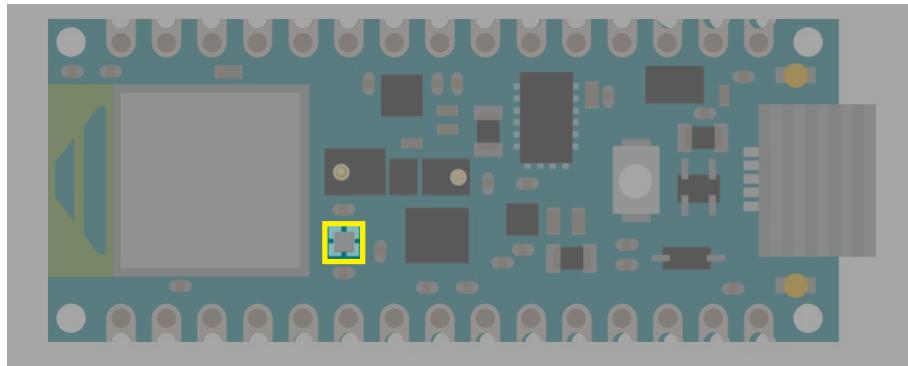


Figure 12.1.: Arduino Nano 33 BLE Sense's RGB LED

WS:tikz picture with
RGB LED

12.3. Specification

The RGB LED occupies pins on the board internally. These pins are defined via variable names `LEDR`, `LEDG`, and `LEDB`.

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`.

WS:test

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.

WS:cite data sheet, power consumption?

WS:What happens if there is another LED at the pin? Both in use?

12.4. Simple Code

In the sketch 12.1, a variable is connected to pin 13. The pins are defined as outputs in the function `setup`. In the function `loop`, every color of the LED is switched on for 1 second and switched off for 1 second so that the LED flashes accordingly.

Listing 12.1.: Simple sketch to test the RGB LED

```

1000 // file: TestLEDRGB
1002 // Define the pins for the RGB LED
1003 #define LEDR 22
1004 #define LEDG 23
1005 #define LEDB 24
1006
1008 void setup() {
1009     // Set the LED pins as outputs
1010     pinMode(LEDR, OUTPUT);
1011     pinMode(LEDG, OUTPUT);
1012     pinMode(LEDB, OUTPUT);
1013 }
1014
1016 void loop() {
1017     // Turn the red LED on
1018     digitalWrite(LEDR, HIGH);
1019     // Wait for one second
1020     delay(1000);
1021     // Turn the LED off
1022     digitalWrite(LEDR, LOW);
1023     // Turn the green LED on
1024     digitalWrite(LEDG, HIGH);
1025     // Wait for one second
1026     delay(1000);
1027     // Turn the LED off
1028     digitalWrite(LEDG, LOW);
1029     // Turn the blue LED on
1030     digitalWrite(LEDB, HIGH);
1031     // Wait for one second
1032     delay(1000);
1033     // Turn the LED off
1034     digitalWrite(LEDB, LOW);
1035     // Wait for one second
1036     delay(1000);
1037 }
```

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

This is just a simple example. The variables `LEDR`, `LEDG`, and `LEDB` are 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.

WS:Arduino Referenz
<https://github.com/pfeerick/elapsedMillis/wiki>

12.5. Tests

12.5.1. Simple Function Test

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

Listing 12.2.: Simple sketch to test the RGB LED

```

1000 // file: TestLEDRGB
1002 // Define the pins for the RGB LED
1003 #define LEDR 22
1004 #define LEDG 23
1005 #define LEDB 24
1006
1008 void setup() {
    // Set the LED pins as outputs
1009     pinMode(LEDR, OUTPUT);
1010     pinMode(LEDG, OUTPUT);
1011     pinMode(LEDB, OUTPUT);
1012 }
1014
1016 void loop() {
    // Turn the red LED on
1017     digitalWrite(LEDR, HIGH);
    // Wait for one second
1018     delay(1000);
    // Turn the LED off
1019     digitalWrite(LEDR, LOW);
    // Turn the green LED on
1020     digitalWrite(LEDG, HIGH);
    // Wait for one second
1021     delay(1000);
    // Turn the LED off
1022     digitalWrite(LEDG, LOW);
    // Turn the blue LED on
1023     digitalWrite(LEDB, HIGH);
    // Wait for one second
1024     delay(1000);
    // Turn the LED off
1025     digitalWrite(LEDB, LOW);
    // Wait for one second
1026     delay(1000);
}
1036 }
```

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

12.5.2. 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 12.3 tests different colors of the LED. The color of the LED changes every 1 second.

Listing 12.3.: value between 255 - 0 to write to the RGB LED

```

1000 // file: TestLEDRGB
1002 // Define the pins for the RGB LED
#define LEDR 22
#define LEDG 23
#define LEDB 24
1006
1008 void setup() {
    // Set the LED pins as outputs
    pinMode(LEDR, OUTPUT);
    pinMode(LEDG, OUTPUT);
    pinMode(LEDB, OUTPUT);
}
1014
1016 void setRGB(int r, int g, int b)
{
    // Write the PWM values to the LED pins
    analogWrite(LEDR, r);
    analogWrite(LEDG, g);
    analogWrite(LEDB, b);
}
1022
1024
1026 void loop() {
    // red
    setRGB(255,0,0);
    // Wait for one second
    delay(1000);
    // green
    setRGB(0,255,0);
    // Wait for one second
    delay(1000);
    // blue
    setRGB(0,0,255);
    // Wait for one second
    delay(1000);
    // yellow
}
1038

```

```

1040   setRGB(255,255,0);
1041     // Wait for one second
1042     delay(1000);
1043   // cyan
1044   setRGB(0,255,255);
1045     // Wait for one second
1046     delay(1000);
1047   // magenta
1048   setRGB(255,0,255);
1049     // Wait for one second
1050     delay(1000);
1051   // white
1052   setRGB(255,255,255);
1053     // Wait for one second
1054     delay(1000);
1055   // black
1056   setRGB(0,0,0);
1057     // Wait for one second
1058     delay(1000);
1059   // orange
1060   setRGB(255,127,0);
1061     // Wait for one second
1062     delay(1000);
1063   // pink
1064   setRGB(255,192,203);
1065     // Wait for one second
1066     delay(1000);
1067   // purple
1068   setRGB(120,0,128);
1069     // Wait for one second
1070     delay(1000);
1071 }
```

..//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 26.4 will make the RGB LED change colors smoothly by varying the brightness of each LED with different speeds. You can adjust the initial brightness values and the increment/decrement values to get different effects.

Listing 12.4.: Different brightness levels for the RGB LED colors

```

1000 // file: TestLEDRGBBrightness.ino
1001
1002 // Define the pins for the RGB LED
1003 #define RED_PIN 22
1004 #define GREEN_PIN 23
1005 #define BLUE_PIN 24
1006
1007 // Define the initial brightness values for each color (0-255)
1008 int redBrightness = 0;
1009 int greenBrightness = 0;
1010 int blueBrightness = 0;
1011
1012 // Define the increment/decrement value for each color
1013 int redStep = 5;
1014 int greenStep = 3;
1015 int blueStep = 7;
1016
1017 void setup() {
1018   // Set the LED pins as outputs
1019   pinMode(RED_PIN, OUTPUT);
```

```

1020     pinMode(GREEN_PIN, OUTPUT);
1021     pinMode(BLUE_PIN, OUTPUT);
1022 }
1023
1024 void loop() {
1025     // Write the PWM values to the LED pins
1026     analogWrite(RED_PIN, redBrightness);
1027     analogWrite(GREEN_PIN, greenBrightness);
1028     analogWrite(BLUE_PIN, blueBrightness);
1029
1030     // Update the brightness values for each color
1031     redBrightness += redStep;
1032     greenBrightness += greenStep;
1033     blueBrightness += blueStep;
1034
1035     // Check if the brightness values are out of range and reverse the
1036     // direction
1037     if (redBrightness <= 0 || redBrightness >= 255) {
1038         redStep = -redStep;
1039     }
1040     if (greenBrightness <= 0 || greenBrightness >= 255) {
1041         greenStep = -greenStep;
1042     }
1043     if (blueBrightness <= 0 || blueBrightness >= 255) {
1044         blueStep = -blueStep;
1045     }
1046
1047     // Wait for 10 milliseconds
1048     delay(10);
1049 }
```

..//Code/Nano33BLESense/Test/TestLEDRGBrightness.ino

12.6. 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 12.5.

Listing 12.5.: A simple watch dog: A simple watchdog: the built-in LED is switched on for 1 second every 30 seconds.

```

1000 // How to control the built-in LED of the Arduino Nano 33 BLE Sense
1001 //
1002 // file: TestLEDBuiltinApplication.ino
1003 //
1004 // The LED is switched on for 1 second and switched off
1005 // for 29 second so that the LED flashes accordingly.
1006
1007 // Define the pin for the built-in LED
1008 #define LED_BUILTIN 13
1009
1010 // Stores the last time the LED was turned on
1011 unsigned long currentMillis = 0;
1012 // Stores the last time the LED was turned on
1013 unsigned long previousMillis = 0;
1014 // Duty cycle
1015 #define CycleTimeOn 1000
1016 // Switch-off time
1017 #define CycleTimeOff 29000
1018 // state: true - LED on
1019 //           false - LED off
1020 bool SignsOfLifeState = false;
```

```

1022
1024 void setup() {
1025     // Initialize the pin as an output
1026     pinMode(LED_BUILTIN, OUTPUT);
1027 }
1028
1029 // This function checks the elapsed time.
1030 // According the elapsed time the LED is switched on or off
1031 void SignsOfLife() {
1032     // Check if CycleTimeOff have passed since the last LED turn on
1033     currentMillis = millis();
1034     if (SignsOfLifeState == true){
1035         if (currentMillis - previousMillis >= CycleTimeOn) {
1036             digitalWrite(ledPin, LOW);    // Turn LED on
1037             previousMillis = currentMillis;
1038             SignsOfLifeState = false;
1039         }
1040     } else {
1041         if (currentMillis - previousMillis >= CycleTimeOff) {
1042             digitalWrite(ledPin, HIGH);   // Turn LED on
1043             previousMillis = currentMillis;
1044             SignsOfLifeState = true;
1045         }
1046     }
1047 }
1048
1049
1050
1051
1052 void loop() {
1053     // Switch builtin LED on/off
1054     SignsOfLife();
1055
1056     // Application
1057 }
```

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

12.7. Further Readings

WS:citations, Colors

12.8. to do

WS:separat beschreiben

13. Built-in Push Button

13.1. 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).

WS:cite books,
applications, board
image of the board

13.2. Specific Sensor

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.

WS:cite board

13.3. 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.

WS:cite data sheet, power consumption?

The pin 11 can also be used otherwise. Then the button is not in use.

- cite data sheet
- Circuit Diagram

WS:What happens if there is another button at the same pin? Both in use?

13.4. Bibliothek

No special library is required to operate the built-in button.

13.5. 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 13.1.: 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 13.2.: Read the built-in button's state

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

13.6. Tests

The simplest test is the flashing of an LED for 2 seconds, if the button is pressed, see sketch 13.3.

Listing 13.3.: Simple sketch to test the built-in LED

```
1000 // How to read in the built-in button states.
1001 //
1002 // file: TestPushButton.ino
1003 //
1004 // If the button is pressed, the the internal LED
1005 // is turn on for 2 seconds
1006
1007 // Use the onboard push button (BUTTON_B)
1008 #define BUTTON_PIN BUTTON_B
1009 // Define the pin for the built-in LED
1010 #define LED_BUILTIN 13
1011
1012 int buttonState = 0;
1013
1014 void setup() {
1015     // Initialize the pin as an output
1016     pinMode(LED_BUILTIN, OUTPUT);
1017     // Initialize the pin as an input
1018     pinMode(BUTTON_PIN, INPUT_PULLUP);
1019 }
1020
1021 void loop()
1022 {
1023     buttonState = digitalRead(BUTTON_PIN);
1024     if (buttonState == HIGH)
1025     {
1026         // Turn the LED on
1027         digitalWrite(LED_BUILTIN, HIGH);
1028         // Wait for two second
1029         delay(2000);
1030         // Turn the LED off
1031         digitalWrite(LED_BUILTIN, LOW);
1032 }
```

```

1034     // Wait for one second
1035     delay(1000);
1036     buttonState = LOW;
1037 }
```

..//Code/Nano33BLESense/Test/TestPushButton.ino

13.7. Simple Application

In the sketch 13.4, a variable is connected to pin 14. The pin is defined as an output in the function `setup`. An interrupt function is defined, changing the state of a flag. If the flag has the value `true`, the led is switch on for 2 seconds. After the 2 seconds, the led is switch off and the value of the flag is set to `false` back.

Listing 13.4.: Simple sketch to control an external LED. Here, pushing the built-in button is handled by an interrupt. Then the LED switch on for 2 sec.

```

1000 // How to control an external LED
1001 // and using the builtin button
1002 // with the Arduino Nano 33 BLE Sense
1003 //
1004 // file: TestPushButtonInterrupt.ino
1005 //
1006 // If the builtin button is pressed, the LED is switched on
1007 // for 2 second and switched off again
1008 //
1009 // Define the pin for the LED
1010 #define LED_EXT 14
1011 // Use the onboard push button (BUTTON_B)
1012 #define BUTTON_PIN BUTTON_B
1013 //
1014 // Initialize variables
1015 // Flag, whether the button is pressed
1016 // Declare as volatile for interrupt safety
1017 volatile bool pushPressed = false;
1018 // LED>Status zur Verarbeitung
1019 int ledState = 0;
1020
1021 void setup() {
1022     // Initialize the pin as an output
1023     pinMode(LED_EXT, OUTPUT);
1024     // Initialize the pin as an input
1025     pinMode(BUTTON_PIN, INPUT_PULLUP);
1026     // Initialize the interrupt function
1027     attachInterrupt(digitalPinToInterrupt(BUTTON_PIN), buttonPressed,
1028                     FALLING);
1029 }
1030
1031 // Interrupt function
1032 // as short as possible
1033 void buttonPressed()
1034 {
1035     if (pushPressed == false)
1036     {
1037         pushPressed = true;
1038     }
1039 }
1040
1041 void loop()
1042 {
1043     if (pushPressed)
1044     {
1045         // Turn the LED on
1046     }
1047 }
```

```
1046     digitalWrite(LED_EXT, HIGH);  
1047     // Wait for one second  
1048     delay(2000);  
1049     // Turn the LED off  
1050     digitalWrite(LED_EXT, LOW);  
1051     pushPressed = false;  
1052 }
```

..../Code/Nano33BLESense/Test/TestPushButtonInterrupt.ino

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

13.8. Further Readings

WS:interrupts, external
buttons

14. Sensormodul APDS9960 for Gesture, Proximity, and Color Detection

14.1. Sensormodul APDS-9960

Das Modul APDS-9960 ist ein Produkt der Firma Broadcom, ehemals Avago Technologies, mit Sitz in San José, Kalifornien. Es handelt sich hierbei um einen Sensor für die Gesten-, Abstands-, Umgebungslicht- und Farberkennung [Ava15]. Ein sehr bekannter Anwendungsfall für diesen Sensor ist das automatische Ausschalten des Smartphone-Displays beim Anlegen des Smartphones an ein Ohr. Hier erkennt der Sensor mithilfe des Abstandssensors, dass etwas auf oder am Bildschirm anliegt. Hier findet die detaillierte Betrachtung des Sensors in Bezug auf die Anwendung als Farbsensor statt.

Bei der Farberkennung misst der Sensor die Intensität der roten, grünen und blauen Farbe und gibt für jeden Farbkanal einen 16-Bit-Wert zurück. Insgesamt ergibt sich somit auf einen Wertebereich von 0 bis 65535 für eine Farbe.

Zum Nachvollziehen eines Farbsensors kann das menschliche Auge betrachtet werden. Der Mensch nimmt seine Umwelt optisch über Rezeptoren, die Stäbchen und Zapfen, wahr. Hierbei sind “die Stäbchen für den Hell-Dunkel-Kontrast und die Zapfen für die Farberkennung” zuständig [HSH17]. Entscheidend sind die Primärfarben Rot, Grün und Blau. Jede weitere Farbe kann aus diesen drei Farben gemischt werden. Aus Farstabellen lassen sich dann die einzelnen Werte für R-G-B, für die verschiedenen Farben wie beispielsweise Orange, Violett oder Türkis, entnehmen.

Um den APDS-9960 betreiben zu können wird eine Spannung von 3,0 V empfohlen. Ferner sollte der Sensor zwischen -30°C und 85°C betrieben werden.

WS:cite books,
applications, board

14.2. General

General description: Farberkennung

WS:cite books

14.3. Specific Sensor

14.4. Specification

WS:cite board

- cite data sheet

- Circuit Diagram

14.5. Bibliothek

Im Fall des Sensor APDS-9960 bietet Arduino eine Bibliothek [Arduino_APDS9960](#) für das Aufrufen der Farberkennung, des Abstandes oder der Gestenerkennung.

Die Bibliothek für den Sensor ist die Bibliothek [Arduino_APDS9960](#). Diese erlaubt es, Gesten, Farben, Lichtintensität und Abstände mit dem Sensor zu messen. Dabei findet die Kommunikation zwischen dem Chip des Arduino Nano 33 BLE Sense Lite und dem Modul APDS-9960 über eine interne I²C-Schnittstelle statt [Ava15].

Die Bibliothek wird durch den Befehl `#include <Arduino_APDS9960.h>` eingebunden.

Die aktuelle Version ist 1.0.4 (Stand: 17.04.2024).

14.5.1. Description

Die Bibliothek [Arduino_APDS9960](#) stellt eine Klasse `APDS` zur Verfügung. Mit Hilfe dieser Klasse können auf alle Funktionen zugegriffen werden. [Ava15].

The sensor APDS-9960 has the functionality of proximity distance measure, RGB color detection, and Gesture detection too. Go to example, check the library APDS-9960 and upload the complete program as shown in the figure 14.1. Full example includes the RGB color detection code, gesture detection code, and also proximity distance measure code.

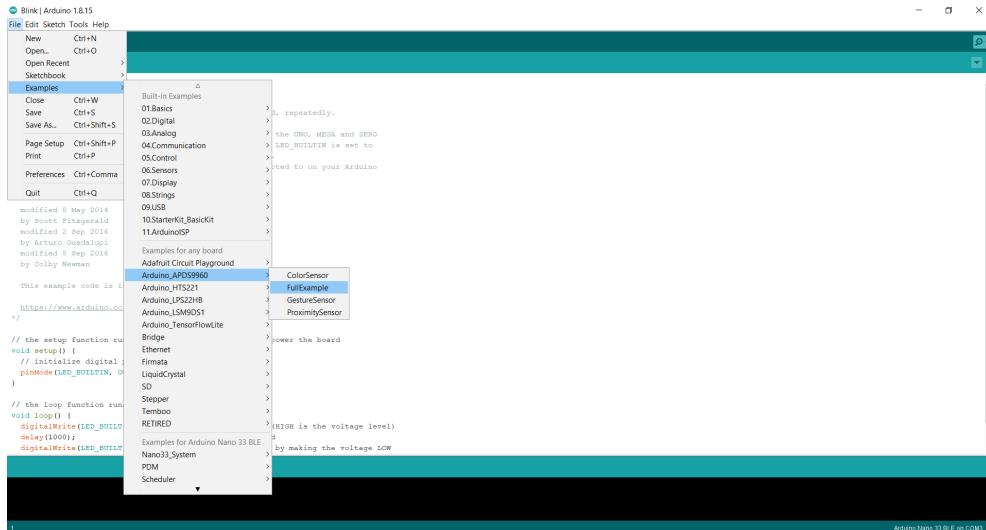


Figure 14.1.: APDS-9960 Gesture, Proximity, Color Sensor

14.5.2. Installation

[Tools](#) > [Manage libraries ...](#)

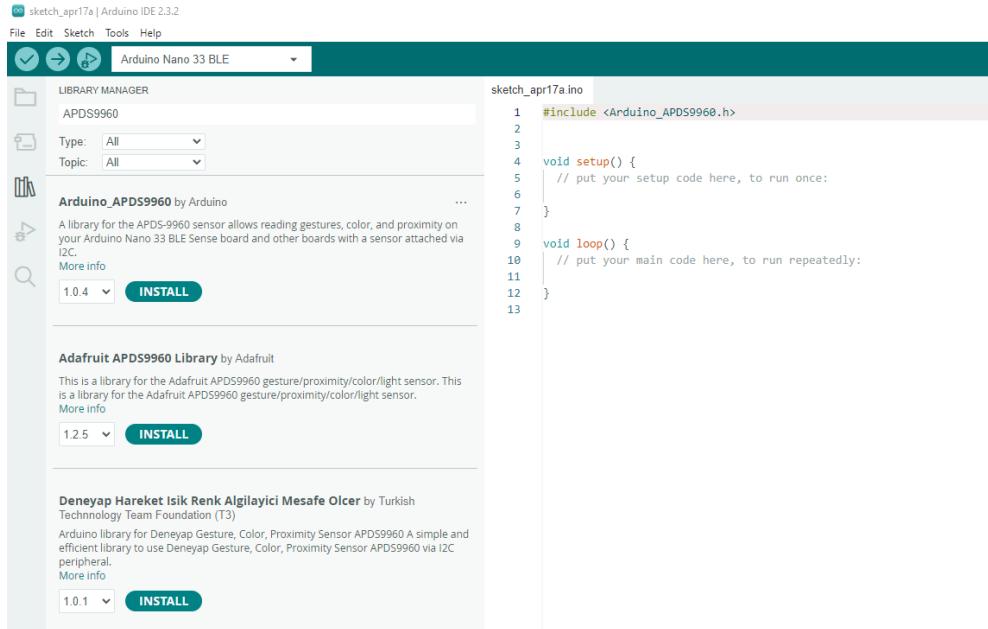


Figure 14.2.: APDS-9960 Gesture, Proximity, Color Sensor

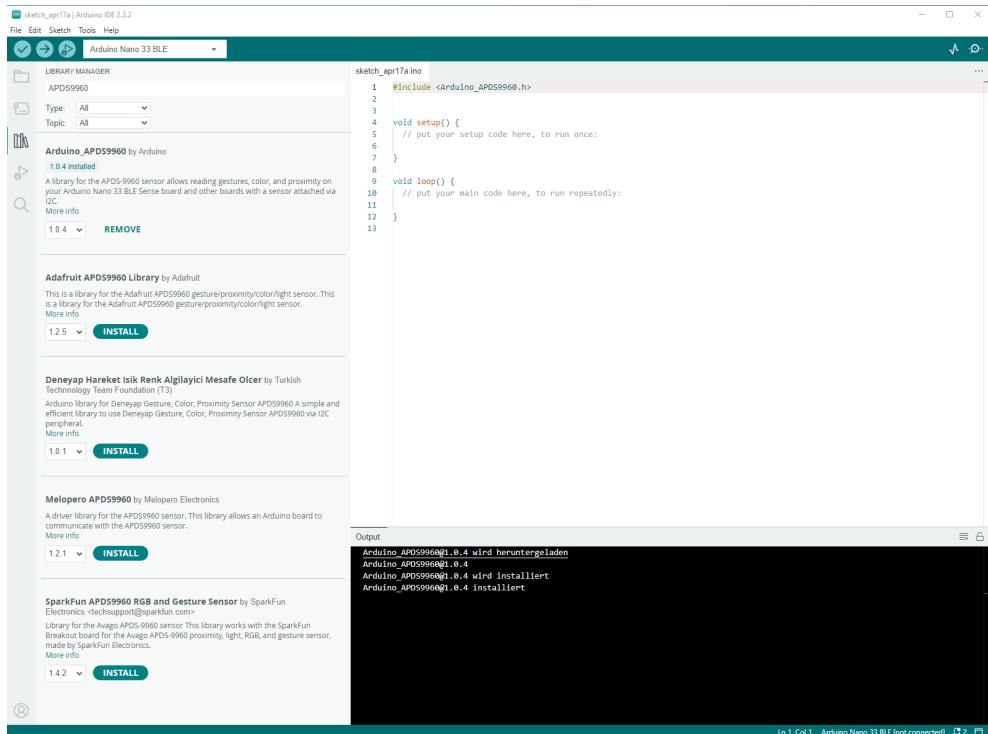


Figure 14.3.: APDS-9960 Gesture, Proximity, Color Sensor

14.5.3. Functions

- **`begin()`**: Die Methode `begin()` aktiviert und initialisiert den Sensor APDS9960. Die Methode wird in der Initialisierung `setup{}` aufgerufen.
 - Rückgabewert `TRUE`: Die Initialisierung ist erfolgreich gewesen.
 - Rückgabewert `FALSE`: Die Initialisierung war nicht erfolgreich.

- `end()`: Die Methode `end()` deaktiviert den Sensor APDS9960.
- `colorAvailable()`: Die Methode überprüft, ob eine Farbinformation abgerufen werden kann.
 - Rückgabewert `TRUE`: Es liegt eine Farbinformation vor.
 - Rückgabewert `FALSE`: Es liegt keine Farbinformation vor.

- `readColor(...)`: Die Methode ermittelt die Farbwerte des Sensors.

Die Methode bestimmt entweder nur die Farbwerte oder die Farbwerte und die Intensität des Umgebungslichts.

Aufruf zur Bestimmung der Farbwerte:

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

Die Variable `r`, `g` und `b` enthalten nach dem Aufruf die aktualisierten Farbwerte. Der Wertebereich ist 0, 1, 2, ..., 255.

Aufruf zur Bestimmung der Farbwerte und der Intensität des Umgebungslichts:

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

Die Variable `r`, `g` und `b` enthalten nach dem Aufruf die aktualisierten Farbwerte. Die Variable `a` enthält die Intensität des Umgebungslichts. Der Wertebereich ist 0, 1, 2, ..., 255.

- `proximityAvailable()`: Die Methode überprüft, ob eine Abstandsinformation abgerufen werden kann.

- Rückgabewert `TRUE`: Es liegt eine Abstandsinformation vor.
- Rückgabewert `FALSE`: Es liegt keine Abstandsinformation vor.

- `readProximity()`: Die Abstandsinformation wird ausgelesen.

- Rückgabewert `int proximity = APDS.int proximity()`: Der Abstand wird als Rückgabewert zurückgegeben.
- Rückgabewert `-1`: Ein Abstand konnte nicht ermittelt werden.

- `gestureAvailable()`: Die Methode überprüft, ob eine Geste erkannt wurde.

- Rückgabewert `TRUE`: Es liegt eine Geste vor.
- Rückgabewert `FALSE`: Es liegt keine Geste vor.

- `setGestureSensitivity()`: Die Gestenerkennung hängt unter anderem von den Lichtverhältnissen, der Geschwindigkeit und des Abstands der Bewegung ab. Hier kann die Empfindlichkeit der Erkennung eingestellt werden. Ein höherer Wert erkennt mehr Gesten, aber die Fehlerwahrscheinlichkeit für das Erkennen einer Geste, die keine gewesen ist, ist höher. Bei einem niedrigen Wert werden vollzogene Gesten eher nicht erkannt.

Der Wertebereich ist `1` bis `100`.

Der voreingestellte Wert ist `80`.

- `readGesture()`: Falls eine erkannte Geste vorliegt, kann diese mit der Methode `readGesture()` abgerufen werden. Dabei stehen 5 mögliche Rückgabewerte zur Verfügung:
 - `GESTURE_UP`: Eine Bewegung von unten nach oben wurde erkannt.
 - `GESTURE_DOWN`: Eine Bewegung von oben nach unten wurde erkannt.
 - `GESTURE_LEFT`: Eine Bewegung von rechts nach links wurde erkannt.
 - `GESTURE_RIGHT`: Eine Bewegung von links nach rechts wurde erkannt.
 - `GESTURE_NONE`: Keine der vorherigen Bewegungen wurde erkannt.
- `setInterruptPin()`: Die Messung wird durch das Setzen eines Eingangs ausgelöst. Dieses Eingang wird in der Regel automatisch erkannt. Mit dieser Funktion kann die Nummer des Pins gesetzt werden.
 Falls der Wert `-1` verwendet wird, dann ist kein Pin verbunden.
 Falls der Wert `0` oder höher ist, so wird dieser Pin verwendet.
 Der Standardwert hängt vom Board ab.
- `setLEDBoost(...)`: In dem Sensor ist eine Infrarot-LED integriert. Die LED kann kurzfristig mehr Leistung um heller zu sein. Es kann bis zum 3-fachen der Nennleistung eingestellt werden. Dies kann hier gestartet werden.

Aufruf:

- `0`: This sets boost to 100% (this is the default power value).
- `1`: This sets boost to 150%.
- `2`: This sets boost to 200%.
- `3`: This sets boost to 300%.

Returns:

- `0`: failure
- `1`: success

14.6. Simple Examples

14.6.1. Example “Color Detection”

For this example, the task is to detect the Red, Green, Blue (RGB) color. Here, an Arduino Nano 33 BLE Sense can be used, which have on-board sensor APDS9960.

14.6.2. Example “Color Detection” - Manual

The sensor is initialized in the function `setup()`. In the function `loop()`, it is checked if colors are measured. If true, the values of each channel is printed in the monitor.

14.6.3. Example “Color Detection” - Code

Listing 14.1.: Simple sketch using the sensor APDS9960 for colors

```

1000 // Test of the sensor APDS9960
1001 // This sketch tests the color sensor
1002 //
1003 // filename: TestAPDS9960Color.ino
1004
1005 // Library for APDS9960 sensor
1006 #include <Arduino_APDS9960.h>
1007
1008 int r; // Measured value for red
1009 int g; // Measured value for green
1010 int b; // Measured value for blue
1011
1012 void setup()
1013 {
1014     Serial.begin(9600);
1015
1016     // initialize the sensor
1017     if (!APDS.begin())
1018         Serial.println("Error initializing APDS9960 sensor.");
1019 }
1020
1021 void loop()
1022 {
1023
1024     // check whether color detection is available
1025     if (APDS.colorAvailable())
1026     {
1027         // read the measured values
1028         APDS.readColor(r, g, b);
1029         // print the measured values
1030         Serial.println("Red = " ,r, " , Green = " ,g, " , BLUE = " , b );
1031     }
1032 }
1033
1034 }
```

..//Code/Nano33BLESense/Test/TestAPDS9960Color.ino

14.6.4. Example “Color Detection” - Files

There is only one file [TestAPDS9960Color.ino](#).

14.6.5. Example “Proximity”

14.6.6. Example “Proximity” - Manual

14.6.7. Example “Proximity” - Code

Listing 14.2.: Simple sketch using the sensor APDS9960 for measuring the proximity

```

1000 // Test of the sensor APDS9960
1001 // This sketch tests the proximity sensor
1002 //
1003 // filename: TestAPDS9960Proximity.ino
1004
1005 // Library for APDS9960 sensor
1006 #include <Arduino_APDS9960.h>
1007
1008 int proximity; // Measured value for the distance
1009
1010 void setup()
1011 {
1012 }
```

```

1014     Serial.begin(9600);
1016     // initialize the sensor
1017     if (!APDS.begin()) {
1018         Serial.println("Error initializing APDS9960 sensor.");
1019     }
1020
1021 void loop()
1022 {
1023     // check whether distance measurement is available, if not 5 ms
1024     if (APDS.proximityAvailable())
1025     {
1026         // read the Measured value
1027         proximity = APDS.readProximity();
1028         // print the measured value
1029         if (proximity != -1) {
1030             Serial.print("Distance: ");
1031             Serial.println(proximity);
1032         }
1033     }
1034 }
```

..//Code/Nano33BLESense/Test/TestAPDS9960Proximity.ino

14.6.8. Example “Proximity” - Files

14.6.9. Example “Gesture Detection”

14.6.10. Example “Gesture Detection” - Manual

14.6.11. Example “Gesture Detection” - Code

Listing 14.3.: Simple sketch using the sensor APDS9960 for gesture detection

```

1000 // Test of the seonsor APDS9960
1001 // This sketch tests the gesture sensor
1002 /**
1003  * filename: TestAPFD9960Gesture.ino
1004
1005 // Library for APDS9960 sensor
1006 #include <Arduino_APDS9960.h>
1007
1008 int proximity; // Measured value for the distance
1009
1010 void setup()
1011 {
1012     Serial.begin(9600);
1013
1014     // initialize the sensor
1015     if (!APDS.begin()) {
1016         Serial.println("Error initializing APDS9960 sensor.");
1017     }
1018 }
1019
1020 void loop()
1021 {
1022     // check whether distance measurement is available, if not 5 ms
1023     if (APDS.proximityAvailable())
1024     {
1025         // read the Measured value
1026         proximity = APDS.readProximity();
1027         // print the measured value
1028         if (proximity != -1) {
```

```

1030     Serial.print("Distance: ");
1031     Serial.println(proximity);
1032   }
1033 }

```

..//Code/Nano33BLESense/Test/TestAPDS9960Gesture.ino

14.6.12. Example “Gesture Detection” - Files

14.7. Calibration

14.7.1. Calibration “Color Detection”

14.7.2. ~~WS~~^{color method} Calibration “Proximity”

14.7.3. ~~WS~~^{gesture method} Calibration “Gesture Detection”

14.8. ~~WS~~^{color method} Tests

14.8.1. Simple Function Test

14.8.2. Test all Functions

14.9. Simple Application

Similarly, by following all the steps for uploading and compiling the program we can see the results of Sensor APDS-9960 on serial monitor too. For seeing the different output, we can change the input for the sensor too, e.g: for color detection we can switch the colors, for gesture detection we can also switch the gestures, and for proximity also do the same. The resulted output as shown in the figure. 14.4

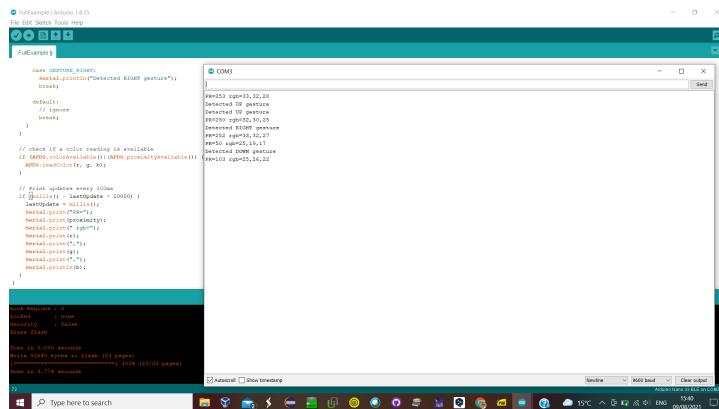


Figure 14.4.: Gesture, Proximity, Color Sensor Output Window

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

Listing 14.4.: Simple sketch using the sensor APDS9960

```

1000 // Library for sensor APDS9960
#include <Arduino_APDS9960.h>

```

```
1002 // Library SSD1306 for text-only output and connection
1004 // with cable via the I2C protocol
1006 #include <SSD1306AsciiWire.h>
1008 // Enter the address of the OLED
1010 #define I2C_ADDRESS 0x3C
1012 // Set the name of the OLED
1014 SSD1306AsciiWire oled;
1016
1018 void setup()
{
    Serial.begin(9600);
1020
1022 // Define pin as INPUT and activate internal pull-up resistor
1024 pinMode(TasterPin, INPUT);
// Define pin as OUTPUT
1026 pinMode(LedPin, OUTPUT);
1028
1028 if (!APDS.begin())
    Serial.println("Error initializing APDS9960 sensor.");
1030 }
1032
1032 // Arduino is hereby registered in the I2C bus,
1034 // as it is to be registered as a master,
1035 // the address SEARCH SOURCE!!!! is omitted.
1036 Wire.begin();
1037 // Clock frequency in Hertz,
1038 // Standard: 100,000
1039 // Fast mode: 400,000
1040 Wire.setClock(400000L);
1041 // first screen size, then reference to I2C address
1042 oled.begin(&Adafruit128x64, I2C_ADDRESS);
1043 oled.setFont(System5x7);
1044 oled.setCursor(0, 40);
1045 oled.println("INITIALISIERUNG!\n");
1046
1048 // External LED flashes 3 times during initialization
1049 for (int zaehler = 1; zaehler < 4; zaehler = zaehler + 1)
1050 {
1051     delay(2000);
1052     digitalWrite(LedPin, HIGH);
1053     delay(200);
1054     digitalWrite(LedPin, LOW);
1055     delay(200);
1056 }
1057 oled.clear();
1058 }
1060
1062 void loop()
{
    oled.println("READY!");
1064
1065 // check whether color detection is available,
1066 // if not wait 5 ms and repeat query
1067 while (!APDS.colorAvailable()) {
1068     delay(5);
1069 }
1070
```

```

1072 // Check whether distance measurement is available,
1073 // if not wait 5 ms and repeat query
1074 while (!APDS.proximityAvailable()) {
1075     delay(5);
1076 }
1077
1078 // Create integer variables to save
1079 // the measured values for red, green and blue
1080 int r, g, b, tasterCount{ 0 };
1081
1082 // Status query of the measurement button
1083 bool zustandTaster = digitalRead(TasterPin);
1084
1085 if (zustandTaster == 0) {
1086     tasterCount = 1;
1087 }
1088
1089 // if clause switch to measuring mode when the button is pressed
1090 if (tasterCount == 1)
1091 {
1092     int proximity = APDS.readProximity();
1093
1094     if (proximity != -1) {
1095         Serial.print("Abstand: ");
1096         Serial.println(proximity);
1097
1098         if (proximity < 20) {
1099             oled.clear();
1100             digitalWrite(LedPin, HIGH);
1101
1102             // 3x Color scan and save in the 3 variables r, g and b
1103             for (int i = 0; i < 3; i++) {
1104
1105                 APDS.readColor(r, g, b);
1106                 delay(1500);
1107             }
1108
1109             // Display largest value via integrated RGB LED,
1110             // option for troubleshooting
1111             if (r > g & r > b) {
1112                 digitalWrite(LEDRed, LOW);
1113                 digitalWrite(LEDGreen, HIGH);
1114                 digitalWrite(LEDBlue, HIGH);
1115             } else if (g > r & g > b) {
1116                 digitalWrite(LEDGreen, LOW);
1117                 digitalWrite(LEDRed, HIGH);
1118                 digitalWrite(LEDBlue, HIGH);
1119             } else if (b > g & b > r) {
1120                 digitalWrite(LEDBlue, LOW);
1121                 digitalWrite(LEDRed, HIGH);
1122                 digitalWrite(LEDGreen, HIGH);
1123             } else {
1124                 digitalWrite(LEDRed, HIGH);
1125                 digitalWrite(LEDGreen, HIGH);
1126                 digitalWrite(LEDBlue, HIGH);
1127             }
1128         }
1129
1130         // Output values in the serial monitor
1131         // for testing and error analysis
1132         Serial.print("Red light = ");
1133         Serial.println(r);
1134         Serial.print("Green light = ");
1135         Serial.println(g);
1136         Serial.print("Blue light = ");
1137         Serial.println(b);
1138         Serial.println();

```

```

1140
1142     if (r > g & r > b) {
1143         oled.println("Das Objekt ist rot!");
1144     } else if (g > r & g > b) {
1145         oled.println("Das Objekt ist gruen!");
1146     } else if (b > g & b > r) {
1147         oled.println("Das Objekt ist blau!");
1148     }
1149
1150     delay(4000);
1151     oled.clear();
1152 }
1153
1154 else {
1155     oled.clear();
1156     oled.println("Kein Objekt \nvorhanden!\n");
1157     oled.println("Halten Sie ein\nObjekt vor den \nSensor \noder");
1158     veraendern Sie\ndie Position!");
1159     digitalWrite(LED_R, HIGH);
1160     digitalWrite(LED_G, HIGH);
1161     digitalWrite(LED_B, HIGH);
1162     delay(4000);
1163     oled.clear();
1164 }
1165 } else {
1166     oled.println("Abstandsmessung fehlgeschlagen!");
1167 }
1168 }
1169
1170 else {
1171     // Switch off LED if not pressed
1172     digitalWrite(LedPin, LOW);
1173
1174     oled.setFont(System5x7);
1175     oled.setCursor(0, 40);
1176     oled.println("READY!");
1177
1178     digitalWrite(LED_R, HIGH);
1179     digitalWrite(LED_G, HIGH);
1180     digitalWrite(LED_B, HIGH);
1181 }
1182
1183 // Set button to 0
1184 zustandTaster = 0;
1185
1186 }

```

.. /Code/Nano33BLESense/Test/TestAPDS9960.ino

14.10. Further Readings

15. Mikrophone MP34DT05-A

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

Some Arduino boards have a microphone on board which can be used for the application. Das Mikrofon des Arduino Nano 33 BLE Sense ist ein ultrakompaktes Modul, das den MP34DT05-Sensor verwendet. Dieser Sensor nutzt die Puls-Dichtemodulation (PDM), um ein analoges Signal in ein binäres Signal umzuwandeln.

Der MP34DT05 ist ein winziges Mikrofon, das speziell für den Arduino Nano 33 BLE Sense entwickelt wurde. Das Mikrofon verwendet PDM, um Schall in digitale Daten umzuwandeln. Durch PDM wird ein analoges Signal in binäre Werte umgewandelt. Der Sensor hat folgende Spezifikationen:

- Signal-Rausch-Verhältnis: 64 dB
- Empfindlichkeit: -26 dBFS ± 3 dB
- Temperaturbereich: -40 bis 85 °C

Mikrofone werden in Mobilgeräten, Spracherkennungssystemen und sogar in Gaming- und Virtual-Reality-Eingabegeräten eingesetzt. Mit dem eingebetteten MP34DT05-Sensor können Sie die Schallwerte Ihrer Umgebung messen und anzeigen.

15.1. Puls-Dichtemodulation

Pulsdichtemodulation (PDM) ist eine faszinierende Technik, die im Arduino Nano 33 BLE Sense verwendet wird, um Audiosignale zu digitalisieren. PDM wandelt analoge Audiosignale in digitale Daten um, indem es die Dichte der Impulse misst. Statt kontinuierlicher Werte verwendet PDM nur zwei Zustände: 1 (Impuls vorhanden) oder 0 (kein Impuls). Um das Mikrofon zu verwenden, steht die Bibliothek [pdm.h](#) zur Verfügung. PDM arbeitet mit einer hohen Samplingrate, um genaue Audioinformationen zu erfassen. Die Impulse werden als Bitstream übertragen, der später in Audiodaten umgewandelt wird.

Die Puls-Dichtemodulation (PDM) ist eine interessante Technik, die sowohl Vorteile als auch Nachteile aufweist.

Die Puls-Dichtemodulation (PDM) hat folgende Vorteile:

- Einfachheit: PDM ist weniger komplex als einige andere Modulationsverfahren. Die Implementierung von Sendern und Empfängern für PDM ist vergleichsweise unkompliziert.
- Robustheit gegenüber Störungen: PDM ist widerstandsfähig gegenüber Rauschen und Interferenzen.

- Hohe Samplingrate: PDM arbeitet mit einer hohen Abtastrate, um genaue Audioinformationen zu erfassen.
- Geringe Verluste: Im Vergleich zu analogen Steuerungen verursacht PDM weniger Verluste, da es die Versorgungsspannung schnell ein- und ausschaltet.

Die Nachteile der Puls-Dichtemodulation (PDM) sind:

- Große Lasten: Die Steuerung großer Verbraucher erfordert hohe Spannungen und Ströme. Standard-Anologschaltungen und DACs sind hierbei weniger effizient.
- Wärmeentwicklung: Leistungstransistoren, die in PDM-Schaltungen verwendet werden, erzeugen Wärme und Verluste.

Insgesamt ist PDM eine leistungsstarke Technik, die in verschiedenen Anwendungen wie Spracherkennung, Klanganalyse und Datenübertragung eingesetzt wird. Es ist wichtig, die Vor- und Nachteile je nach Kontext zu berücksichtigen

WS:citations

15.2. Bibliothek `pdm.h`

Die PDM-Bibliothek ermöglicht die einfache Verwendung des integrierten Mikrofons und ist auch über die Bibliothek `ArduinoSound` zugänglich.

Die PDM-Bibliothek für den Arduino Nano 33 BLE Sense ermöglicht die Verwendung von Puls-Dichtemodulation (PDM)-Mikrofonen, wie dem integrierten MP34DT05 auf dem Board. Hier sind die wichtigsten Funktionen, die diese Bibliothek bereitstellt:

`begin()` : Diese Funktion initialisiert die PDM-Schnittstelle. Sie nimmt zwei Parameter entgegen:

Parameter `channels` : Die Anzahl der Kanäle (1 für Mono, 2 für Stereo).

Parameter `sampleRate` : Die gewünschte Abtastrate in Hertz.

Beispiel:

```
1000 if (!PDM.begin(1, 16000)) {
1001     Serial.println("Fehler beim Starten der PDM!");
1002     while (1);
1003 }
```

`end()` : Mit dieser Funktion wird die PDM-Schnittstelle deinitialisiert:

```
1000 PDM.end();
```

`available()` : Diese Funktion gibt die Anzahl der verfügbaren Bytes im PDM-Empfangspuffer zurück. Das sind bereits eingetroffene Daten, die darauf warten, gelesen zu werden:

```
1000 int bytesAvailable = PDM.available();
```

read() : Mit dieser Funktion liest man Daten aus dem PDM in einen angegebenen Puffer:

```
1000 short sampleBuffer[256]; // Puffer fuer Samples (16-Bit)
  int bytesRead = PDM.read(sampleBuffer, bytesAvailable);
```

onReceive() : Diese Funktion setzt die Callback-Funktion, die aufgerufen wird, wenn neue PDM-Daten zum Lesen bereit sind:

```
1000 void onPDMdata() {
  int bytesAvailable = PDM.available();
  // Weitere Verarbeitung der Daten...
}
1004 PDM.onReceive(onPDMdata);
```

15.3. General Information

15.4. Decibels

In electronics one often wants to represent the ratio of two signals on a logarithmic scale. For this we use the decibel, dB . If we have two powers P_1 and P_2

$$dB = 10 \log_{10} \left(\frac{P_2}{P_1} \right)$$

or equivalently – when comparing two signals with the same kind of waveform –

$$dB = 20 \log_{10} \left(\frac{V_2}{VP_1} \right).$$

Note dB is a relative unit of measurement, i.e. *This amplifier has a gain of 10 dB*. dB can be positive or negative. For example, $+10dB$ corresponds to P_2 greater than P_1 by a factor of 10, and $-3dB$ corresponds to P_2 less than P_1 by approximately a factor of 2.

For an absolute measure on a logarithmic scale there are a variety of other units. A common one is dBm .

$$dBm = 10 \log_{10}(P[mW])$$

Zero dBm corresponds to $1mW$ of power. And in a 50Ω system $0dBm$ corresponds to $220mVrms$.

15.5. Specific Sensor

15.6. Specification

15.7. Simple Code

Der Code verwendet die folgenden Bibliotheken:

- [PDM.h](#): Diese Bibliothek ermöglicht die Kommunikation mit dem Mikrofon zur Aufnahme von PDM-Signalen [Arde]

Listing 15.1.: Simple sketch to control the built-in LED

```

1000 // How to control the built-in LED of the Arduino Nano 33 BLE Sense
1001 //
1002 // file: TestLEDBuiltin.ino
1003 //
1004 // The LED is switched on for 1 second and switched off
1005 // for 1 second so that the LED flashes accordingly.
1006 //
1007 // Define the pin for the built-in LED
1008 #define LED_BUILTIN 13

1010 void setup() {
1011     // Initialize the pin as an output
1012     pinMode(LED_BUILTIN, OUTPUT);
1013 }

1014 void loop() {
1015     // Turn the LED on
1016     digitalWrite(LED_BUILTIN, HIGH);
1017     // Wait for one second
1018     delay(1000);
1019     // Turn the LED off
1020     digitalWrite(LED_BUILTIN, LOW);
1021     // Wait for one second
1022     delay(1000);
1023 }

```

..../Code/Nano33BLESense/Test/TestLEDBuiltin.ino

15.8. Tests

15.8.1. Simple Function Test

15.8.2. Test all Functions

15.9. Simple Application

15.9.1. Pin-Konfiguration

Die verwendeten Pins werden zu Beginn des Codes definiert:

- `microphoneDataPin`: Der Datenpin für das Mikrofon
- `microphoneClockPin`: Der Clock-Pin für das Mikrofon

15.9.2. Initialisierung und Setup

In der Funktion `setup()` werden die erforderlichen Initialisierungen durchgeführt:

- `PDM.onReceive(onPDMdata)`: Registriert den Empfang des Mikrofonsignals als PDM-Signal

15.9.3. Messungssteuerung

Die Funktion `onPDMdata()` wird aufgerufen, wenn Daten vom Mikrofon verfügbar sind. Dies liest die Daten in einen Array ein und zählt die Anzahl der gelesenen Samples:

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

```

61 void onPDMdata() {
62     int bytesAvailable = PDM.available();
63     PDM.read(sampleBuffer, bytesAvailable);
64     samplesRead = bytesAvailable / 2;
65 }

```

Figure 15.1.: Code Einlesen/Zählen

```

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

```

Figure 15.2.: Code Messungssteuerung

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

- Wenn der blaue Taster gedrückt wird, wird die Messung gestartet, indem `isMeasuring` auf `true` gesetzt wird und die Funktion `StartSampling()` aufgerufen wird.
- Wenn der rote Taster gedrückt wird, wird die Messung beendet, indem `isMeasuring` auf `false` gesetzt wird und die Funktion `StopSampling()` aufgerufen wird. Je nach vorherigem Zustand des roten Tasters wird entweder der maximale Wert SPL in dB angezeigt (`ShowMaxdBspl()`) oder der durchschnittliche dB SPL-Wert (`ShowAveragedBspl()`).
- `isDelayOver` wird auf `true` gesetzt, wenn die Startverzögerung (definiert durch `measureThresholdMilliseconds`) von 1200ms abgelaufen ist.
- Es gibt eine kurze Verzögerung (`delay(50)`) zwischen den Schleifendurchläufen, um Tastenprellungen zu vermeiden.

15.9.4. Mikrofondatenverarbeitung

Die Funktion `StartSampling()` initialisiert die Datenverarbeitung:

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

15.9.5. Anzeige der Ergebnisse

Die Funktionen `ShowResult()` und `ShowMicrophoneValues()` sind für die Anzeige der Messergebnisse auf dem OLED-Display verantwortlich. `ShowResult()` zeigt den aktuellen SPL-Wert in dB auf dem Display an und aktualisiert den maximalen Wert SPL in dB, wenn nach der Startverzögerung ein neuer Höchstwert erreicht wird. `ShowMicrophoneValues()` verarbeitet die vom Mikrofon empfangenen Signale. Der maximale Samplewert wird ermittelt und verwendet, um den Wert SPL in dB zu berechnen. Die Funktion berechnet auch einen Durchschnittswert über eine bestimmte Zeit, `averagingTime = 200ms`, und zeigt das Ergebnis auf dem Display an. Die Funktionen `ShowMaxdBspl()` und `ShowAveragedBspl()` zeigen den maximalen bzw. den durchschnittlichen Wert SPL in dB auf dem Display an.

15.9.6. Umrechnung auf den Wert dB SPL

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

`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 „ $20 * \log10(\text{Voltage} / \text{Vrms}) + \text{sensitivity}$ “ berechnet [Quelle der Formel: PDF Decibels Formula https://physicscourses.colorado.edu/phys3330/phys3330_sp19/resources/Decibels_Phys_3330.pdf University of Colorado System], wobei „Voltage“ der berechnete „maxSampleVoltage“ ist, und „sensitivity“ die Empfindlichkeit des Mikrofons in dBV (Dezibel-Volt) ist. „Vrms“ ist die RMS-Spannung (Root Mean Square), die einem Schalldruckpegel von 94 dB SPL entspricht. Dieser Wert wurde experimentell ermittelt.

15.9.7. Benutzeroberfläche

Die Funktion „HandleUI()“ aktualisiert OLED-Display Anzeige:

Wenn eine Messung aktiv ist („isMeasuring == true“), werden die aktuellen Messwerte auf dem Display angezeigt. Andernfalls wird der Displayinhalt aktualisiert, ohne neue Werte anzuzeigen.

15.10. Further Readings

WS:citations

```

72 void HandleInput() {
73     bool blueButtonIsPressed = analogRead(blueButtonPin) == LOW;
74     bool redButtonIsPressed = analogRead(redButtonPin) == LOW;
75
76     if (blueButtonIsPressed)
77         isMeasuring = true;
78
79     if (redButtonIsPressed)
80         isMeasuring = false;
81
82     bool doNextStep = redButtonIsPressed || blueButtonIsPressed;
83     if (doNextStep)
84     {
85         if (isMeasuring)
86         {
87             StartSampling();
88             absoluteSum = 0.0;
89             absoluteCount = 0;
90             maxdBspl = 0.0; // Zurücksetzen des höchsten dB SPL Werts bei Start einer neuen Messung
91             isDelayOver = false; // Startverzögerung zurücksetzen
92             startTime = millis(); // Startzeit erfassen
93         }
94     else
95     {
96         StopSampling();
97         if (redButtonResult == 0)
98         {
99             ShowMaxdBspl(); // Anzeigen des höchsten dB SPL Werts am Ende der Messung
100            redButtonResult = 1;
101        }
102    else
103    {
104        ShowAveragedBspl();
105        redButtonResult = 0;
106    }
107 }
108 }
109
110 delay(50);

```

Figure 15.3.: Code Überwachung

```

119 void StartSampling() {
120     if (!PDM.begin(1, 16000)) {
121         Serial.println("Failed to start Measurement!");
122         while (1);
123     }
124 }
125
126 void StopSampling() {
127     PDM.end();
128 }

```

Figure 15.4.: Code Initialisierung der Datenverarbeitung

```
148 float getDbValueFromPMC(int pmcValue) {  
149     float maxSampleVoltage = maxSampleValue * Vref / 32767.0;  
150     float dBspl = 20.0 * log10(maxSampleVoltage / Vrms) + sensitivity;  
151     return dBspl;  
152 }
```

Figure 15.5.: Code Umrechnung Mikrofonsignal

```
216 void HandleUI() {  
217     if (isMeasuring) {  
218         ShowMicrophoneValues();  
219     } else {  
220         //display.clearDisplay();  
221         display.display();  
222     }  
223 }
```

Figure 15.6.: Code OLED-Aktualisierung

16. Test of the Hardware

Die Kapitel 16 und 17 müssen neu gestaltet werden:

- Einen Test der schnell zeigt, ob die Hardware funktioniert.
- Ausführliche Tests der einzelnen Sensoren
- einen gesamten Test für alles

16.1. General Tests

All the Arduino boards need power to operate, either it comes from the USB connection with Laptop, Ac power adapter, Battery or a regulated power supply. The most easiest way to operate arduino board is USB connection with laptop, normally these boards need 5V direct current (DC) to operate. Arduino Nano 33 BLE sense also need these types of power sources for functionality, when applying one of the above mention power source the green LED glows as shown in the figure 16.1, it shows the sign of Arduino board working.



Figure 16.1.: Power On Arduino Nano 33 BLE Sense

16.2. Testing the On-Board Sensor

Arduino Nano 33 BLE sense have a set of on-board sensor embed on it. These sensor also start working when arduino board operate. These are the following set of sensors available in Arduino Nano 33 BLE Sense board.

- The IMU is a LSM9DS1 and it is managed through I2C.
- The LPS22HB reads barometric pressure and environmental temperature.
- The HTS221 senses relative humidity.
- The ADPS-9960 is a digital proximity, ambient light, RGB and gesture sensor.
- The MP34DT05 is the digital microphone.

16.2.1. LSM9DS1 (Accelerometer, Gyroscope, and Magnetometre Sensor)

The 9-axis inertial measurement unit (IMU) sensor is work as a accelerometer, gyroscope, and magnetic sensor. It measures 3D linear acceleration, 3D angular Velocity and 3D magnetic field aroud the sensor. This 9-axis IMU sensor use to measure Position, vibration, orientation and magnetic field around the sensor. To check the functionality of this sensor there is avialable library in the example section as shown in the figure. 16.2 Go to examples check LSM9DS1 sensor, it shows all the functionality of this sensor i.e: (Accelerometer, Gyroscope, and Magnetometer).

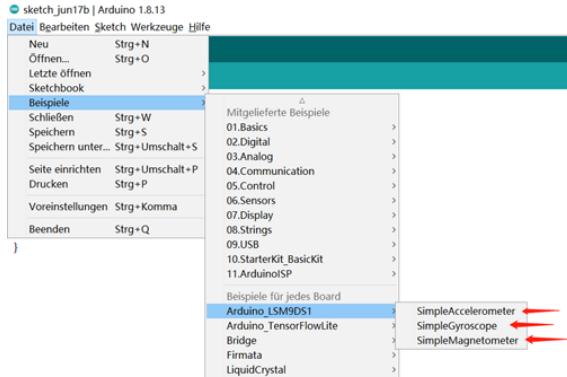


Figure 16.2.: 9-Axis IMU Sensor

By getting the results and output of the 9-axis IMU sensor, first we need to upload the available library program into Arduino nano 33 BLE Sense. To avoid any trouble during uploading the program, make sure to follow all the steps as we discussed in previous chapter e.g; (choose the board, select the port during upload and change the port when need to see output in serial monitor, reset arduino when needed). By uploading the program successfully, open the serial monitor and see the result as shown in figure.16.3 Make sure to change the position and orientation of board and also place some Magnet aroud the borad, it shows visible changes in the output too. The above 9-Axis IMU output, shows the 3-axis values for each accelerometer, gyroscope and magnetometer. It shows how powerful the Arduino Nano 33 ble sense board and compatible with Arduino IDE the integrated development environment. These values changes as the sensor observes some changing in the sorroundings too, and it shows in the form of graph in the serial monitor.

The Arduino Nano 33 BLE Sense has a built-in IMU (inertial measurement unit) that consists of two modules: the 6-axis BMI270 and the 3-axis BMM150. These modules can measure acceleration, rotation, and magnetic fields in 3D space. To use the IMU, you need to install the official Arduino library for the sensor.

Here is a simple example of how to use the IMU to read the values of the accelerometer and print them to the serial monitor:

[WS:How to install the library PDM](#)

16.2.2. LPS22HB (Pressure Sensor)

Arduino Nano 33 BLE sense also capable of measuring environmental or ambient pressure with the help of LPS22HB on-board embed sensor on it. There is also available library program in Arduino IDE, we need to install the appropriate library and use it or modify it as per the required application. The procedure for uploading and compiling the code is same as the above sensors have. By opening the example

```
1000 #include <Arduino_LSM9DS1.h> // include the IMU library
1002
1003 void setup() {
1004     Serial.begin(9600); // initialize serial communication
1005     while (!Serial); // wait for serial monitor to open
1006
1007     if (!IMU.begin()) { // initialize IMU
1008         Serial.println("Failed to initialize IMU!");
1009         while (1);
1010     }
1011
1012     void loop() {
1013         float x, y, z; // variables to store the accelerometer
1014         values
1015
1016         if (IMU.accelerationAvailable()) { // check if new data is
1017             available
1018             IMU.readAcceleration(x, y, z); // read the acceleration
1019             values
1020             Serial.print("Acceleration x: ");
1021             Serial.print(x);
1022             Serial.print(" y: ");
1023             Serial.print(y);
1024             Serial.print(" z: ");
1025             Serial.println(z);
1026         }
1027     }
1028 }
```

Listing 16.1.: Simple example using of the builtin IMU of the Arduino Nano 33 BLE Sense

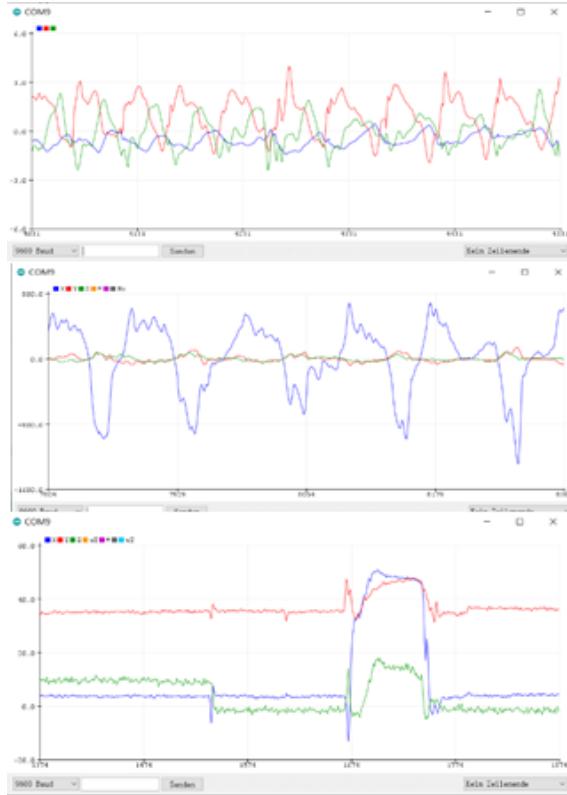


Figure 16.3.: Results of IMU-Tests

section from file, then go to LPS22HB and click on read or measuring pressure as shown in the figure 16.4.

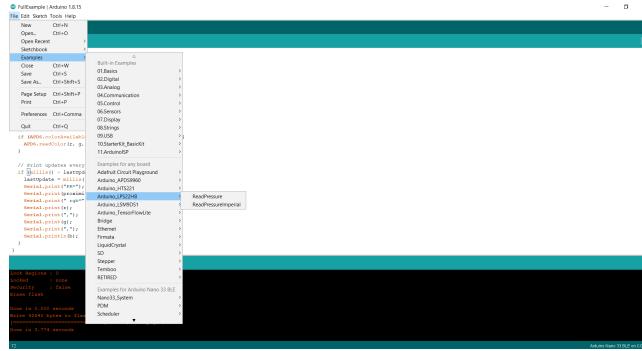


Figure 16.4.: LPS22HB, Pressure and Temperature sensor

The results for LPS22HB the environmental pressure also showed in the serial monitor as shown in the figure. 16.5 With the help of these values and modifying the code we can operate some external devices or also just measure the external or internal pressure and apply it wherever it needs.

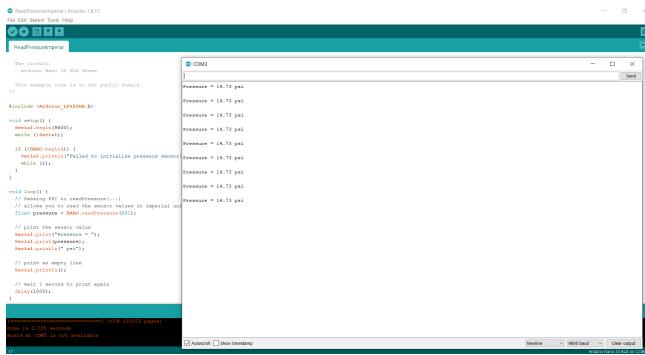


Figure 16.5.: LPS22HB, Output

16.2.3. HTS221 (Humidity and Temperature Sensor)

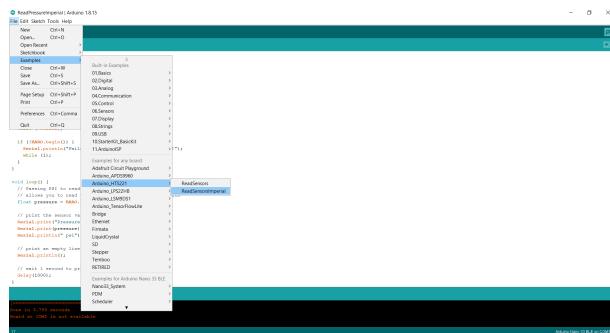


Figure 16.6.: HTS221, Humidity and Temperature Sensor

The on-board embed HTS221 sensor on Arduino Nano 33 BLE Sense has the functionality to measure the relative humidity and temperature in the environment. The procedure for getting access to his library and code also the same as the other sensors have as shown in the figure. 16.6

After Uploading and compiling the program, the environmental humidity and temperature also display in the serial monitor as shown in the figure. 16.7 By getting access to these values, change the port again after compiling the program for opening the serial monitor, these values help us to make the application with electrical appliances regarding energy savings or modify the code and add some external devices with the GPIO of Arduino Nano 33 BLE Sense too and control it as per the humidity and temperature values.

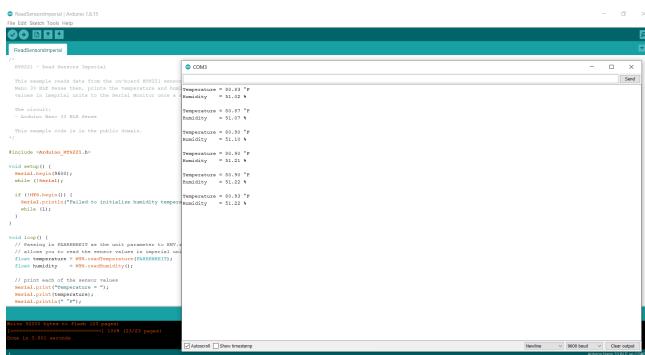


Figure 16.7.: HTS221, Output Window

16.2.4. MP34DT05 (Digital Microphone)

The embed on-board MP34DT05 sensor in Arduino Nano 33 BLE Sense has the functionality to sense audio voice from the environment. There is build in Arduino library for this particular sensor, which is PDM as shown in the figure. 16.8

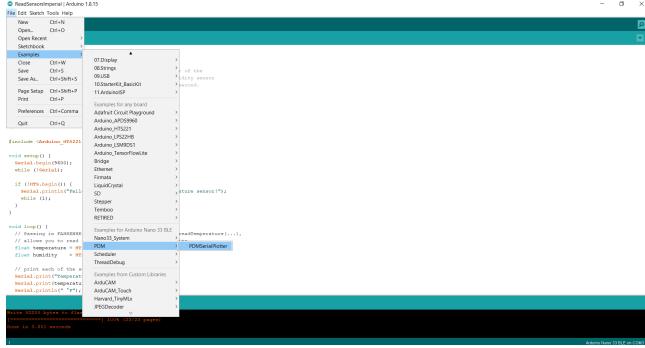


Figure 16.8.: MP34DT05, Digital Microphone

By following the same steps, for this sensor we can see the output the resulted frequency in the serial plotter instead of serial monitor as shown in figure. 16.9 It is more convenient to understand if we change the loudness of voice the plotter shows us a different frequency curve. MP34DT05 use to detect different voices or words too, with the help of these functionality we can easily make the valuable application with Arduino Nano 33 BLE Sense by adding some external devices with GPIO pins with the help of 3.3V relay.

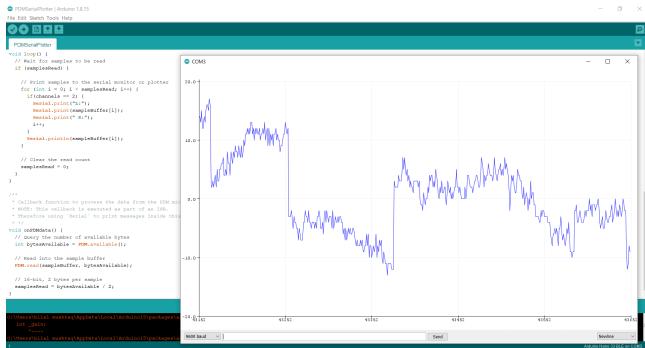


Figure 16.9.: MP34DT05, Serial Plotter

16.3. Test with Bluetooth Module Connection

The same procedure we need to follow for making the successful bleutooth coonection, the one we follow for on-board sensors. Below figure 16.10 shows the ArduinoBLE library in the example section of Arduino IDE.

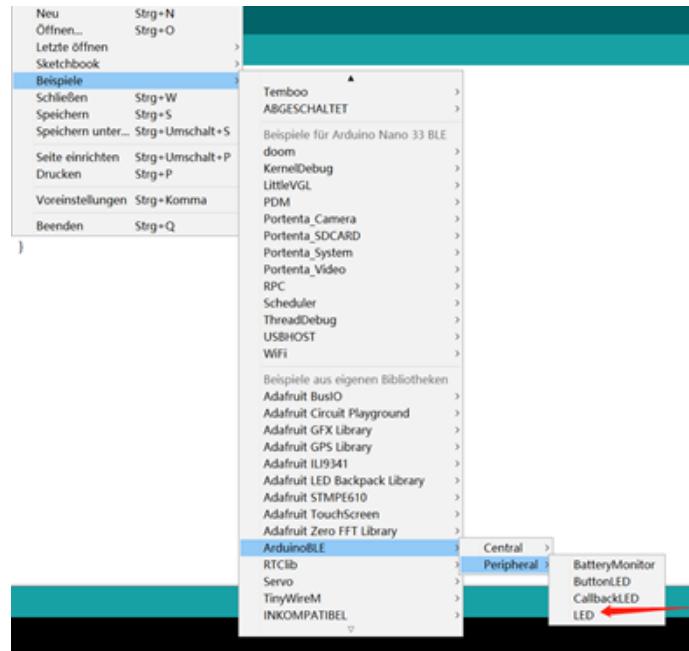


Figure 16.10.: Bluetooth Connection

Bluetooth wireless technology allows us to share the data, the voice, the music, the video, and a lot of information between paired devices. It is built into many products, from mobile phones, cars to medical devices and computers. It has lower power consumption. It is easily upgradeable. It has range better than Infrared communication. Bluetooth is used for voice and data transfer, we can communicate by receiving and sending the data with other bluetooth connected devices. It is also possible to output a value on the phone side or also on the laptop by making the proper pairing.

16.4. Reset of the Arduino

Due to several errors with the serial interface, it was initially assumed that the Arduino had a defect. After further tests, it turned out that the Arduino was not responsive; it had to be reset. With the help of the small button on the Arduino board, it boots into the bootloader and can then be reloaded with a manual change of the COM port.

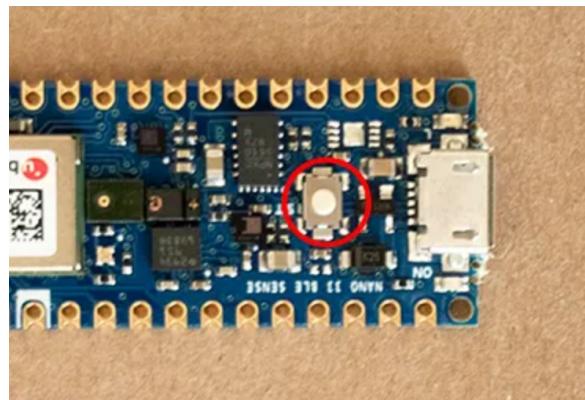


Figure 16.11.: The Reset Button of the Arduino

17. Testspezifikation

Ein 100% reibungsfreier Ablauf kann nie gewährleistet werden, es können stets zufällige Fehler auftreten. Hardwareseitig kann man vor Gebrauch eine Sichtprüfung machen und die Applikation auf Schäden untersuchen. Softwareseitig ist es möglich nach dem Einschalten ein Testprogramm zu durchlaufen.

17.1. Sketch “Hello World”

Zur Prüfung der Entwicklungsumgebung und der grundsätzlichen Funktion der Hardware, bietet sich ein einfacher Beispielsketch an, der nur eine LED ansteuert. Dieser Sketch stellt die Arduino Integrated Development Environment (IDE) zur Verfügung. Unter dem Pfad [File/Examples/01.Basics](#) können kleine Beispielsketchen ausgewählt werden. Hier wird das Beispiel [Fade](#) genutzt. Hierbei wird die eingebaute LED, welche eine RGB-LED ist, genutzt.

```
1000 int brightness = 0; // how bright the LED is
1001 int fadeAmount = 5; // how many points to fade the LED by
1002
1003 // the setup routine runs once when you press reset:
1004 void setup() {
1005     // declare LED_BUILTIN to be an output:
1006     pinMode(LED_BUILTIN, OUTPUT);
1007 }
1008
1009 // the loop routine runs over and over again forever:
1010 void loop() {
1011     // set the brightness of LED_BUILTIN:
1012     analogWrite(LED_BUILTIN, brightness);
1013
1014     // change the brightness for next time through
1015     //the loop:
1016     brightness = brightness + fadeAmount;
1017
1018     // reverse the direction of the fading at the ends
1019     //of the fade:
1020     if (brightness <= 0 || brightness >= 255) {
1021         fadeAmount = -fadeAmount;
1022     }
1023     // wait for 30 milliseconds to see the dimming
1024     //effect
1025     delay(30);
1026 }
```

Als Ergebnis sollte die Helligkeit der eingebauten LED stufenweise an- und ausgehen.

17.2. Test der Sensoren eines Arduino Nano 33 BLE Sense

Zur Aufgabe der Software auf dem Arduino Nano 33 BLE Sense Lite gehört das Auslesen und Übermitteln der Sensordaten an den Computer. In diesem Abschnitt erfolgt eine Beschreibung der dafür entwickelten Software.

```
1000 #include <Arduino_LSM9DS1.h> // IMU
1001 #include <Arduino_APDS9960.h> // Farbsensor
1002 #include <Arduino_LPS22HB.h> // Druck- und Temperatursensor
```

Zu Beginn des Skriptes werden die Bibliotheken zur Ansteuerung der Sensoren eingebunden. Dafür wird der **#include** Befehl verwendet.

```
1000 void setup() {
1001   Serial.begin(9600);
1002   while (!Serial);

1004   // Initialisierung des Druck- und Temperatursensors
1005   if (!BARO.begin()) {
1006     Serial.println("Initialisierung des Druck- und Temperatursensor
1007       fehlgeschlagen.");
1008   }

1010   // Initialisierung der IMU
1011   if (!IMU.begin()) {
1012     Serial.println("Initialisierung der IMU fehlgeschlagen.");
1013   }

1014   // Initialisierung des Farbsensors
1015   if (!APDS.begin()) {
1016     Serial.println("Initialisierung des Farbsensors fehlgeschlagen.");
1017   }
}
```

Zu Beginn wird einmalig die `setup()` Funktion aufgerufen. Innerhalb dieser Funktionen erfolgt das Initialisieren der Bibliotheken für die jeweiligen Sensoren. Sollte eine Initialisierung fehlgeschlagen sein, liefert die Funktion `begin()` eine null als Rückgabewert, die zu einer eins negiert wird und es folgt die Mitteilung der fehlgeschlagenen Initialisierung an den seriellen Monitor.

```
1000 void loop() {
1001   int c1 = 0; // Zaehler fur Verfugbarkeitsprufung der
1002   Beschleunigungsdaten
1003   int c2 = 0; // Zaehler fur Verfugbarkeitsprufung der Farbsensordaten

1004   // Initialisierung der Variablen
1005   float x, y, z; // IMU-Variablen
1006   int r, g, b; // Farbsensor-Variablen
```

Die Funktion `loop()` wird als Dauerschleife auf dem Arduino ausgeführt. Zunächst werden die Zähler für eine spätere Verfügbarkeitsprüfung der Beschleunigungs- und Farbsensordaten initialisiert. Anschließend erfolgt das Deklarieren der Variablen für das spätere Speichern der Sensordaten.

```
1000 float druck = BARO.readPressure(); // Lesen des Drucksensors
1001 float temperatur = BARO.readTemperature(); // Lesen des
1002   Temperatursensors

1003   Serial.println();
1004   Serial.println("Testwerte des LPS22HB-Sensors:");
1005   Serial.println();
1006   // Ausgabe der gemessenen Temperatur
1007   Serial.print("Die Temperatur T = ");
1008   Serial.print(temperatur);
1009   Serial.println(" C wurde gemessen.");

1010  // Ausgabe des gemessenen Drucks
1011  Serial.print("Der Druck p = ");
```

```
1014 Serial.print(druck);
    Serial.println(" kPa wurde gemessen.");
```

In diesem Abschnitt werden die Werte für den Druck und die Temperatur erfasst. Die BARO.readPressure()-Funktion liefert als Rückgabewert den aktuellen Druck in kPa. Mit diesem Wert wird die Variable druck initialisiert. Nach dem identischen Vorgehen wird die temperatur Variable mit dem Rückgabewert der BARO.readTemperature()-Funktion initialisiert. Anschließend erfolgt das Senden der gemessenen Information an den Port des angeschlossenen Computers.

```
1000 // Testen der IMU
1001 Serial.println("Test der IMU-Einheit:");
1002 Serial.println();
1003 while (!IMU.accelerationAvailable()) {
1004     delay(5);
1005     c1++;
1006     if (c1 > 15)
1007     {
1008         break;
1009     }
1010 }
```

Für den Test der inertialen Messeinheit wird zunächst mit einer while-Schleife **while** (! IMU.accelerationAvailable()) geprüft, ob die zu messenden Beschleunigungsdaten verfügbar sind. Sobald der Sensor bereit ist oder 16-mal auf die Verfügbarkeit der Sensordaten geprüft wurde **if**(c1 > 15), wird die Schleife verlassen und es folgt eine weitere Prüfung, deren Ergebnis im positiven Fall das Auslesen der Sensordaten zur Folge hat und im negativen Fall das Auslesen der Beschleunigungswerte überspringt.

```
1000 IMU.readAcceleration(x, y, z);
1001
1002     Serial.print("Die Beschleunigung in x-Richtung beträgt ");
1003     Serial.print(x*100);
1004     Serial.println("% der Erdbeschleunigung.");
1005     Serial.print("Die Beschleunigung in y-Richtung beträgt ");
1006     Serial.print(y*100);
1007     Serial.println("% der Erdbeschleunigung.");
1008     Serial.print("Die Beschleunigung in z-Richtung beträgt ");
1009     Serial.print(z*100);
1010     Serial.println("% der Erdbeschleunigung.");
1011     Serial.println();
1012     Serial.print("Der Betrag der addierten Beschleunigungsvektoren");
1013     Serial.print(" beträgt ");
1014     Serial.print(sqrt(x*x+y*y+z*z)*100);
1015     Serial.println("% der Erdbeschleunigung (Sollwert bei Ruhelage ca.");
1016     Serial.println("100 %).");
```

Das Auslesen der Beschleunigungswerte erfolgt mit der IMU.readAcceleration()-Funktion. Es wird jeweils das Verhältnis der gemessenen Beschleunigung zur herkömmlichen Erdbeschleunigung $g = 9,81 \text{ m/s}^2$ für die drei Koordinaten-Richtungen x, y und z zurückgegeben. Zum Schluss erfolgt eine Plausibilitätsprüfung der gemessenen Werte. Dazu wird der Betrag des summierten Vektors der drei Koordinaten-Richtungen berechnet. Je nach geographischer Lage sollte der Betrag bei ca. 100 % der Erdbeschleunigung liegen.

```
1000 while (!APDS.colorAvailable()) {
```

```

1002     delay(5);
1003     c2++;
1004     if(c2 > 15)
1005     {
1006         break;
1007     }
1008

```

Für den Farbsensor erfolgt der identische Test auf Verfügbarkeit der Sensordaten wie zuvor bei der IMU.

```

1000 // Auslesen der Farbwerte
1001 APDS.readColor(r, g, b);
1002
1003 // Ausgeben der Farbwerte
1004 Serial.print("Der Rotwert r = ");
1005 Serial.print(r);
1006 Serial.println(" wurde gemessen.");
1007 Serial.print("Der Grunwert g = ");
1008 Serial.print(g);
1009 Serial.println(" wurde gemessen.");
1010 Serial.print("Der Blauwert b = ");
1011 Serial.print(b);
1012 Serial.println(" wurde gemessen.");
1013 Serial.println();
1014

```

Die Farbwerte für das rote, grüne und blaue Licht können mit der APDS.readColor()-Funktion ausgelesen werden. Nach erfolgreichem Auslesen der Farbwerte werden diese an den seriellen Port gesendet.

```

1000 // Nach 3,8 Sekunden werden die Sensordaten erneut ausgelesen.
1001 delay(3800);
1002

```

Zum Ende des vollständigen Durchlaufs der loop()-Funktion wird das Fortschreiten des Skriptes für 3,8 Sekunden verzögert. Diese Zeit ist auf die Öffnungsduer von vier Sekunden des seriellen Monitors auf dem Computer angepasst. In seltenen Fällen kann es dadurch zu einer doppelten Anzeige der Sensordaten in der Log-Datei kommen.

18. Serial Communication between PC and Arduino

To communicate from the PC with the Arduino Nano 33 BLE 33 Sense using Python via USB, using the serial protocol is a good idea.

In this chapter, the simple example is a program which controls the RGB LED via the serial port.

To do this, there are four steps:

1. Installation of a library on the PC
2. Development of the Python program on the PC
3. Preparation of the Arduino and Development of the Arduino Sketch
4. Development of the Arduino Sketch
5. Identical configuration of the serial port and the Arduino sketch

18.1. Installation of a library on the PC

The serial protocol allows you to exchange data between devices using a serial port. You can use the package [PySerial](#) in Python under Windows to interact with serial devices.

The first step is to install the package [PySerial](#) using the program [pip](#):

```
pip install pyserial
```

18.2. Development of the Python program on the PC

After the installation of the package [PySerial](#), the class `serial`. The method `serial.Serial` creates an object that represents the connection to the Arduino. The port name (e.g. COM7) and the baud rate (e.g. 9600) must match to the Arduino sketch. With the methods `write` and `read` data can be sent and received to and from the Arduino. Here is a simple example to send a letter to the Arduino and receive a response:

Listing 18.1.: Python program for the Windows PC for communication with an Arduino via the serial interface.

```
1000 # Python code to send a letter to the Arduino
1001 import serial
1002 import time
1003
1004 # Adapt the port of the Arduino; e.g. 'COM3' under Windows
1005 ArduinoPort = 'COM3';
1006
1007 # Set baud rate; corresponds to the baud rate used in the Arduino sketch
1008 Baudrate = 9600;
1009
1010 try:
1011     # Open the serial port to which the Arduino is connected
1012     ser = serial.Serial(ArduinoPort, Baudrate)
```

```

1014     # Wait until the connection is established.
1016     time.sleep(3)
1018
1019     print(f"Connection to the Arduino established via {ArduinoPort}.")
1020
1021     # Send data
1022     ser.write(b"Hello, Arduino!")
1023
1024     # Receive data
1025     received_data = ser.readline().decode().strip()
1026     print(f"Received data: {received_data}")
1027
1028     # Close serial connection
1029     ser.close()
1030
1031
1032
1033     # Send the letter 'A' to the Arduino.
1034     ser.write(b"A")
1035
1036     # Read the answer from the Arduino.
1037     res = ser.read()
1038
1039     # Print the answer.
1040     print(res)
1041
1042     # Close the port
1043     ser.close()
1044

```

..//Code/Nano33BLESense/Serial/TestSerialPC.py

18.3. Development of the Arduino Sketch and Preparation of the Arduino

Listing 18.2.: Sketch of an Arduino for communication with a Windows PC via the serial interface.

```

1000 // Arduino code to react to a letter
1001 void setup() {
1002     // Initialize the serial communication.
1003     Serial.begin(9600);
1004
1005     // Define a pin as an output.
1006     pinMode(13, OUTPUT);
1007 }
1008
1009 void loop() {
1010     // Check whether data is available.
1011     if (Serial.available()) {
1012         // Read the byte received.
1013         char c = Serial.read();
1014
1015         // React differently depending on the byte.
1016         switch (c) {
1017             case 'A':
1018                 // Switch on the LED.
1019                 digitalWrite(13, HIGH);
1020
1021                 // Send back a confirmation.
1022                 Serial.write("OK");

```

```

1024         // Wait a second.
1026         delay(1000);

1028         // Switch the LED off.
1029         digitalWrite(13, LOW);
1030         break;
1031     }
1032 }
1034 }
```

..//Code/Nano33BLESense/Serial/TestSerial.ino

- Connect the Arduino to the computer using a USB cable.
- Upload the desired sketch to the Arduino.

18.4. Identical configuration of the serial port and the Arduino sketch

Make adjustments

- Adjust the port `ArduinoPort` according to your Arduino connection.
- Make sure that the baud rate `Baudrate` matches the baud rate used in the Arduino sketch.

18.5. Further Readings

- [Python Serial Port Communication Between PC and Arduino Using PySerial Library](#)
- [How would I establish a serial connection with python without using the serial monitor?](#)

Use serial monitor:

- Open the Arduino IDE Serial Monitor and make sure that the baud rate is also set correctly there.
- You can use the Serial Monitor to send and receive data between the Arduino and the computer.

You can find an example of a Python program that can control the RGB LED of the Arduino Nano 33 BLE 33 Sense using the pacakge [pySerial here](#). You can also find the corresponding Arduino sketch that uses the serial library [here](#)

18.6. Example: Transfer of an Image using the Serial Port

TinyML Cookbook - 2nd edition, [Iod23]
p. 361ff

19. Protokoll I²C

Die Firma Philips führte Anfang der 1980er Jahre eine neue serielle Zweidraht-Kommunikationsstandard ein, den *Inter-Integrated-Circuit Bus*, kurz I²C oder I²C. Die Datenübertragung findet über eine Leitung statt, die Serial Data (SDA) Leitung. Über Serial Clock (SCL) wird ein Taktsignal vorgegeben. Die Taktfrequenz liegt zwischen 100 kHz (Standard-Mode) bis 400 kHz (Speed-Mode).

Der erste Schritt zur Datenübertragung in diesem Protokoll ist die Startbedingung. Der Master übermittelt eine Bausteinadresse an den Slave, siehe Abbildung 19.1[GW22]. Der Master wird im Protokoll I²C durch die Code-Zeile `Wire.begin();` definiert. Durch das Weglassen einer Adresse innerhalb der Klammern, wird der Arduino als Master initialisiert[Ardf].

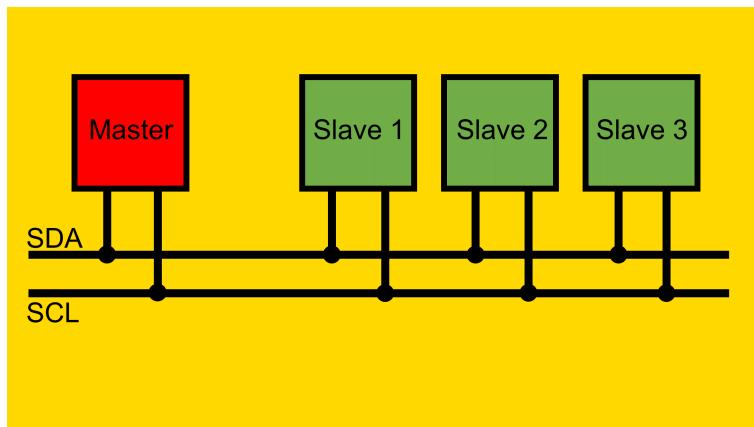


Figure 19.1.: Aufbau des Protokolls I²C [GW22]

Ist diese gesendete Bausteinadresse identisch mit der Adresse des Slaves, kann der Slave an der Kommunikation mit dem Master teilnehmen. So können auch mehrere Slaves angesprochen werden. Im nächsten Schritt wird ein einzelnes Bit gesendet mit dem angezeigt wird, ob der Master vom Slave Daten lesen möchte (1:Lesen/Read) oder Daten an den Slave übermitteln möchte (0:Schreiben/Write). Im Anschluss werden dann die Daten an den Slave gesendet bzw. vom Master empfangen. Dieser Datenaustausch wird dann bestätigt und weitere Daten können ausgetauscht werden. Soll die Kommunikation beendet werden wird eine Stopp-Bedingung gesendet, siehe Abbildung 19.2. [GW22]

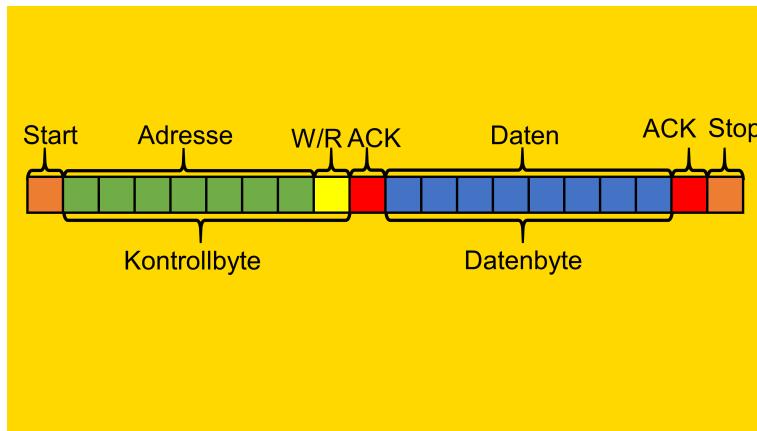


Figure 19.2.: I²C-Protokoll Quelle: [GW22]

Das Protokoll wird auch bei der Verwendung interner Sensoren, wie zum Beispiel des Sensors APDS-9960, benutzt. Ebenfalls die Anbindung eines OLED-Displays erfolgt über I²C.

19.1. Pin-Belegung für I²C beim Arduino Nano 33 BLE Sense

Der Arduino Nanao 33 BLE Sense nutzt zur Realisierung den Pin A4 als Datenleitung SDA und Pin A5 als Taktleitung SCL.

Einige Sensoren sind per Inter-Integrated-Circuit, auch I²C genannt, mit dem Arduino vernetzt. Die Schnittstelle besteht aus einer Daten- und einer Taktleitung, weshalb die Daten auch über längere Strecken übertragen werden können als beispielsweise von der SPI-Schnittstelle. Aufgrund der zwei benötigten Leitungen wird der I²C auch als TWI (Two Wire) Schnittstelle bezeichnet. Alle zu übertragenden Daten werden nach dem FIFO-Prinzip/ seriell hintereinander gesendet. Der Arduino ist beim I²C der Master und die Sensoren sind die Slaves.[Meroth2021] Master und Slave sind jeweils über SDA und SCL Leitung verbunden. Die SDA (Serial Data) Leitung arbeitet in beide Richtungen, also bidirektional, und ist für den Datenaustausch zuständig. Die SCL (Serial Clock) Leitung sendet die Taktimpulse, welcher bei allen Teilnehmern synchron abläuft. Kommuniziert wird mit 128 Teilnehmern, bei einem Arduino können also noch 127 Sensoren am Bus angeschlossen werden.[I²C]

20. Serial Peripheral Interface

Das Akronym SPI steht für Serial Peripheral Interface. Dieses Protokoll wird von Mikrocontrollern genutzt, um eine schnelle Kommunikation mit einem oder mehreren Peripheriegeräten zu ermöglichen. Es gibt drei gemeinsame Pins für alle Peripheriegeräte:

- SCK: Dies steht für Serial Clock und erzeugt Taktimpulse zur Synchronisation der Datenübertragung.
- MISO: Dies steht für Master Input/Slave Output und wird genutzt, um Daten an den Master zu senden.
- MOSI: Dies steht für Master Output/Slave Input und wird genutzt, um Daten an die Slaves/Peripheriegeräte zu senden.

Die SPI-Pins auf der Platine sind D13 (SCK), D12 (MISO) und D11 (MOSI). RXD und TXD werden für die serielle Kommunikation genutzt. Der TXD-Pin wird zur Übertragung von Daten genutzt. Die RXD-Pin für den Empfang von Daten, während der seriellen Kommunikation. Die Pins stellen auch den erfolgreichen Datenfluss vom Computer zur Platine her. Die UART-Pins auf der Platine sind D0 (TX) und D1 (RX).

21. Inertial Measurement Unit

21.1. 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.

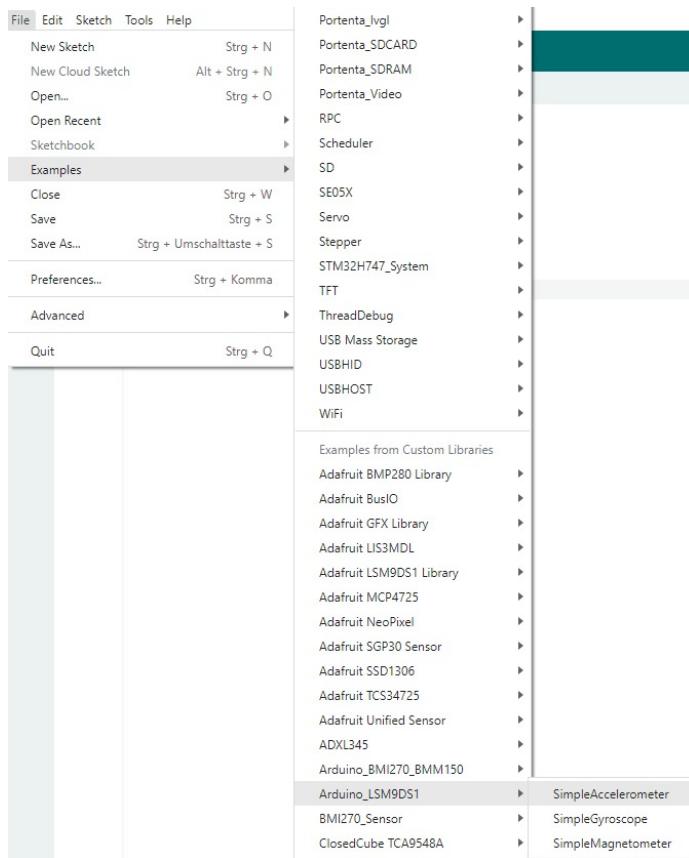


Figure 21.1.: Examples in the library [Arduino_LSM9DS1](#)

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]

21.2. 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.

21.3. 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.

21.4. 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 21.2

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 21.3

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.

21.4.1. Accelerometer:

[Chi+06]

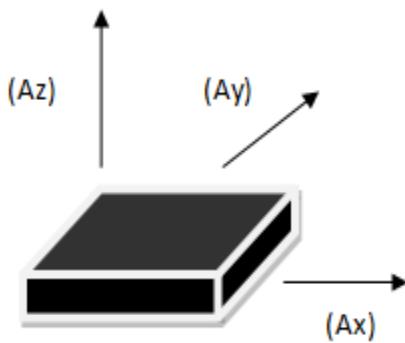


Figure 21.2.: Axis Acceleration of Accelerometer Sensor
WS:tikz!

- 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)

21.4.2. Gyroscope

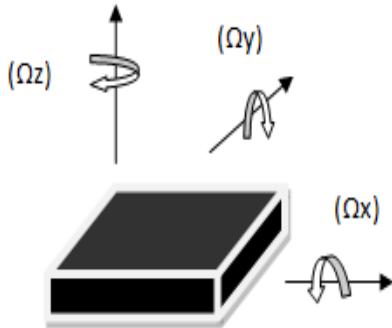


Figure 21.3.: Angular Speed of Gyroscope Sensor

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

21.5. 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.

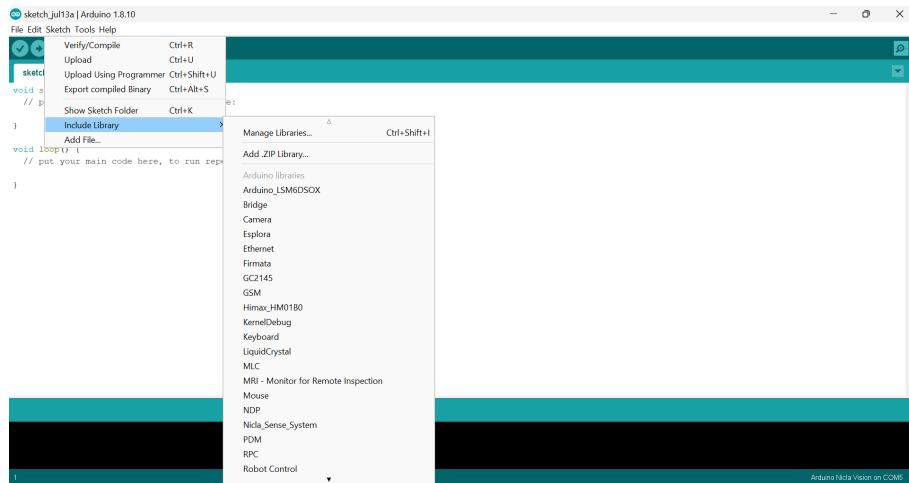


Figure 21.4.: Library setup in Arduino IDE

21.6. 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.

21.7. 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

WS:citations for [benötigt](#).

21.8. Beispiele auf dem Mikrocontroller

21.8.1. Testen des Sensors LSM9DS1

Das Beispiel [SimpleAccelerometer](#) wurde geladen und der Sensor gibt erfolgreich die Daten der Beschleunigung aus.



The screenshot shows the Arduino Serial Monitor interface. The title bar says "Output" and "Serial Monitor x". The message area contains the text "Message (Enter to send message to 'Arduino Nano 33 BLE' on 'COM4')". Below this, the status bar shows "Started" and "Accelerometer sample rate = 119.00 Hz". The main content area displays a table of acceleration data 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	0.97	

The status bar at the bottom right shows "Ln 1, Col 1" and "Arduino Nano 33 BLE on COM4".

Figure 21.5.: Output des Testprogramms

21.9. Programmierung

21.9.1. Programmablaufplan

21.9.2. Programmcode und Dokumentation

```

1 #include <Arduino_LSM9DS1.h>
2 #include <Wire.h>
3 #include "SSD1306Ascii.h"
4 #include "SSD1306AsciiWire.h"
5
6 SSD1306AsciiWire oled;
7
8 #define I2C_ADDRESS 0x3C
9 #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.

```

1 float x, y, z;
2 int degreesX = 0;
3 int degreesY = 0;
4 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.

```

1 void setup()
2 {
3     Wire.begin();
4     Wire.setClock(400000L);

```

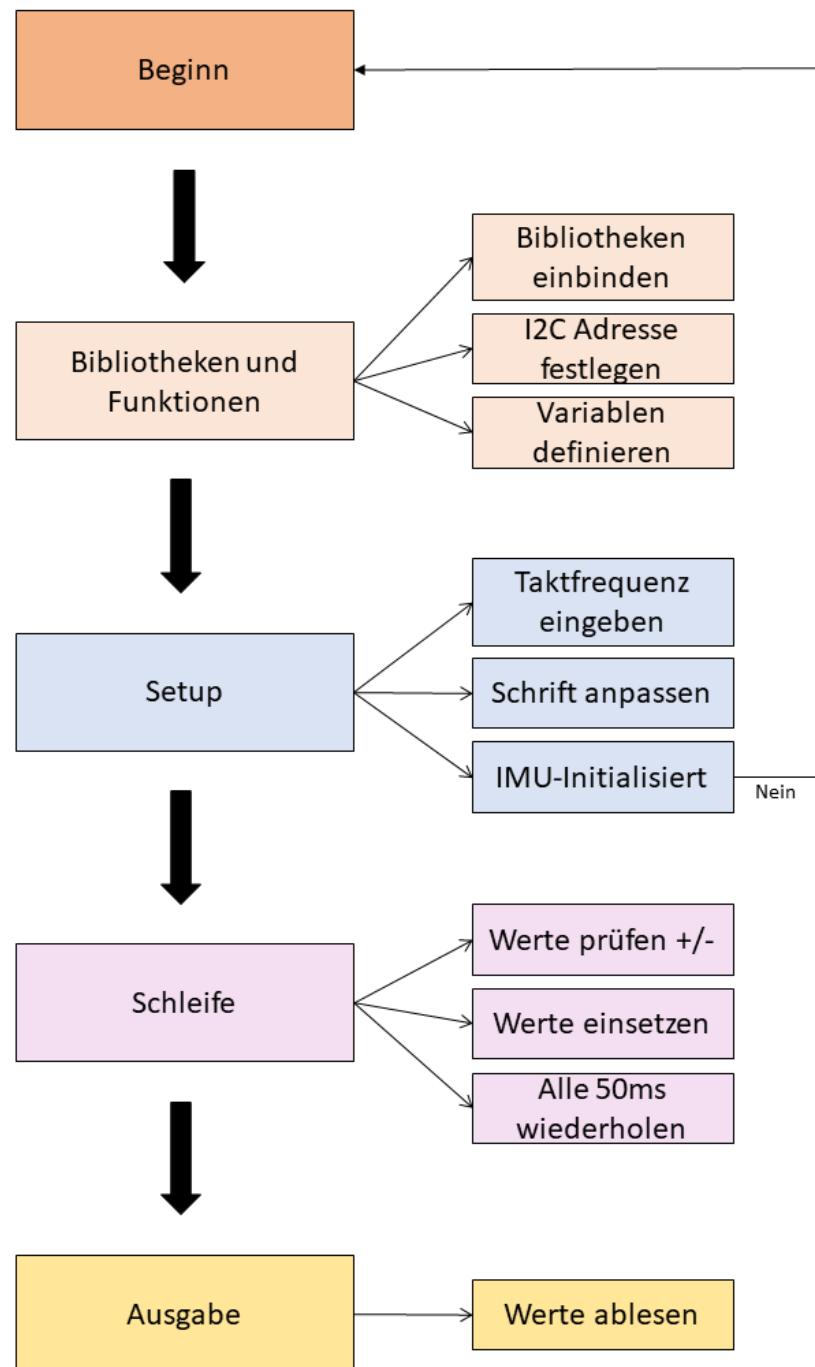


Figure 21.6.: Der Programmablaufplan

```

5
6 #if RST_PIN >= 0
7 oled.begin(&Adafruit128x64, I2C_ADDRESS, RST_PIN);
8 #else // RST_PIN >= 0
9 oled.begin(&Adafruit128x64, I2C_ADDRESS);
10#endif // RST_PIN >= 0
11
12 oled.setFont(TimesNewRoman16_bold);
13 oled.clear();
14 oled.println("IMU Bereit");
15
16 if (!IMU.begin())
17 {
18     oled.clear();
19     oled.println("Failed to initialize IMU!");
20     while (1);
21 }
22
23
24 xPre = " ";
25 yPre = " ";
26 }
```

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.

```

1 void loop()
2 {
3     if (IMU.accelerationAvailable())
```

```

4   {
5     IMU.readAcceleration(x, y, z);
6   }
7   if (x > 0.1)
8   {
9     x = 100 * x;
10    degreesX = map(x, 0, 97, 0, 90);
11
12    xPre = "+";
13    oled.clear();
14    oled.println();
15
16    if (degreesY <= 10)
17    {
18      oled.println("XX0Grad");
19    }
20  else
21  {
22    oled.print("X");
23    oled.print(yPre);
24    oled.print(" ");
25    oled.print(degreesY);
26    oled.print("Grad");
27    oled.println();
28  }
29  if (degreesX <= 10)
30  {
31    oled.println("YY0Grad");
32  }
33  else
34  {
35    oled.print("Y+");
36    oled.print(degreesX);
37    oled.print("Grad");
38    oled.println();
39  }
40}

```

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.

```

1   if (x < -0.1)
2   {
3     x = 100 * x;
4     degreesX = map(x, 0, -100, 0, 90);
5
6     xPre = "-";
7     oled.clear();
8     oled.println();
9
10    if (degreesY <= 10)
11    {
12      oled.println("XX0Grad");

```

```

13         }
14     else
15     {
16         oled.print("X");
17         oled.print(yPre);
18         oled.print("°");
19         oled.print(degreesY);
20         oled.print("Grad");
21         oled.println();
22     }
23
24     if (degreesX <= 10)
25     {
26         oled.println("Y0Grad");
27     }
28     else
29     {
30         oled.print("Y-");
31         oled.print(degreesX);
32         oled.print("Grad");
33         oled.println();
34     }
35 }
```

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.

```

1     if (y > 0.1)
2     {
3         y = 100 * y;
4         degreesY = map(y, 0, 97, 0, 90);
5
6         yPre = "+";
7         oled.clear();
8         oled.println();
9
10        if (degreesY <= 10)
11        {
12            oled.println("X0Grad");
13        }
14        else
15        {
16            oled.print("X+");
17            oled.print(degreesY);
18            oled.print("Grad");
19            oled.println();
20        }
21
22        if (degreesX <= 10)
23        {
24            oled.println("Y0Grad");
25        }
26        else
27        {
28            oled.print("Y");
29        }
30    }
31
32    if (y < -0.1)
33    {
34        y = 100 * y;
35        degreesY = map(y, -97, 0, -90, 0);
36
37        yPre = "-";
38        oled.clear();
39        oled.println();
40
41        if (degreesY <= 10)
42        {
43            oled.println("X0Grad");
44        }
45        else
46        {
47            oled.print("X");
48            oled.print(degreesY);
49            oled.print("Grad");
50            oled.println();
51        }
52    }
53
54    if (y == 0)
55    {
56        oled.print("Y");
57        oled.print("0");
58        oled.print("Grad");
59        oled.println();
60    }
61
62    if (y < 0)
63    {
64        oled.print("Y");
65        oled.print(yPre);
66        oled.print("°");
67        oled.print(degreesY);
68        oled.print("Grad");
69        oled.println();
70    }
71
72    if (y > 100)
73    {
74        oled.print("Y");
75        oled.print("100");
76        oled.print("Grad");
77        oled.println();
78    }
79
80    if (y < -100)
81    {
82        oled.print("Y");
83        oled.print("100");
84        oled.print("Grad");
85        oled.println();
86    }
87
88    if (y < -0.1)
89    {
90        y = 100 * y;
91        degreesY = map(y, -97, 0, -90, 0);
92
93        yPre = "-";
94        oled.clear();
95        oled.println();
96
97        if (degreesY <= 10)
98        {
99            oled.println("X0Grad");
100       }
101      else
102      {
103          oled.print("X");
104          oled.print(degreesY);
105          oled.print("Grad");
106          oled.println();
107      }
108  }
```

```

29         oled.print(xPre);
30         oled.print(" „");
31         oled.print(degreesX);
32         oled.print(" „Grad");
33         oled.println();
34     }
35 }
36 if (y < -0.1)
37 {
38     y = 100 * y;
39     degreesY = map(y, 0, -100, 0, 90);
40
41     yPre = " „-";
42     oled.clear();
43     oled.println();
44
45     if (degreesY <= 10)
46     {
47         oled.println(" „X„„„„0„„„„Grad");
48     }
49     else
50     {
51         oled.print(" „X„„- „");
52         oled.print(degreesY);
53         oled.print(" „Grad");
54         oled.println();
55     }
56
57     if (degreesX <= 10)
58     {
59         oled.println(" „Y„„„„0„„„„Grad");
60     }
61     else
62     {
63         oled.print(" „Y „");
64         oled.print(xPre);
65         oled.print(" „");
66         oled.print(degreesX);
67         oled.print(" „Grad");
68         oled.println();
69     }
70 }
71 delay(50);
72 }
```

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!

21.9.3. 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 21.7.: Anzeige des Arduino OLED Displays sobald der Arduino eingeschaltet ist

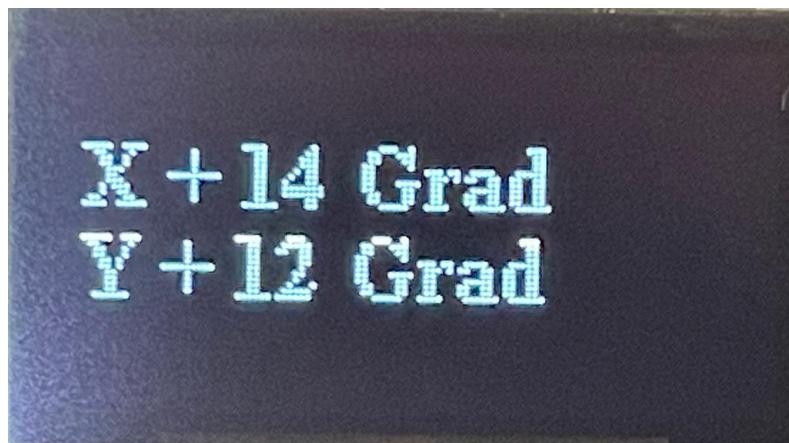


Figure 21.8.: Ausgabe von Messdaten

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.

21.10. 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.

21.11. 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.

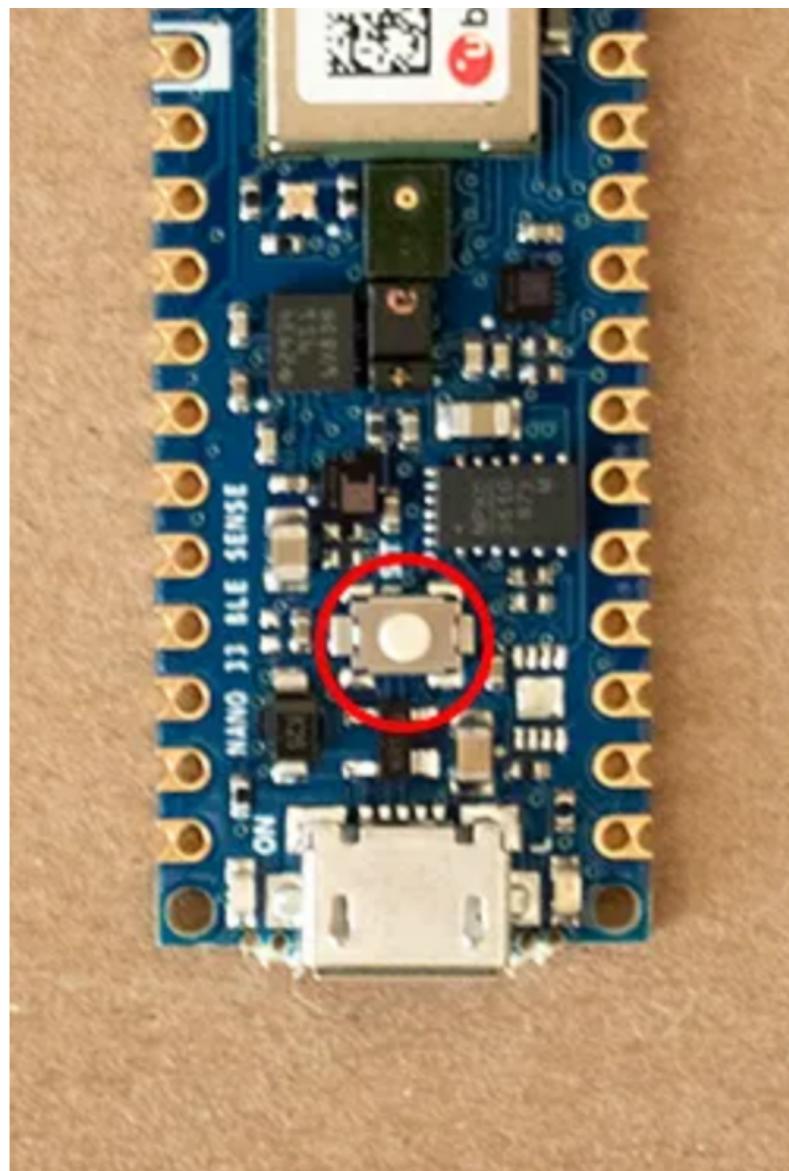


Figure 21.9.: Der Reset Button des Arduino

22. Calibration

22.1. 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]

22.2. 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.

22.2.1. 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].

22.2.2. 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].

22.2.3. 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 22.1.: 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 22.2.: 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 22.3.: Example ADPS-9960

23. Errors

23.1. 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]

23.2. Affected Parameters

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

23.2.1. 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.

23.2.2. 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]

24. IMU LSM6DSOX - Libraries and Functions

24.1. 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.

24.1.1. 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.[Ardc; Ari21]

24.1.2. 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]

24.1.3. 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]

24.1.4. 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.

24.2. 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.

24.2.1. **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]

24.2.2. **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]

24.2.3. **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.

24.2.4. **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.

24.2.5. **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.

24.2.6. 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.

24.2.7. 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.

24.2.8. 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.

24.2.9. 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.

24.2.10. 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.

24.2.11. 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.

24.2.12. 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.

24.2.13. 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.

24.2.14. 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.

24.2.15. 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.

24.2.16. 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.

24.2.17. 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.

24.2.18. 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.

24.2.19. 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.

24.2.20. `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.

24.2.21. `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.

24.2.22. `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.

24.2.23. `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.

24.2.24. 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.

24.2.25. 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.

24.2.26. 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.

Part III.

Arduino Nano 33 BLE Sense - Externe Sensoren und Aktoren

25. External Standard LED

25.1. General

LED stands Light-Emitting Diode. A diode is an electronic component that only allows current to pass in one direction. Most diodes are made of semiconductor materials; in the case of light-emitting diodes, it is usually silicon.

The way an LED works is known from the animal world. For example, the energetic firefly is a flying LED. The only difference is how the atoms inside are excited and thus made to glow. The fireflies achieve this through a chemical reaction. In the LED, this happens with the help of an electric current - in short: electroluminescence.

A light-emitting diode consists of an anode (positive pole) and a cathode (negative pole). A wire, the so-called bonding wire, ensures the flow of current between the two poles. The energy required for electroluminescence flows through this wire. The semiconductor crystal (also known as the LED chip) sits on the cathode. It is surrounded by a reflector trough that directs the light from the LED chip upwards. The entire structure is embedded in a protective plastic lens. It distributes the light in the room, thereby increasing both the efficiency and the luminous efficacy.

The chip is a semiconductor crystal made from two different materials that are doped differently. Doping means that there is an excess of positive or negative charge carriers. When current flows, the electrons react and energy is released in the form of light flashes (photons). The LED lights up.

The color of the light depends on the doping of the semiconductor layers. Depending on the energy level, photons with different wavelengths, i.e. light color, are released. For example, a high energy level produces short-wave blue light and a low energy level produces long-wave red light. The superposition of the three primary colors red, blue and green produces white light. The colors red and green produce yellow. Red and blue become magenta, green and blue produce cyan.

A multicolor light-emitting diode therefore contains three semiconductor crystals, each of which produces one of the three primary colors and expands the color palette in different compositions.

Depending on how the brightness of red, green and blue is controlled, white is produced in different shades between very bright cool white light (indicated on the packaging as "5000 Kelvin" and above) or dimmed warm feel-good light (around 3000 Kelvin). As this variant of the mixture is expensive, there is now a new manufacturing option: blue LED chips are coated with a yellowish phosphor layer.

The colors available for acleds are quite diverse, ranging from the basic red, green, and blue (RGB) to a much wider spectrum depending on the technology and application. [Qua] Here's a breakdown of the different types:

Red: This is the most common LED color, achieved by using materials like gallium arsenide phosphide (GaAsP). It has a dominant wavelength around 620-750 nm and appears vibrant red to the human eye.

Green: Another common LED color, typically made with indium gallium nitride (InGaN). Its dominant wavelength falls between 500-570 nm, producing a bright green color.

Blue: Blue LEDs are often made with gallium nitride (GaN) and emit light with a dominant wavelength of 450-480 nm. They appear as a deep, rich blue to our eyes.

Combinations and White Light:

White: While not a single color, white LEDs are crucial for various applications. They are typically achieved in two ways:

- RGB combination: Combining red, green, and blue LEDs in specific ratios can create white light. This method offers flexibility in color temperature control.
- Phosphor conversion: A blue or ultraviolet LED is coated with a phosphor that converts part of the emitted light to longer wavelengths, generating white light. This method is more efficient but offers less color control.

Beyond the Basics:

Amber: Often used in traffic signals and indicator lights, amber LEDs have a dominant wavelength around 585-605 nm and appear yellowish-orange.

Yellow: True yellow LEDs are less common but achievable with specific materials like aluminum gallium indium phosphide (AlGaInP). Their dominant wavelength is around 580-590 nm, appearing as a pure yellow color.

Other colors: Specialized LEDs can produce various other colors, including violet, pink, cyan, and even deep red and green for wider spectrum applications. These often involve more complex materials and manufacturing processes.

It's important to note that the specific color of an acled can vary slightly depending on factors like manufacturing tolerances, viewing angle, and operating conditions. Additionally, new acled technologies are emerging constantly, potentially expanding the color range further in the future. [Kai09; HEG21; HS23; Bai18]

25.2. Specification

A LED need to have a resistor placed in series with it. Otherwise, the unrestricted current will quickly burn out the LED. The resistor can be any value between 100 Ohms and about 10K Ohms. It is necessary to check the data sheet. Lower value resistors will allow more current to flow, which makes the LED brighter. Higher value resistors will restrict the current flow, which makes the LED dimmer. A typically value is 220 Ohm. Using the data sheet of the LED, the value of the resistor can be determined. [Qua; Hui] The maximal current to operate an LED is 20mA. It works also using 4mA until 15 mA. [Qua]

Most LEDs have polarity, which means that they need to be connected the right way around. Usually, the LED's shortest lead connects to the ground side, see figure 25.2

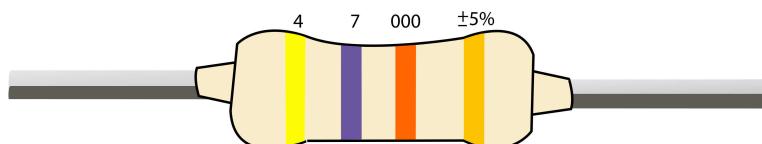


Figure 25.1.: Resistor with 220 Ohm for an external LED.

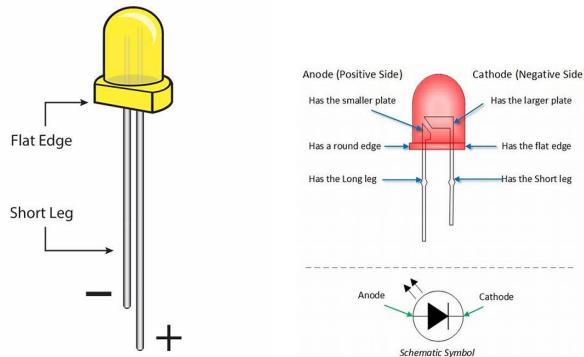


Figure 25.2.: Parts of a LED.

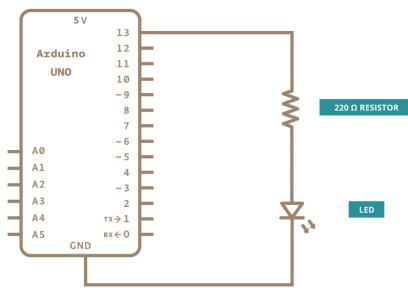


Figure 25.3.: Schematic Symbol for a LED.

The Arduino can control as many LEDs as long as there are enough pins available.

25.3. Using Standard External LED

25.3.1. Connecting a LED to the Arduino Board

25.3.2. Pins

As mentioned, the Arduino can control as many LEDs as long as there are enough pins available. There are two different types of output pins. The standard output pin can just switch between high and low. For some pins the PWM is usable. Using such a pin, the brightness of the LED can be controlled.

PWM A0 - A7 Welche Nummer? 16-23? 19 - 26

Digital D2-D13 Welche Nummer Pin 5 -16

25.4. Bibliothek

No special library is required to operate an external LED.

25.5. Simple Code

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

1. In the first step, the acled must be connected to a pin:

Listing 25.1.: Defining the pin for an external LED

```
1000 #define LED_EXT 14
```

2. In the second step, the pin is configured in the function `setup`:

Listing 25.2.: Defining the pin for an external LED

```
1000 pinMode(LED_EXT, OUTOUT)
```

3. In the third step, the acled can be used in the function `loop`. To turn OFF the LEDs, write a state `LOW` to the LED:

Listing 25.3.: Swichting On the LED

```
1000 // Swichting On the LED
1002 digitalWrite(LED, HIGH);
1003 //Waiting 1s
1004 delay(1000)
1005 // Swichting Off the LED
1006 digitalWrite(LED, LOW);
```

25.6. Tests

25.6.1. Simple Function Test

The simplest test is the flashing of the LED at 2 Hz, see sketch 25.4.

Listing 25.4.: Simple sketch to test a LED

```
1000 // How to control an external LED with the Arduino Nano 33 BLE Sense
1001 //
1002 // file: TestLED.ino
1003 //
1004 // The LED is switched on for 1 second and switched off
1005 // for 1 second so that the LED flashes accordingly.
1006 //
1007 // Define the pin for the LED
1008 #define LED_EXT 14
1009
1010 void setup() {
1011     // Initialize the pin as an output
1012     pinMode(LED_EXT, OUTPUT);
1013 }
1014
1015 void loop() {
1016     // Turn the LED on
1017     digitalWrite(LED_EXT, HIGH);
```

```

1018     // Wait for one second
1019     delay(1000);
1020     // Turn the LED off
1021     digitalWrite(LED_EXT, LOW);
1022     // Wait for one second
1023     delay(1000);
1024 }
```

..../Code/Nano33BLESense/Test/TestLED.ino

25.6.2. Test all Functions

Standard pins can be used, but pins that support pulse width modulation can also be used. One such pin is used in the example 25.5. The brightness can then be varied.

Listing 25.5.: Simple sketch to test a LED with pulse width modulation

```

1000 // file: TestLEDBrightness.ino
1002 // Define the pin for the external LED
1003 #define LED_EXT 21
1004 // Define the initial brightness values (0–255)
1005 int ledBrightness = 0;
1006 // Define the increment/decrement value
1007 int ledStep = 5;
1008 void setup() {
1009     // Set the LED pin as output
1010     pinMode(LED_EXT, OUTPUT);
1011 }
1012 void loop() {
1013     // Write the PWM value to the LED pin
1014     analogWrite(LED_EXT, ledBrightness);
1015
1016     // Update the brightness value
1017     ledBrightness += ledStep;
1018
1019     // Check if the brightness value is out of range and reverse the
1020     // direction
1021     if (ledBrightness <= 0 || ledBrightness >= 255) {
1022         ledStep = -ledStep;
1023     }
1024     // Wait for 10 milliseconds
1025     delay(10);
1026 }
```

..../Code/Nano33BLESense/Test/TestLEDBrightness.ino

25.7. Simple Application

In the sketch 25.6, a variable is connected to pin 14. The pin is defined as an output in the function `setup`. An interrupt function is defined, changing the state of a flag. If the flag has the value `true`, the led is switch on for 2 seconds. After the 2 seconds, the led is switch off and the value of the flag is set to `false` back.

Listing 25.6.: Simple sketch to control an external LED. Here, pushing the built-in button is handled by an interrupt. Then the LED switch on for 2 sec.

```

1000 // How to control an external LED
1001 // and using the builtin button
```

```

1002 // with the Arduino Nano 33 BLE Sense
1003 //
1004 // file: TestPushButtonInterrupt.ino
1005 //
1006 // If the builtin button is pressed, the LED is switched on
1007 // for 2 second and switched off again
1008 //
1009 // Define the pin for the LED
1010 #define LED_EXT 14
1011 // Use the onboard push button (BUTTON_B)
1012 #define BUTTON_PIN BUTTON_B

1013 // Initialize variables
1014 // Flag, whether the button is pressed
1015 // Declare as volatile for interrupt safety
1016 volatile bool pushPressed = false;
1017 // LED>Status zur Verarbeitung
1018 int ledState = 0;

1019 void setup() {
1020     // Initialize the pin as an output
1021     pinMode(LED_EXT, OUTPUT);
1022     // Initialize the pin as an input
1023     pinMode(BUTTON_PIN, INPUT_PULLUP);
1024     // Initialize the interrupt function
1025     attachInterrupt(digitalPinToInterrupt(BUTTON_PIN), buttonPressed,
1026                     FALLING);
1027 }

1028

1029 // Interrupt function
1030 // as short as possible
1031 void buttonPressed()
1032 {
1033     if (pushPressed == false)
1034     {
1035         pushPressed = true;
1036     }
1037 }

1038 void loop()
1039 {
1040     if (pushPressed)
1041     {
1042         // Turn the LED on
1043         digitalWrite(LED_EXT, HIGH);
1044         // Wait for one second
1045         delay(2000);
1046         // Turn the LED off
1047         digitalWrite(LED_EXT, LOW);
1048         pushPressed = false;
1049     }
1050 }

```

..../Code/Nano33BLESense/Test/TestPushButtonInterrupt.ino

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

25.8. Further Readings

WS:citations

26. External RGB LED

26.1. Standard RGB-LED

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

To connect a RGB LED to an Arduino Nano 33 BLE Sense board, you need to use three resistors, one for each color pin. The value of the resistors depends on the type and specifications of the RGB LED, but a common value is 220 ohms. You also need to connect the common cathode or anode of the RGB LED to the ground or 3.3V pin of the board, respectively. Here is a diagram that shows how to connect a common cathode RGB LED to the board:

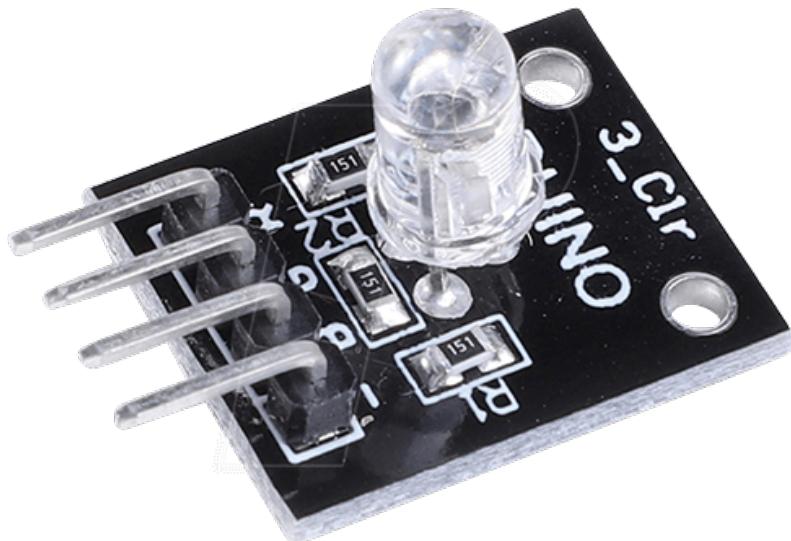


Figure 26.1.: External RGB LED with Resistors [Sima]

External RGB LED with Resistors [Sima; Kin]

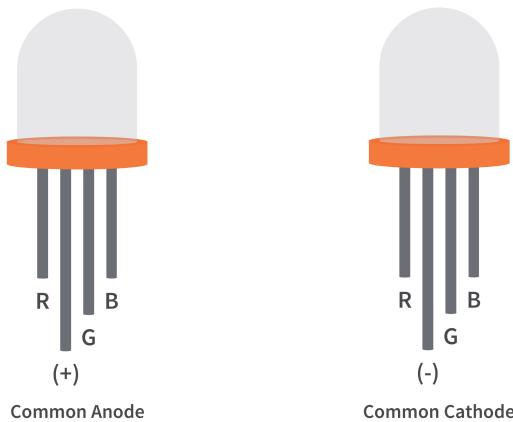


Figure 26.2.: Connectors of a RGB LED.

26.2. Specific Sensor

26.2.1. Pins

~~VS:take a LED or RGB LED~~

Here, you can define some pins for LEDs.
This must ~~VS:be observed~~ when using the pins.

Red LED: LEDR = Pin 22

Green LED: LEDG = Pin 23

Blue LED: LEDB = Pin 24

26.3. Specification

cite data sheet

26.4. Calibration

cite method

26.5. Simple Code

26.6. Simple Application

26.7. Tests

26.7.1. Simple Function Test

To turn ON the LEDs, write a state **HIGH** to the LED:

```
1000   digitalWrite (LEDR, HIGH) ; //RED
1002   digitalWrite (LEDG, HIGH) ; //GREEN
          digitalWrite (LEDB, HIGH) ; //BLUE
```

Listing 26.1.: Swichting On the LEDs

To turn OFF the LEDs, write a state `LOW` to the LED:

```
1000 digitalWrite(LED_R, LOW); //RED
1002 digitalWrite(LED_G, LOW); //GREEN
1002 digitalWrite(LED_B, LOW); //BLUE
```

Listing 26.2.: Swichting Off the LEDs

26.7.2. Test all Functions**26.7.3. Brightness of the RGB-LED**

In a short cut, using values between 255 - 0 to write to the RGB LED is possible, too. Then the brightness is defined.

```
1000 analogWrite(LED_R, 72); //GREEN
1002 analogWrite(LED_G, 122); //BLUE
1002 analogWrite(LED_B, 234); //RED
```

Listing 26.3.: value between 255 - 0 to write to the RGB LED

This sketch 26.4 will make the RGB LED change colors smoothly by varying the brightness of each LED with different speeds. You can adjust the initial brightness values and the increment/decrement values to get different effects.

```
1000 // Define the pins for the RGB LED
1002 #define RED_PIN 22
1002 #define GREEN_PIN 23
1002 #define BLUE_PIN 24
1004
1006 // Define the initial brightness values for each color (0-255)
1006 int redBrightness = 0;
1006 int greenBrightness = 0;
1008 int blueBrightness = 0;
1010
1010 // Define the increment/decrement value for each color
1010 int redStep = 5;
1012 int greenStep = 3;
1012 int blueStep = 7;
1014
1016 void setup() {
1016     // Set the LED pins as outputs
1018     pinMode(RED_PIN, OUTPUT);
1018     pinMode(GREEN_PIN, OUTPUT);
1018     pinMode(BLUE_PIN, OUTPUT);
1020 }
1022
1022 void loop() {
1022     // Write the PWM values to the LED pins
1024     analogWrite(RED_PIN, redBrightness);
1024     analogWrite(GREEN_PIN, greenBrightness);
1024     analogWrite(BLUE_PIN, blueBrightness);
```

```

1028 // Update the brightness values for each color
1029 redBrightness += redStep;
1030 greenBrightness += greenStep;
1031 blueBrightness += blueStep;
1032
1033 // Check if the brightness values are out of range and reverse the
1034 direction
1035 if (redBrightness <= 0 || redBrightness >= 255) {
1036     redStep = -redStep;
1037 }
1038 if (greenBrightness <= 0 || greenBrightness >= 255) {
1039     greenStep = -greenStep;
1040 }
1041 if (blueBrightness <= 0 || blueBrightness >= 255) {
1042     blueStep = -blueStep;
1043 }
1044
1045 // Wait for 10 milliseconds
1046 delay(10);
1047

```

Listing 26.4.: Different brightness levels for the RGB LED colors

Colors

A RGB LED is a device that can emit light of different colors by mixing the primary colors of red, green, and blue. The color of the light depends on the relative brightness of each LED, which can be controlled by PWM signals. By varying the brightness of each LED, the RGB LED can produce a wide range of colors, such as yellow, cyan, magenta, white, and more. Some examples of the colors and their corresponding brightness values are:

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

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

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

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

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

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

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

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

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

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

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

26.8. Simple Application**26.9. Further Readings**

27. Sensor BME280 für Temperatur, Luftfeuchtigkeit und den Luftdruck

27.1. Beschreibung der Hardware

Der BME280 ist ein Temperatursensor, der von Bosch Sensortec entwickelt wurde. Der Sensor bietet die Möglichkeit, die Temperatur, Luftfeuchtigkeit und den Luftdruck in der Umgebung zu messen, siehe Abbildung 27.1). [Fun]

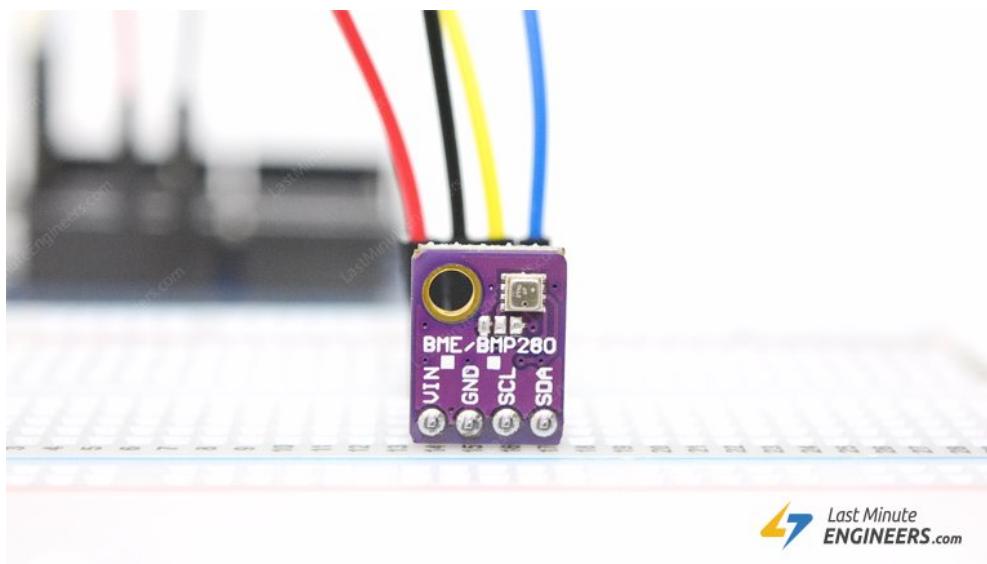


Figure 27.1.: Sensor BME280
[Las]

Bei dem BME280 handelt es sich um einen kombinierten Sensor für die Messung von Feuchtigkeit, Luftdruck und Temperatur. Dieser ist auf dem Entwicklerboard DEBO BME280 verbaut. Das Entwicklerboard hat die Maße 15,3 x 11,5 x 2,5 mm (LxBxH), wobei der Sensor BME280 die Maße 2,5 x 2,5 x 0,93 mm (LxBxH) aufweist. Die Schnittstellen I2C und SPI des Entwicklerboards ermöglichen die Kommunikation zwischen dem Arduino und dem Sensor. Die Versorgungsspannung bewegt sich zwischen 1,72 V und 3,6 V. Die gemessene Luftfeuchtigkeit erfolgt mit einer Genauigkeitstoleranz von ± 3 Prozent relativer Luftfeuchtigkeit und einer Reaktionszeit von einer Sekunde. Der Druckbereich für den Luftdruck beträgt 300 bis 1100 hPa mit einer relativen Genauigkeit von $\pm 0,12$ hPa und einer absoluten Genauigkeiten von ± 1 hPa. Der Temperatursensor hat einen Bereich von -40 bis +85 °C und besitzt eine voll Genauigkeit im Bereich von 0° bis 65° C. Der durchschnittliche Stromverbrauch des Sensors bei einer Frequenz von 1 Hz beträgt $1.8\mu A$ für die Messung von Feuchtigkeit und Temperatur, $2.8\mu A$ für die Messung von Luftdruck und Temperatur und $3.6\mu A$ für die Messung von Feuchtigkeit, Luftdruck und Temperatur. Der Sensor BME280 verfügt über mehrere Pins, die für verschiedene Zwecke verwendet werden. Hierbei ist es wichtig die richtige Pin-Belegung für den Sensor zu kennen. Im Allgemeinen sind die Pins wie folgt belegt: Der VCC-Pin wird mit der positiven Stromversorgung (+3.3 V oder +5

V) des Systems verbunden. Der GND-Pin muss mit dem Masseanschluss (GND) Ihres Systems verbunden werden. Der SDA-Pin ist der Datenpin für die Kommunikation I2C. Dieser wird mit dem entsprechenden SDA-Pin auf dem Mikrocontroller verbunden. Der SCL-Pin ist der Takt-/Clock-Pin für die Kommunikation I2C. Dieser muss mit dem entsprechenden SCL-Pin auf dem Mikrocontroller verbunden werden. SDI sorgt für die SPI-Kommunikation. [Bosa; Bosb]

27.2. Schaltplan des Aufbaus

Der folgende Schaltplan stellt die Komponenten des Arduino Nano BLE Sense Lite, des Sensors BME280 und des OLED-Displays dar. Als Spannungsquelle dient ein Computer, der den Arduino Nano 33 BLE Sense über ein USB-A auf Mikro-USB Kabel versorgt. Die Komponenten sind sinngemäß miteinander verbunden und in Abbildung 29.8 abgebildet.

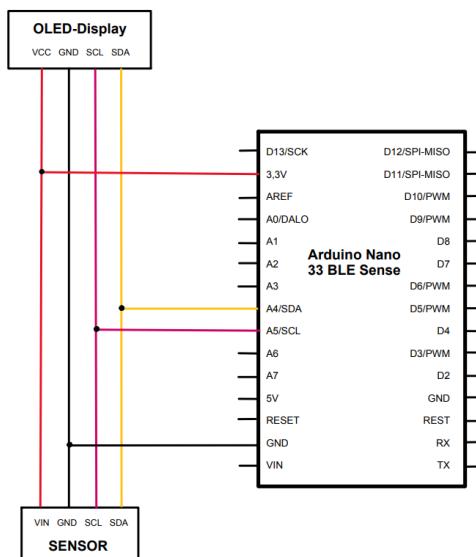


Figure 27.2.: Stationärer Aufbau der Wetterstation
[Arda]

27.2.1. Bibliothek [cactus_io_BME280](#) für den Sensor BME280

Die Bibliothek [cactus_io_BME280_I2C](#) ist eine spezielle Arduino Bibliothek zur Kommunikation mit dem Sensor BME280. Die Bibliothek erleichtert die Interaktion mit dem Sensor BME280 und bietet eine einfache Möglichkeit, Messwerte abzurufen und sie in Ihren Arduino-Projekten zu verwenden. Um die Bibliothek [cactus_io_BME280_I2C](#) zu verwenden, muss sie erst als ZIP-Datei heruntergeladen und anschließend von dem Arduino Library Manager installiert werden. Mit Hilfe von `#include<"cactus_io_BME280_I2C">` können diese in dem Arduino-Code eingebunden werden. [Iot]

27.2.2. Testen des OLED-Displays

Für die Programmierung des Displays gibt es viele Möglichkeiten. Da es viele Librarys gibt, muss zuerst einmal überlegt werden, welche verwendet werden sollen. Um einen ersten Eindruck über die Programmierung des Displays zu bekommen, wurde zunächst ein Beispielsketch getestet. Es handelt sich hier um den Sketch "Hello World",

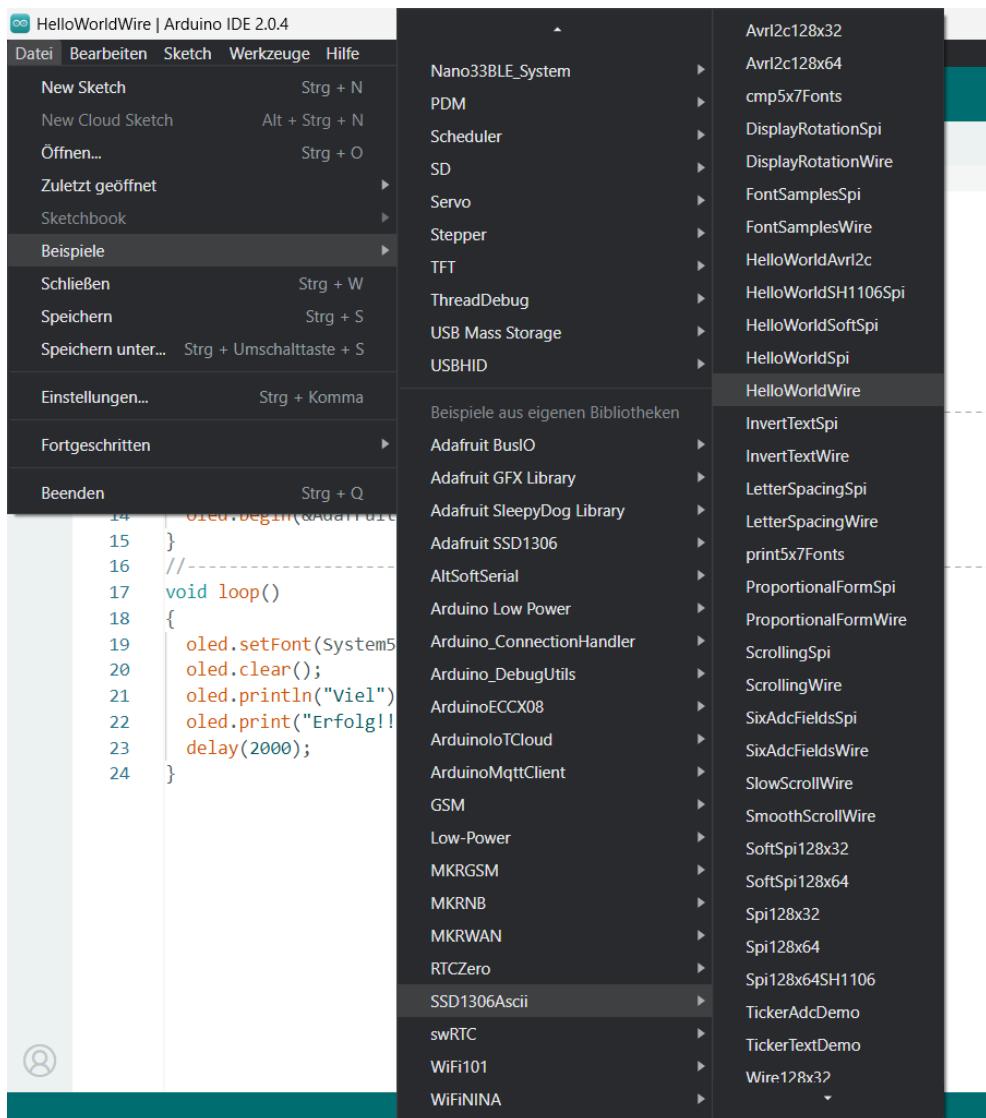


Figure 27.3.: Pfad des Testprogramms.

welcher unter `Datei -> SSD1206Ascii -> HelloWorldWire` zu finden ist (siehe Abbildung 29.9).

Das Testprogramm "Hello World" (siehe Seite 208) wird dafür benutzt, um den Text Hello World auf dem Display auszugeben (siehe Abbildung 29.10). Es wird hier verwendet, um sich mit der Software vertraut zu machen und um sicherzustellen, dass die Entwicklungsumgebung richtig eingerichtet ist. Außerdem wird getestet, ob das Programmieren funktioniert und alle Kabel richtig angesteckt sind.

Nachdem der Quellcode kompiliert und an den Arduino geschickt wurde, wurde auf dem Display der Text Viel Erfolg ausgegeben (siehe Abbildung 29.10). Hiermit ist sichergestellt worden, dass die Entwicklungsumgebung und der Compiler korrekt funktionieren.

```
1000 #include <Wire.h>
1002 #include <SSD1306Ascii.h>
1002 #include <SSD1306AsciiWire.h>
1004 #define I2C_ADDRESS 0x3c
1004
1006 SSD1306AsciiWire oled;
1006
1008 void setup()
1008 {
1009     Wire.begin();
1010     Wire.setClock(400000L);
1011     oled.begin(&Adafruit128x64, I2C_ADDRESS);
1012 }
1014
1016 void loop()
1016 {
1017     oled.setFont(System5x7);
1018     oled.clear();
1019     oled.println("Viel ");
1020     oled.print("Erfolg !!!");
1021     delay(2000);
1022 }
```

Listing 27.1.: Testprogramm für ein OLED-Display



Figure 27.4.: Erste Ausgabe Display

28. Servomotor JAMRA 033212

Elektromotoren besitzen bei geringen Drehzahlen eine geringe Kraftentfaltung. Aus Platz zu sparen, werden deshalb Getriebe verwendet. Die hohe Motordrehzahl wird in ein langsames, aber hohes Drehmoment übersetzt. Dadurch bekommen Servomotoren ihre besonderen Eigenschaften: Sehr genaue Positions- und Geschwindigkeitsregelung. [Ber18]

Unterteilt werden Servomotoren in zwei Kategorien: Open Loop und Closed Loop. Der Open Loop-Motor hat keine Begrenzung im Drehwinkel und kann sich um 360° frei drehen. Dieser wird auch als Schrittmotor bezeichnet. Die realisierte Anwendung zur Sonnennachführung benötigt jedoch keine volle Kreisbahn, weshalb der ausgewählte Motor in die Kategorie Closed Loop fällt. Die in der Industrie eingesetzten Servomotoren besitzen nicht zwingend ein Getriebe, sind daher deutlich größer als ein Modellbauservo. Basierend auf einem Brushless-Motor ist Magnet und Spule sehr groß ausgelegt, um die Winkelgenauigkeit zu halten.

Bei dem verwendeten Servomotor ist das Chassis aus eingefärbtem, durchsichtigem Kunststoff gefertigt, weshalb ein Blick den Aufbau und die wesentlichen Bauteile zeigt.

Motor: Als Aktuator ist im JAMRA 033212 ein Gleichstrommotor verbaut. Da der Motor in der Baugruppe das schwerste Bauteil ist, muss je nach Anwendung aus Einsatzzweck und Realisierung durch Getriebe und Motor ein korrektes Verhältnis abgeleitet werden. In der Sonnennachführung verläuft die Hauptlast vertikal nach unten und wird dort gelagert. Der Servomotor muss also nur ein vergleichsweise geringes Drehmoment aufbringen.

Getriebe: Um bei kleinen Motoren das notwendige Drehmoment aufzubringen, wird ein mehrstufiges Getriebe mit hoher Untersetzung eingesetzt. Bei hochwertigen, teuren Servomotoren besteht das Getriebe aus Metallzahnradern und ist kugelgelagert, der vorhandene Servomotor besitzt ein kostensparendes Kunststoffgetriebe.

Positionssensor: Bei dem Servomotor ist an der Ausgangswelle ein Potentiometer angebracht. Ein Potentiometer ist ein verstellbarer Widerstand welcher als Spannungsteiler ausgelegt ist. Bei Drehung der Welle um einen bestimmten Winkel wird der Widerstand des Potentiometers verändert. Das mittlere Anschlussbein ist der Wischer, welcher am Schleifring entlang schleift.

Motorsteuerung: In der Motor Steuerung kommen Spannung, Position und externes Steuersignal zusammen. Geregelt wird abhängig von der Last die Spannung und von der externen Steuerung die Position. Unterschiede gibt es in Analog- und Digitaler Steuerung. Bei der digitalen Steuerung ist ein zusätzlicher Mikrocontroller verbaut welcher die Position exakter anfahren und halten kann. Dieser ist jedoch auch teurer, weshalb wir uns für die analoge Ausführung entschieden haben.

Die Spannung aus dem Spannungsteiler (Potentiometer) wird mit der Spannung aus dem Impulssteller verglichen. Aus dem Impulssteller kommt das veränderte Signal vom Arduino. Das Signal vom Arduino kommt über eine Signalleitung mit 50 Hz. Je nachdem wie breit das Signal ist, kann die Winkelstellung

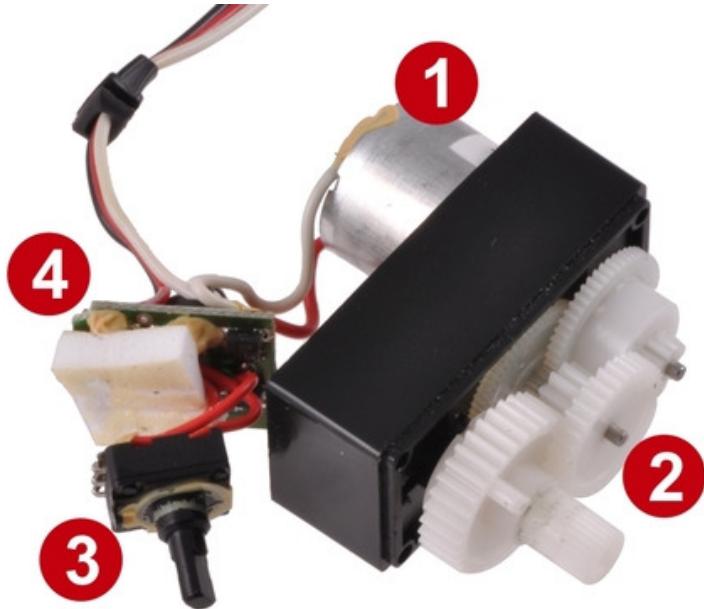


Figure 28.1.: Aufbau eines Servomotors Quelle

durch den Motor Treiber verändert werden. Die Änderung der Signalbreite wird auch als Puls Weiten Modulation bezeichnet, kurz PWM. [Dej18; Ibr18]

Dadurch ist es möglich den Servomotor mit nur einer Steuerleitung anzusteuern. Bei steigender Spannung erhöht sich das Impulssignal bei gleichbleibender Breite. Der Servomotor ändert mit höherer Spannung seine Position schneller und auch die Stellkraft steigt. Beim Halten der Position ist der Stromverbrauch sehr gering, weshalb Servomotoren auch als Verstelleinheit eingesetzt werden.

28.1. Datenblatt JAMRA 033212

Die Zusatzbezeichnung 9g bezieht sich auf das Eigengewicht, welches ungefähr 9 Gramm beträgt und die Bauklasse kennzeichnet. Zum Beispiel im Flugzeugmodellbau als Klappen- oder Fahrwerkssteuerung. Wichtig ist für die Anwendung vor allem die Stellkraft. Im Vorfeld muss die aufzubringende Kraft bekannt sein, um einen Passenden Servomotor auszuwählen. Am Datenblatt wird nochmal deutlich was bei steigender Spannung passiert. Die Kraft und die Geschwindigkeit nehmen zu. Den einfachen, günstigen Servomotor erkennt man auch am Kunststoffgetriebe und an fehlenden Kugellagern.

28.2. Schaltplan

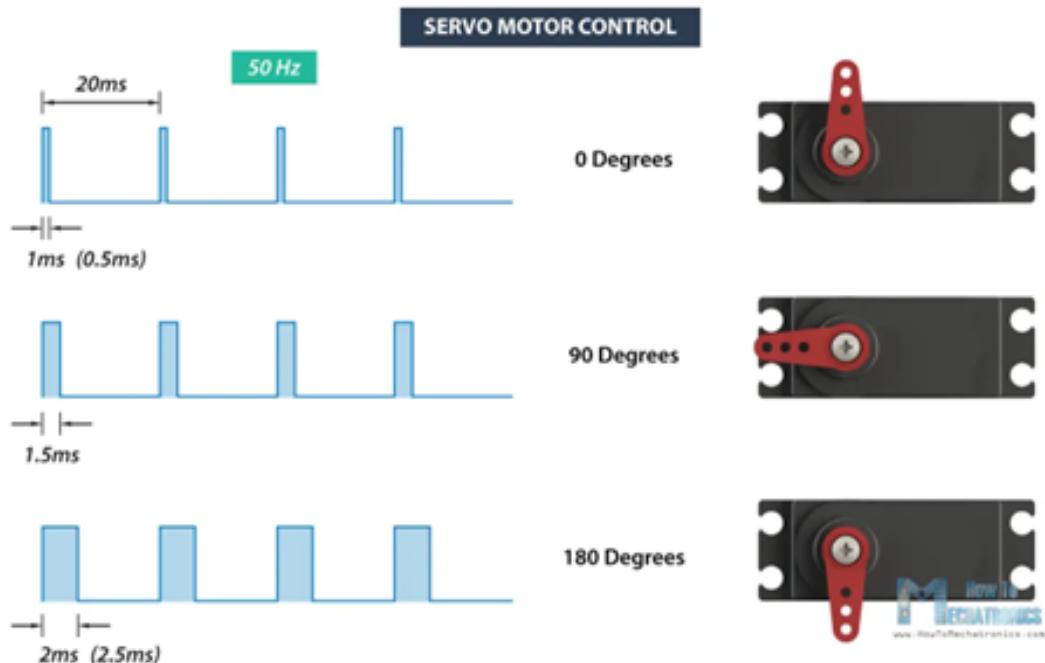


Figure 28.2.: PWM am Servomotor Quelle

WS: keine geeignete
Quelle, richtig zitieren;
eigene Zeichnung mit
tikz

Technische Daten:

Spannung	4,8V ~ 6,0 V
Stellkraft (kg/cm)	1,2 kg/cm (4,8 V) 1,4 kg/cm (6,0 V)
Stellzeit (Sek./60°)	0,11 Sek. (4,8 V) 0,09 Sek. (6,0 V)
Getriebe	Kunststoff
Kugellager	nein
Abmessungen (mm)	32 x 23 x 29,5 mm
Gewicht (g)	12 g

Figure 28.3.: Auszug aus Gebrauchsanleitung JAMRA 033212, Seite 1 [Jam]

WS: Informationen
herausziehen, eigene
Tabelle

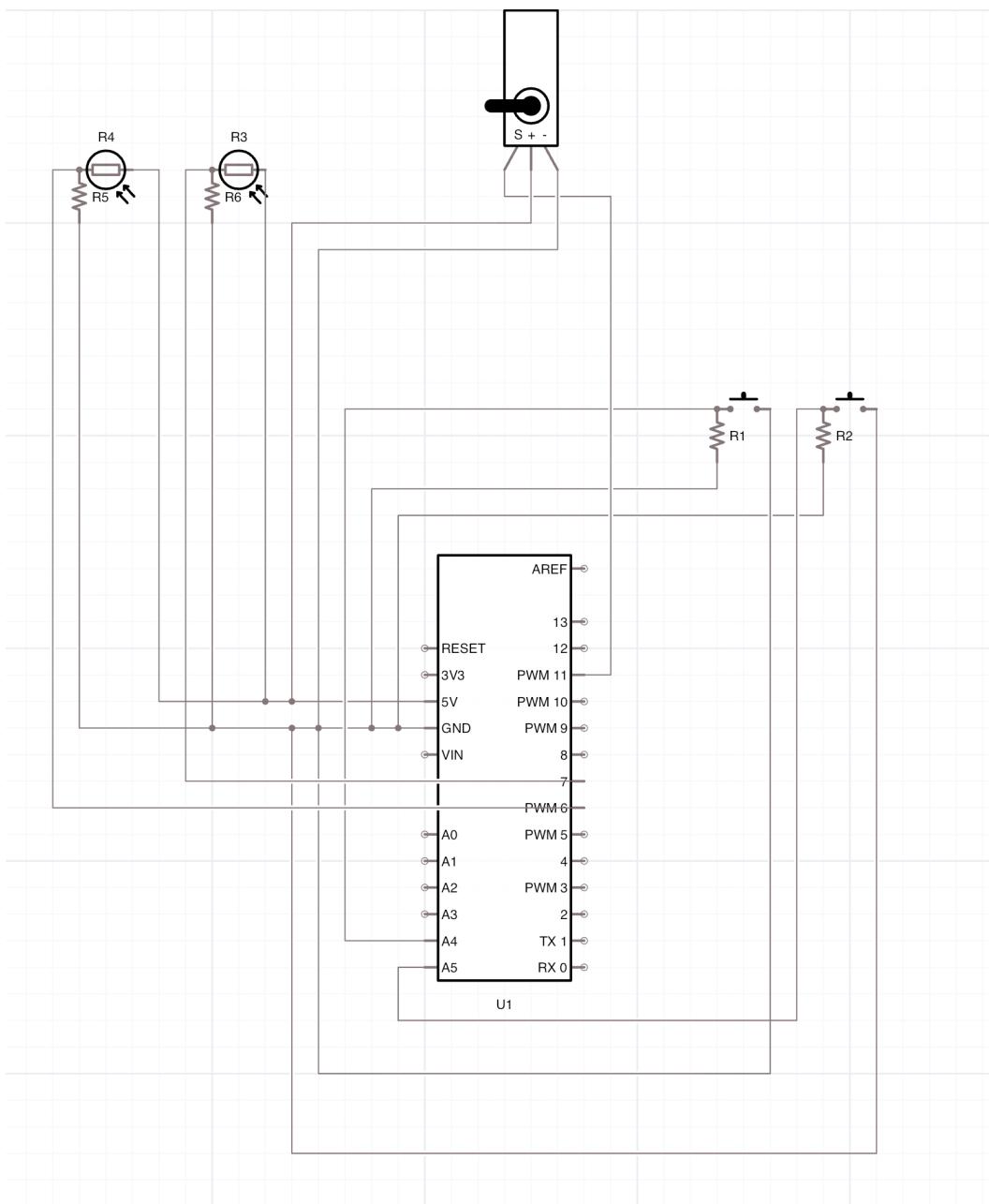


Figure 28.4.: Schaltplan zum Anschluss eines Servomotors JAMRA 033212

WS:Wo ist das
Grove-Kabel?
Motherboard?

29. OLED-Display

29.1. 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.[Simb]

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

29.2. Anschluss

In der Abbildung 29.1 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.

29.3. Programmierung

29.3.1. [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.

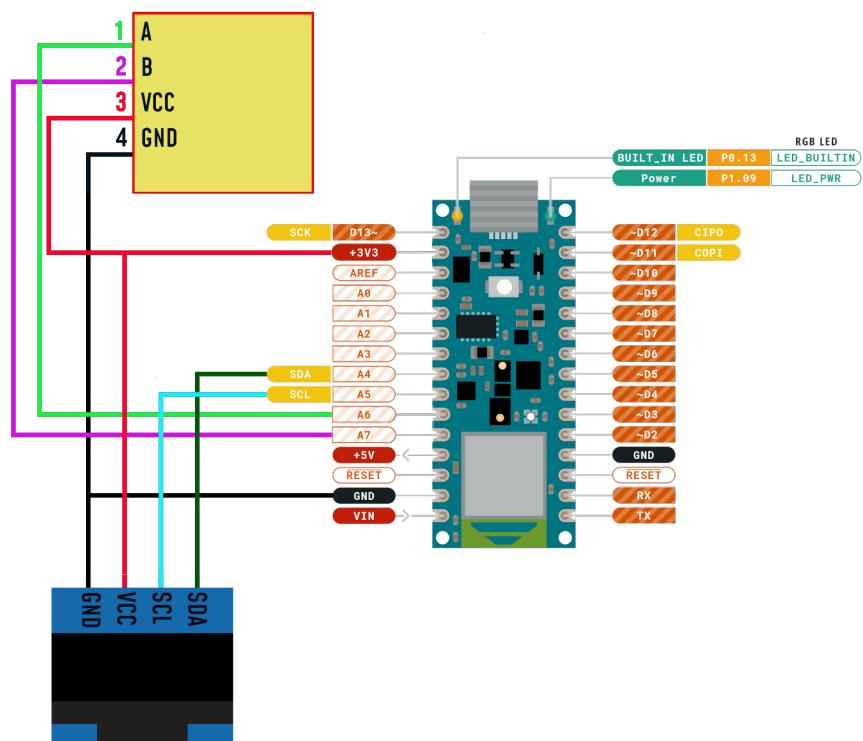


Figure 29.1.: Gesamter Schaltplan

29.3.2. 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.

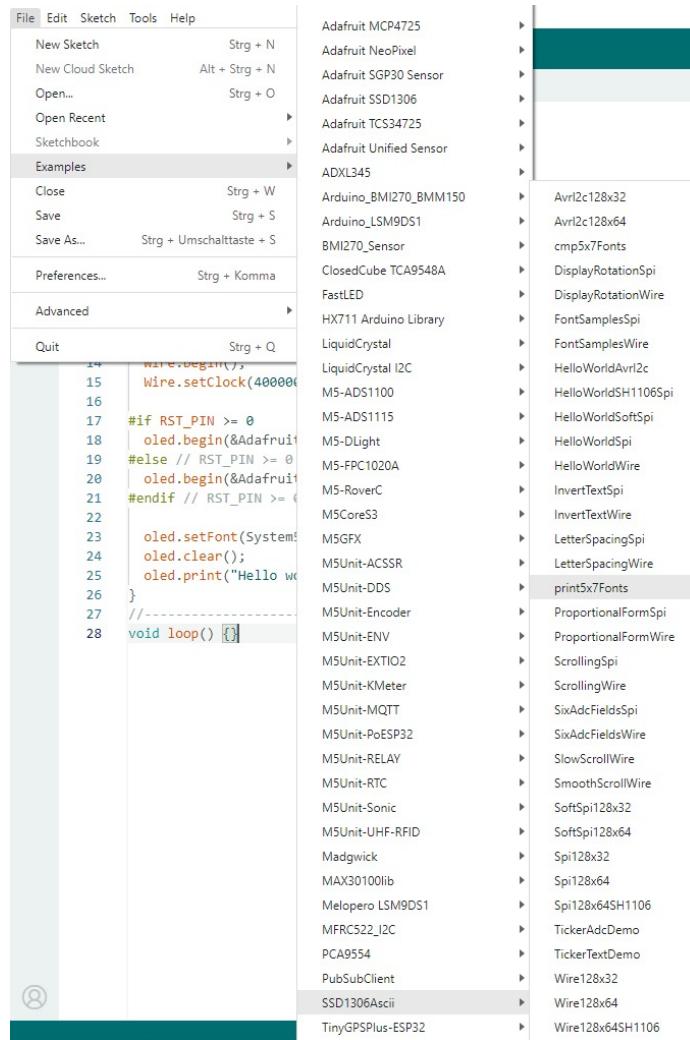


Figure 29.2.: Beispiele in der OLED Bibliothek

29.3.3. 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.

29.4. Software

29.4.1. Verwendete Bibliotheken

Zur Ansteuerung des Displays werden folgende Bibliotheken verwendet:

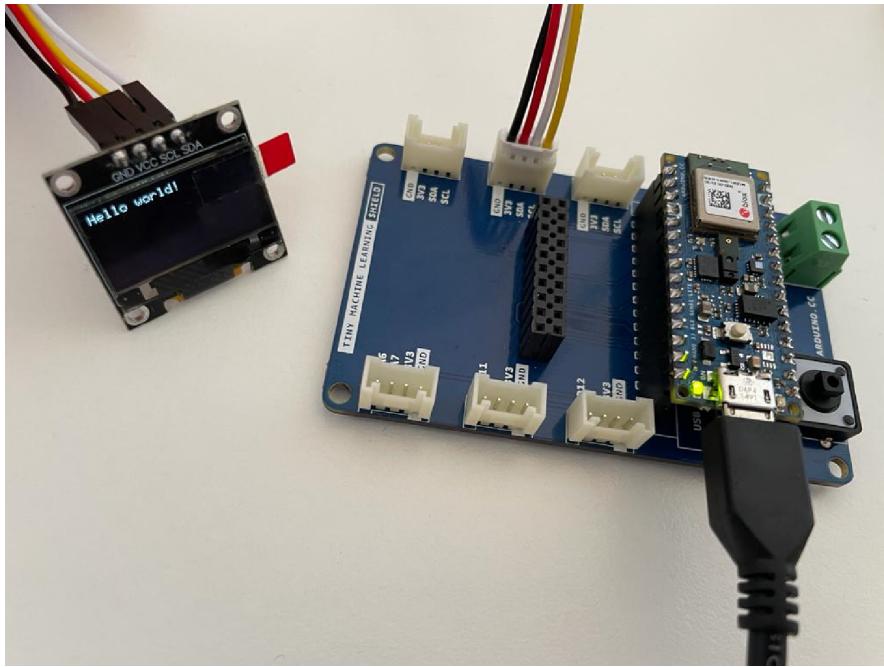


Figure 29.3.: Testausgabe des OLED Display

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

29.4.2. 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
```

29.4.3. 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

29.4.4. 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 29.4.: Code Einlesen/Zählen

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

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

Figure 29.5.: Code Messungssteuerung

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

Listing 29.1.: Monitoring

```
1000 void HandleInput() {
1001     bool blueButtonPressed = analogRead(blueButtonPin) == LOW;
1002     bool redButtonPressed = analogRead(redButtonPin) == LOW;
1003
1004     if (blueButtonPressed)
1005         isMeasuring = true;
1006
1007     if (redButtonPressed)
1008         isMeasuring = false;
1009
1010     bool doNextStep = redButtonPressed || blueButtonPressed;
1011     if (doNextStep)
1012     {
1013         if (isMeasuring)
1014         {
1015             StartSampling();
1016             absoluteSum = 0.0;
1017             absoluteCount = 0;
1018             // Resetting the highest dB SPL value at the start of a new
1019             // measurement
1020             maxdBspl = 0.0;
```

```

1020 // Reset start delay
1021     isDelayOver = false;
1022 // Record start time
1023     startTime = millis();
1024 }
1025 else
1026 {
1027     StopSampling();
1028     if (redButtonResult == 0)
1029     {
1030 // Display of the highest dB SPL value at the end of the measurement
1031         ShowMaxdBspl();
1032         redButtonResult = 1;
1033     }
1034     else
1035     {
1036         ShowAveragedBspl();
1037         redButtonResult = 0;
1038     }
1039 }
1040 }

1041 delay(50);

1042 // Check whether the start delay has expired
1043 bool isMeasureThresholdReached = (millis() - startTime) >=
1044     measureThresholdMilliseconds;
1045 if (isMeasuring && !isDelayOver && isMeasureThresholdReached) {
1046     isDelayOver = true;
1047 }
1048 }
```

..../Code/Arduino/OLED/DEBO -OLED2 0.96/TestDeboOLED.ino

- 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 Tastenprellungen zu vermeiden.

29.4.5. Mikrofondatenverarbeitung

Die Funktion `StartSampling()` initialisiert die Datenverarbeitung:

Listing 29.2.: Start and stop sampling

```

1000 void StartSampling() {
1001     if (!PDM.begin(1, 16000)) {
1002         Serial.println("Failed to start Measurement!");
1003         while (1);
```

```

1004 }
1006 void StopSampling() {
1008 PDM.end();
}

..//Code/Arduino/OLED/DEBO -OLED2 0.96/TestDeboOLED.ino

```

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

29.4.6. 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.

29.4.7. Umrechnung auf den Wert dB SPL

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

Listing 29.3.: Conversion of the microphone signal

```

1000 float getDbValueFromPMC(int pmcValue) {
1001     float maxSampleVoltage = maxSampleValue * Vref / 32767.0;
1002     float dBspl = 20.0 * log10(maxSampleVoltage / Vrms) + sensitivity;
1003     return dBspl;
1004 }

..//Code/Arduino/OLED/DEBO -OLED2 0.96/TestDeboOLED.ino

```

`MaxSampleVoltage` wird berechnet, indem der maximale Samplewert `maxSampleValue` mit der Referenzspannung des Mikrocontrollers („`Vref = 3,3 V`“) multipliziert und durch den maximalen Wert eines 16-Bit-Signed-Integers (32767) dividiert wird. 32767 ist der maximale Wert eines 16-Bit-Signed-Integers, der der maximalen Amplitude des Audiosignals entspricht. Der dB SPL-Wert („`dBspl`“) wird mit der Formel „`20 * log10(Voltage / Vrms) + sensitivity`“ berechnet [Quelle der Formel: PDF Decibels Formula https://physicscourses.colorado.edu/phys3330/phys3330_sp19/resources/Decibels_Phys_3330.pdf University of Colorado System], wobei „`Voltage`“ der berechnete „`maxSampleVoltage`“ ist, und „`sensitivity`“ die Empfindlichkeit des Mikrofons in dBV (Dezibel-Volt) ist. „`Vrms`“ ist die RMS-Spannung (Root Mean Square), die einem Schalldruckpegel von 94 dB SPL entspricht. Dieser Wert wurde experimentell ermittelt.

29.4.8. Benutzeroberfläche

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

Listing 29.4.: OLED-Aktualisierung

```

1000 void HandleUI() {
1001   if (isMeasuring) {
1002     ShowMicrophoneValues();
1003   } else {
1004     //display.clearDisplay();
1005     display.display();
1006   }
}

```

..../Code/Arduino/OLED/DEBO -OLED2 0.96/TestDeboOLED.ino

```

216 void HandleUI() {
217   if (isMeasuring) {
218     ShowMicrophoneValues();
219   } else {
220     //display.clearDisplay();
221     display.display();
222   }
223 }

```

Figure 29.6.: Code

Wenn eine Messung aktiv ist („isMeasuring == true“), werden die aktuellen Messwerte auf dem Display angezeigt. Andernfalls wird der Displayinhalt aktualisiert, ohne neue Werte anzuzeigen.

29.5. 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 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

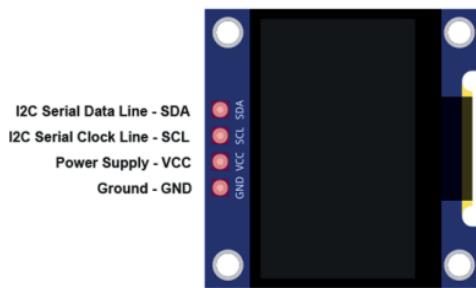


Figure 29.7.: Pins des OLED-Displays.

Das Display verfügt über vier Pins, welche in der Abbildung 29.7 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]

29.6. 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 29.8 abgebildet.

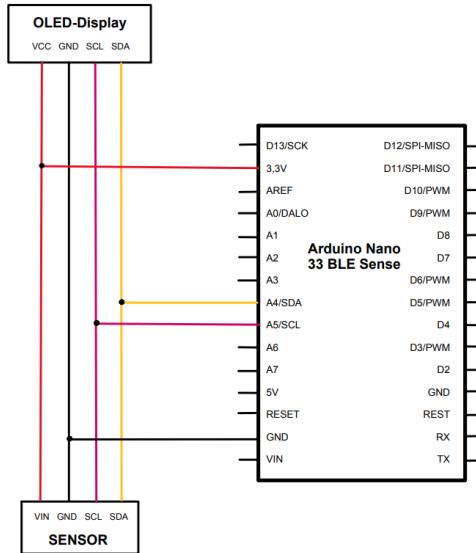


Figure 29.8.: Stationärer Aufbau der Wetterstation
[Arda]

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

29.7.1. Bibliothek Wire.h

Die Bibliothek Wire.h ist eine Standardbibliothek für Arduino-Plattformen, welche Funktionen und Methoden für die Kommunikation zur Verfügung stellt. Mit Hilfe der Schnittstelle I2C ermöglicht die Bibliothek die Kommunikation zwischen einem Arduino und einem anderen I2C-fähigen Gerät wie z.B. Sensoren oder Displays. I2C ist ein Kommunikationsbus, der im Vergleich zu seriellen Schnittstellen den Vorteil hat, dass er mit mehr als zwei Geräten kommunizieren kann. Ein I2C-Bus benötigt zwei Leitungen: SCL für ein Taktsignal und SDA für Daten. Um die Wire.h Bibliotek anwenden zu können, muss sie am Anfang des Sketchs eingebunden werden: `#include<...>` [W3c]

29.7.2. 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. [Funa]

29.7.3. Testen des OLED-Displays

Für die Programmierung des Displays gibt es viele Möglichkeiten. Da es viele Librarys gibt, muss zuerst einmal überlegt werden, welche verwendet werden sollen. Um einen ersten Eindruck über die Programmierung des Displays zu bekommen, wurde zunächst ein Beispielsketch getestet. Es handelt sich hier um den Sketch [Hello World](#), welcher unter Datei -> SSD1206Ascii -> HelloWorldWire zu finden ist (siehe Abbildung 29.9).

Das Testprogramm [Hello World](#) (siehe Seite 208) wird dafür benutzt, um den Text Hello World auf dem Display auszugeben (siehe Abbildung 29.10). 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 29.5.: 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 29.10). Hiermit ist sichergestellt worden, dass die Entwicklungsumgebung und der Compiler korrekt funktionieren. [Funa]

29.8. 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.

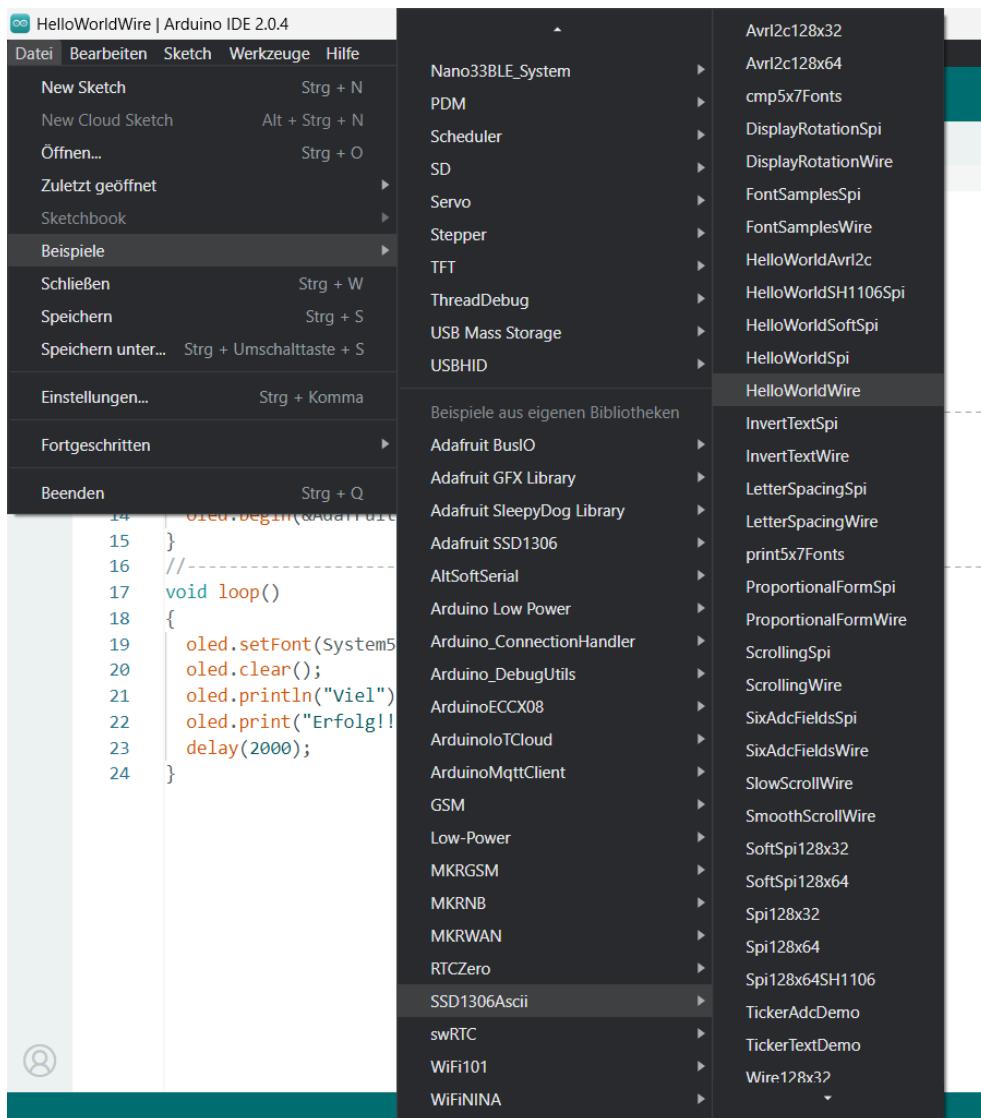


Figure 29.9.: Pfad des Testprogramms.



Figure 29.10.: Erste Ausgabe Display
[Funb]

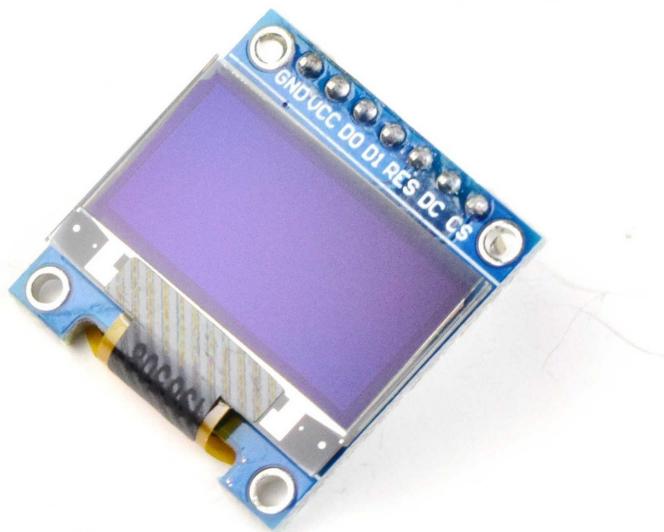


Figure 29.11.: Das Display zur Ausgabe der Messwerte

[Quelle](#) [Alle�hlte](#), [ID](#) [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

29.8.1. OLED

Als weiteren Punkt werden Klassenobjekte im Header initialisiert und Adressen von Peripheriegeräten festgelegt. In diesem Fall sind es das Objekt *oled* der Klasse *SSD1306AsciiWire* und die Definition der Adresse des OLED-Displays. Die Adresse wurde zuvor durch einen I²-Scanner bestätigt.

```
1000 #define I2C_ADDRESS 0x3C
      SSD1306AsciiWire oled;
```

Ein weiterer Punkt ist die Initialisierung und Deklarierung von globalen Variablen. Beim Farberkennungsautomaten sind es zwei Variablen. Über Pin D11, kurz 11, wird der Messtaster abgefragt. Pin D12, kurz 12, gibt bei Bedarf ein Signal an die externe RGB-LED. Durch das Voranstellen von *const* werden beide Variablen zu Konstanten und könne außer über den Source-Code nicht geändert werden. Dies ergibt Sinn, da die Verkabelung sich nicht verändern wird und somit auch nicht die gewählten Pins.

```
1000 const int TasterPin = 11;
      const int LedPin = 12;
```

void setup{}

In der *setup()*-Funktion wird zuerst die serielle Kommunikation mit einer Baudrate von 9600 bit/s gestartet. Baudrate oder auch Bitrate beschreibt die Übertragungsdauer eines Bits. Bei einer Baudrate von 9600 dauert es 104,17 µs um ein Bit zu übertragen. Je höher die Baudrate ist, desto schneller wird ein Bit übertragen. Der Empfänger tastet meist mehrmals pro gesendetem Bit ab und bildet dann nach dreimaligem Abtasten den Mittelwert, welcher dann als empfangenes Bit weiterverarbeitet wird [GW22]. Nach dem Starten der seriellen Kommunikation werden die Modi der zwei genutzten Pins definiert.

```
1000 Serial.begin(9600);
      pinMode(buttonPin, INPUT);
1002   pinMode(ledPin, OUTPUT);
```

Manche Pins können als Input oder als Output genutzt werden. Im Fall des Arduino Nano 33 BLE Sense Lite sind die Pins D13, AREF, A0-A7, TX, RX und D2-D12 als General Purpose Input Output (GPIO) nutzbar. Über das genutzte Shield kann auf A6, A7, D11 und D12 zugegriffen werden. Genutzt werden D11 und D12. Der Tasterpin D11 wird als Input definiert um das Signal aufnehmen zu können, welches durch Drücken des Tasters ausgelöst wird. Pin D12 wird als Output definiert um die externe Rot-Grün-Blau (RGB)-LED einzuschalten. Der Befehl *pinMode()* beinhaltet zusätzlich noch die Möglichkeit einen internen Pull-Up-Widerstand einzuschalten [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 [Ardd].

```
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();
```

30. Kamera-Modul ArduCAM OV2640

30.1. Indroduction

ArduCAM is Arduino based open source camera platform which is well mated to Arduino boards. It is a high definition 2MP SPI camera, which reduce the complexity of the camera control interface. It integrates 2MP CMOS image sensor OV2640, and provides miniature size, as well as the easy to use hardware interface and open source code library. The ArduCAM mini can be used in any platforms like Arduino, Raspberry Pi, Maple, Chipkit, Beaglebone black, as long as they have SPI and I2C interface and can be well mated with standard Arduino boards. The figure ?? shows the mini ArduCAM Camera. The mini ArduCAM OV2640 is well suited for tinyML application, it is easy to configure with Arduino boards. For making the ML applications, especially Images caputuring, object and gesture detection, it supports to take capture and send back to Arduino microcontroller for getting desire results. [Arducam]

<https://www.arducam.com/ov2640/>

<https://www.arducam.com/focal-length-calculator/>

30.2. Produktbeschreibung

Arducam-M-2MP ist eine optimierte Version von Arducam Shield Rev.C und ist eine hochauflösende 2MP-SPI-Kamera, die die Komplexität der Kamerasteuerung verringert. Sie verfügt über einen 2-MP-CMOS-Bildsensor OV2640 und hat eine Miniaturgröße sowie eine einfach zu bedienende Hardware-Schnittstelle und die Open-Source-Code-Bibliothek. Die Arducam Mini-Kamera kann auf allen Plattformen wie Arduino, Raspberry Pi, Maple, Chipkit, Beaglebone Black verwendet werden, solange sie über eine SPI- und I2C-Schnittstelle verfügen und mit Standard-Arduino Boards verbunden werden können. Die Arducam Mini-Kamera bietet nicht nur die Möglichkeit, eine Kamera-Schnittstelle hinzuzufügen, die in einigen Mikrocontrollern nicht vorhanden ist, sondern bietet auch die Möglichkeit, mehrere Kameras zu einem einzigen Mikrocontroller hinzuzufügen.

Anwendung:

- IoT-Kameras.
- Roboterkameras.
- Wildlife-Kameras.

Andere batteriebetriebene Produkte. Kann auf Plattformen wie MCU, Raspberry Pi, ARM, DSP, FPGA verwendet werden.

Eigenschaften:

- 2-Megapixel-Bildsensor OV2640.
- M12-Mount- oder CS-Mount-Objektivhalter mit wechselbaren Objektivoptionen.
- IR-empfindlich mit entsprechender Objektivkombination.
- I2C-Schnittstelle für die Sensorkonfiguration.



Figure 30.1.: Kamera IMX477 der Firma Arducam; [Ard21]

- SPI-Schnittstelle für Kamera-Befehle und Datenstrom.
- Alle E/A-Anschlüsse sind für 5 V/3,3 V geeignet.
- Unterstützt JPEG-Komprimierungsmodus, Einzel- und Mehrfachaufnahmemodus, einmaliges Erfassen mehrerer Lesevorgänge, Burst-Lese-Operation, Niedrige-Energie-Modus usw..
- Kann mit Standard-Arduino-Boards verbunden werden.
- Open-Source-Code-Bibliothek für Arduino, STM32, Chipkit, Raspberry Pi, BeagleBone Black.
- Schlanke Form.

Lieferumfang:

1 x Arducam Mini-Modul Kameraschutz mit OV2640 2 MP, Objektiv, für Arduino UNO Mega2560 Board.

Hinweis: Arduino UNO ist nicht enthalten.

https://www.amazon.com/dp/B07D58GDDV/ref=sr_1_17_sspa?__mk_de_DE=%C3%85%C3%A5%C3%96~O~N&dchild=1&keywords=arducam&qid=1622358684&sr=8-17-spons&psc=1&spLa=ZW5jcnlwdGVkUXVhbGlmaWVYPUEyWUDNSVhJSFNUQUtLJmVuY3J5cHRLZE1kPUEwNjI4NT

30.2.1. Pin Configuration of Arducam 0V2640 2MP Mini

Arducam Mini 2MP OV2640 is a small mini size camera, we can easily embed this camera with any kind of Arduino or other electronics boards, if they have the serial peripheral interface (SPI) and chip select (CS) . It has has 8 pins, the following figure 30.2 shows the functionality of each pin.

It offers to add a camera interface with microcontroller the one who dont have camera capability, also there is a option to add multiple cameras with microcontroller.

Pin No.	Pin Name	Type	Description
1	CS	Input	SPI slave chip select input
2	MOSI	Input	SPI master output slave input
3	MISO	Output	SPI master input slave output
4	SCLK	Input	SPI serial clock
5	GND	Ground	Power ground
6	+5V	POWER	5V Power supply
7	SDA	Bi-directional	Two-Wire Serial Interface Data I/O
8	SCL	Input	Two-Wire Serial Interface Clock

Figure 30.2.: ArduCAM Pin Config

30.3. ArduCAM Interface with Arduino

ArduCAM OV2640 needs the SPI and I2C connection with the arduino boards. It will be connecting until these two connections are make sure. The figure 30.3 shows the ArduCAM connection with Arduino Mega 2560, the same connection will need with the other arduino boards too until the availability of SPI and I2C connection. These ArduCAM cameras are easy to configure with arduino and depends upon the application, it is possible to connect multiple camera with single board to make the edge computing application. Arducam Interface

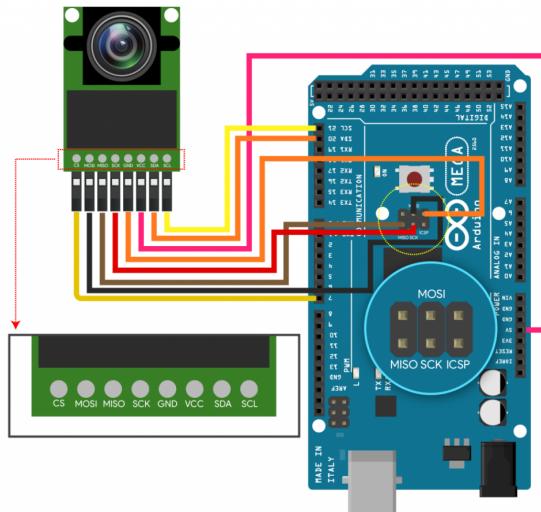


Figure 30.3.: ArduCAM Interface with Arduin Mega 2560

ArduCAM is very small in size, even it is possible to fix the camera on the Arduino board. There is no external battery required for ArduCam to operate, it needs 5V/70mA operating power supply, so it will get the power from the arduino board too. By having the innovative functionality with arduino boards, this can be used in the following applications.

30.4. ArduCAM Library Introduction

<https://github.com/ArduCAM/Arduino>

Dies ist eine Open-Source-Bibliothek für die Aufnahme von hochauflösenden Standbildern und kurzen Videoclips auf Arduino-basierten Plattformen unter Verwendung der Kameramodule von ArduCAM. Die Kamera-Breakout-Boards sollten vor dem

Anschluss an die Arduino-Boards mit dem ArduCAM-Shield funktionieren. ArduCAM-Kameramodule der Mini-Serie wie Mini-2MP, Mini-5MP(Plus) können direkt an Arduino-Boards angeschlossen werden. Zusätzlich zu Arduino kann die Bibliothek auf beliebige Hardware-Plattformen portiert werden, solange sie über eine I2C- und SPI-Schnittstelle verfügen, die auf dieser ArduCAM-Bibliothek basiert.

Now Supported Cameras

- OV7660 0.3MP
- OV7670 0.3MP
- OV7675 0.3MP
- OV7725 0.3MP
- MT9V111 0.3MP
- MT9M112 1.3MP
- MT9M001 1.3MP
- MT9D111 2MP
- OV2640 2MP JPEG
- MT9T112 3MP
- OV3640 3MP
- OV5642 5MP JPEG
- OV5640 5MP JPEG

Supported MCU Platform

Theoretically support all Arduino families

- Arduino UNO R3 (Tested)
- Arduino MEGA2560 R3 (Tested)
- Arduino Leonardo R3 (Tested)
- Arduino Nano (Tested)
- Arduino DUE (Tested)
- Arduino Genuino 101 (Tested)
- Raspberry Pi (Tested)
- ESP8266-12 (Tested) (http://www.arducam.com/downloads/ESP8266_UNO/package_ArduCAM_index.json)
- Feather M0 (Tested with OV5642)

Note: ArduCAM library for ESP8266 is maintained in another repository ESP8266 using a json board manager script.

30.5. Libraries Structure

Die Basisbibliotheken bestehen aus zwei Unterbibliotheken: [ArduCAM](#) und [UTFT4ArduCAM_SPI](#). Diese beiden Bibliotheken sollten direkt unter die Bibliotheken des Arduino-Verzeichnisses kopiert werden, damit sie von der Arduino-IDE erkannt werden.

Die ArduCAM-Bibliothek ist die Kernbibliothek für ArduCAM-Shields. Sie enthält unterstützte Bildsensortreiber und Benutzerland-API-Funktionen, die Befehle zum Erfassen oder Lesen von Bilddaten erteilen. Es gibt auch ein Beispielverzeichnis innerhalb der ArduCAM-Bibliothek, das die meisten Funktionen der ArduCAM-Shields illustriert. Die vorhandenen Beispiele sind Plug-and-Play, ohne dass eine einzige Zeile Code geschrieben werden muss.

Die Bibliothek [UTFT4ArduCAM_SPI](#) ist eine modifizierte Version von UTFT, die von Henning Karlsen geschrieben wurde. Wir haben sie portiert, um das ArduCAM-Shield mit LCD-Bildschirm zu unterstützen. Daher wird die Bibliothek [UTFT4ArduCAM_SPI](#) nur benötigt, wenn das ArduCAM-LF-Modell verwendet wird.

30.6. How to use

Die Bibliotheken sollten vor dem Ausführen von Beispielen konfiguriert werden, ansonsten erhalten Sie eine Fehlermeldung beim Kompilieren.

30.6.1. Edit `memoriesaver.h` file

Öffnen Sie die Datei [memoriesaver.h](#) im ArduCAM-Ordner und aktivieren Sie die Hardwareplattform und das Kameramodul, das zu Ihrer Hardware passt, indem Sie die Makrodefinition in der Datei auskommentieren oder auskommentieren. Wenn Sie zum Beispiel eine ArduCAM-Mini-2MP haben, sollten Sie die Zeile `#define OV2640_MINI_2MP` auskommentieren und alle anderen Zeilen auskommentieren. Und wenn Sie ein ArduCAM-Shield-V2 und ein OV5642-Kameramodul haben, sollten Sie die Zeile `#define ARDUCAM_SHIELD_V2` und die Zeile `#define OV5642_CAM` auskommentieren und dann alle anderen Zeilen.

30.6.2. Choose correct CS pin for your camera

Öffnen Sie eines der Beispiele und verdrahten Sie die SPI- und I2C-Schnittstelle, insbesondere die CS-Pins, entsprechend den Beispielen mit dem ArduCAM-Shield. Hardware und Software sollten konsistent sein, um die Beispiele korrekt auszuführen.

30.6.3. Upload the examples

Im Beispielordner befinden sich sieben Unterverzeichnisse für verschiedene ArduCAM-Modelle und die Host-Anwendung. Der Ordner Mini ist für die Module ArduCAM-Mini-2MP und ArduCAM-Mini-5MP.

1. Der Ordner [Mini_5MP_Plus](#) ist für ArduCAM-Mini-5MP-Plus (OV5640/OV5642) Module.
2. Der Ordner [RevC](#) ist für ArduCAM-Shield-RevC oder ArduCAM-Shield-RevC+ Shields.
3. Der Ordner [Shield_V2](#) ist für das ArduCAM-Shield-V2 Schild.
4. Der Ordner [host_app](#) ist die Host-Erfassungs- und Anzeigeanwendung für alle ArduCAM-Module.
5. Der Ordner [RaspberryPi](#) ist eine Beispielanwendung für die Raspberry Pi-Plattform, siehe weitere Anleitung.

6. Der Ordner [ESP8266](#) ist für ArduCAM-ESP8266-UNO-Board-Beispiele für Bibliothekskompatibilität. Bitte versuchen Sie stattdessen, ESP8266 mit dem Skript json board manager zu repositoryen.

Selecting correct COM port and Arduino boards then upload the sketches.

Arducam MINI Kamera Demo Tutorial für Arduino
Arducam Kamera-Schild V2 Demo Tutorial für Arduino

30.6.4. How To Connect Bluetooth Module

Mit dieser Demo

https://github.com/ArduCAM/Arduino/blob/master/ArduCAM/examples/mini/ArduCAM_Mini_Video_Streaming_Bluetooth

So laden Sie den Host V2:

- For ArduCAM_Host_V2.0_Mac.app, please refer to this link:
www.arducam.com/downloads/app/ArduCAM_Host_V2.0_Mac.app.zip
- For ArduCAM_Mini_V2.0_Linux_x86_64bit, Please refer to this link:
www.arducam.com/downloads/app/ArduCAM_Mini_V2.0_Linux_x86_64bit.zip

31. Lens Calibration Tool

Arducam Lens Calibration Tool, Sichtfeld (Field of View, FoV) Test Chart Folding Card, Pack of 2

https://www.amazon.com/-/de/dp/B0872Q1RLD/ref=sr_1_40?__mk_de_DE=%C3%85%C3%A5%C3%96%C3%9F&dchild=1&keywords=arducam&qid=1622358908&sr=8-40

Multifunktional für Objektiv: Objektivfokus kalibrieren, FoV messen und Schärfe abschätzen

Einfach zu bedienen: Schnelle Einrichtung in Sekunden und Messung in Minuten mit Online-Videotutorial.

Ein handliches Werkzeug: Einfaches Ermitteln des Sichtfelds des Objektivs ohne Berechnung

Sauber und aufgeräumt: Faltbarer Kartenstil, mehrfach gefaltet und aufgerollt für schnelle Einstellung und bessere Lagerung

Anwendung: Fokuskalibrierung, Schärfeabschätzung und Bildfeld-Schnellmessung für M12, CS-Mount, C-Mount und DSLR-Objektive.

<https://www.arducam.com/product/arducam-lens-calibration-tool-field-of-view-fov-test-chart/>

31.1. Übersicht

Sind Sie immer noch frustriert von der Berechnung des FOV Ihrer Objektive und den unscharfen Bildern? Arducam hat jetzt ein multifunktionales Tool für Objektive veröffentlicht, mit dem Sie das Sichtfeld des Objektivs ohne Berechnung erhalten und den Objektivfokus schnell und einfach kalibrieren können.

31.2. Applications

Focus calibration, sharpness estimation and field of view quick measuring for M12, CS-Mount, C-Mount, and DSLR lenses



Figure 31.1.: Kalibrierungswerkzeug der Firma Arducam; [Ard21]

31.3. Package Contents

2*Foldable Lens Calibration Card

32. BlueTooth

- <https://www.instructables.com/Arduino-NANO-33-Made-Easy-BLE-Sense-and-IoT/>
- <https://www.hackster.io/sridhar-rajagopal/control-arduino-nano-ble-with-blue>
- <https://docs.arduino.cc/tutorials/nano-33-ble-sense/ble-device-to-device>
- <https://projecthub.arduino.cc/8bitkick/sensor-data-streaming-with-arduino-a6>
- <https://www.okdo.com/getting-started/get-started-with-arduino-nano-33-sense/>
- <https://rootsaid.com/arduino-ble-example/>

32.1. Quick Start

This tutorial shows you how to use the free pfodDesignerV3 V3.0.3774+ Android app to create a general purpose Bluetooth Low Energy (BLE) and WiFi connection for Arduino NANO 33 boards without doing any programming. There are three (3) Arduino NANO 33 boards, the NANO 33 BLE and NANO 33 BLE Sense, which connect by BLE only and the NANO 33 IoT which can connect via BLE or WiFi. Connecting using pfodApp is the most flexible way to connect via BLE (or WiFi). See Using pfodApp to connect to the NANO 33 BLE (Step 4) and Using pfodApp to connect to the NANO 33 IoT via BLE (Step 7) and Using pfodApp to connect to the NANO 33 IoT via WiFi (Step 9) below. However simple sketches are also provided to send user defined command words via Telnet, for WiFi, or via the free Nordic nRF UART 2.0 for BLE. See Using the Nordic nRF UART 2.0 app to connect to NANO 33 BLE (Step 5) and Using the Nordic nRF UART 2.0 app to connect to the NANO 33 IoT via BLE (Step 8) and Using a Telnet terminal program to connect to the NANO 33 IoT via WiFi (Step 10) below. This tutorial is also available on-line at Arduino NANO 33 Made Easy.

32.2. Supplies

Arduino NANO 33 – either BLE or Sense or IoT
optionally pfodDesignerV3 and pfodApp

32.3. Step 1: Introduction

There are a number of problems with BLE. See this page for BLE problems and solutions and there are some BLE trouble shooting tips. The learning curve is steep and the specification has hundreds of specialise connect services each of which requires its own mobile application to connect to. This tutorial shows you how to generate Arduino code for a general purpose Nordic UART BLE connection over which you can send and receive a stream of commands and data to a general purpose BLE UART mobile application. The free pfodDesignerV3 Android application is used to generate the Arduino code. The output is designed to connect to the paid pfodApp Android

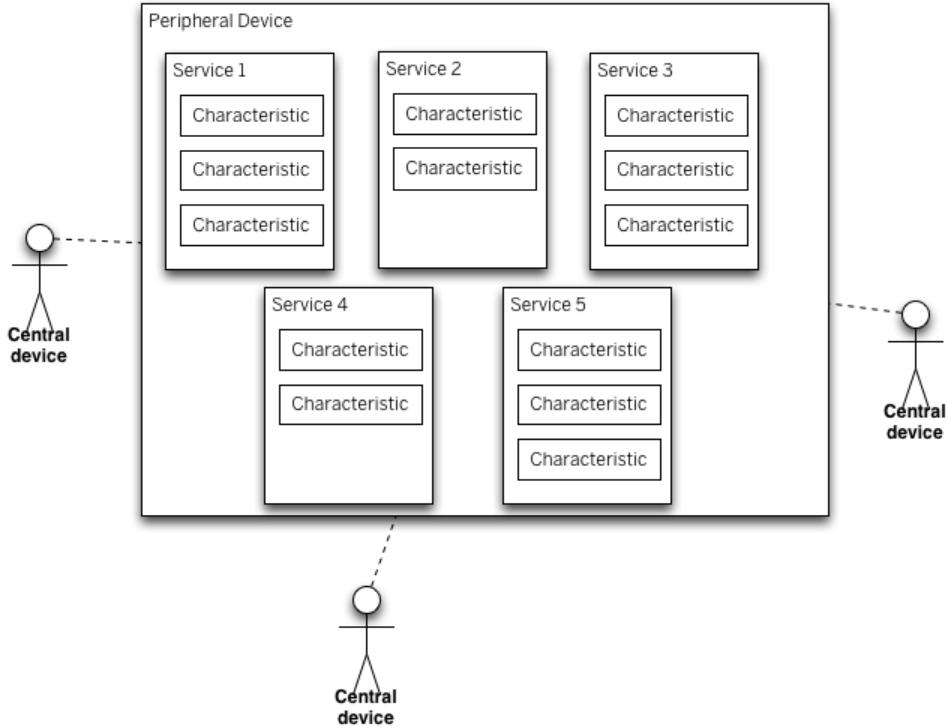


Figure 32.1.: BLE Devices – Peripheral and Central Devices

application which can display menus, send command, log data and show charts. No Android programming is required. The Arduino code has complete control over what is displayed by pfodApp.

For the NANO 33 IoT you can also connect via WiFi. Again the pfodDesignerV3 generates all the Arduino code and by default is designed to connect to pfodApp with optional 128bit security.

However you do not need to use pfodApp, you can connect to the generated code using the free Nordic nRF UART 2.0 or a Telnet programs (for the WiFi connection). Sketches are included which provide command words to control the boards.

The free pfodDesigner V3.0.3774+ will generate Arduino code for a wide range of boards and connection types including Serial connections, Bluetooth Low Energy (BLE), WiFi, SMS, Radio/LoRa, Bluetooth Classic and Ethernet. For examples Arduino code for a wide range of BLE boards see Bluetooth Low Energy (BLE) made simple with pfodApp. Here we will be using a BLE connection for NANO 33 BLE, Sense and IoT and a WiFi connection for the IoT.

Each of the NANO 33 boards has extra sensor components that differ between the three (3) boards. The pfodDesignerV3 generates code to read/write the digital outputs and perform analogReads and analogWrites. In the examples below we will turn the board LED on and off and read the voltage at A0 and log and plot it. Once that sketch is running you can add each board's specialised sensor libraries and sent their data in place of the analogRead(A0).

pfodDesigner generates simple menus and charts, however you can also program custom graphical interfaces in your Arduino sketch. Above is an example of slider control adjusting a guage. All the code for this control is in your Arduino sketch. No Android programming necessary. See Custom Arduino Controls for more examples.

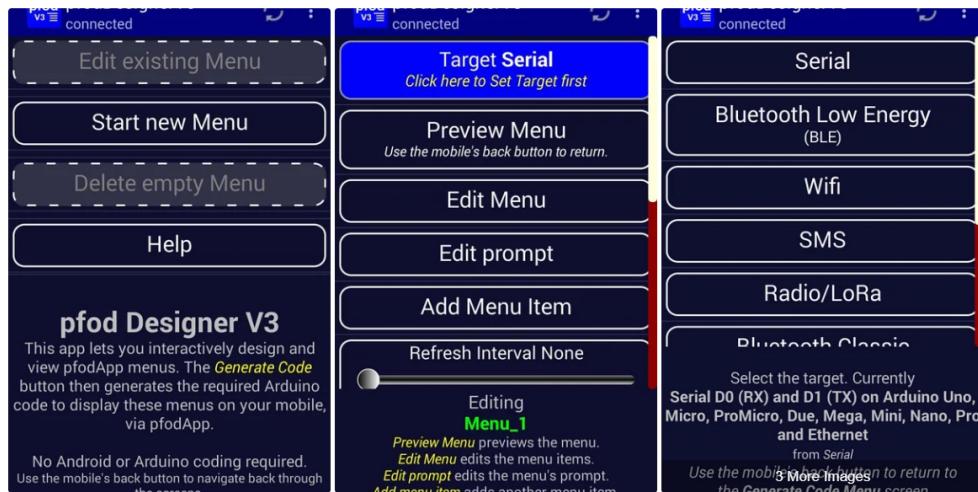


Figure 32.2.: Step 2: Creating the Custom Android Menus and Generating the Code

32.4. How pfodApp is optimised for short BLE style messages

Bluetooth Low Energy (BLE) or Bluetooth V4 is a completely different version of Bluetooth. BLE has been optimised for very low power consumption. pfodApp is a general purpose Android app whose screens, menus, buttons, sliders and plots are completely defined by the device you connect to.

BLE only sends 20 bytes in each message. Fortunately the pfod Specification was designed around very small messages. Almost all of pfod's command are less than 20 bytes. The usual exception is the initial main menu message which specifies what text, menus, buttons, etc. pfodApp should display to the user, but the size of this message is completely controlled by you and you can use sub-menus to reduce the size of the main menu.

The pfod specification also has a number of features to reduce the message size. While the BLE device must respond to every command the pfodApp sends, the response can be as simple as (an empty response). If you need to update the menu the user is viewing in response to a command or due to a re-request, you need only send back the changes in the existing menu, rather than resending the entire menu. These features keep the almost all messages to less than 20 bytes. pfodApp caches menus across re-connections so that the whole menu only needs to be sent once. Thereafter short menu updates can be sent.

32.5. Step 2: Creating the Custom Android Menus and Generating the Code

Before looking at each of these BLE modules, pfodDesignerV3 will first be used to create a custom menu to turn a Led on and off and plot the voltage read at A0. pfodDesignerV3 can then generate code tailored to the particular hardware you select. You can skip over this step and come back to it later if you like. The section on each module, below, includes the completed code sketch for this example menu generated for that module and also includes sketches that do not need pfodApp

The free pfodDesignerV3 is used to create the menu and show you an accurate preview of how the menu will look on your mobile. The pfodDesignerV3 allows you to create menus and sub-menus with buttons and sliders optionally connected to I/O

pins and generate the sketch code for you (see the pfodDesigner example tutorials) but the pfodDesignerV3 does not cover all the features pfodApp supports. See the pfodSpecification.pdf for a complete list including data logging and plotting, multi- and single- selections screens, sliders, text input, etc.

Create the Custom menu to turn the Arduino LED on and off and Plot A0 Start a new menu and select as a target Bluetooth Low Energy (BLE) and then select NANO 33 BLE (and Sense). pfodDesignerV3.0.3770+ has support for NANO 33 boards.

Then follow the tutorial Design a Custom menu to turn the Arduino Led on and off for step by step instructions for creating a LED on/off menu using pfodDesignerV3.

If you don't like the colours of font sizes or the text, you can easily edit them in pfodDesignerV3 to whatever you want and see a WYSIWYG (What You See Is What You Get) display of the designed menu.

Now we will add a Chart button to display the A0 reading. The steps to do this in pfodDesignerV3 are shown in Adding a Chart and Logging Data The AtoD range is 0 to 1023 for 0 to 3.3V

Generating the code from pfodDesignerV3 gives the this sketch [Nano33BLE_Led_A0.ino](#)

32.6. BLE Mobile App “Nordic Semiconductors nRF Connect”

- On your mobile, install the App “Nordic Semiconductors nRF Connect”. There are Android and IOS versions.
- In the Arduino IDE open Serial Monitor by clicking on the magnifying glass icon on the top right.

The Nano will start sending BLE advertising packets and wait for a Central (Mobile Phone) to connect.

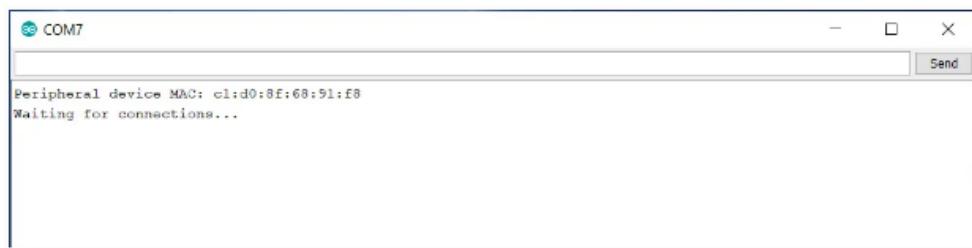


Figure 32.3.: App nRF is searching the Arduino Nano BLE Sense

In the App nRF on your mobile select **Scan** and you should see **Nano33BLE** as a device you can connect to.

Attention: BLE Peripherals can only connect to one device at a time – if the Nano33BLE is already connected it could be to another device nearby.

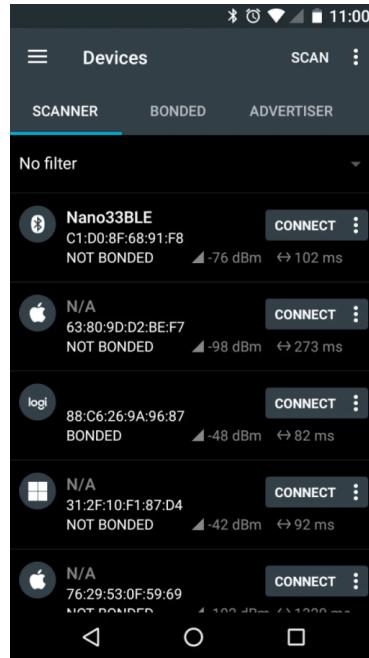


Figure 32.4.: App nRF is searching the Arduino Nano BLE Sense

Connect to the Nano and select the Unknown Service.

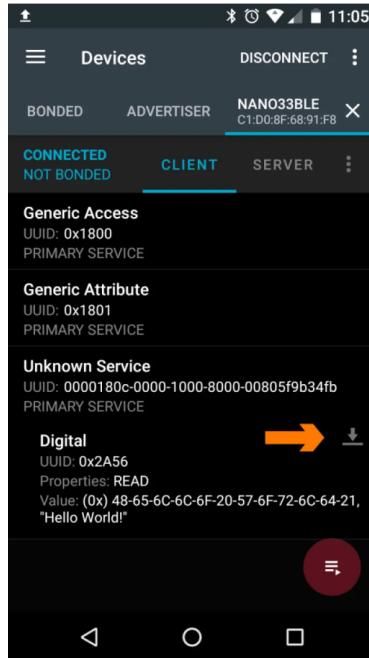


Figure 32.5.: App nRF is getting data the Arduino Nano BLE Sense

Touch the **down arrow** next to the characteristic **Digital** and it will read the message "Hello World!" being broadcast by the Nano33BLE.

32.7. Arduino BLE Example – Explained Step by Step

32.7.1. Arduino BLE Example Code Explained

In this tutorial series, I will give you a basic idea you need to know about Bluetooth Low Energy and I will show you how you can make Arduino BLE Chipset to send and receive data wirelessly from mobile phones and other Arduino boards. Let's Get Started.

Arduino Nano 33 BLE Sense, Nano with BLE connectivity focussing on IOT, which is packed with a wide variety of sensors such as 9 axis Inertial Measurement Unit, pressure, light, and even gestures sensors and a microphone.

It is powered by Nina B306 module that supports BLE as well as Bluetooth 5 connection. The inbuilt Bluetooth module consumes very low power and can be easily accessed using Arduino libraries. This makes it easier to program and enable wireless connectivity to any of your projects in no time. You won't have to use external Bluetooth modules to add Bluetooth capability to your project. Save space and power.

32.7.2. Arduino BLE – Bluetooth Low Energy Introduction

BLE is a version of Bluetooth which is optimized for very low power consuming situations with very low data rate. We can even operate these devices using a coin cell for weeks or even months.

Arduino have a wonderful introduction to BLE but here in this [WS site](#) will give you a brief introduction for you to get started with BLE communication.

Basically, there are two types of devices when we consider a BLE communication block.

- The Peripheral Device
- The Central Device

Peripheral Device is like a Notice board, from where we can read data from various notices or pin new notices to the board. It posts data for all devices that needs this information.

Central Devices are like people who are reading notices from the notice board. Multiple users can read and get data from the notice board at the same time. Similarly multiple central devices can read data from the peripheral device at the same time.

The information that is given by the Peripheral devices are structured as Services. And These services are further divided into characteristics. Think of Services as different notices in the notice board and services as different paragraphs in each notice board. If Accelerometer is a service, then their values X, Y and Z can be three characteristics. Now let's take a look at a simple Arduino BLE example.

32.7.3. Arduino BLE Example 1 – Battery Level Indicator

In this example, I will explain how you can read the level of a battery connected to pin A0 of an Arduino using a smartphone via BLE. This is the code here. This is pretty much the same as that of the example code for Battery Monitor with minor changes. I will explain it for you.

First you have to install the library ArduinoBLE from the library manager.

Just go to [Sketch -> Include Library -> Manage Library](#) and Search for [ArduinoBLE](#) and simply install it.

```
1000 #include <ArduinoBLE.h>
1001 BLEService batteryService("1101");
1002 BLEUnsignedCharCharacteristic batteryLevelChar("2101", BLERead | BLENotify);
1003
1004 void setup() {
1005     Serial.begin(9600);
1006     while (!Serial);
1007
1008     pinMode(LED_BUILTIN, OUTPUT);
1009     if (!BLE.begin())
1010     {
1011         Serial.println("starting BLE failed!");
1012         while (1);
1013     }
1014
1015     BLE.setLocalName("BatteryMonitor");
1016     BLE.setAdvertisedService(batteryService);
1017     batteryService.addCharacteristic(batteryLevelChar);
1018     BLE.addService(batteryService);
1019
1020     BLE.advertise();
1021     Serial.println("Bluetooth device active, waiting for connections...");
1022 }
1023
1024 void loop()
1025 {
1026     BLEDevice central = BLE.central();
1027
1028     if (central)
1029     {
1030         Serial.print("Connected to central: ");
1031         Serial.println(central.address());
1032         digitalWrite(LED_BUILTIN, HIGH);
1033
1034         while (central.connected()) {
1035
1036             int battery = analogRead(A0);
1037             int batteryLevel = map(battery, 0, 1023, 0, 100);
1038             Serial.print("Battery Level % is now: ");
1039             Serial.println(batteryLevel);
1040             batteryLevelChar.writeValue(batteryLevel);
1041             delay(200);
1042
1043         }
1044     }
1045     digitalWrite(LED_BUILTIN, LOW);
1046     Serial.print("Disconnected from central: ");
1047     Serial.println(central.address());
1048 }
```

Listing 32.1.: Arduino BLE Tutorial Battery Level Indicator Code

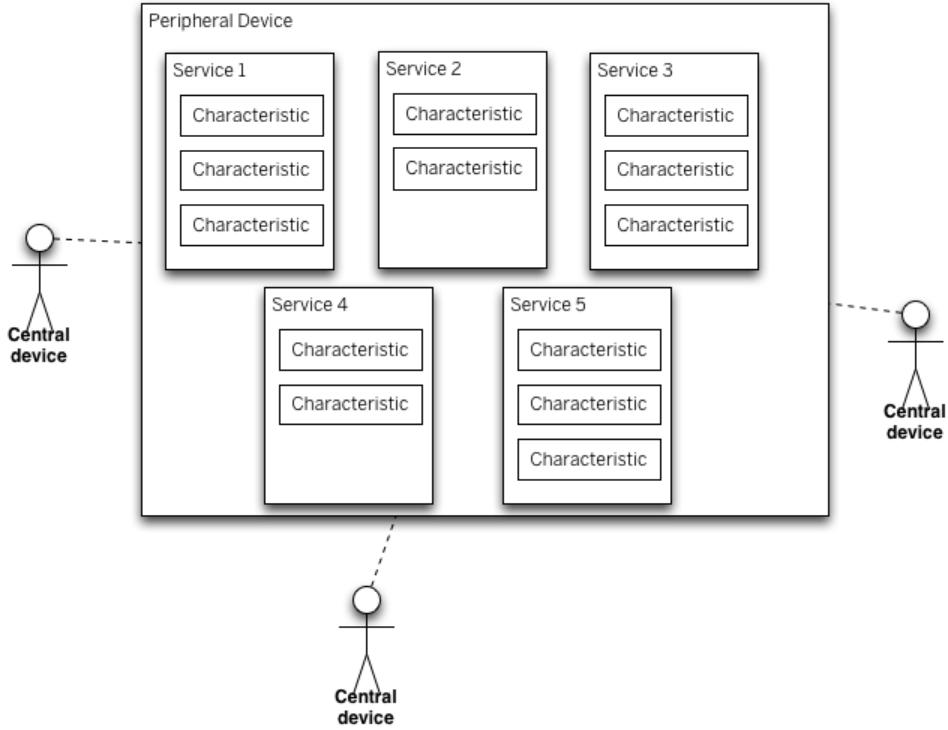


Figure 32.6.: BLE Devices – Peripheral and Central Devices

32.7.4. Arduino BLE Tutorial Battery Level Indicator Code

32.7.5. Arduino Bluetooth Battery Level Indicator Code Explained

```
#include <ArduinoBLE.h>
BLEService batteryService("1101");
BLEUnsignedCharCharacteristic batteryLevelChar("2101", BLERead | BLENotify);
```

The first line of the code is to include the file `ArduinoBLE.h`. Then we will declare the Battery Service as well the battery level characteristics here. Here we will be giving two permissions – `BLERead` and `BLENotify`.

`BLERead` will allow central devices (Mobile Phone) to read data from the Peripheral device (Arduino). And `BLENotify` allows remote clients to get notifications if this characteristic changes.

Now we will jump on to the Setup function.

```
Serial.begin(9600);
while (!Serial);
pinMode(LED_BUILTIN, OUTPUT);
if (!BLE.begin()) {
    Serial.println("starting BLE failed!");
    while (1);
}
```

Here it will initialize the Serial Communication and BLE and wait for serial monitor to open.

Set a local name for the BLE device. This name will appear in advertising packets and can be used by remote devices to identify this BLE device.

```
BLE.setLocalName("BatteryMonitor");
```

```
BLE.setAdvertisedService(batteryService);
batteryService.addCharacteristic(batteryLevelChar);
BLE.addService(batteryService);
```

Here we will add and set the value for the Service UUID and the Characteristic.

```
BLE.advertise();
Serial.println("Bluetooth device active , waiting for connections ...");
```

And here, we will Start advertising BLE. It will start continuously transmitting BLE advertising packets and will be visible to remote BLE central devices until it receives a new connection.

```
BLEDevice central = BLE.central();
if (central) {
    Serial.print("Connected to central: ");
    Serial.println(central.address());
    digitalWrite(LED_BUILTIN, HIGH);
```

And here, the loop function. Once everything is setup and have started advertising, the device will wait for any central device. Once it is connected, it will display the MAC address of the device and it will turn on the builtin LED.

```
while (central.connected()) {
    int battery = analogRead(A0);
    int batteryLevel = map(battery, 0, 1023, 0, 100);
    Serial.print("Battery Level % is now: ");
    Serial.println(batteryLevel);
    batteryLevelChar.writeValue(batteryLevel);
    delay(200);
}
```

Now, it will start to read analog voltage from A0, which will be a value in between 0 and 1023 and will map it with in the 0 to 100 range. It will print out the battery level in the serial monitor and the value will be written for the batteryLevelchar characteristics and waits for 200 ms. After that the whole loop will be executed again as long as the central device is connected to this peripheral device.

```
digitalWrite(LED_BUILTIN, LOW);
Serial.print("Disconnected from central: ");
Serial.println(central.address());
Once it is disconnected, a message will be shown on the central device and LE
```

32.7.6. Installing the App for Android

In your Android smartphone, install the app “nRF Connect”. Open it and start the scanner. You will see the device “Battery Monitor” in the device list. Now tap on connect and a new tab will be opened.

Go to that and you will see the services and characteristics of the device. Tap on Battery service and you will the battery levels being read from the Arduino.

32.7.7. Example 2 – Arduino BLE Accelerometer Tutorial

I showed you a very easy example to show you how you can send simple data via Bluetooth. If you are new to this, try this example first. It will help to give you a better understanding of the BLE.

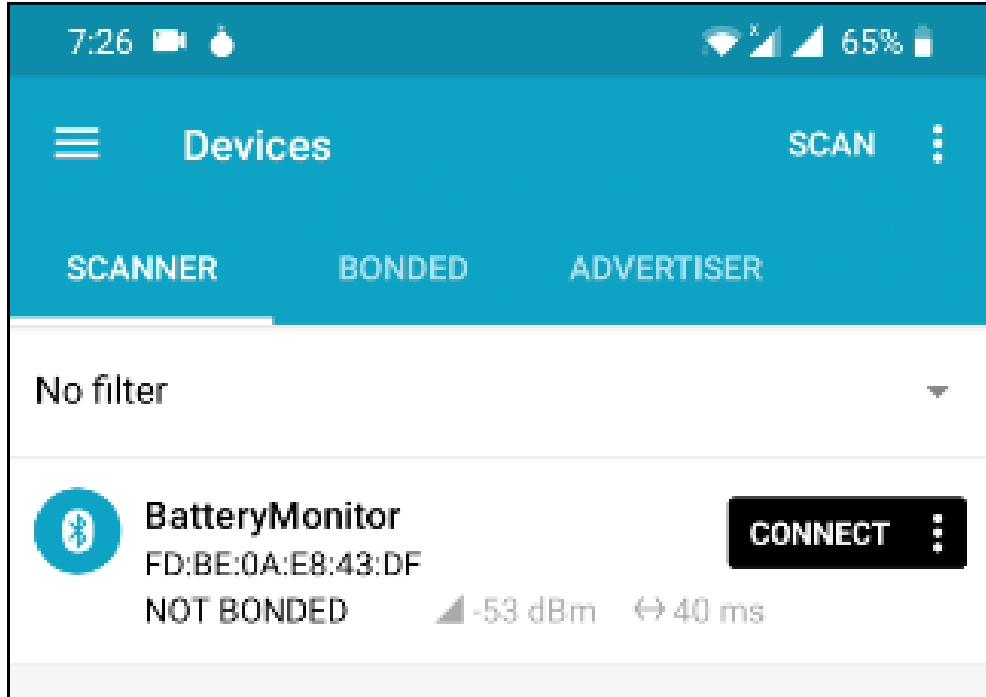


Figure 32.7.: Battery app “nRF Connect”

Step 1 – Installing Libraries

For this example, we will need two libraries

- [ArduinoBLE](#) – To Send Data via Bluetooth
- [LSM9DS1](#) – To Read data from inbuilt Accelerometer

Both these libraries are available in library manager. Simply search for that using the name and click on install.

Step 2 – Test Accelerometer Code (Optional)

Now we will try running the accelerometer code to make sure that these data are being read properly. For that, use the below code.

```
#include <Arduino_LSM9DS1.h>

void setup() {
    Serial.begin(9600);
    while (!Serial);
    Serial.println("Started");

    if (!IMU.begin()) {
        Serial.println("Failed to initialize IMU!");
        while (1);
    }

    Serial.print("Accelerometer sample rate = ");
    Serial.print(IMU.accelerationSampleRate());
    Serial.println(" Hz");
}
```

```

    Serial.println();
    Serial.println("Acceleration in G's");
    Serial.println("XtYtZ");
}

void loop() {
    float x, y, z;

    if (IMU.accelerationAvailable()) {
        IMU.readAcceleration(x, y, z);

        Serial.print(x);
        Serial.print('t');
        Serial.print(y);
        Serial.print('t');
        Serial.println(z);
    }
}

```

Once uploaded, start the serial monitor. You will see the data being populated. Try tilting the board in all direction and you will see the value changes accordingly.

Step 3 – Upload the Code

Now its time to upload the complete code. Copy the code below and paste it in the IDE.

```

#include <ArduinoBLE.h>
#include <Arduino_LSM9DS1.h>

int accelX=1;
int accelY=1;
float x, y, z;

BLEService customService("1101");
BLEUnsignedIntCharacteristic customXChar("2101", BLERead | BLENotify);
BLEUnsignedIntCharacteristic customYChar("2102", BLERead | BLENotify);

void setup() {
    IMU.begin();
    Serial.begin(9600);
    while (!Serial);

    pinMode(LED_BUILTIN, OUTPUT);

    if (!BLE.begin()) {
        Serial.println("BLE failed to initiate");
        delay(500);
        while (1);
    }

    BLE.setLocalName("Arduino Accelerometer");
    BLE.setAdvertisedService(customService);
    customService.addCharacteristic(customXChar);
    customService.addCharacteristic(customYChar);
    BLE.addService(customService);
}

```

```

customXChar.writeValue(accelX);
customYChar.writeValue(accelY);

BLE.advertise();

Serial.println("Bluetooth device is now active , waiting for connections ...");
}

void loop() {
    BLEDevice central = BLE.central();
    if (central) {
        Serial.print("Connected to central:");
        Serial.println(central.address());
        digitalWrite(LED_BUILTIN, HIGH);
        while (central.connected()) {
            delay(200);
            read_Accel();

            customXChar.writeValue(accelX);
            customYChar.writeValue(accelY);

            Serial.print("At Main Function");
            Serial.println("");
            Serial.print(accelX);
            Serial.print(" - ");
            Serial.println(accelY);
            Serial.println("");
            Serial.println("");
        }
    }
    digitalWrite(LED_BUILTIN, LOW);
    Serial.print("Disconnected from central:");
    Serial.println(central.address());
}

void read_Accel() {
    if (IMU.accelerationAvailable()) {
        IMU.readAcceleration(x, y, z);
        accelX = (1+x)*100;
        accelY = (1+y)*100;

    }
}

```

Select the right port and board. Click on upload.

Step 4 – Testing Arduino BLE Accelerometer

In your Android smartphone, install the app “nRF Connect”. Open it and start the scanner. You will see the device “Arduino Accelerometer” in the device list. Now tap on connect and a new tab will be opened.

Go to that and you will see the services and characteristics of the device.

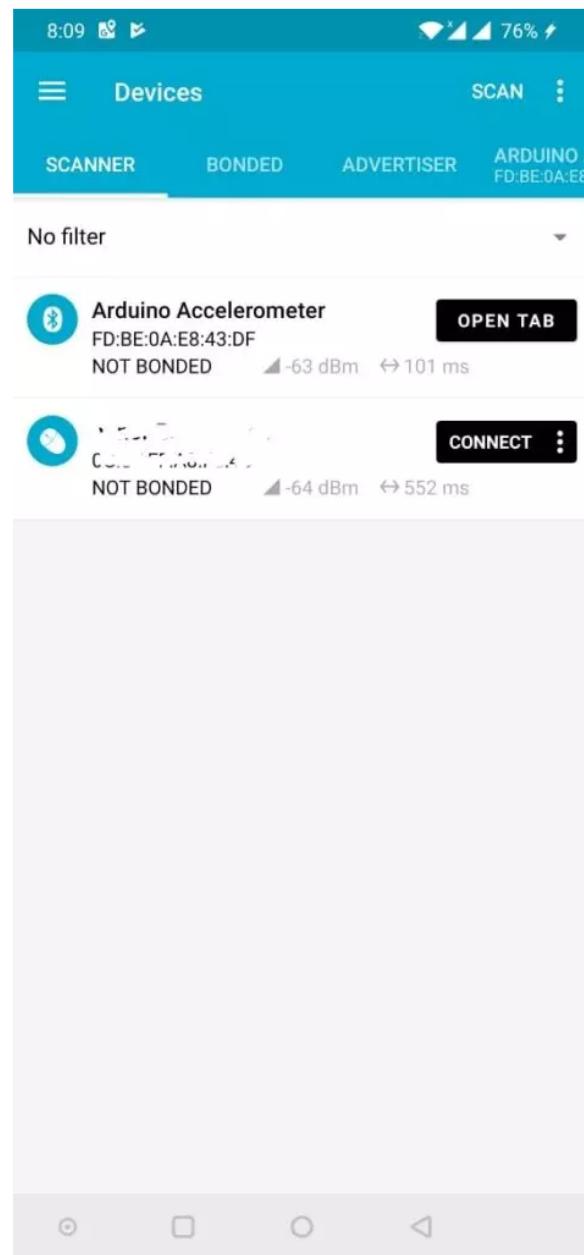


Figure 32.8.: nRF Connect Screen

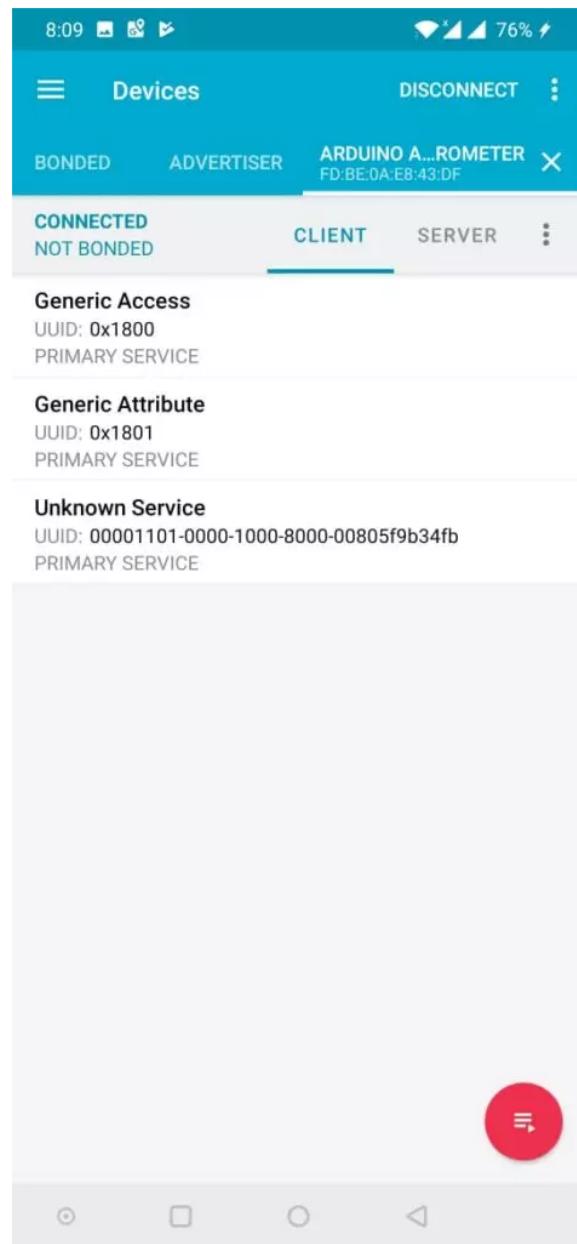


Figure 32.9.: Accelerometer Values Read from Arduino Using BLE

Tap on Unknown Service and you will see the accelerometer values being read from the Arduino.

In the next post, I will show you how you can send inbuilt sensor values such as accelerometer, gyroscope, color sensor and gesture sensor from the Arduino to your phone as well as another Arduino via BLE.

32.8. Arduino Library [BLEAK](#)

Low Energy (BLE) protocol. BLE allows you to exchange data between devices wirelessly and efficiently. You can use the bleak library in Python to interact with [BLE devices](#).

To get started, you need to install the bleak library using pip:

```
pip install bleak
```

Then, you need to write a Python program that can scan for BLE devices, connect to the Arduino Nano 33 BLE 33 Sense, and read or write data to its characteristics. Characteristics are the attributes that define the data and behavior of a BLE device. [For example, the Arduino Nano 33 BLE 33 Sense has a characteristic for each color of its RGB LED.](#)

You can find an example of a Python program that can control the RGB LED of the Arduino Nano 33 BLE 33 Sense using bleak [here3](#). You can also find the corresponding Arduino sketch that sets up the BLE service and characteristics for the Nano 33 BLE 33 Sense [here4](#).

Part IV.

Projects

33. TinyML and Edge Computing

33.1. TinyML

“TinyML(Tiny Machine Learning) is a technology that can be used to develop embedded low power consuming devices to run both machine and deep learning models” [Gup20]. It requires simple procedures for training the model created. The amount of data required for creating the model is significantly less compared to other platforms. Tiny ML also works on battery powered devices and can be used for various applications including environmental monitoring.

TinyML focuses on application design, development, deployment of different models which can be transformed in to a large scale ML model. This approach mainly removes the barriers of traditional style ML models and allows users to develop, deploy and implement applications seamlessly.” TinyML permits a wide range of new applications that traditional ML cannot deliver because of bandwidth, latency, economics, reliability, and privacy (BLERP) limitations.“ [RPW21] Some of the common applications of TinyML includes keyword spotting, wake word detection, person detection, gesture detection and much more. Keyword spotting generally refers to identification of words that typically act as part of a cascade architecture to kick-start or control a system, such as a mobile phone responding to voice commands [RPW21]. Visual wake words involve parsing image data to find an individual (human or animal) or object which can be used for security systems and for intelligent lighting.

33.2. Real Time

Real Time can be considered as a way of training the model by making it run through live data constantly in order to improve the model. This contradicts the traditional way of machine learning when machine learning engineers were dependent already existent data inputs for creating the model. A method by which we can implement real time learning in to a machine learning model is by continuously feeding it with a data stream and improving the model over time. This is achieved by using an event driven architecture. Event driven architecture is a modern design and development approach that centers about the events occurring in the model. Event-driven architectures create, detect, consume, and react to events.[Haz]. By using a scalable event driven architecture, large number of events in real time can be responded, to which are suitable for loosely coupled softwares(For eg: web services). They also work well with unpredictable and non linear events by making it versatile. [Haz]

33.3. Edge computing

The placement of computer and storage resources at the point where data produced is known as edge computing. This computes and stores the data source at the network edge, which is optimal. In other words, instead of sending raw data to a main data centre for processing and analysis, this work is done right where the data is generated. Edge computing is used to handle discrete tasks like determining if someone answered “yes” and responding appropriately. Instead of comparing the data with that on the web, the audio analysis is done on the edge. This drastically decreases expenses and complexity while also reducing the risk of data breaches.[Big21]

Edge computing has gained traction as a viable solution to a variety of issues related to transporting the massive amounts of data that today's businesses generate and consume. It's not just a matter of quantity, it's also a matter of time; applications increasingly rely on processing and responses that are time-sensitive.

34. TensorFlow Lite

34.1. TensorFlow Lite

“TensorFlow Lite is an open-source, product ready, cross-platform deep learning framework that converts a pre-trained model in TensorFlow to a special format that can be optimized for speed or storage” [Kha20]. The special format model can be deployed on edge devices like mobiles, embedded devices or Microcontrollers. by quantization and weight pruning TensorFlow Lite achieves optimization and it reduces the size of the model or improves the latency. Figure ?? shows the workflow of TensorFlow Lite.

34.2. What is TensorFlow Lite?

Often the trained models are not to be used on the PC on which they were trained, but on mobile devices such as mobile phones or microcontrollers or other embedded systems. However, limited computing and storage capacities are available there. TensorFlow Lite is a whole framework that allows the conversion of models into a special format and their optimisation in terms of size and speed, so that the models can then be run with TensorFlow Lite interpreters on mobile, embedded and IoT devices. [Goo20][WS20]

TensorFlow Lite is a „light“ version of TensorFlow designed for mobile and embedded devices. It achieves low-latency inference in a small binary size - both the TensorFlow Lite models are much smaller. However, you cannot train a model directly with TensorFlow Lite, it must first be created with TensorFlow, then you must convert the model from a TensorFlow file to a TensorFlow Lite file using the TensorFlow Lite converter. [Goo11] The reason is that TensorFlow models usually calculate in 32-bit floating point numbers for the neural networks. The weights can assume very small values close to zero. Therefore, the models cannot be run on systems that cannot calculate with long floating point numbers, but only with 8-bit integers. This is especially the case with small AI processors, on which only a few transistors can be accommodated due to size and power consumption. Therefore, the neural network must undergo a transformation, „in which long floating-point numbers with varying precision over the representable range of values – floating-point numbers represent the range of numbers around the zero point much more finely than very large or small values – become short integers with constant precision in a limited range of values“ [RA20]. This quantisation takes place during the conversion to the TensorFlow Lite format.

34.3. Procedure

The process for using TensorFlow Lite consists of four steps:

1. Choice of a model:

An existing, trained model can be adopted or trained yourself, or a model can be created from scratch and trained.

2. Conversion of the model:

If a custom model is used that is accordingly not in TensorFlow Lite format, it must be converted to the format using the TensorFlow Lite converter.

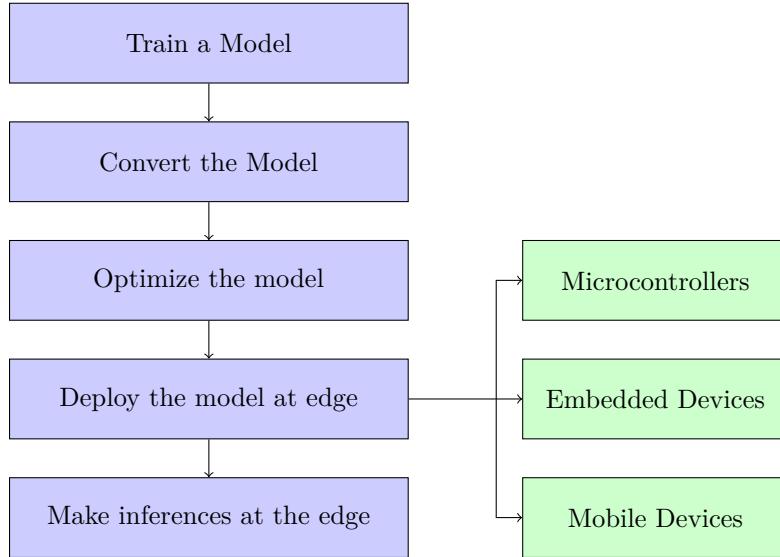


Figure 34.1.: Tensorflow Lite workflow [Kha20]

3. deployment on the end device:

To run the model on the device, the TensorFlow Lite interpreter is available with APIs in many languages.

4. Optimisation of the model:

Other optimisation tools are available to improve model size and execution speed. For example, the model can be quantised by converting 32-bit floating point numbers to more efficient 8-bit integers. With the TensorFlow Lite converter, TensorFlow models are converted into an optimised FlatBuffer format so that they can be used by the TensorFlow Lite interpreter. [Goo19b]

The figure 34.2 illustrates the basic process for creating a model that can be used on an edge computer. Most of the process uses standard TensorFlow tools.

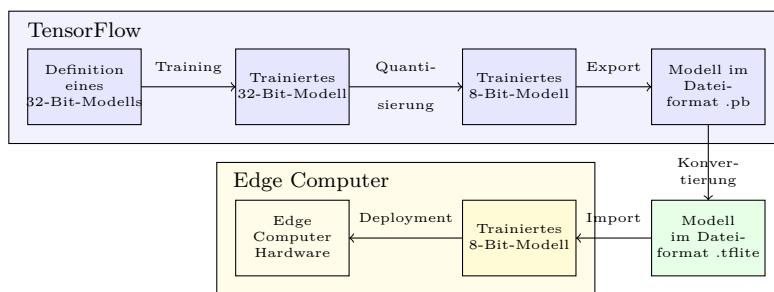


Figure 34.2.: The basic process for creating a model for an edge computer[Goo19a]

34.4. TensorFlow Lite Converter

The TensorFlow Lite Converter is a tool available as a Python API that converts trained TensorFlow models into the TensorFlow Lite format. This is done by converting the Keras model into the form of a FlatBuffer, a special space-saving file format. In

general, converting models reduces file size and introduces optimisations without compromising accuracy. The TensorFlow Lite converter continues to offer options to further reduce file size and increase execution speed, with some trade-offs in accuracy. [Goo20][WS20]

One way to optimise is quantisation. While the weights of the models are usually stored as 32-bit floating point numbers, they can be converted to 8-bit integers with only a minimal loss in the accuracy of the model. This saves memory and also allows for faster computation, as a Central Processing Unit (CPU) works more efficiently with integers. [WS20]

34.5. TensorFlow Lite Interpreter

The TensorFlow Lite interpreter is a library that executes an appropriately converted TensorFlow Lite model using the most efficient operations for the given device, applying the operations defined in the model to the input and providing access to the output. It works across platforms and provides a simple API to run TensorFlow Lite models from Java, Swift, Objective-C, C ++ and Python.[Goo20][WS20]

To use hardware acceleration on different devices, the TensorFlow Lite interpreter can be configured with „delegates“. These use device accelerators such as an Graphics Processing Unit (GPU) or a digital signal processor (DSP). [Goo20]

35. Introduction

Tiny Machine Learning (TinyML), is a rapidly growing subfield of applied ML. This area focuses on deploying simple yet powerful models on extremely low-power, low-cost microcontrollers at the network edge. TinyML models require relatively small amounts of data, and their training can employ simple procedures.

Furthermore, as TinyML can run on microcontroller development boards with extensive hardware abstraction, such as Arduino products, deploying an application onto hardware is easy. TinyML systems working in concert at the “edge” of the cloud-computing network.[RPW21]

This report discusses about Arduino and its environment and how we have implemented its functions in our project. Arduino is a development platform that can be used for easy integration of software and hardware. The boards by Arduino enables a user to read the input, and turn it into an output. The Arduino can be programmed according to the needs required by the user with a set of commands(Arduino Programming Language) using the Arduino IDE. [Ard21] The report focusses on the hardware Arduino Nano BLE33 Sense which is a 3.3V AI enabled board. The Nano 33 BLE sense board is occupied three axis accelerometers.

This implies that the sensor is able to detect the motion and in turn enables to calculate the acceleration produced in three directions. The objective of this report is to create a “Magic Wand” and implement the same into an Arduino Nano BLE33 sense board. Gestures namely “wing”, “ring” and “slope” will be recognised by the board when the wand is waved. The direction of motion for the recognition of the gestures are shown in the figure 35.1.

The Arduino can also be programmed such that the LED the hardware blinks according to the commands given in relation to the gestures waved by the hand. For example , the Arduino can be programmed to indicate a green light when a “wing” is waved , red for “slope” and blue for “ring”.

When the wand is moved in the direction as shown in the figure, the gesture interpreted by the board is transformed into a visual output.

To construct magic wand, attach the Arduino Nano BLE 33 sense board to the end of a stick. Then feed the accelerometer’s output into a deep learning model, which will perform classification to tell whether a known gesture was made. [WS20]

Gestures collected using the built-in multi-dimensional sensor on the Arduino board are able to understand the complex data and the model can be trained such that it understands the complex data and embeds them into a microcontroller that is Arduino Nano 33 BLE Sense. Specifically, it uses TensorFlow Lite to run Convolutional Neural Network model to recognize gestures with an accelerometer. If you’re successful casting the spells, you will see the corresponding gesture as a visual output on the screen and the Arduino board should be lit according to the command given by the user.

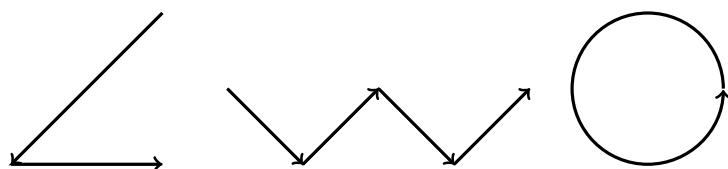


Figure 35.1.: Gestures used namely Slope, W and ring

Heuristic function is used for this application, which is a function that expresses the knowledge of gestures as a function in code. To create a heuristic we need domain knowledge, programming and mathematical expertise. A heuristic algorithm expresses human knowledge and understanding. The data obtained from the accelerator is mathematically converted to get a desired output or the gestures which we want to cast.

Instead of designing a heuristic algorithm from scratch, a machine learning developer can find a suitable model architecture, collect and label a dataset, and iteratively create a model through training and evaluation. [WS20]

There is no pre-processing done in the model, instead the accelerometer values are directly taken as input. Then it runs inference and then processes the output. The gestures trained are wing (w), ring (o) and slope (\angle) and if the input is recognized as valid, it prints a visual output of the gestures on the terminal and it reacts to each spell by lighting an LED. In case any of the gestures are not recognised, there will also be an output to notify that the gesture is "unknown".

From the initial planning phase, various challenges encountered are:-

- Creating a dataset on the 4 gestures , especially unknown gesture as the amount of data to be recorded for the unknown gesture has no limit and would require a concrete database.
- Gesture should have a minimum wave time of 5 seconds, else it may give false readings.
- Gestures waved should not be of a very long time duration which might make the accelerometer to record false reading giving a false output.
- Programming the Arduino Nano.
- Interference on cheap or low power hardware.
- Incorporating Arduino and the Magic Wand.

Most of the challenges faced were addressed with knowledge imparted from the book TinyML. For programming the Arduino, the Arduino guide was referred to and the application required was downloaded from the Arduino website.

The report initially talks about the Domain Knowledge acquired for - Magic Wand-Arduino Nano BLE33 Sense and other components involved in the project, special emphasis is given to IMU sensors (LSM9DS1). The involvement of Convolutional Neural Network (CNN) in the project is discussed in the next section followed by the application of Knowledge Discovery in Databases (KDD) processes. The KDD section discusses about the steps involved in creating a target dataset, selection, preparation and transformation of the dataset followed by data mining which discusses about the patterns found in the database.

The report further talks about the implementation and deployment of the known results obtained from the initial phase, it then concludes with the future scope and the improvements that can be incorporated in the future.

36. Project Magic Wand

36.1. Development

This section talks about how the KDD process is implemented and the steps followed for each of the KDD Processes starting with the preparation of Databases

36.1.1. Database

Since there are no existing databases for the model a new database will be created. The database is a .txt file which contains raw data from the accelerometer readings recognized by the Arduino Nano 33 BLE Sense. Since the way how the gesture is recorded by a person varies from one to another, gestures of three people are recorded as we are a group of three members. In order for the database to have a vast amount of data, each person will record a minimum of 50 trials for each gesture. In order for the database to be more accurate data from both hands of the three users were used. This would improve the efficiency and effectiveness of the database making it more concrete. There is also a possibility to add further data into our model which would improve our model over time. The size of the database thus created will be very small that would be less than 2 MB. Obtaining sufficient data is one of the major concerns in a machine learning project and the amount of data involved in our data set is very small. The data thus created will be further used in the next step “Data Preparation and Transformation” where this raw data is further processed and the outliers are identified and removed if necessary. [WS20]

36.1.2. Data Preparation

This section discusses about how to shape our data and how we can use it for training. Three gestures are to be recognized by our machine learning model namely wing, slope and ring. Since we are not intending to use any existing databases, in order to prepare the data we will give multiple inputs for a single gesture and save it as model data for the gesture. When a gesture for example - “wing” is waved using the wand, the accelerometer present in the microcontroller detects the movement by using the coordinates in x,y and z axes. The same process is repeated multiple number of times for the dataset to be as large as possible. The advantage of creating a database with more inputs allows the machine learning model to be more accurate and provide better and efficient results. The same process is repeated for the remaining gestures namely ring, slope and unknown. With the data available for the above gestures, we can use it to train a model. The data that is available is split in to two namely as training data and test data.

The major challenge while preparing data can be the presence of outliers or anomalies where there are unexpected values. This can be overcome by feeding our model with more data so that the model is efficient. [WS20]

The different steps involved in preparing data can be as follows according to [WS20]

Understanding the problem The problem, which in our case is collection of data consisting of gestures. The main problem is sub divided in to many parts for easy preparation of data.

Preparation of data The step deals with converting the raw data in to machine learning language that can be interpreted by the compiler.

Evaluation of data This step deals with evaluating the model after collection of the necessary data. This data is then checked for its correctness and analysis is done according to the accuracy of output obtained.

Finalizing the data The model is tested with various parameters and the data is validated. If the results obtained are according to our requirements, the resulting dataset is finalized.

The same steps are repeated for various gestures and the final database is created.

Preparation of Datasets for the Magic Wand

Since the amount of data required for creation of a database is very small the database will be prepared by us. The first set of data that we are trying to capture the movement is for the gesture “W”. The steps involved in capturing a W is mentioned below. [WS20]



Figure 36.1.: Wing Gesture [WS20]

- The device is first moved down and to the right.
- Then the device is moved up and to the right.
- The device is then moved down and to the right.
- The device is then moved up and to the right again.

Shows a sample of real data captured during the “wing” gesture, measured in milli-Gs. The process involved in capturing a ring are as follows. [WS20]

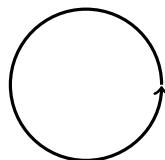


Figure 36.2.: Ring Gesture [WS20]

- Trace a clockwise circle using the wand.
- Aim again to take around a second to perform gesture.

The steps involved in waving the slope gesture is mentioned below [WS20]:

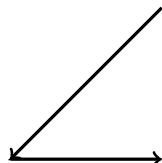


Figure 36.3.: Slope Gesture [WS20]

- The device is first moved down and to the left.

- Then the device is moved to the right.
- Finally you should get a corner of the triangle as shown in the figure.

The process is repeated until around 15 readings are captured by the wand for Data Preparation and Transformation.

In order to consider the unknown readings, we will be carrying out another set of procedures to feed with unknown data. This would help the wand recognize any unknown gesture and gives the output stating that it is false. [WS20]

All readings that are recorded in a unique text file for each gesture. The files `.txt` contains raw accelerometer data that would later be transformed into machine learning language that should be interpreted by the compiler. [WS20]

36.1.3. Data Transformation & Mining

It is in the data mining step that the model is actually built. However before we start the process the algorithm that needs to be used has to be decided. As discussed in the previous section, the dataset is divided equally so that model testing and evaluation is also done to re-confirm the model works fine.

We need to run a simple program to log accelerometer data to the serial port when the gestures are being performed. [WS20]

All the codes used from Github are re-edited to suit the dataset.

Training the Model

To get started we need to modify one of the examples in SparkFun Edge Board Support Package (BSP) such that we can input the data set that we captured. First, follow SparkFun's "Using SparkFun Edge Board with Ambiq Apollo3 SDK" guide to set up the Ambiq SDK and SparkFun Edge BSP. Then the code needs to be changed. [WS20] The program will then be ready to take the dataset as the input. We then need to build the example application and flash it to the device.

The next step is to capture the required data, as mentioned in the previous section, the 3 members of the group will perform the required gestures and create the dataset. For that, we need to open the terminal window and run the following command:

```
script output.txt
```

In the next screen connect to "[115200](#)" Device. Post the measurements from the accelerometer will show up on the screen. The values will also be saved to a file `output.txt`. [WS20]

The text file will contain the data in accordance how it is required by the training set. In order to get a good quality dataset, the same gestures are repeated again as much as possible and then the program is exited. Then the next group member makes the similar file `output.txt`. We need to press the button marked 14 to stop logging the acceleration data. [WS20]

We need to rename the `output.txt` folders as per the names of the person who performed the gestures, this helps to differentiate the datasets for testing and validation purposes. [WS20]

Data for unknown category also is then added, so that when a different gesture is shown, the model identifies it as an unknown gesture. The following are the steps for training the model:

- Loading the tensor board.
- Codes are run to begin the training on Google Colab.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 128, 3, 8)	104
max_pooling2d (MaxPooling2D)	(None, 42, 1, 8)	0
dropout (Dropout)	(None, 42, 1, 8)	0
conv2d_1 (Conv2D)	(None, 42, 1, 16)	528
max_pooling2d_1 (MaxPooling2 (None, 14, 1, 16)		0
dropout_1 (Dropout)	(None, 14, 1, 16)	0
flatten (Flatten)	(None, 224)	0
dense (Dense)	(None, 16)	3600
dropout_2 (Dropout)	(None, 16)	0
dense_1 (Dense)	(None, 4)	68

Figure 36.4.: CNN sequence to classify gestures. Source : [WS20]

- `data_augmentation` file is run to train the model better as it has changed values of acceleration as will it give the model more input values.
- We will then start to see output values appearing on the screen (These are the memory size of the model)

The training then ramps up and the validation accuracy gets better with time.

36.1.4. Model

In model for this project, we transform a sequence of 128 three-axis accelerometer readings, representing around five seconds of time, into an array of four probabilities: one for each gesture, and one for “unknown”. CNNs are used when the relationships between adjacent values contain important information[WS20]. A CNN with numerous layers may learn to recognize each gesture by analyzing its component parts. For example, a network might learn to recognize up-and-down motions and that combining two of them with the appropriate z- and y-axis movements results in a "wing" gesture[WS20]. A CNN accomplishes this by learning a series of filters organized in layers. Each filter learns to recognize a specific type of data feature. When it identifies this feature, it sends this high-level data to the network’s next tier. One filter in the network’s first layer, for example, might learn to recognize something as simple as a period of upward acceleration. When it identifies such a structure, it passes this information to the next layer of the network.[WS20] The outputs of earlier, simpler filters are mixed together to build larger structures in subsequent layers of filters. For example, the "W" shape in the "wing" gesture could be represented by a series of four alternating upward and downward accelerations. The noisy input data is gradually

turned into a high-level, symbolic representation in this process. Network's subsequent layers can use this symbolic representation to figure out which gesture was made. [WS20]

For data acquisition we are using IMU signals^{42.6}.IMU is the electronic component which contains the accelerometer. The IMU object comes from the Arduino LSM9DS1 library.

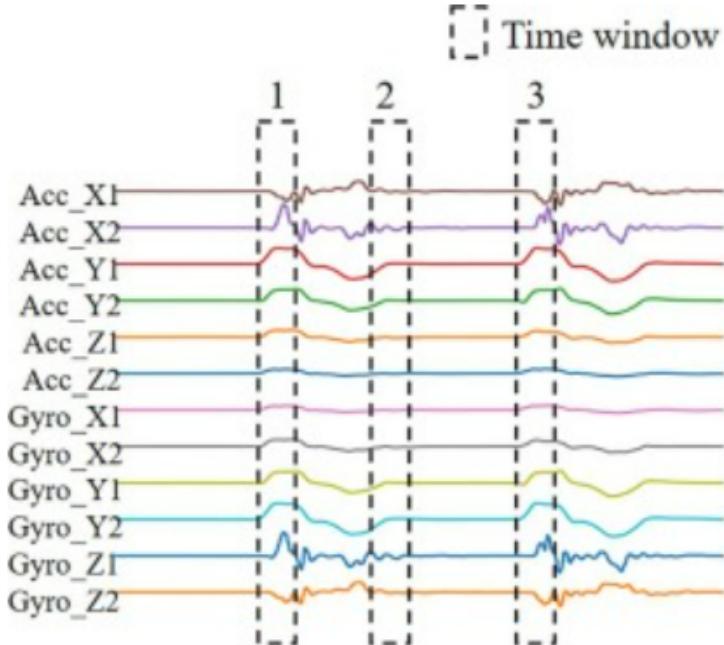


Figure 36.5.: IMU signals [Xu+22]

Convolutional layer directly receives network's input, which is a sequence of raw accelerometer data. The input's shape is provided in the input shape argument. It's set to (seqlength, 3, 1), where seqlength is the total number of accelerometer measurements that are passed (128 by default). Each measurement is composed of three values, representing the x, y, and z-axes. [WS20]

The job of the convolutional layer is to take this raw data and extract some basic features that can be interpreted by subsequent layers. The arguments to the Conv2D() function determine how many features will be extracted.

Conv2D() is where we provide the dimensions of this window. In this case, it's (4, 3). This means that the features for which our filters are hunting span four consecutive accelerometer measurements and all three axes. Because the window spans four measurements, each filter analyses a small snapshot of time, meaning it can generate features that represent a change in acceleration over time. You can see how this works in 36.7 [WS20]

The padding argument determines how the window will be moved across the data. When padding is set to "same", the layer's output will have the same length (128) and width (3) as the input. Because every movement of the filter window results in a single output value, the "same" argument means the window must be moved three times across the data, and 128 times down it. As soon as the convolution window has moved across all the data, using each filter to create eight different feature maps, the output will be passed to our next layer, MaxPool2D. This MaxPool2D layer takes the output of the previous layer, a (128, 3, 8) tensor, and shrinks it down to a (42, 1, 8) tensor a third of its original size. It does this by looking at a window of input

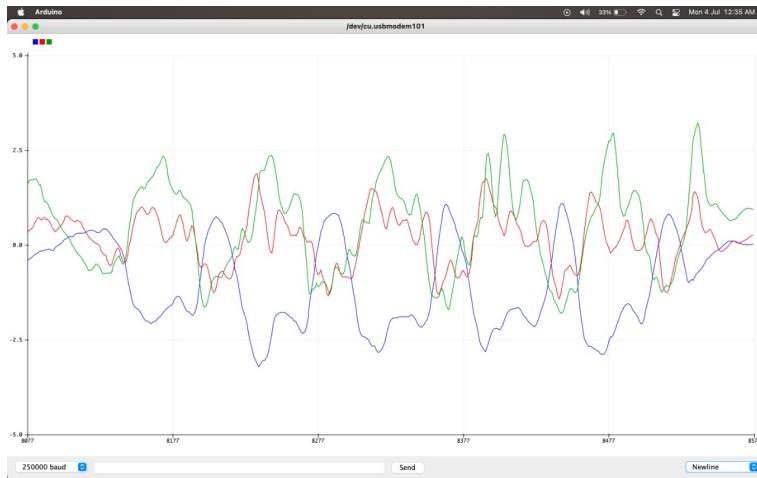


Figure 36.6.: IMU Accelerometer Graph [WS20]

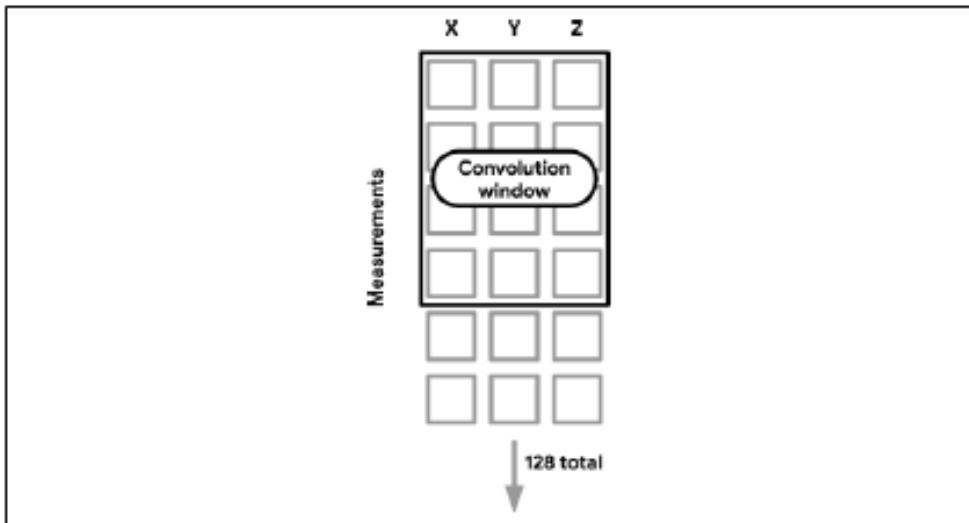


Figure 36.7.: A convolution window overlaid on the data. [WS20]

data and then selecting the largest value in the window and propagating only that value to the output. The process is then repeated with the next window of data. The argument provided to the MaxPool2D function, (3, 3), specifies that a 3×3 window should be used. By default, the window is always moved so that it contains entirely new data. 36.8 shows how this process works. [WS20]

The goal of a CNN is to transform a big, complex input tensor into a small, simple output. The MaxPool2D layer helps make this happen. It boils down the output of our first convolutional layer into a concentrated, high-level representation of the relevant information that it contains. By concentrating the information, we begin to strip out things that aren't relevant to the task of identifying which gesture was contained within the input. Only the most significant features, which were maximally represented in the first convolutional layer's output, are preserved. After we've shrunk the data down, it goes through a Dropout layer. Dropout is a regularization technique; regularization is the process of improving machine learning models so that they are less likely to overfit their training data. Dropout is a simple but effective way to limit overfitting. By randomly removing some data between one layer and the next, we

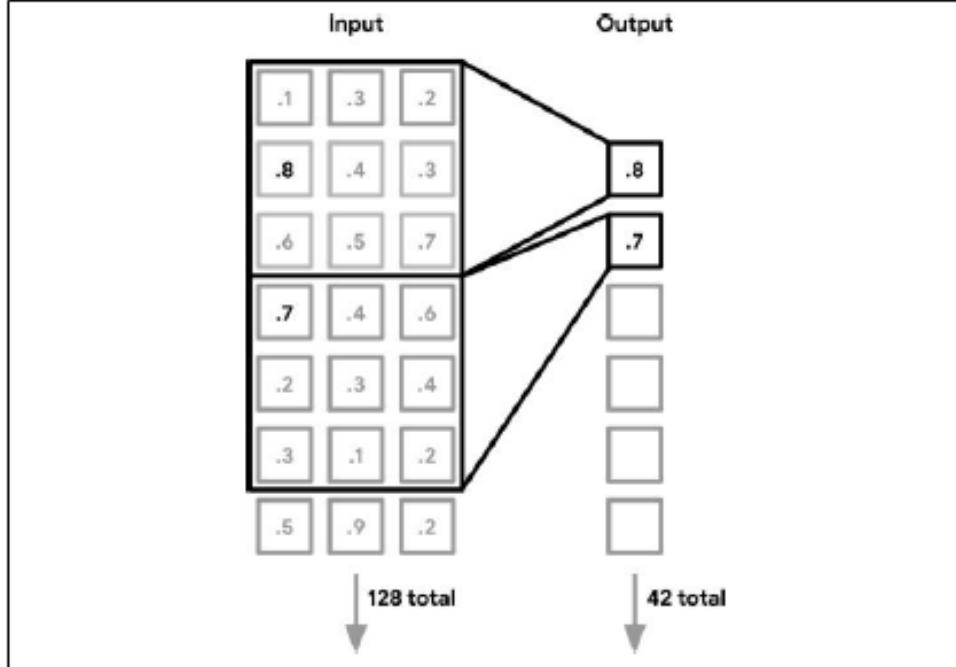


Figure 36.8.: Max Pooling. [WS20]

force the neural network to learn how to cope with unexpected noise and variation. Adding dropout between layers is a common and effective practice. The dropout layer is only active during training. During inference, it has no effect; all of the data is allowed through. [WS20]

This continues the process of distilling the original input down to a smaller, more manageable representation. The output, with a shape of (14, 1, 16), is a multidimensional tensor that symbolically represents only the most significant structures contained within the input data.

If we want to, we can continue the process of convolution and pooling. The number of layers in a CNN is just another hyperparameter that we can tune during model development. However, during the development of this model, we found that two convolutional layers was sufficient.36.9 shows the sequence.

We flatten the data and feed it into a Dense layer (also known as a fully connected layer) to find out the major features within our input. The Flatten layer is used to transform a multidimensional tensor into one with a single dimension. In this case, our (14, 1, 16) tensor is squished down into a single dimension with shape (224). “It’s then fed into a Dense layer with 16 neurons. This is one of the most basic tools in the deep learning toolbox: a layer where every input is connected to every neuron. By considering all of the data, all at once, this layer can learn the meanings of various combinations of inputs. The output of this Dense layer will be a set of 16 values representing the content of the original input in a highly compressed form”. [WS20] Our final task is to shrink these 16 values down into 4 classes. This layer has four neurons; one representing each class of gesture. Each of them is connected to all 16 of the outputs from the previous layer. During training, each neuron will learn the combination of previous-layer activations that correspond to the gesture it represents. The layer is configured with a “softmax” activation function, which results in the layer’s output being a set of probabilities that sum to 1. This output is what we see in the model’s output tensor. “This type of model architecture—a combination of convolutional and fully connected layers—is very useful in classifying time-series sensor

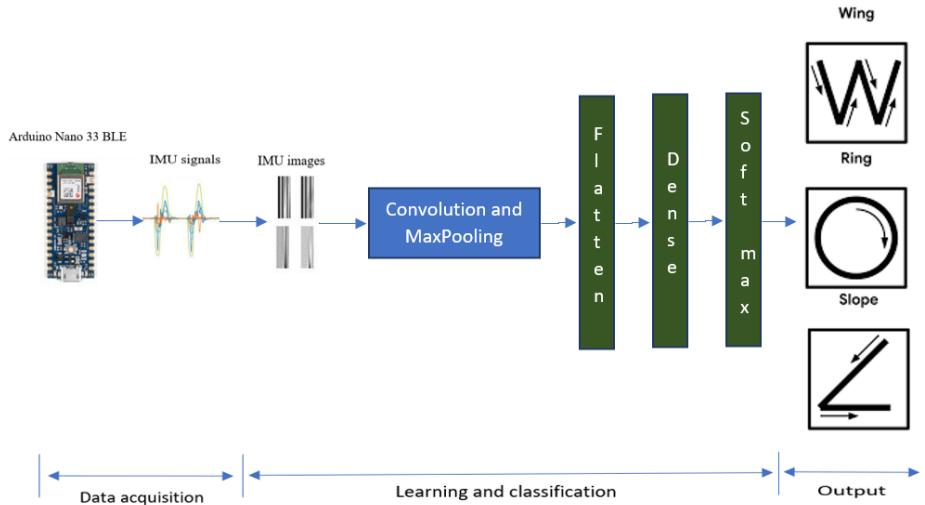


Figure 36.9.: CNN sequence to classify Wing,Ring and Slope.

data like the measurements we obtain from our accelerometer. The model learns to identify the high-level features that represent the “gesture” of a particular class of input".[WS20]

Once we have a result from all the inferences in the model, inorder to ensure there is no false positive the model’s output tensor is given as input to a function called PredictGesture().

Two main actions are carried out by this function. It checks whether the getsures probability meets a minimum threshold and also checks if the gesture has been detected over a certain number of inferences.

The minimum number of inferences to confirm a gesture varies from the type of gesture as different gestures take different times to perform. The number of inferences required for each gesture is located in the file constants.cc. [WS20]

This function returns the predicted gesture by returning numeric values 0,1, 2,3 for Wing, Ring, Slope and unknown respectively. Initially the "prediction scores" obtained from the main is passed on to the function in which a "maxPredictionScore" and "maxPrediction Index". These values above are then compared "kNoGesture" and "kDetectionThrehsold" and then a value of found gesture variable is found which is equal to max prediciton index. This value is then returned to the main function and passed to output_handler() which then displays the recognised gesture. [WS20]

36.1.5. Evaluation and Verification

Once we get the model ready, it is important to test the model with a different dataset. This helps us confirm our model will work fine with gestures from other persons as well. To test the model we need to now call the Keras’s `model.evaluate()` function.[WS20] We can also use a confusion matrix to check the performance of the model. An example of a confusion matrix according to the [WS20] is shown in 36.10 which is calculated by `tf.math.confusion_matrix()` function:

The confusion matrix fuction tells us how much the predicted class of each input in the test dataset agrees to the actual value. It basically tells us the weak points of our model. We can then prepare the dataset again knowing the weak points and train the model once again.[WS20] which improves our model over time

```
tf.Tensor(
[[ 75   3   0   4]
 [ 0  69   0  15]
 [ 0   0  85   3]
 [ 0   0  1 129]], shape=(4, 4), dtype=int32)
```

Figure 36.10.: Sample Confusion Matrix for the dataset [WS20]

36.2. Deployment

In the above sections, the complete details with regards to implementing the project is discussed. In the deployment phase the knowledge acquired in the development phase is applied and implemented.

36.2.1. Software

The below section discusses about the development platform used, the major functions, softwares and different libraries used.

Installation

The first step starts by downloading the Arduino software from the official website <https://www.arduino.cc/en/software>. On visiting the link the following prompt appears as shown in 36.11. Depending on the operating system environment, the necessary software is downloaded and installed.

Downloads

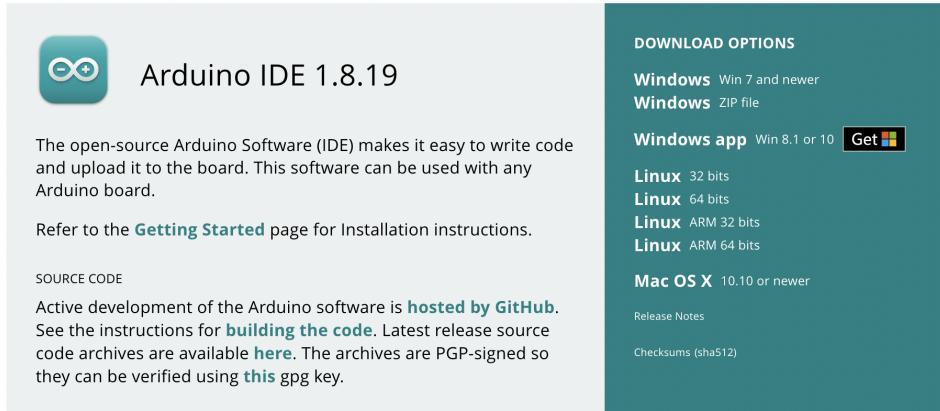


Figure 36.11.: Arduino Nano website downloads page

In order to carry out the project offline, the software should be downloaded in to the desktop. The latest version of the software is preferred to be downloaded as it contains all the performance improvements and bug fixes. The software can be installed in to multiple operating systems like Windows, Mac OS and Linux. For this project version 1.8.19 was downloaded. After installation the libraries has to be configured for the Arduino Nano BLE 33 sense board to work which is explained in the next section



Figure 36.12.: Arduino Nano software version number

Configuration

After installation of Arduino IDE, a window as 36.13 is shown.

In this project an Arduino Nano BLE 33 sense is used, that specific board has to be installed. This done by toggling through option navigating to Tools -> Boards -> Boards Manager as shown in the screenshot 36.14

On opening the boards manager, a search bar is present that lets you to search, download and install the necessary libraries. A compatible board has to be downloaded and installed for the hardware to work properly. In this scenario, an Arduino Nano mbed OS nano board has to be downloaded and installed as shown in figure 36.15.

After installing the boards, the board has to be selected from the menu as shown in 36.16. For the magic wand project, Arduino Nano 33 BLE has to be selected as the board.

Connect the board to a computer using a USB cable and select the correct port from the menu by going to Tools -> Port -> Arduino Nano BLE 33 as shown in 36.17. If the board is not seen in the drop down menu, ensure that the board libraries are downloaded and installed.

In order to confirm the board is connected and to check the details of the connected board, navigate to Tools -> Get Board Info where a window is produced as shown in 36.18

For the Arduino Nano BLE 33 sense to work necessary libraries has to be installed in order to function properly. This can be done my navigating through Tools -> Manage Libraries as shown in 36.19. Search for Tensor flow libraries and then download and install them

Recognizing Gestures

This code snippet in 36.20 recognizes gestures. The program initially checks if the wand moves and then a signal is given to the compiler for recording the gestures. A switch case statement checks whether the wand stays still or the wand keeps moving.

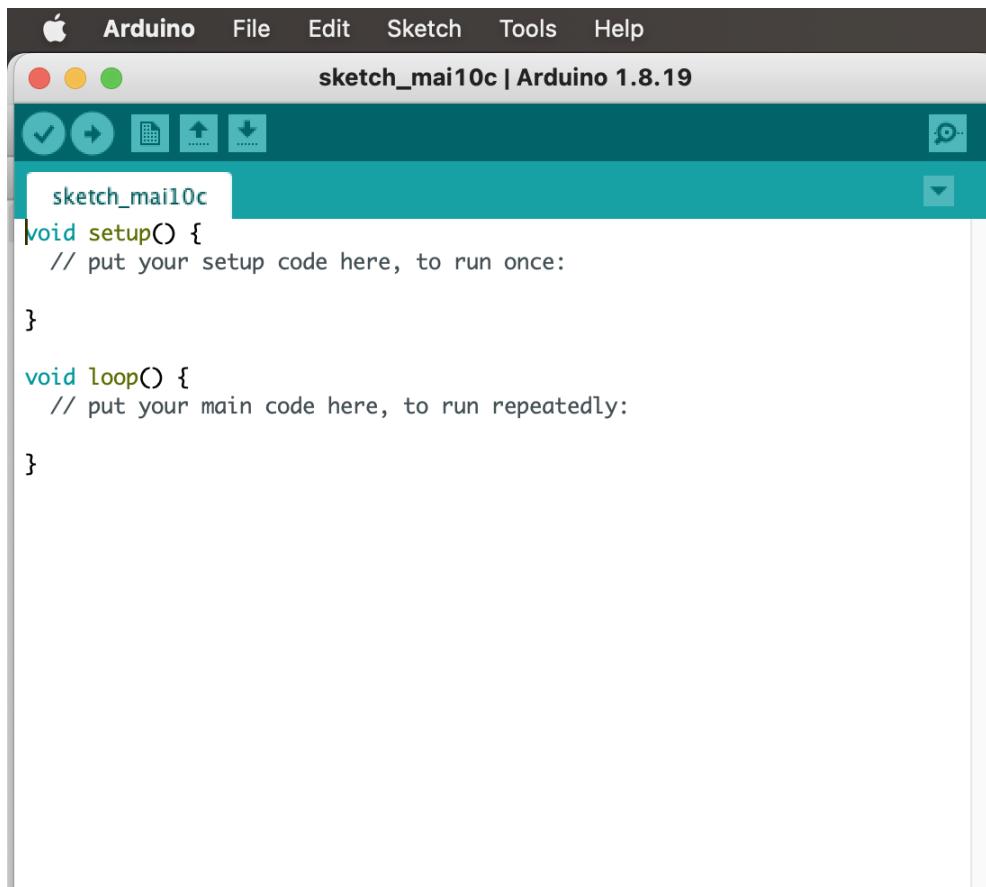


Figure 36.13.: Arduino IDE initial software window

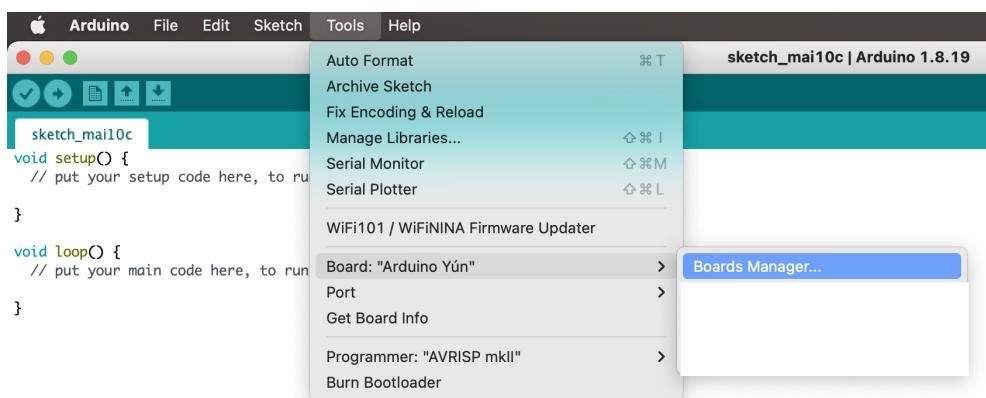


Figure 36.14.: Arduino IDE Boards Manager menu

Prediction of Gestures

The code snippet in 37.3 how a gesture is being recorded is shown in this code snippet. The gesture waved is passed to a function PredictGesture() and the value is stored in to a variable. Which is then passed to the function HandleOutput() to display the gesture waved. The session ends by displaying "Done prediction" and the hardware waits for the new gesture to be fed.

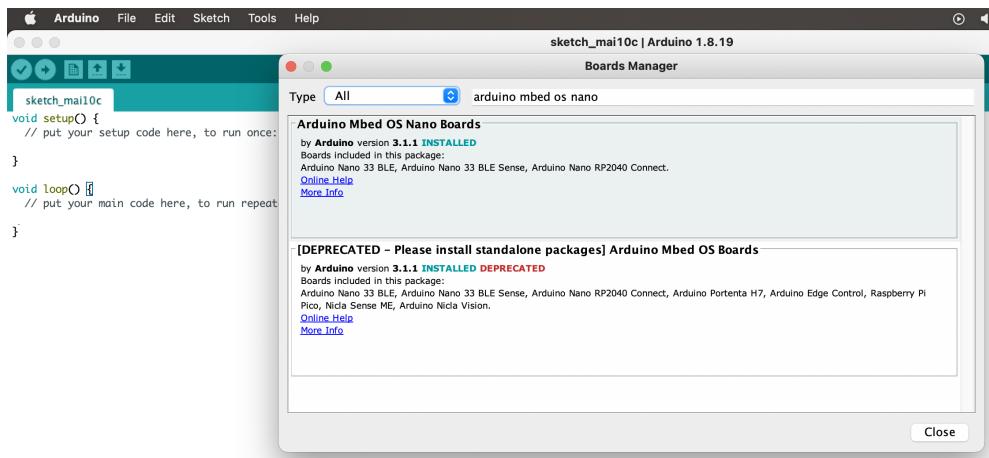


Figure 36.15.: Arduino IDE Boards Manager list

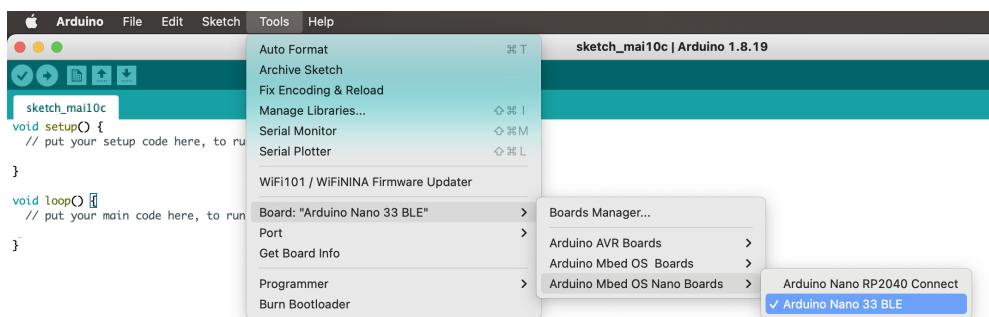


Figure 36.16.: Selection of correct board for the program

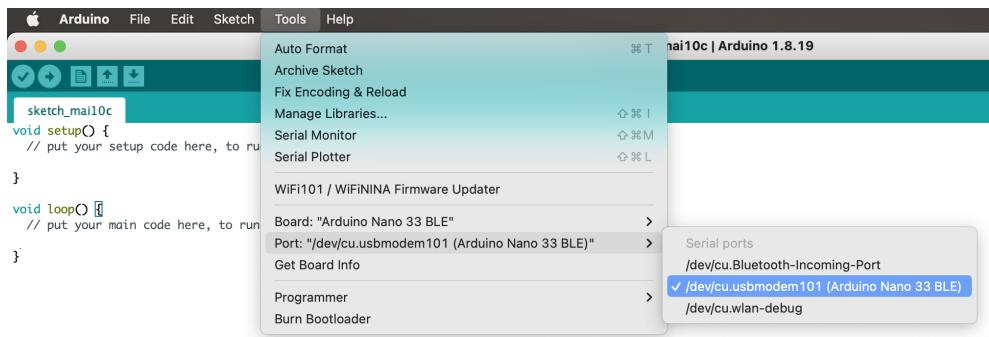


Figure 36.17.: Selection of correct port for uploading the program

Pending Movement

The code in 36.22 checks for any pending movement and waits for a new gesture to be recorded.

Training or Recognizing

The code in 36.23 snippet has a boolean expression at the end for which if the value is "false" the code works like the data to be captured. Change the expression in to "true" to start training the model and gather the values.

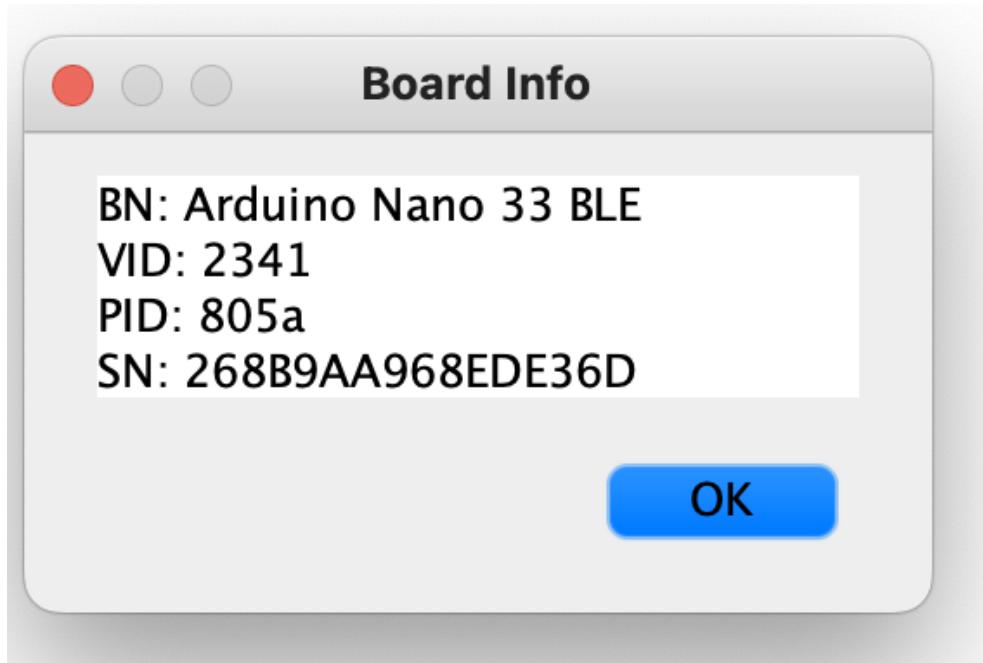


Figure 36.18.: Arduino Board Info

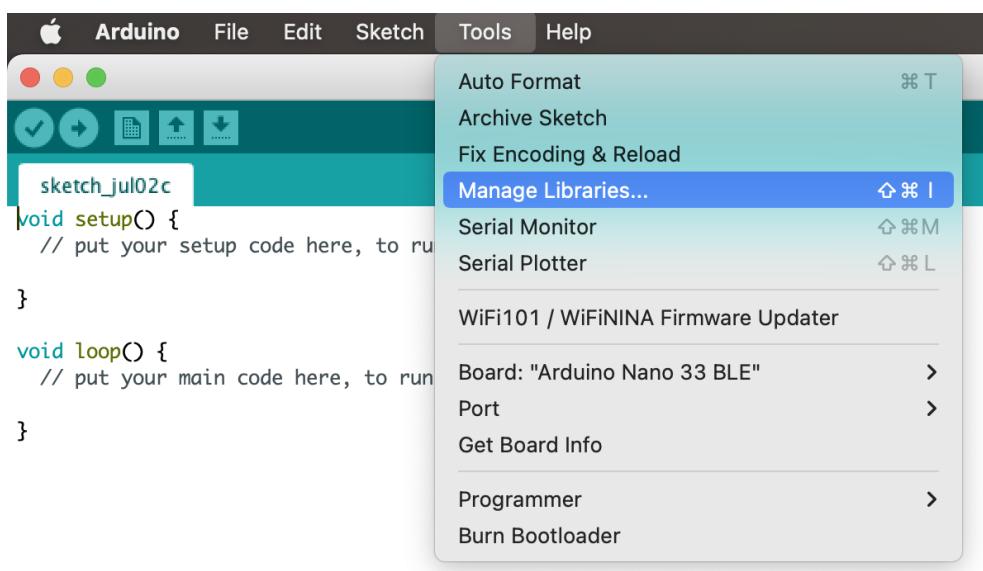


Figure 36.19.: Managing libraries in the Arduino IDE

Printing the output

The code snippet in 36.24 shows how the output is being printed using "*" using different line spacing methods. An example of how the wing gesture is being displayed is shown in the figure.

Prediction Score

The code snippet in 36.25 allows the user to tweak the threshold values of various gestures. A higher value of kDetectionThreshold gives more robust values. But if the

```

void RecognizeGestures() {
    const bool is_moving = IsMoving();

    static int counter = 0;
    static enum {
        ePendingStillness,
        eInStillness,
        ePendingMovement,
        eRecordingGesture
    } state = ePendingStillness;

```

Figure 36.20.: Function to Recognize gestures

```

    case eRecordingGesture: {
        const int recording_time = 100;
        if ((counter - gesture_start_time) > recording_time) {
            Serial.println("OK");
            // Run inference, and report any error.
            TfLiteStatus invoke_status = interpreter->Invoke();
            if (invoke_status != kTfLiteOk) {
                TF_LITE_REPORT_ERROR(error_reporter, "Invoke failed on index: %d\n",
                                     begin_index);
                return;
            }
            Serial.println("processing");

            const float* prediction_scores = interpreter->output(0)->data.f;
            const int found_gesture = PredictGesture(prediction_scores);

            // Produce an output
            HandleOutput(error_reporter, found_gesture);
            Serial.println("Done pred");

```

Figure 36.21.: Code snippet that shows processing of the waved gestures

```

    case ePendingMovement: {
        if (is_moving) {
            state = eRecordingGesture;
            Serial.println("new gesture");
            gesture_start_time = counter;
        }
    } break;

```

Figure 36.22.: Switch case statement, Pending Movement

```

const bool should_capture_data = false;
if (should_capture_data) {
    CaptureGestureData();
} else {
    RecognizeGestures();
}
}

```

Figure 36.23.: Code snippet that explains to train or to capture gestures

```

void HandleOutput(tfLite::ErrorReporter* error_reporter, int kind) {
    // The first time this method runs, set up our LED
    static bool is_initialized = false;
    if (!is_initialized) {
        pinMode(LED_BUILTIN, OUTPUT);
        is_initialized = true;
    }

    if (kind == 0) {
        TF_LITE_REPORT_ERROR(
            error_reporter,
            "WING:\n\r*      *      *\n\r*      *      *\n\r*      *      *\n\r*      *      *\n\r*      *      *\n\r");
    }
}

```

Figure 36.24.: Code snippet displaying a Wing gesture

results achieved are not very good, more training has to be carried out.

```

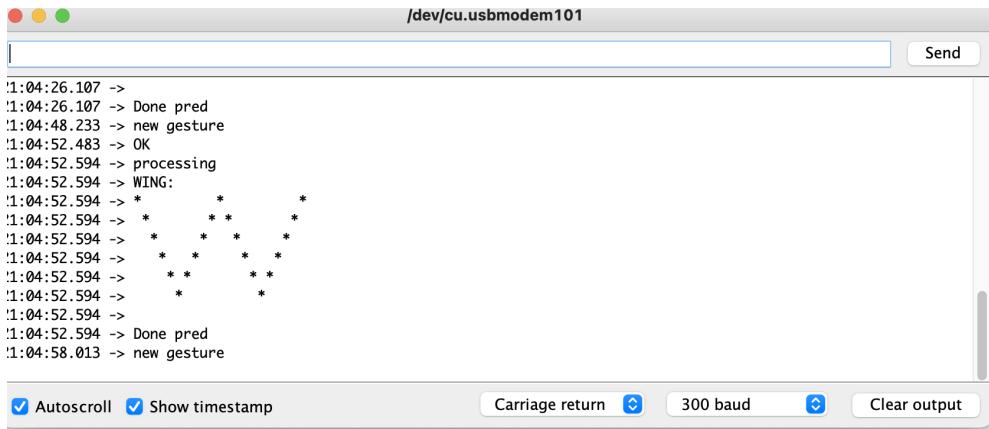
constexpr float kDetectionThreshold = 0.7f;
constexpr int kPredictionHistoryLength = 5;
constexpr int kPredictionSuppressionDuration = 25;

```

Figure 36.25.: The value of constants and constant thresholds in the program

ASCII outputs for Wing, Ring , Slope and Unknown

This code snippets show the various outputs displayed for 36.26 for Wing, 36.27 for Slope, 36.28 and for unknown 36.29



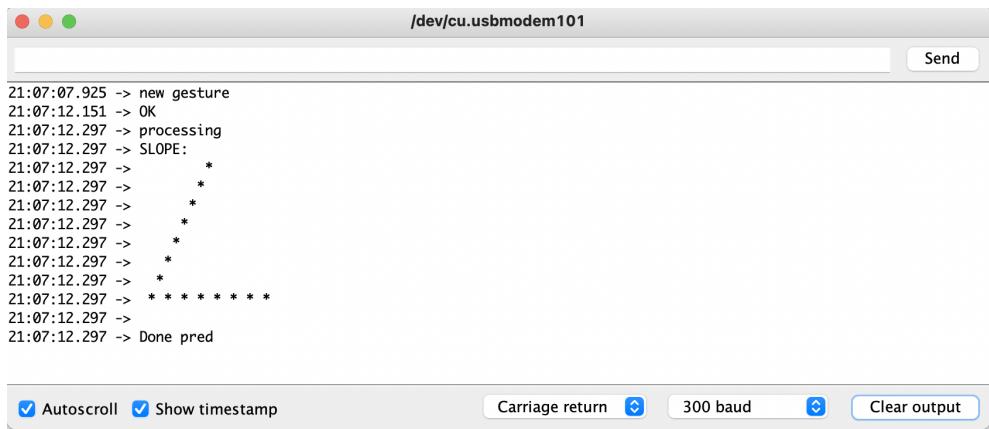
The screenshot shows a terminal window titled '/dev/cu.usbmodem101'. The window contains a text log of gesture recognition events. The log starts with 'new gesture' and ends with 'Done pred'. Between these, there are several 'OK' and 'processing' messages, followed by a sequence of asterisks (*). The terminal has standard OS X-style window controls (red, yellow, green) and a toolbar at the bottom with checkboxes for 'Autoscroll' and 'Show timestamp', and buttons for 'Carriage return', '300 baud', and 'Clear output'.

```

1:04:26.107 ->
1:04:26.107 -> Done pred
1:04:48.233 -> new gesture
1:04:52.483 -> OK
1:04:52.594 -> processing
1:04:52.594 -> WING:
1:04:52.594 -> *      *
1:04:52.594 -> *  *   *
1:04:52.594 -> *      *   *
1:04:52.594 -> *  *   *   *
1:04:52.594 -> *      *   *
1:04:52.594 -> *  *   *
1:04:52.594 -> *      *
1:04:52.594 -> Done pred
1:04:58.013 -> new gesture

```

Figure 36.26.: The displayed Wing gesture



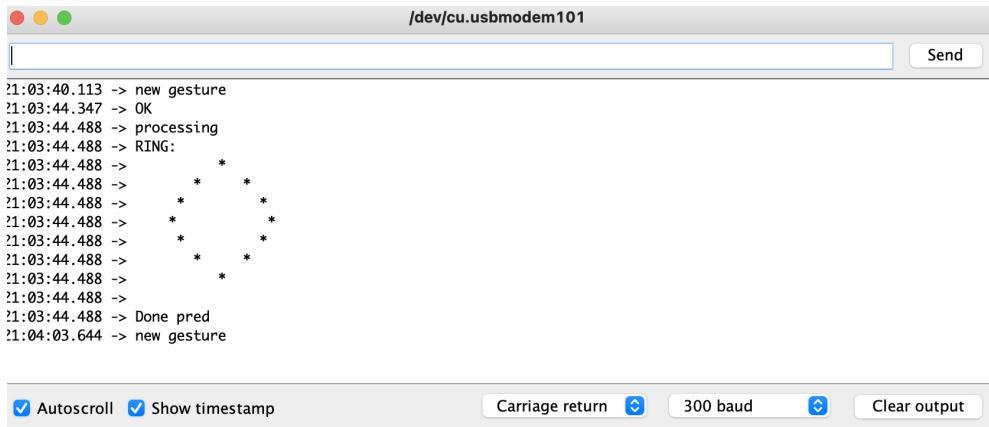
The screenshot shows a terminal window titled '/dev/cu.usbmodem101'. The window contains a text log of gesture recognition events. It begins with 'new gesture', followed by 'OK', 'processing', and 'SLOPE:' messages. The log then shows a series of asterisks (*) being processed sequentially. The terminal includes standard OS X window controls and a toolbar with 'Autoscroll', 'Show timestamp', 'Carriage return', '300 baud', and 'Clear output' buttons.

```

21:07:07.925 -> new gesture
21:07:12.151 -> OK
21:07:12.297 -> processing
21:07:12.297 -> SLOPE:
21:07:12.297 -> *
21:07:12.297 -> *
21:07:12.297 -> *
21:07:12.297 -> *
21:07:12.297 -> *
21:07:12.297 -> *
21:07:12.297 -> *
21:07:12.297 -> *
21:07:12.297 -> *
21:07:12.297 -> *
21:07:12.297 -> *
21:07:12.297 -> *
21:07:12.297 -> * * * * * * *
21:07:12.297 ->
21:07:12.297 -> Done pred

```

Figure 36.27.: The displayed slope gesture



The screenshot shows a terminal window titled '/dev/cu.usbmodem101'. The window displays a text log of gesture recognition events. It starts with 'new gesture', followed by 'OK', 'processing', and 'RING:' messages. The log then shows a sequence of asterisks (*) being processed. The terminal features standard OS X window controls and a toolbar with 'Autoscroll', 'Show timestamp', 'Carriage return', '300 baud', and 'Clear output' buttons.

```

?1:03:40.113 -> new gesture
?1:03:44.347 -> OK
?1:03:44.488 -> processing
?1:03:44.488 -> RING:
?1:03:44.488 -> *
?1:03:44.488 -> *  *
?1:03:44.488 -> *      *
?1:03:44.488 -> *      *
?1:03:44.488 -> *  *
?1:03:44.488 -> *      *
?1:03:44.488 -> *
?1:03:44.488 -> *
?1:03:44.488 -> Done pred
?1:04:03.644 -> new gesture

```

Figure 36.28.: The displayed ring gesture

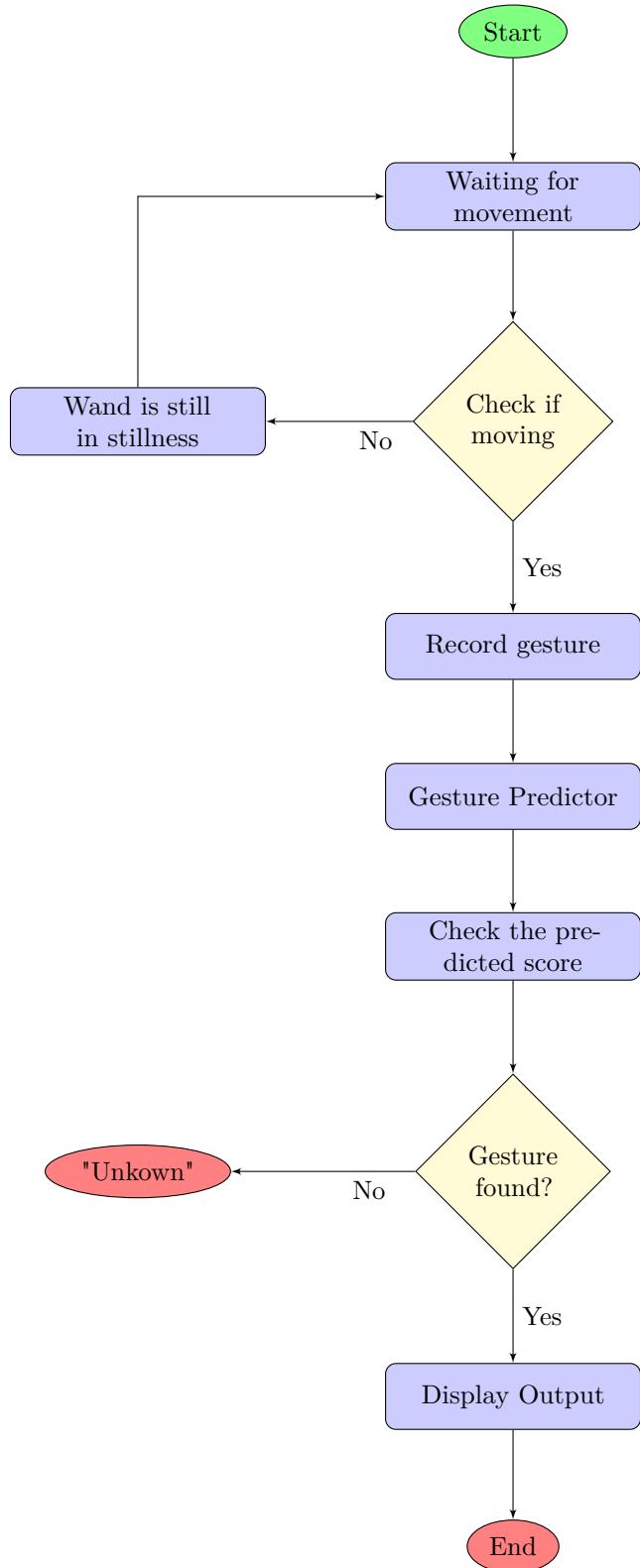
36.2.2. Program Flow Code

On clicking the option "Manage Libraries" a new window shows up where a person can search for the libraries and install them.



Figure 36.29.: The unknown gesture

37. Magic Wand Program Workflow



wand starts moving, the program recognizes the movement followed by recording and predicting the gesture. Then the most probable gesture to the one that was waved by the user is displayed in the serial monitor window.

37.1. Development

37.1.1. Introduction

This developer manual is created to help developers to understand the project and to implement their own ideas in to this project after understanding the major aspects of the project.

37.1.2. 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

This section describes the system requirements needed for the program to work.

Since the software is light, the software can be installed on almost all platforms and can be worked up on.

ARM Cortex M4 Microcontroller (MCU) is present on the Arduino Nano BLE 33 Sense, attached on a stick.

The software is optimized to run on low power MCU that only runs on 64MHz and 256 kbs of RAM.

A USB cable is required to connect the arduino board to the computer. However, the wooden stick for the wand is optional. The board orientation plays a major role in prediction of gestures and a change in orientation may result in false positives.

37.1.3. Overview of the Program

This program is designed to detect gestures waved by the user. The model was initially trained with three gestures namely wing , ring and slope where the training data has been stored in to a data set. This program can be further developed by adding more cases for further gestures and adding more cases in the GesturePredictor.cpp file which will be explained in the further sections.

37.2. Code Structure

The code includes 12 major files. These files are coded to be compatible for the Arduino Platform.

Magic_Wand

This code snippet 37.1 is the main component file of the program. This file comprises of all the header files and functions that are used in the program. The program starts by inclusion of all the libraries and header files needed.

```
#include <TensorFlowLite.h>

#include "main_functions.h"

#include "accelerometer_handler.h"
#include "constants.h"
#include "gesture_predictor.h"
#include "magic_wand_model_data.h"
#include "output_handler.h"
#include "tensorflow/lite/micro/kernels/micro_ops.h"
#include "tensorflow/lite/micro/micro_error_reporter.h"
#include "tensorflow/lite/micro/micro_interpreter.h"
#include "tensorflow/lite/micro/micro_mutable_op_resolver.h"
#include "tensorflow/lite/schema/schema_generated.h"
#include "tensorflow/lite/version.h"
```

Figure 37.1.: Code Snippet function Main header files()

```
bool IsMoving() {
    // Look at the most recent accelerometer values.
    const float* input_data = model_input->data.f;
    const float last_x = input_data[input_length - 3];
    const float last_y = input_data[input_length - 2];
    const float last_z = input_data[input_length - 1];

    // Figure out the total amount of acceleration being felt by the device.
    const float last_x_squared = last_x * last_x;
    const float last_y_squared = last_y * last_y;
    const float last_z_squared = last_z * last_z;
    const float acceleration_magnitude =
        sqrtf(last_x_squared + last_y_squared + last_z_squared);

    // Acceleration is in milli-Gs, so normal gravity is 1,000 units.
    const float gravity = 1000.0f;

    // Subtract out gravity to get the actual movement magnitude.
    const float movement = acceleration_magnitude - gravity;

    // How much acceleration is needed before it's considered movement.
    const float movement_threshold = 30.0f;
    const bool is_moving = (movement > movement_threshold);

    return is_moving;
}
```

Figure 37.2.: Code snippet for isMoving() function

isMoving()

Using the function isMoving() shown in 37.2 the program checks if the wand is still or moving and returns a boolean value 0 or 1. The isMoving() function works by checking the most recent accelerometer values.

A constant called "gravity" is initialised and equated to 1000 milli G's. The movement

is predicted by subtracting this gravity from the obtained accelerometer values from x,y and z.

We have inputed a threshold of 40.0f in order to detect any movement. This can be tweaked by the user if the person wants even slightest of the movements to be detected.

RecognizeGestures()

```

case eRecordingGesture: {
    const int recording_time = 100;
    if ((counter - gesture_start_time) > recording_time) {
        Serial.println("OK");
        // Run inference, and report any error.
        TfLiteStatus invoke_status = interpreter->Invoke();
        if (invoke_status != kTfLiteOk) {
            TF_LITE_REPORT_ERROR(error_reporter, "Invoke failed on index: %d\n",
                begin_index);
            return;
        }
        Serial.println("processing");

        const float* prediction_scores = interpreter->output(0)->data.f;
        const int found_gesture = PredictGesture(prediction_scores);

        // Produce an output
        HandleOutput(error_reporter, found_gesture);
        Serial.println("Done pred");
}

```

Figure 37.3.: Code snippet that shows processing of the waved gestures

The code snippet shown in 37.3 shows a void function and does not return any values. The main objective of this function is to check whether the wand is still or moving. A variable called state is assigned initially and is checked for multiple cases which mentions whether the wand moves or if the wand is kept still. Incase it is moving, the model then records and predicts the gesture.

Cases - ePendingStillness

These switch cases shown in 37.4 ensure that the wand is still by utilising the isMoving() function mentioned earlier. A incremental "counter" variable is used at the end of the switch case statement that increments after every iteration.

ePendingStillness

Initially the state is assigned to as ePendingStillness in the code snippet 37.4 . The switch case checks if the wand is moving or not. An "if" statement checks if its moving or not and then the value of the counter is equated to a variable "still_found_time".The state of the wand is assigned as eInStillness if the "if" condition is satisfied.

eInStillness

An if else statement is used in this case statement as shown in 37.6. If the boolean value returned from the isMoving() function is true , the "state" is considered as "eInStillness" which means the wand is stationary. If that condition is not satisfied, a new variable "duration" is initialised and calculated by subtracting the value of "still_found_time" from the "counter". If the calculated duration is greater than 25

```

switch (state) {
    case ePendingStillness: {
        if (!is_moving) {
            still_found_time = counter;
            state = eInStillness;
        }
    } break;

    case eInStillness: {
        if (is_moving) {
            state = ePendingStillness;
        } else {
            const int duration = counter - still_found_time;
            if (duration > 12) {
                state = ePendingMovement;
            }
        }
    } break;
}

```

Figure 37.4.: The pending switch case that checks if wand is moving or not

```

switch (state) {
    case ePendingStillness: {
        if (!is_moving) {
            still_found_time = counter;
            state = eInStillness;
        }
    } break;

    case eInStillness: {
        if (is_moving) {
            state = ePendingStillness;
        } else {
            const int duration = counter - still_found_time;
            if (duration > 12) {
                state = ePendingMovement;
            }
        }
    } break;
}

```

Figure 37.5.: Code snippet checking if the wand moves or not

which is input by the user, state is assigned as ePendingMovement. This means that the wand is moving and is ready to accept and record gestures from the wand. An if statement is used to check the value of the variable "duration", this can be tweaked by the user for lesser values to interpret even the slightest movement.

```

case eInStillness: {
    if (is_moving) {
        state = ePendingStillness;
    } else {
        const int duration = counter - still_found_time;
        if (duration > 12) {
            state = ePendingMovement;
        }
    }
} break;

```

Figure 37.6.: Code snippet to check if the wand is still

```

case ePendingMovement: {
    if (is_moving) {
        state = eRecordingGesture;
        Serial.println("new gesture");
        gesture_start_time = counter;
    }
} break;

case eRecordingGesture: {
    const int recording_time = 100;
    if ((counter - gesture_start_time) > recording_time) {
        Serial.println("OK");
        // Run inference, and report any error.
        TfLiteStatus invoke_status = interpreter->Invoke();
        if (invoke_status != kTfLiteOk)
}

```

Figure 37.7.: Code snippet to check if the wand is still

37.2.1. Cases - ePendingMovement, eRecordingGesture

These cases shown in 37.7 form the second part of the switch case statements. These cases makes the computer to accept, record and check the gestures done by the user.

ePendingMovement

Checks the state of the wand and if the condition is accepted, the command is given to accept new gestures by the output "New Gesture".

eRecordingGesture

Once the wand starts accepting new gestures, these gestures has to be recorded in order to be checked and compiled. After waving the gesture, the output window shows "processing" and these values are stored in to Prediction scores and this is passed in to a function PredictGesture(). The value obtained earlier is then passed through the function Handle Output() in order to display the gesture waved. The state is then changed back to ePendingStillness and the program continues its iteration.

default

This case is evoked if none of the above cases are satisfied by showing an error.

```

void CaptureGestureData() {
    const bool is_moving = IsMoving();

    // Static state used to control the capturing process.
    static int counter = 0;
    static int gesture_count = 0;
    static enum {
        eStarting,
        ePendingStillness,
        eInStillness,
        ePendingMovement,
        eRecordingGesture
    } state = eStarting;
    static int still_found_time;
    static int gesture_start_time;
    static const char* next_gesture = nullptr;
    // State machine that controls gathering user input.
    switch (state) {
        case eStarting: {
            if (!next_gesture || (strcmp(next_gesture, "other") == 0)) {
                next_gesture = "wing";
            } else if (strcmp(next_gesture, "wing") == 0) {
                next_gesture = "ring";
            } else if (strcmp(next_gesture, "ring") == 0) {
                next_gesture = "slope";
            } else {
                next_gesture = "other";
            }
            Serial.println(next_gesture);
        }
    }
}

```

Figure 37.8.: Code snippet showing capturing gesture function for training

CaptureGestureData()

This function shown in 37.8 is used for training the magic wand. Different prompts are given to the user to wave a gesture and these data can be fed in to the Python code to gather training data. This ensures that your model is more robust.

The function is similar to the RecognizeGestures() and has all the switch cases mentioned earlier. The only difference about this function is that it gathers training data and does not predict any gesture or record them.

loop()

This function fragment shown in 37.9 attempts to read new accelerometer data. A variable named "should_capture_data" is initialised and assigned a "false" value. This means that the program is in prediction mode. Changing this value to "true" changes it into training mode and the user can wave gestures, and feed the data to a python notebook to create model data.

37.2.2. arduino_accelerometer_handler

The function shown in 37.10 uses input tensor with accelerometer data to calculate the necessary values.

```

void loop() {
    // Attempt to read new data from the accelerometer.
    bool got_data =
        ReadAccelerometer(error_reporter, model_input->data.f, input_length);
    // If there was no new data, wait until next time.
    if (!got_data) return;

    // In the future we should decide whether to capture data based on a user
    // action (like pressing a button), but since some of the devices we're
    // targeting don't have any built-in input devices you'll need to manually
    // switch between recognizing gestures and capturing training data by changing
    // this variable and recompiling.
    //SHOULDCAPTURE TRUE = TRAINING MODE, IF FALSE PREDICTION MODE
    const bool should_capture_data = false;
    if (should_capture_data) {
        CaptureGestureData();
    } else {
        RecognizeGestures();
    }
}

```

Figure 37.9.: Code snippet showing the loop() function in the program

The value of the accelerometer is recorded in this function when the gesture is waved. Three variables are initialized to store the x, y, and z axis readings. These readings are then saved in to an array named "saved_data" . The array indices for the array is used by the variable "begin_index"

These readings are then saved in adjacent array indices with the help of a "++" increment operator. 20 readings per gesture are collected. When the array index crosses 600, the counter resets and a new gesture can be waved. These values can be increased or decreased according to users preferences.

gesture_predictor

The code snippet in 37.11 shows the code snippet for gesture predictor. After the collection of data, the program would be filled with lots of data and has to indicate to us which gesture has been made. Two main things are carried out by this function Check whether the gesture probability meets a minimum threshold and also checks if the gesture has been detected over a certain number of inferences.

This function returns the predicted gesture by returning numeric values 0,1, 2 and 3 for Wing, Ring, Slope and unknown respectively. Initially the "prediction scores" obtained from the main is passed on to the function in which a "maxPredictionScore" and "maxPrediction Index". These values above are then compared to "kNoGesture" and "kDetectionThrehsold", then the value of found gesture variable is found which is equal to max prediction index.

37.2.3. Arduino_Output_Handler

The function shown in 37.12 output handler is to display the output according to the value it receives from the PredictGesture() function. It simply displays the gesture waved, which in this case is Wing, Ring and Slope.

37.2.4. main_functions

The code snippet shown in 37.13 contains two functions, setup() and loop(). The setup() function intialises all the necessary values required for the program to function. The loop() function runs once and iterates the data to predict the gesture.

```

const float norm_x = -z;
const float norm_y = y;
const float norm_z = x;
save_data[begin_index++] = norm_x * 1000;
save_data[begin_index++] = norm_y * 1000;
save_data[begin_index++] = norm_z * 1000;
// Since we took a sample, reset the skip counter
sample_skip_counter = 1;
// If we reached the end of the circle buffer, reset
if (begin_index >= 600) {
    begin_index = 0;
}
new_data = true;
}

// Skip this round if data is not ready yet
if (!new_data) {
    return false;
}

// Check if we are ready for prediction or still pending more initial data
if (pending_initial_data && begin_index >= 200) {
    pending_initial_data = false;
}

// Return if we don't have enough data
if (pending_initial_data) {
    return false;
}

// Copy the requested number of bytes to the provided input tensor
for (int i = 0; i < length; ++i) {
    int ring_array_index = begin_index + i - length;
    if (ring_array_index < 0) {
        ring_array_index += 600;
    }
}

```

Figure 37.10.: The accelerometer handler function

37.2.5. constants.h

This code snippet shown in 37.14 contains the expected accelerometer data frequency. A numerical value is assigned to the gestures, Wing, Ring, Slope and unknown as 0,1,2 and 3 respectively. A 'kDetectionthreshold' variable is present which can be tweaked to ensure the quality of results or outputs obtained. Higher values of the variable means that the gesture wave should be more accurate to the training data.

```
#include "gesture_predictor.h"

#include "constants.h"

// Return the result of the last prediction
// 0: wing("W"), 1: ring("O"), 2: slope("angle"), 3: unknown
int PredictGesture(const float* prediction_scores) {
    int max_prediction_index = -1;
    float max_prediction_score = 0.0f;
    for (int i = 0; i < kGestureCount; i++) {
        const float prediction_score = prediction_scores[i];
        if ((max_prediction_index == -1) ||
            (prediction_score > max_prediction_score)) {
            max_prediction_score = prediction_score;
            max_prediction_index = i;
        }
    }
}
```

Figure 37.11.: Gesture Predictor function

```
void HandleOutput(tfLite::ErrorReporter* error_reporter, int kind) {
    // The first time this method runs, set up our LED
    static bool is_initialized = false;
    if (!is_initialized) {
        pinMode(LED_BUILTIN, OUTPUT);
        is_initialized = true;
    }

    if (kind == 0) {
        TF_LITE_REPORT_ERROR(
            error_reporter,
            "WING:\n\r*      *      *\n\r*      *      *\n\r*      *      *\n\r*      *\n\r");
    }
}
```

Figure 37.12.: The code handling the output

```

Arduino File Edit Sketch Tools Help
magic_wand - main_functions.h | Arduino 1.8.19
magic_wand accelerometer_handler.h arduino_accelerometer_handler.cpp arduino_main.cpp arduino_output_handler.cpp constants.h gesture_predictor.cpp gesture_predictor.h magic_wand_model_data.p
/* Copyright 2019 The TensorFlow Authors. All Rights Reserved.
Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at
http://www.apache.org/licenses/LICENSE-2.0
Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
=====
#ifndef TENSORFLOW_LITE_MICRO_EXAMPLES_MAGIC_WAND_MAIN_FUNCTIONS_H_
#define TENSORFLOW_LITE_MICRO_EXAMPLES_MAGIC_WAND_MAIN_FUNCTIONS_H_
// Expose a C friendly interface for main functions.
#ifndef __cplusplus
extern "C" {
#endif
// Initializes all data needed for the example. The name is important, and needs
// to be setup() for Arduino compatibility.
void setup();
// Runs one iteration of data gathering and inference. This should be called
// repeatedly from the application code. The name needs to be loop() for Arduino
// compatibility.
void loop();
#endif // TENSORFLOW_LITE_MICRO_EXAMPLES_MAGIC_WAND_MAIN_FUNCTIONS_H_

```

Figure 37.13.: Code snippet showing the `loop()` function in the program

```

constexpr float kDetectionThreshold = 0.7f;
constexpr int kPredictionHistoryLength = 5;
constexpr int kPredictionSuppressionDuration = 25;

```

Figure 37.14.: Code displaying threshold values

37.2.6. Magic Wand Program Workflow

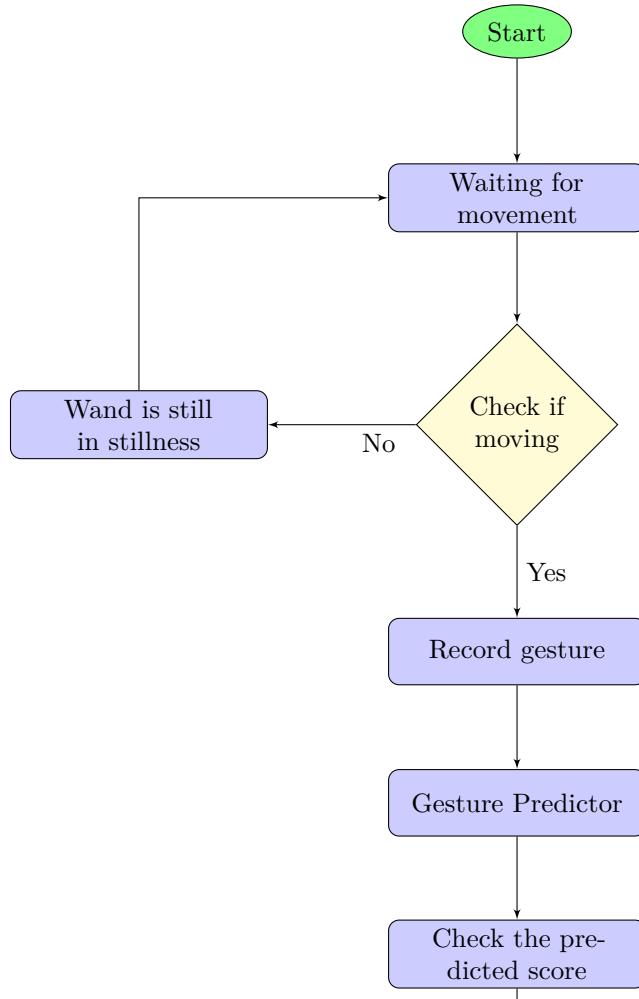


Figure 37.15.: Magic Wand Program Workflow

37.2.7. Frequently Asked Questions

- Q1 - The Program is not uploading.
 - A1 - The board is not connected to the app.
 - Q2 - The program does not give the required values.
 - A2 - Ensure more training is done. Try to reduce threshold values. Try resetting the board.
 - Q3 - The program doesn't compile.
 - A3 - Ensure the necessary libraries are installed in the Arduino IDE.
 - Q4 - The board cannot be found.
 - A4 - Ensure the necessary boards are installed in the Arduino IDE.
 - Q5 - Where can I see the output ?
 - A5 - The answer can be seen in the serial monitor option in tools.
 - Q6 - Why is the output not showing in the monitor ?
 - A6 - The wand needs a small movement to detect movement. These values can be tweaked in the accelerometer handler file.
 - Q7 - How can I train new data in to the program ?
 - A7 - The new data can be trained using google colab or by using a python environment.
 - Q8- I am not getting the right gestures in the serial monitor
 - A8 - Make sure the device is calibrated to ensure proper readings.
 - Q9 - How should I hold the wand while performing gestures ?
 - A9 - While casting the wing and slope, the cable has to be point towards the user. When the ring is being casted , ensure the cable is down as the model was trained in such a manner.

37.3. Getting to Know Arduino Nano Magic Wand

37.3.1. Key Features

- Can detect various gestures.
- Three gestures trained here are “wing” , “ring” and “slope”
- Compatible with Windows, linux and MacOS

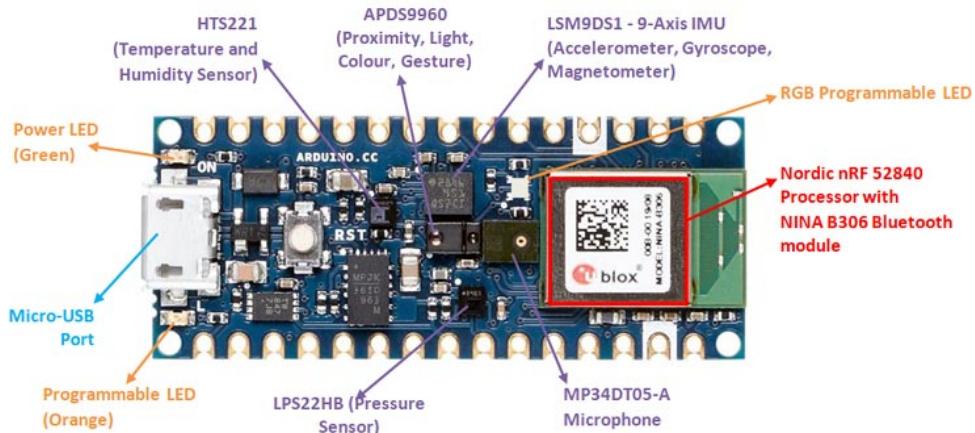


Figure 37.16.: Components in Arduino Nano 33 BLE Sense [Raj19]

37.3.2. Functional Parts

1. **LSM9DS1-** IMU features a 3D accelerometer, gyroscope and magnetometer and allows you to detect orientation, motion or vibrations in your project.
2. **APDS-9960-** The APDS-9960 chip allows for measuring digital proximity and ambient light as well as for detecting RGB colors and gestures.
3. **LPS22HB-** The LPS22HB picks up barometric pressure and allows for a 24-bit pressure data output between 260 to 1260 hPa. This data can also be processed to calculate the height above sea level of the current location.
4. **HTS221-** The HTS221 capacitive digital sensor measures relative humidity and temperature. It has a temperature accuracy of ± 0.5 °C (between 15-40 °C) and is thereby perfectly suited to detect ambient temperature.
5. **MP34DT05-** The MP34DT05 microphone allows to capture and analyze sound in real time and can be used to create a voice interface for your project.
6. **USB port-** USB port allows you to connect Arduino Nano 33 BLE sense to your machine.
7. **LEDs-** There are 3 different LEDs that can be accessed on the Nano BLE Sense: RGB(Programmable LED), the built-in LED(Programmable LED) and the power LED.

Processor

The Main Processor is a Cortex M4F running at up to 64MHz. Most of its pins are connected to the external headers, however some are reserved for internal communication with the wireless module and the on-board internal I²C peripheral.

37.4. Board Operation

Getting Started - IDE

If you want to program your Arduino Nano 33 BLE while offline you need to install the Arduino Desktop IDE. To connect the Arduino Nano 33 BLE to your computer, you need a Micro-B USB cable. This also provides power to the board, as indicated by the LED.

Download the Arduino IDE

In order to deploy the program to the Arduino Microcontroller, we use the Arduino IDE.

Connecting to Computer

In order to connect with Arduino IDE and Arduino Nano 33 BLE sense, connect it with Micro-B USB cable. Connect USB B cable to the Arduino Nano 33 BLE sense port and USB part to the computer.



Figure 37.17.: How to connect with computer

Connector Pinouts

Schematic connection diagram

Technical Specifications of Arduino Nano 33 BLE Sense

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 37.1.: Technical Specifications of Arduino Nano 33 BLE Sense [Ard21]

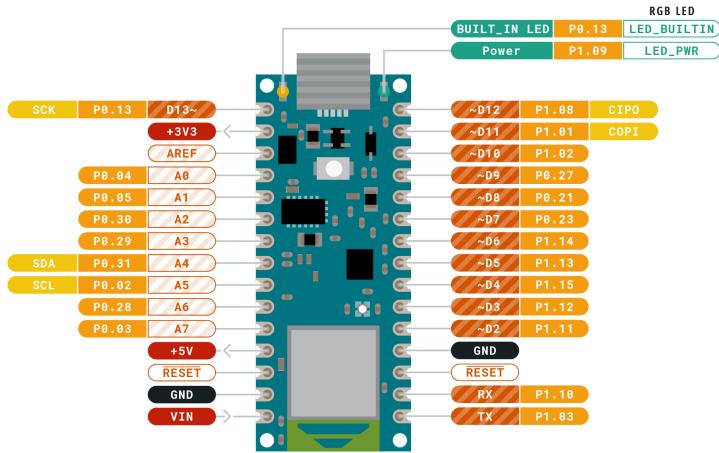


Figure 37.18.: Pinout
[Ard21]

37.5. LSM9DS1(IMU)

37.6. 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.

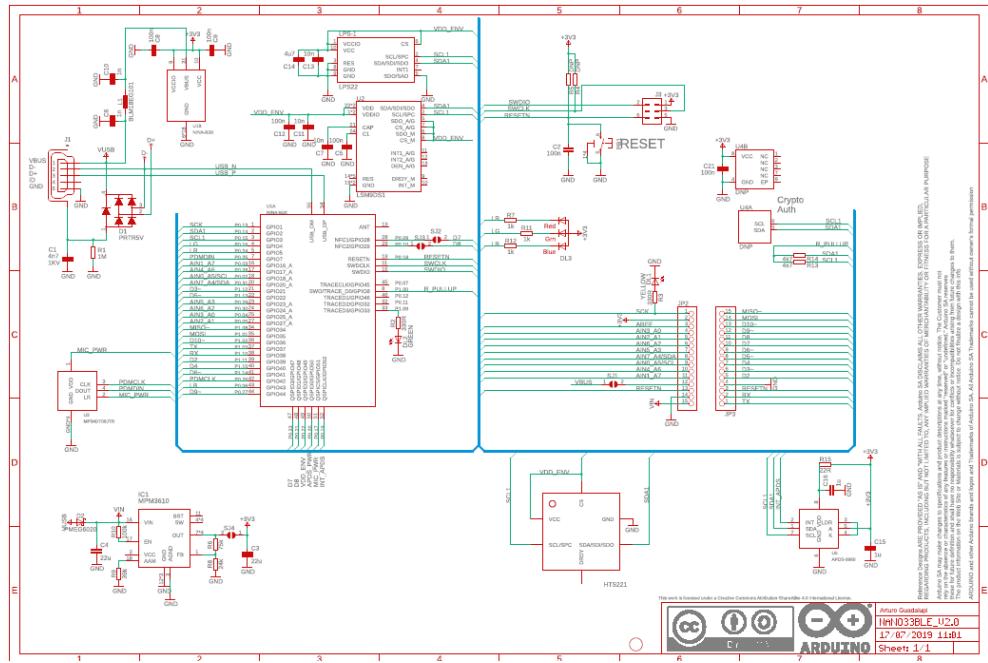


Figure 37.19.: Schematic connection diagram
[Ard21]

Connectivity	BLE 5.0		
Chip	NINA-b3 (nRF52840)		
Clock	64 MHz		
Memory	1 MB FLASH	256 KB SRAM	
Interfaces	USB SPI I2C I2S UART		
Voltages	5V INPUT-USB	4.5-21V INPUT-VIN	3.3V OPERATING
Pinout	14 DIGITAL	6 PWM	8 ANALOG
Dimensions	18 x 45 mm		

Figure 37.20.: Technical Specification

- 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.

37.7. Pin description LSM9DS1

37.8. Pin connections LSM9DS1

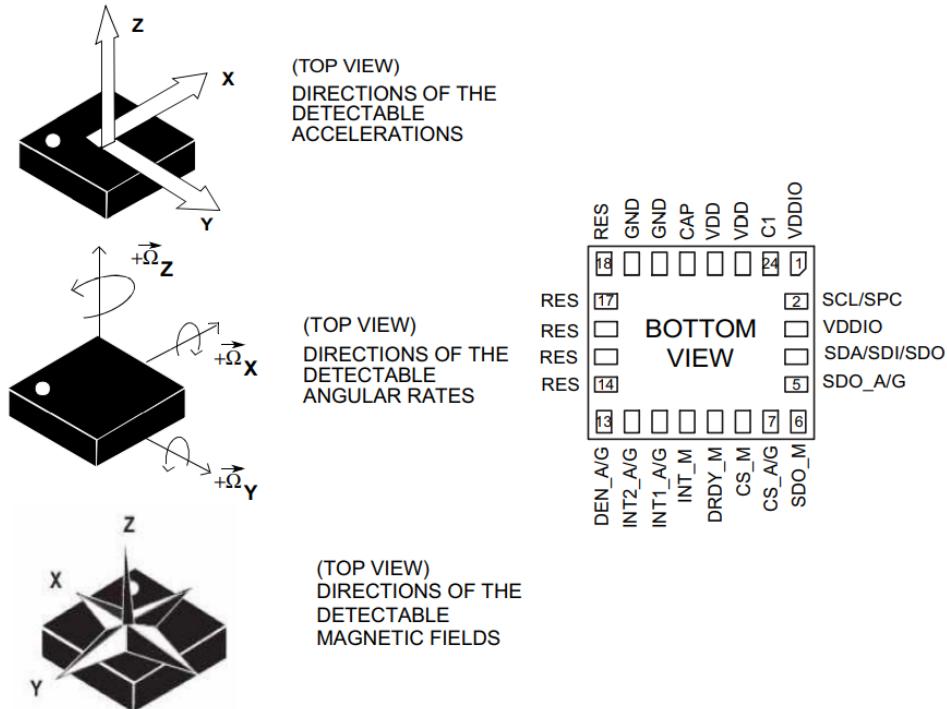


Figure 37.21.: Pin connections LSM9DS1
[Stmb]

Pin description LSM9DS1

37.8.1. Module specifications

Sensor characteristics

Temperature Sensor characteristics

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.

37.8.2. Block Diagram

Accelerometer and gyroscope digital block diagram

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 37.22.: Pin description LSM9DS1
[Stmb]

Magnetometer digital block diagram

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 37.23.: Sensor Characteristics
[Stmb]

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 37.24.: Temperature Sensor Characteristics
[Stmb]

LSM9DS1 electrical connections

37.8.3. Using Magic Wand

37.9. Connect with Computer

Connect the device using Micro-B USB cable.

37.10. Open Arduino IDE

Open Arduino IDE on your computer.

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
AUNP	Acceleration (any axis)	3,000 for 0.5 ms 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 37.25.: Absolute Maximum Temperature
[Stmb]

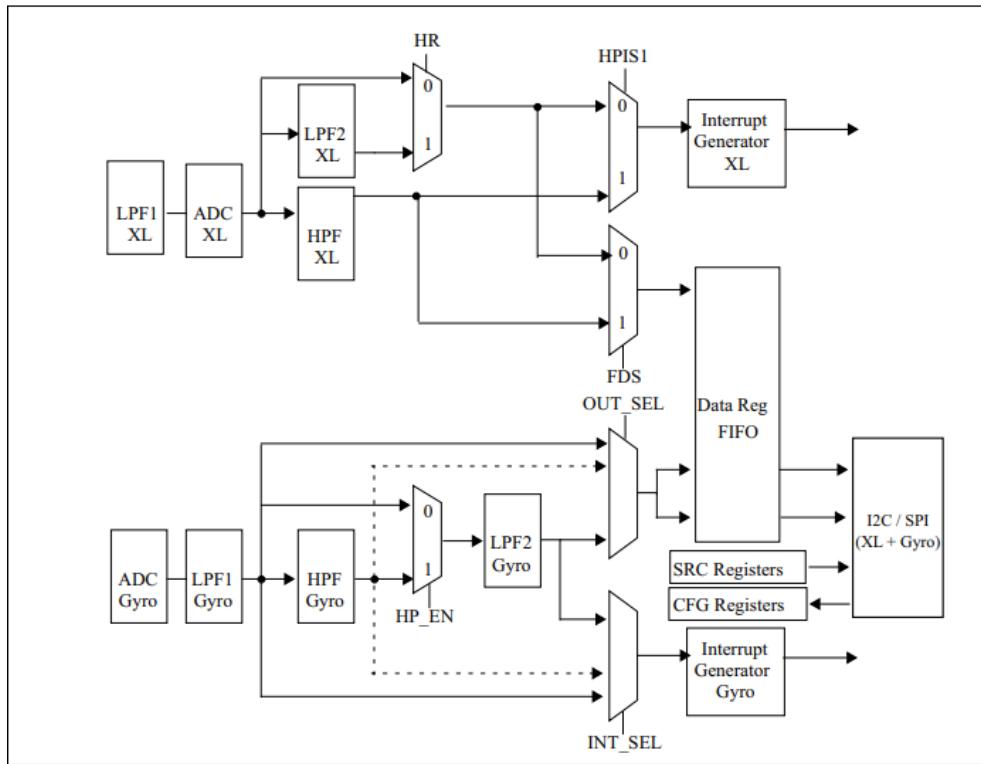


Figure 37.26.: Accelerometer and gyroscope block diagram
[Stmb]

37.11. Select Port

Tools → Board → Arduino Nano 33 BLE

37.12. Run the application

Hit the upload button to compile and upload the code to the Arduino device.

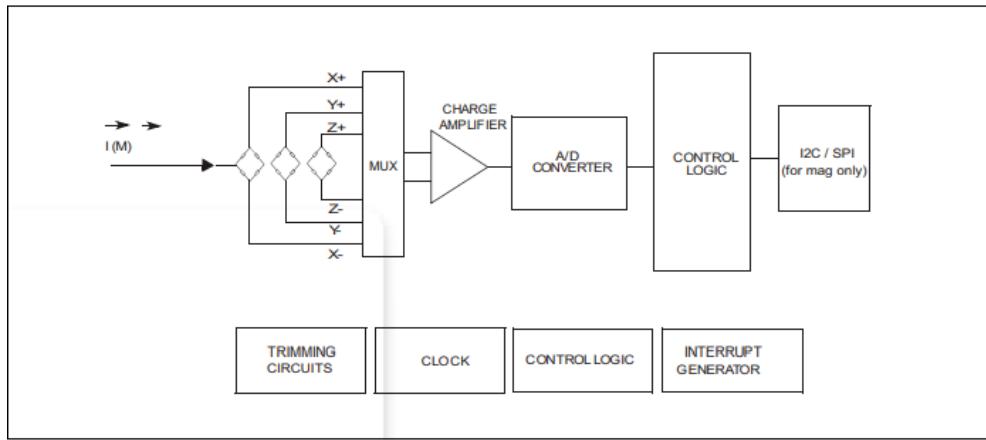


Figure 37.27.: Magnetometer block diagram
[Stmb]

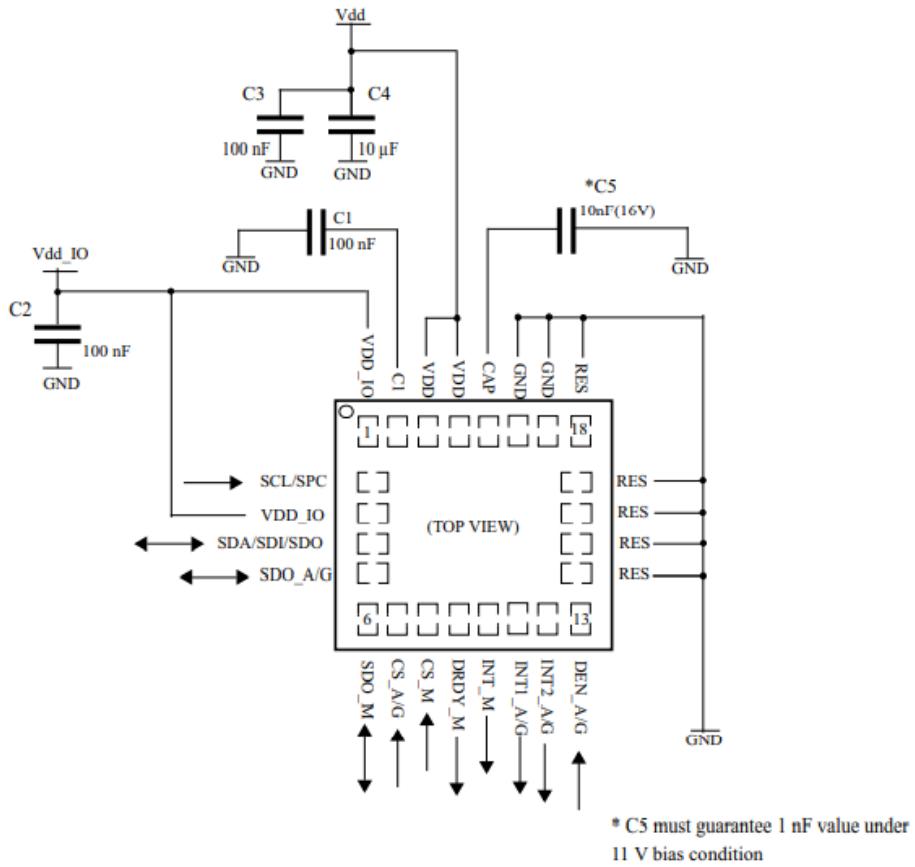


Figure 37.28.: LSM9DS1 electrical connections
[Stmb]

37.13. Open Serial Monitor

Tools → Serial Monitor

37.14. Start waving magic wand

Start waving for the desired output. Gestures will be predicted according to the movement of the wand.

37.14.1. 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

37.14.2. 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.

37.14.3. FAQs

1. Why LEDs are not blinking?

A. Check the connection, also confirm the program is uploaded properly without any errors.

2. Why does the orange light blink sometimes?

A. It is a programmable LED and in this case it blinks when a process is running.

3. What is the button present on the board?

A. It is the reset button, which you can use for resetting the program.

4. How to reset the magic wand?
 - A. Press the reset button present on the board.
5. How should I hold the magic wand?
 - A. Hold the magic wand according to how it was trained. However handle with care by not harming the components and sensors present on the board.
6. Why I am not getting the gesture which I waved?
 - A. Hold the wand in a manner how you trained it, also train it with more datasets.
7. Is it water and fire resistant?
 - A. No, it not. keep away from water or fire.
8. What is input voltage required for the magic wand?
 - A. Arduino Nano 33 BLE only supports 3.3V. So make sure you do not connect it to any other input signals to this board or else it will be damaged.
9. Is it wireless, Can I connect it with Bluetooth?
 - A. It can be connected with the Bluetooth, but as there is no inbuilt battery it should be powered by connecting it with the Micro-B USB cable to the computer.
10. How to turn on/turn off the magic wand?
 - A. The board can be powered via USB connector. Once turned on, Power LED lights up.

38. Project Implementation Steps

The main focus to implement this project is to train the Arduino Nano 33 BLE Sense for edge computing application. The edge computer will behave and react as the human do after deploying machine learning algorithm and computer vision technique. These are the following outputs we need for completing this project [Edge Impulse].

- Gesture Detection
- Object Detection
- Color Detection

Initially I have tried multiple technique to implement the color and object detection part of this project. One of them is Edge Impulse.

38.1. Edge Impulse

Edge Impulse was designed for software developers, engineers and domain experts to solve real problems using machine learning on edge devices without a prior in machine learning. I have tried this platform for color detection, Key word detection, and object detection but later I have switched to other techniques too. Fig38.1 shows the different steps involved for training the model.

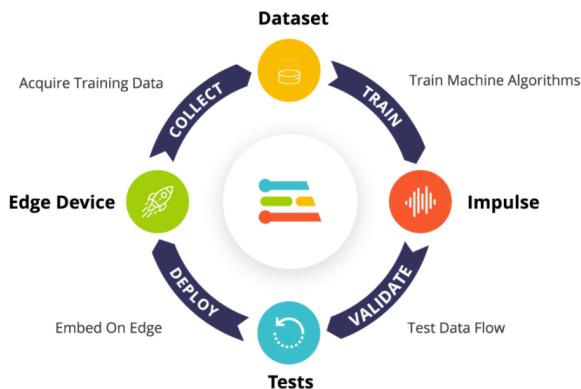


Figure 38.1.: Edge Impulse Flow Chart

Edge Impulse can be the great start for beginner, it allows us to make the data set and also assist us during the whole procedure. This platform supports particular type of edge computer, Arduino Nano 33 BLE Sense is also one of them. For getting started with edge impulse we need to make an account on edge impulse website. [Edge Impulse].

- Edge Impulse is the leading development platform for embedded machine learning.
- It supports a number of embedded hardwares including (Arduino Nano 33 BLE Sense).

- Help us to create data set for Machine learning model.
- Shows the uncertainty in data.
- Split the data into training (80 percent) and test (20 percent).
- Predict the most suitable Machine learning algorithm to train the model.
- Allow us to test the model and see the accuracy.
- After doing all these above steps, we can deploy the train model in our Arduino nano 33 BLE Sense

38.2. Gesture Detection

Gesture detection is the main task for doing this project. On the basis of different gestures we need to train the Arduino Nano 33 BLE Sense to behave differently on each gesture. My task is to define the type of gesture what I want to detect, and also recognize these gestures for getting the certain output from Arduino Nano 33 BLE Sense.

38.2.1. First Approach for Gesture Detection

During my research, while going through for defining the type of gestures, I have found very few solutions. One of the most visible solution is the available sensor on Arduino Nano 33 BLE Sense is APDS9960. This sensor can detect four different types of gesture (Moving left, Moving right, Up, Down) these 4 gesture sense the sensor when i move my hand in front of sensor. This was the first solution, and the available library and code is also available in the arduino IDE. I was not satisfy with this solution, because every time the sensor sense the movement when I put my hand very close to the sensor approximately 15mm. So, I switch from sensor detection to Computer vision and Machine learning techniques.

38.3. Gesture Detection Test Using MediaPipe

MediaPipe is the framework in computer vision and support so many Machine learning algorithm to make usefull application. It help us to define to gesture using hand by making the gesture on the basis of hand landmarks. Initially as a test, I have tried to count the finger of hand and show them on the computer using the OpenCV and MediaPipe module. It is detecting very fast and accurate with good frame per second (FPS). Fig38.2 shows the result of counting finger using the MediaPipe technique on the basis of hand landmarks.

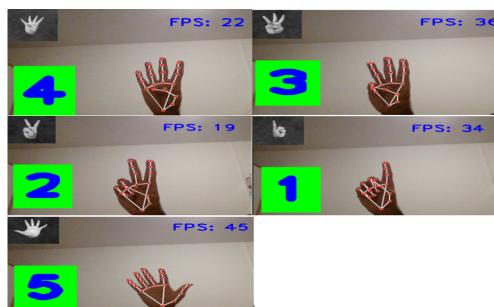


Figure 38.2.: Counting Finger Using MediaPipe and OpenCV

38.3.1. Successful Approach of Gesture Detection

For gesture detection, I choose my hand to define the different types of gesture for edge devices. The most suitable module I found for this is MediaPipe, it help me to make the different types of gesture using the hand landmarks. MediaPipe Hand Landmark Solution There are 21 hand Landmarks on each hand, we can use the both hand for making the more complex gesture too by changing the position of our fingures or even the individual landmark on hand. Fig38.3 shows the 21 3D hand landmarks.



Figure 38.3.: Hand Landmarks using MediaPipe

MediaPipe module uses the convolutional neural network and took 30,000 3D images to train this module. It has very good accuracy and precision, there are some functions written in this module to detect which landmarks we want to detect. Fig38.4 shows for detecting the 21 hand landmarks of the hand are:

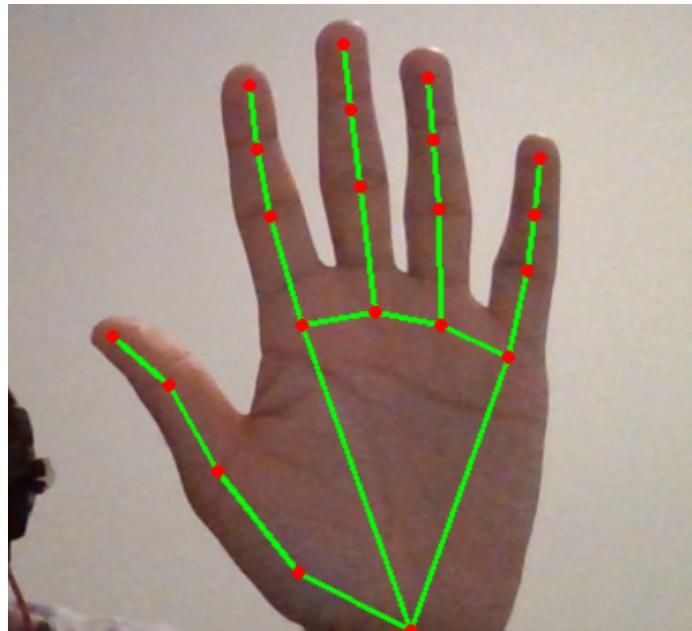


Figure 38.4.: Hand Landmarks using MediaPipe Function

With the help of these hand landmarks, we can define the different types of gestures. For example by opening or closing the finger we are able to define and detect gestures or by changing the position of landmarks, by changing position the landmarks also changes the x and y value. MediaPipe help us to detect these changes and also recognize it for getting certain output. The Following two solution I have tried this with the help of MediaPipe Hand Landmarks detection are:

38.3.2. Gesture Detection on Arduino Nano 33 BLE Sense

For doing the Gesture detection on Arduino Nano 33 BLE Sense, I used the following set of Software, framework, Libraries, Modules and Hardwares.

- Arduino IDE
- Pycharm
- Logitech Camera
- MediaPipe Algorithm
- OpenCV
- Pyfirmata

These are the sets of hardware and software pieces I have used for detecting the gestures with Arduino Nano 33 BLE Sense. For the detection part I have define six different outputs on each six different gesture detection. The outputs are (Run, Stop, Slow, Fast, Turn Left, Turn Right). I wrote the Python program in Pycharm software, because MediaPipe module is written in python. Arduino IDE is support only c++, and it is not directly support python libraries and module. Pyfirmata use for serial communication protocol between python supportive host computer and embedded device have Arduino IDE. By Installing pyfirmata we are able to run the python program on host computer and by serial communication get the results on Arduino Nano 33 BLE Sense. Similarly Logitech camera is for capturing the gesture, I have switch to Logitech camera for better resolution and pixels. OpenCV library use for better visualization, writing color function making circles and image processing.

38.3.3. Gesture Detection Results On Arduino Nano 33 BLE Sense

There are six different types of Gestures define and detect by using Hand landmark function of MediaPipe module. The detected signs on the basis of gestures are (Run, Stop, Slow, Please Turn Left, Please Turn Right). These signs are detect by using the landmarks of hand, and we can define and detect as many gesture by using hand landmark technique. Fig38.5 shows the detected gestures and sign results are as follow.

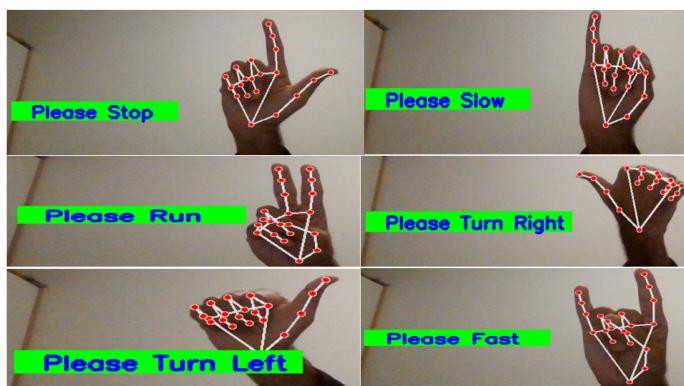


Figure 38.5.: Gesture Detection on Arduino

38.4. Gesture Detection Using LSTM Layer and Tensor Flow

This Model is also equally perform well for detecting the gestures for high processing power computers. The Following sets of Libraries, Software, Module, Hardware, and Packages.

- Anaconda (For Jupyter Notebook)
- Numpy
- Matplotlib
- sklearn
- TensorFlow 2.6
- OpenCV
- MediaPipe
- CUDA 11.0
- Cudnn 8.2

For training the model I have used the LSTM layer, with Tensorflow for MediaPipe gesture detection. It also detect different types of sign on different gestures. The gestures detect the following five sign (Run, Stop, Slow, Fast, Going Good). I have tried 2000 epochs for training the model but this technique gives the 99 percent accuracy after 300 epochs.

38.4.1. Gesture Detection Results Using LSTM and TensorFlow using MediaPipe

The similar type of gesture detection technique has been tried with long short term memory (LSTM) layer and TensorFlow framework. The model is trained for 5 different classes, (Run, Fast, Slow, Stop, and Going Good(GGood)), these gestures are also detecting using MediaPipe hand Landmarks technique. Fig38.6 shows the result of train model detecting the different gestures.

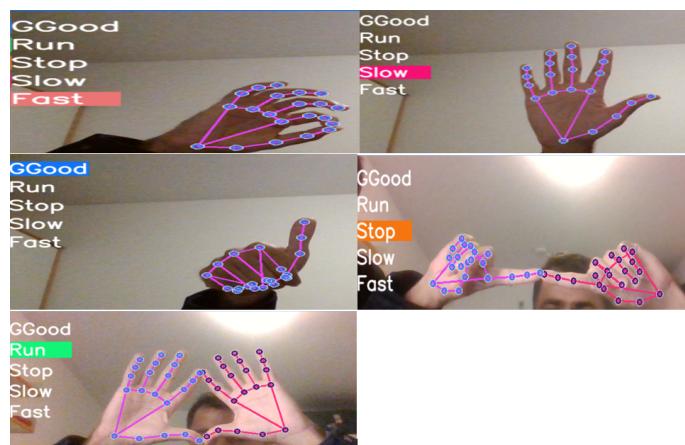


Figure 38.6.: Gesture Detection Using LSTM, TensorFlow, and MediaPipe

38.5. Object Detection

There are multiple solution I have tried for object detection too, one with the MediaPipe and other with state of the art object detection models like OpenCV Gpu support using Darknet Yolov4, (You Only Look Once Version 3) Yolov3, (You Only Look Once Version 4) Yolov4, and Deep sort.

38.5.1. Yolov3 and Yolov4 Object Detection Model

You only look once (YOLO) versions are also performing equally well on COCO data for detecting 80 different classes. YOLO v4 train object detectoin on a single GPU with a smaller mini-batch size. Other models require many GPUs for training with a large mini-batch size, and doing this with one GPU makes the training slow and impractical. The result shows that, YoloV4 has better frame per second (FPS) and Average precision (AP) among others. Fig38.7 shows the comparison of YOLO version and other models for object detection is:

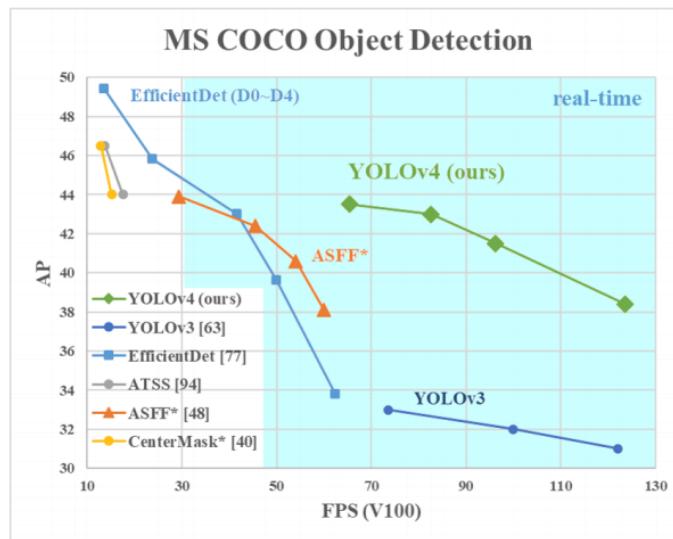


Figure 38.7.: Object detection Model Comparison

38.5.2. OpenCV Gpu support using Darknet YOLOV4

Darknet is an open source neural network framework written in C and CUDA. It is fast, easy to install, and supports CPU and GPU computation. Darknet Yolov4 is the state of the art object detection model, it uses the COCO data set having 80 different classes. It uses the pretrained Yolov4 weights and able to detect images from photos and videos. The following pre-requisite steps need to follow for setting up the environment:

- Install Open-CV from Source with Gpu-support.
- Install the CUDA and CUDNN compatible version with open-cv for
- Darknet support.
- Visual Studio.
- Anaconda

38.5.3. Deep Sort Object Tracking Model

Deep sort is the model in machine learning use for tracking the object from the video having good frame per second on gpu. It also track 80 different classes and uses COCO data set. It tracks based on not just distance, and velocity but also what that object looks like. Deep sort allows us to add this feature by computing deep features for every bounding box and using the similarity between deep features to also factor into the tracking logic.

39. Open Question and Possible Solution

It is the time we can say that we have achieve the desired results and implement on Arduino Nano 33 BLE Sense for making the edge computing application. But there are still some checkpoints need to be overview, Although, it is not easy to apply the AI and computer vision both at the same time on such a small and low processing computer. Instead of low processing power, the Arduino Nano 33 BLE Sense like the other Arduino board supports only Arduino Integrated development environment (IDE) for writing or erasing the code. It is the environment for C++ and not support any python module. The following checkpoints need to be overlook are as follow.

Although, I already get the desire output and results for Arduino Nano 33 BLE Sense. For training the edge computer I used various techniques, all were mentioned in the previous chapters too. There are still some checkpoints (which have alternative solutions available) but we can still work on that more.

39.1. Checkpoint 1 (Arduino IDE and Python supportive Packages)

The Arduino IDE is written only in C++ language and is not supported the other programming languages. Some of the Module I used specially for executing the Gesture detection part of this project is only support python language e.g., MediaPipe and OpenCV. MediaPipe and OpenCV module are the most important part for gesture detection, for detecting the landmarks on hand the supported function is written in python. Without the MediaPipe Module, the hand landmarks technique is not possible to implement. At the moment, there is no such module or library who can support directly MediaPipe Module in Arduino IDE and C++, because MediaPipe Module is written in python.

39.1.1. Possible Solution

The only possible solution I have found yet is to run the Python Module in Arduino by using the Pyfirmata library available in Arduino environment. Firmata is use as an intermediate protocol that connects an embedded system to a host computer, using pyfirmata the program will run on host computer and get the certain output on embedded Devices. Fig 39.1 shows the firmata Protocol is use as the communication protocol between the two different environments (C++ and Python). Initially, the python program needs to run once on any python supported desktop, and by installing the firmata library on Arduino it will make the communication between embedded device and desktop for getting the desire results on Arduino. The Below figure also shows the two environments, the Raspberry pi and Arduino make a communication using firmata protocol. Arduino and Python

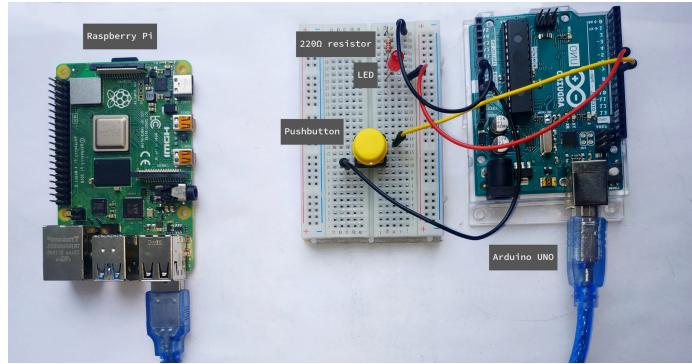


Figure 39.1.: Pyfirmata Communication for Python on Arduino

39.2. Checkpoint 2 (Arducam Mini 2MP OV2640 Replacement)

For getting the high precision and good accuracy, when we are trying to detect some images, we need a high-resolution camera. Nevertheless, the Arducam Mini 2MP is the best suit for Arduino and is most cost effective too, but for better result and accuracy, it is not best suited for the AI and Machine learning application. The main issue with these types of cameras are they need a continue SPI and CS signal. It has eight pins; all need a continue output from an Arduino Nano 33 BLE Sense at the same time. If any of this damage or loose, the image will not be captured from the Arducam. The other main issue with this camera is that it is not as much accurate for the real time application. For getting the better results we need to come very near to camera every time, it is not suited for any process. There are some of the possible solutions exist, which is easy to implement and making the good results too.

39.2.1. Possible solution

There are many good resolution cameras available for the embedded devices like Arduino Nano 33 BLE Sense, which is easy to use and not as much complicated as the Arducam Mini 2MP is. One of them is (Logitech). It is easy to use, and we can change or displace the camera position very easily. Fig 39.2 shows the Logitech needs just one Universal Serial Bus (USB) input, and it can operate from desktop or even from any embedded device. The results, reliability, and robustness we have gain by using Logitech camera is much better for machine vision and Artificial Intelligence application.

39.3. Checkpoint 3 (TensorFlow lite not supported complex object)

There are still some remaining check points, where we can say that we can apply multiple solutions. There is a possibility to run the already written TensorFlow Lite library in the Arduino IDE for detecting the person detection. It is working equally fine as the other solutions are, but it is only supported the person detection part. We need a Arducam Mini 2MP camera for deploying this Model on Arduino Nano 33 BLE Sense, which has already some limitation because it can detect only when the person is very close to edge computer because edge computer and Arducam must be on the same place all the time due to SPI and CS communication. The other object detection is not possible with TensorFlow lite technique on Arduino Nano 33 BLE



Figure 39.2.: Logitech Camera Replacement

Sense due to C++ language environment issues.

39.3.1. Possible solution

Similarly, due to only C++ supported environment, we are not able to run the OpenCV module directly on Arduino IDE. For the person detection, there is a MediaPipe Holistic function called face detection, it can detect the person face with much better accuracy as compared to above mentioned solution. Due to Python supported MediaPipe Module, we can make the communication between python program and Arduino using the Pyfirmata protocol and getting the desired result on Arduino by running the Python and C++ program together. For the image detection we can change the Arducam Mini 2MP with Logitech camera for better and long-distance detection and reduce the complexity.

39.4. Object Detection Part Solution

For the object detection part other than person, it is not possible directly with Arduino or TensorFlow lite. We need a python supportive module OpenCV, full version of TensorFlow and running some of the state of art object detection Model (Yolo v4, Yolo v3). There are some state of art model for object detection having predefined weights file for detecting 80 different classes are Detectron-2, Deep Sort and Yolo v5. These models need high processing power for detecting the object in random condition, they can detect object as well as track object from the video too. For running these types of models at edge computer we need more Ram, which is impossible on Arduino. These models support OpenCV only when it is installed from the source, which means that for compatibility we need a specific version of TensorFlow and NumPy libraries.

39.5. Possible operating system options for Machine Learning Model

There is more advancement in this field, either you have to train your model with Central processing unit (CPU) or Graphic processing unit (GPU). Whether for deep learning applications, massive parallelism, intense 3D gaming, or another demanding workload, systems today are being asked to do more than ever before. A central processing unit (CPU) and a graphics processing unit (GPU) have very different roles.

In short, the CPU Constructed from millions of transistors, the CPU can have multiple processing cores and is commonly referred to as the brain of the computer, while GPU is a processor that is made up of many smaller and more specialized cores. By working together, the cores deliver massive performance when a processing task can be divided up and processed across many cores. GPU vs CPU

39.5.1. Nvidia GPU Installation requirement

The Systems having Nvidia GPU capabilities are perform well and quicker when we are trying to train and run the Machine Learning Model as compared to CPU. For running the Model on GPU, we need to install the appropriate version of Computer unified device architecture (CUDA) and cudnn deep neural network library with the OpenCV too. The NVIDIA CUDA® Deep Neural Network library (cuDNN) is a GPU-accelerated library of primitives for deep neural networks. ... It allows them to focus on training neural networks and developing software applications rather than spending time on low-level GPU performance tuning.

The Arduino support only person detection using the TensorFlow lite, but for the object detection other than person we need a high processing computer and need to install some python module like OpenCV for object detection in random condition.

40. Source Codes

40.0.1. Hand Landmark Tracking

```
import cv2                      # Import OpenCV Library
import mediapipe as mp          # Import the MediaPipe Module
import time                      # Import time Module

cap = cv2.VideoCapture(0)        # Setting up the Webcam

mpHands = mp.solutions.hands      # Use the Hand Landmark function of MediaPipe
hands = mpHands.Hands()
mpDraw = mp.solutions.drawing_utils # For making the connection between Landmarks

pTime = 0                        # (Initialize variables for Frame per second (FPS)
cTime = 0                        # Current time

while True:                      # Using While Loop for continues detecting
    success, img = cap.read()     # Read the detecting image
    imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) # Convert Color Using OpenCV
    results = hands.process(imgRGB)      # Save the converted Image
    #print(results.multi_hand_landmarks)    # Print the X,Y,Z Value of each landmarks

    if results.multi_hand_landmarks:      # Check if there is any detecting hand
        for handLms in results.multi_hand_landmarks: # When detecting the Landmark
            for id, lm in enumerate(handLms.landmark): # Handlandmarks according to Id no
                print(id, lm)                      # Print Handlandmarks as per ID
                h, w, c = img.shape               # Image shape
                cx, cy = int(lm.x * w), int(lm.y * h) # Saving the x, and y position of Landmark
                #print(id, cx, cy)                 # Print the X,Y Value of Each Landmark

            # if id == 4:                      # Only For Hand Landmarks # 4
            cv2.circle(img, (cx, cy), 5, (255, 0, 255), cv2.FILLED) #Draw circle again each LMS

        mpDraw.draw_landmarks(img, handLms, mpHands.HAND_CONNECTIONS) # Drawing LMS connect

    cTime = time.time()      # This is the calculation for Frame per second
    fps = 1 / (cTime - pTime)
    pTime = cTime
    # Frame Per second calculation end
    cv2.putText(img, str(int(fps)), (10, 70), cv2.FONT_HERSHEY_PLAIN, 3,
    (255, 0, 255), 3)      # Print the FPS for Hand Landmarks detection

    cv2.imshow("Image", img)  # For getting the result on desktop
    cv2.waitKey(1)           # Wait for one second
```

40.0.2. Test Example for Counting figure Using Landmark detection

```
import cv2                      # Import the OpenCV Library
```

```

import time          # Importing the time
import os           # Import os for folder selection
import HandTrackingModule as htm  # Import the Hand Tracking Module

wCam, hCam = 640, 480      # Setting the webcam width and height

cap = cv2.VideoCapture(0)    # Activate the Webcam for capturing
cap.set(3, wCam)
cap.set(4, hCam)

folderPath = "FingerImages"      # File name of save pictures
myList = os.listdir(folderPath)   # save the directory into variable myList
print(myList)                  # Print the myList variable
overlayList = []                # Initialize the array for saving images
for imPath in myList:           # To check which image is detect
    image = cv2.imread(f'{folderPath}/{imPath}')  # Take images from folder a
# print(f'{folderPath}/{imPath}')
overlayList.append(image)       # Appending the image

print(len(overlayList))         # Print the total finger

pTime = 0                      # For Per second

detector = htm.handDetector(detectionCon=0.75) # Setting the Detection Threshold

tipIds = [4, 8, 12, 16, 20]      # Saving the Fingertips Landmarks

while True:                     # While Loop for continues capturing
    success, img = cap.read()    # Read the image is there hand or not
    img = detector.findHands(img) # save the detected img
    lmList = detector.findPosition(img, draw=False) # Seting the lmList folder
# print(lmList)                 # It will show all the LMS

    if len(lmList) != 0:          # Check if there is any image available in the list
        fingers = []              # Initialize the empty array
        #if lmList[8][2] < lmList[6][2]: # For specific Case
        #print('Index finger open')    # Index Finger Landmark

        # Thumb (This is the code for thumb detection using tipIds)
        if lmList[tipIds[0]][1] > lmList[tipIds[0] - 1][1]:
            fingers.append(1)
        else:
            fingers.append(0)
        # Thumb detection code end

        # Fingers (This is the code for all remaining four fingers)
        for id in range(1, 5):  # For going through all the fingers which one is detected
            if lmList[tipIds[id]][2] < lmList[tipIds[id] - 2][2]: # Using tipIds of fingers
                fingers.append(1)          # Append the finger if it is detected
            else:
                fingers.append(0)          # If not detected leave it
            # Finger detection code end
            # print(fingers) # Print all the fingers
        totalFingers = fingers.count(1)  # Save the detecting finger in variable

```

```

print(totalFingers)                      # Print finger detect on Console

h, w, c = overlayList[totalFingers - 1].shape    # Setting the frame dimension
img[0:h, 0:w] = overlayList[totalFingers - 1]

# Making the rectangle using OpenCV for Number Counting
cv2.rectangle(img, (20, 225), (170, 425), (0, 255, 0), cv2.FILLED)
# Write the text in rectangle using OpenCV
cv2.putText(img, str(totalFingers), (45, 375), cv2.FONT_HERSHEY_PLAIN,
10, (255, 0, 0), 25)

# This is the code for Frame Per second (FPS)
cTime = time.time()
fps = 1 / (cTime - pTime)
pTime = cTime
# Frame per second code end

# This is OpenCV function for showing FPS on the screen, setting color and font size
cv2.putText(img, f'FPS: {int(fps)}', (400, 70), cv2.FONT_HERSHEY_PLAIN,
3, (255, 0, 0), 3)

cv2.imshow("Image", img)      # OpenCV function for showing the image
cv2.waitKey(1)                # Wait for one second

```

40.0.3. HandTracking Module

```

import cv2          # Import OpenCV Library
import mediapipe as mp  # Import MediaPipe Library
import time         # Import Time

class handDetector():    # Make the Hand Detector class for calling in main program
    def __init__(self, mode=False, maxHands=2, detectionCon=0.5, trackCon=0.5):
        self.mode = mode
        self.maxHands = maxHands
        self.detectionCon = detectionCon
        self.trackCon = trackCon

        self.mpHands = mp.solutions.hands
        self.hands = self.mpHands.Hands(self.mode, self.maxHands,
                                        self.detectionCon, self.trackCon)
        self.mpDraw = mp.solutions.drawing_utils

    def findHands(self, img, draw=True): # Write the findHands function for detecting hand
        imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        self.results = self.hands.process(imgRGB)
        # print(results.multi_hand_landmarks)

        if self.results.multi_hand_landmarks:
            for handLms in self.results.multi_hand_landmarks:
                if draw:
                    self.mpDraw.draw_landmarks(img, handLms,
                                              self.mpHands.HAND_CONNECTIONS)
            return img

```

```

def findPosition(self, img, handNo=0, draw=True): # Finding Position (X, Y) of each LMS

    lmList = []
    if self.results.multi_hand_landmarks:
        myHand = self.results.multi_hand_landmarks[handNo]
        for id, lm in enumerate(myHand.landmark):
            # print(id, lm)
            h, w, c = img.shape
            cx, cy = int(lm.x * w), int(lm.y * h)
            # print(id, cx, cy)
            lmList.append([id, cx, cy])
        if draw:
            cv2.circle(img, (cx, cy), 8, (255, 0, 255), cv2.FILLED)

    return lmList


def main(): # Main Function
    pTime = 0
    cTime = 0
    cap = cv2.VideoCapture(0)
    detector = handDetector()

    while True: # For continues detecting
        success, img = cap.read()
        img = detector.findHands(img)
        lmList = detector.findPosition(img)
        if len(lmList) != 0:
            print(lmList[4])

        cTime = time.time()      # This is for Frame per second
        fps = 1 / (cTime - pTime)
        pTime = cTime

        cv2.putText(img, str(int(fps)), (10, 70), cv2.FONT_HERSHEY_PLAIN, 3,
                   (255, 0, 255), 3) # For Writing text using OpenCV

        cv2.imshow("Image", img)
        cv2.waitKey(1)

    if __name__ == "__main__":
        main()

```

40.0.4. Detecting 6 Different Gesture Code Using Hand Landmarks

```

import cv2          # Import OpenCV Library
import mediapipe as mp # Import MediaPipe Module
import time         # Import Time

# import controllerad as cnt    # Import the Controllerad module

time.sleep(2.0)      # Sleep for 2 second

```

```
mp_draw = mp.solutions.drawing_utils # Drawing and Capturing the Hand landmarks
mp_hand = mp.solutions.hands # Saving in mp_hand variable

tipIds = [4, 8, 12, 16, 20] # Use the tip of each finger

video = cv2.VideoCapture(0) # Activate the Webcam

with mp_hand.Hands(min_detection_confidence=0.9,
min_tracking_confidence=0.9) as hands: # Set the detection and tracking confidence
while True: # For continues detection
    ret, image = video.read() # Read the image
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) # Convert color using OpenCV
    image.flags.writeable = False
    results = hands.process(image) # Save the results
    image.flags.writeable = True
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR) # Save images in variable
    lmList = [] # Initialize the empty list for image saving

    if results.multi_hand_landmarks: # Check if there is any Hand in the frame or not
        for hand_landmark in results.multi_hand_landmarks: # Detecting the Landmarks of hand
            myHands = results.multi_hand_landmarks[0] # Saving the landmarks in variable
            for id, lm in enumerate(myHands.landmark): # Check all the Ids of fingers
                h, w, c = image.shape # Saving the image shape as width and height
                cx, cy = int(lm.x * w), int(lm.y * h) # X, and Y Position
                lmList.append([id, cx, cy]) # Append the X, Y and id of each gesture

        mp_draw.draw_landmarks(image, hand_landmark, mp_hand.HAND_CONNECTIONS) # LMS Connec
        fingers = [] # Initialize empty array for saving detection

        if len(lmList) != 0: # Check if there is any hand detection or not
            fingers = []

            if (lmList[8][2] < lmList[6][2]) and (
                lmList[20][2] < lmList[18][2]): # For specific Case When Index and small finger open.
                print('Please Fast') # Sign will be Please fast for this gesture
                Reaction = 'Fast' # Save fast in reaction variable
                cv2.rectangle(image, (15, 345), (270, 380), (0, 255, 0), cv2.FILLED) # Draw rectangle
                cv2.putText(image, "Please Fast", (45, 375), cv2.FONT_HERSHEY_SIMPLEX,
                1, (255, 0, 0), 3) # Write text in rectangle
                cnt.led(Reaction) # Getting certain action on Arduino as per the Gesture

            elif (lmList[8][2] < lmList[6][2]) and (
                lmList[12][2] < lmList[10][2]): # For specific Case When Index and middle finger open
                print('Please Run') # Sign will be Please Run for this gesture
                Reaction = 'Run' # Save fast in reaction variable
                cv2.rectangle(image, (15, 345), (270, 380), (0, 255, 0), cv2.FILLED) # Draw rectangle
                cv2.putText(image, "Please Run", (45, 375), cv2.FONT_HERSHEY_SIMPLEX,
                1, (255, 0, 0), 3) # Write text in rectangle
                cnt.led(Reaction) # Getting certain action on Arduino as per the Gesture

            elif lmList[20][2] < lmList[18][2]: # For specific Case When only small finger open
                print('Please Slow') # Sign will be Please Slow for this gesture
                Reaction = 'Slow' # Save Slow in reaction variable
```

```

cv2.rectangle(image, (15, 345), (270, 380), (0, 255, 0), cv2.FILLED) # Draw rectangle
cv2.putText(image, "Please Slow", (45, 375), cv2.FONT_HERSHEY_SIMPLEX,
1, (255, 0, 0), 3) # Write text in rectangle
cnt.led(Reaction) Getting certain action on Arduino as per the Gesture

elif lmList[tipIds[0]][1] > lmList[tipIds[0] - 1][1] and (lmList[8][2] < lmList[6][2])
print('please stop') # Sign will be Please stop for this gesture
Reaction = 'Stop' # Save Stop in reaction variable
cv2.rectangle(image, (15, 345), (270, 380), (0, 255, 0), cv2.FILLED) # Draw rectangle
cv2.putText(image, "Please Stop", (45, 375), cv2.FONT_HERSHEY_SIMPLEX,
1, (255, 0, 0), 3) # Write text in rectangle
cnt.led(Reaction) # Getting certain action on Arduino as per the Gesture

elif lmList[tipIds[0]][1] > lmList[tipIds[0] - 1][1]: # When thumb shows left direction
print("Please Turn Left") # Sign will be Please turn left for this gesture
Reaction = 'Turn Left' # Save the Turn Left in Reaction variable
cv2.rectangle(image, (15, 345), (320, 380), (0, 255, 0), cv2.FILLED) # Draw rectangle
cv2.putText(image, "Please Turn Left", (45, 375), cv2.FONT_HERSHEY_SIMPLEX,
1, (255, 0, 0), 3) # Write text in rectangle
cnt.led(Reaction) # Getting certain action on Arduino as per the Gesture

elif lmList[tipIds[0]][1] < lmList[tipIds[0] - 1][1]: # When thumb shows right direction
print("Please Turn Right") # Sign will be Please turn right for this gesture
Reaction = 'Turn Right' # Save the Turn right in Reaction variable
cv2.rectangle(image, (15, 345), (335, 380), (0, 255, 0), cv2.FILLED) # Draw rectangle
cv2.putText(image, "Please Turn Right", (45, 375), cv2.FONT_HERSHEY_SIMPLEX,
1, (255, 0, 0), 3) # Write text in rectangle
cnt.led(Reaction) # Getting certain action on Arduino as per the Gesture

cv2.imshow("Frame", image) # Showing the results on desktop
if cv2.waitKey(10) & 0xFF == ord('q'): # For smooth quit please press q
break
video.release() # Getting out of programm
cv2.destroyAllWindows() # Stop all screen

```

40.0.5. Module for Running code on Arduino

```

import pyfirmata # Import the Pyfirmata Library

comport = 'COM3' # Selecting the COM3

board = pyfirmata.Arduino(comport) # Serial communication with Arduino

led_1 = board.get_pin('d:12:o') # Initialize Led on Digital pin 12
led_2 = board.get_pin('d:10:o') # Initialize Led on Digital pin 10
led_3 = board.get_pin('d:8:o') # Initialize Led on Digital pin 8
led_4 = board.get_pin('d:6:o') # Initialize Led on Digital pin 6
led_5 = board.get_pin('d:4:o') # Initialize Led on Digital pin 4
led_6 = board.get_pin('d:2:o') # Initialize Led on Digital pin 2

def led(Reaction): # Writing the Led Function
if Reaction == 'Run': # Check the reaction condition
# Only led_1 turn on, all the others off

```

```
led_1.write(1)      #
led_2.write(0)
led_3.write(0)
led_4.write(0)
led_5.write(0)
led_6.write(0)
elif Reaction == 'Fast': # Check the reaction condition
# Only led_2 turn on, all the others off
led_1.write(0)
led_2.write(1)
led_3.write(0)
led_4.write(0)
led_5.write(0)
led_6.write(0)
elif Reaction == 'Stop': # Check the reaction condition
# Only led_1, led_3, led_5 turn on, all the others off
led_1.write(1)
led_2.write(0)
led_3.write(1)
led_4.write(0)
led_5.write(1)
led_6.write(0)
elif Reaction == 'Slow': # Check the reaction condition
# Only led_4 turn on, all the others off
led_1.write(0)
led_2.write(0)
led_3.write(0)
led_4.write(1)
led_5.write(0)
led_6.write(0)
elif Reaction == 'Turn Left': # Check the reaction condition
# Only led_5 turn on, all the others off
led_1.write(0)
led_2.write(0)
led_3.write(0)
led_4.write(0)
led_5.write(1)
led_6.write(0)
elif Reaction == 'Turn Right': # Check the reaction condition
# Only led_6 turn on, all the others off
led_1.write(0)
led_2.write(0)
led_3.write(0)
led_4.write(0)
led_5.write(0)
led_6.write(1)
```


41. Magic Wand

- Als Mikrocontrollerboard wird ein Arduino Nano 33 BLE verwendet:

<https://docs.arduino.cc/hardware/nano-33-ble/>

- Betrieben wird das Controllerboard mit einer Lithium Batterie CR2032: [Ene]

<https://data.energizer.com/pdfs/cr2032.pdf>

- Der Stab ist aus Acryl:

<https://acrylhaus.com/Rundstab-aus-Acrylglas-XT-mit-Luftblasen>

- Für den 3D-Druck wurde PETG Filament verwendet:

[https://www.material4print.de/filament/petg/31/petg-tiefschwarz?
c=15](https://www.material4print.de/filament/petg/31/petg-tiefschwarz?c=15)

41.1. Attention

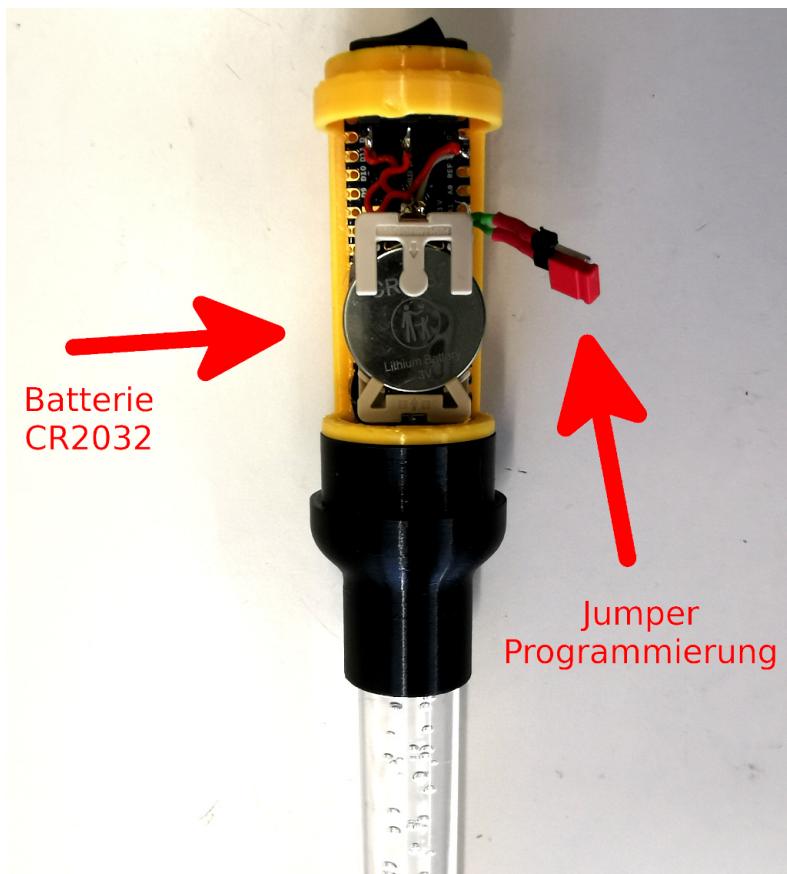
An drei Ausgangpins des Controllerboards ist, über Vorwiderstände von 180 Ohm, eine RGB-LED angeschlossen.

Der Zauberstab lässt sich über die eingebaute Micro-USB Schnittstelle programmieren.

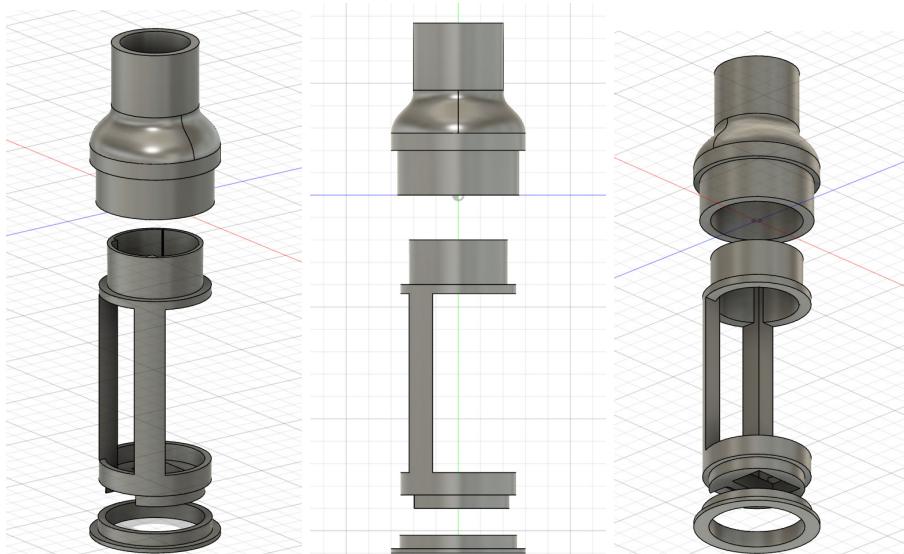
Dazu muss der Zauberstab geöffnet und der Jumper im inneren geschlossen (gesteckt werden).

ACHTUNG vor dem Stecken des Jumpers muss der Stab ausgeschaltet sein und er darf nicht eingeschaltet werden, solange der Jumper steckt!!

41.2. Gehäuse



Die Gehäusekonstruktion wurde mit Fusion 360 erstellt.



Listing 41.1.: Einschalten der Farben der RGB-Led

```

1000 // Zauberstab LED on/off
1001 // 240221 Thomas Peetz
1002 //
  
```

```

// An ein Microcontrollerboard Arduino Nano 33 BLE ist eine RGB-LED
// ueber 180 Ohm Vorwiderstaende angeschlossen .
1004 // Zuordnung der Pins GPIO 02 LED Blau
//          GPIO 03 LED Gruen
1006 //          GPIO 04 LED Rot
// Die verwendeten GPIO Pins koennen ueber PWM Signale angesteuert
// werden , damit ist eine Farbmischung moeglich .
1008 //

1010 int ledBlau = 2;
int ledGruen = 3;
1012 int ledRot = 4;

1014 void setup() {
    Serial.begin(115200);
1016 }

1018 void loop() {
    Serial.println("Rot");
1020 analogWrite(ledRot, 0);
    delay(1000);
    analogWrite(ledRot, 255);

1024 Serial.println("Blau");
    analogWrite(ledBlau, 0);
    delay(1000);
    analogWrite(ledBlau, 255);

1028 Serial.println("Gruen");
1030 analogWrite(ledGruen, 0);
    delay(1000);
    analogWrite(ledGruen, 255);
1032 }
}

```

..../Code/Nano33BLESense/MagicWand/MagicWandOnOff.ino

Listing 41.2.: Farbwechsel der RGB-Led

```

1000 // Zauberstab LED PWM
1001 // 240221 Thomas Peetz
1002 //
1003 // An ein Microcontrollerboard Arduino Nano 33 BLE ist eine RGB-LED
// ueber 180 Ohm Vorwiderstaende angeschlossen .
1004 // Zuordnung der Pins GPIO 02 LED Blau
//          GPIO 03 LED Gruen
1006 //          GPIO 04 LED Rot
// Die verwendeten GPIO Pins koennen ueber PWM Signale angesteuert
// werden , damit ist eine Farbmischung moeglich .
1008 //

1010 int ledBlau = 2;
int ledGruen = 3;
1012 int ledRot = 4;
int speed = 10;

1014 void setup() {
1016 }

1018 void loop() {
    for (int i=255; i>=0; i--){
        analogWrite(ledRot, i);
        delay(speed);
    }
    for (int i=0; i<=255; i++){
        analogWrite(ledRot, i);
        delay(speed);
    }
    for (int i=255; i>=0; i--){

```

```
1028     analogWrite(ledBlau, i);
1029     delay(speed);
1030 }
1031 for (int i=0; i<=255; i++){
1032     analogWrite(ledBlau, i);
1033     delay(speed);
1034 }
1035 for (int i=255; i>=0; i--){
1036     analogWrite(ledGruen, i);
1037     delay(speed);
1038 }
1039 for (int i=0; i<=255; i++){
1040     analogWrite(ledGruen, i);
1041     delay(speed);
1042 }
```

..../Code/Nano33BLESense/MagicWand/MagicWandPWM.ino

42. Development

This section talks about how the KDD process is implemented and the steps followed for each of the KDD Processes, starting with the preparation of Databases.

42.1. Why KDD for the magic wand?

The KDD process is an ideal and systematic approach for the Magic Wand project due to its comprehensive methodology tailored for deriving valuable insights from complex datasets. In the context of the Magic Wand, where the data involves diverse sensor readings capturing intricate gestures, the initial step of data exploration and understanding is paramount. KDD's emphasis on iterative processes aligns seamlessly with the project's evolving nature, allowing for continuous refinement and improvement based on feedback and insights gained from each iteration [FPSS96].

Gesture data requires thoughtful feature engineering to extract relevant information for model training, and KDD's structured approach addresses this by focusing on feature selection and extraction. Additionally, the KDD process incorporates essential steps such as data cleaning and preprocessing, addressing challenges such as noise, outliers, and missing values, crucial for ensuring the accuracy and reliability of the gesture recognition model [FPSS96].

42.2. Database

In the absence of pre-existing databases tailored for our specific model, we undertook the creation of a new database for our Magic Wand project. The database is structured as a .json file, housing raw data derived from accelerometer readings captured by the Arduino Nano 33 BLE Sense board. Acknowledging the inherent variability in how individuals record gestures, we engaged three members of our group to contribute, resulting in a richer and more diverse dataset. Each participant meticulously recorded a minimum of 70 trials for each gesture W, O, and L ensuring a comprehensive representation of gesture variations. This approach leverages the unique hand movements of three distinct users, enhancing the accuracy, efficiency, and effectiveness of the database.

In the subsequent phase of our project, titled "Data Preparation and Transformation," we plan to process the raw data obtained from the database. This involves identifying and addressing outliers if necessary, further refining the dataset for optimal performance in training our machine learning model. The comprehensive nature of our dataset, enriched by varied contributions and purposely limited in size, lays a solid foundation for the subsequent phases of our project, promising an effective and adaptable model for Magic Wand gesture recognition.

42.3. Data Collection and Creation

For the Magic Wand project, the dataset was meticulously crafted using the Magic Wand Capture sketch on an Arduino Nano 33 BLE Sense board. The following steps outline the process:

- 1. Upload Magic Wand Capture Sketch:**

Utilized the Magic Wand Capture sketch, specifically designed for Arduino Nano 33 BLE Sense boards, to record sensor readings during gesture performances. Uploaded the Magic Wand Capture sketch onto the Arduino Nano 33 BLE Sense board. This sketch is responsible for collecting and logging accelerometer data during wand movements with the help of tinyml website.

The website https://tinyml.seas.harvard.edu/magic_wand/ serves as a platform for the Magic Wand project, providing resources and tools for capturing and creating gesture datasets. Users can upload the Magic Wand Capture sketch to an Arduino Nano 33 BLE Sense board, connect it to a laptop via Bluetooth, and record distinct gestures.

2. Bluetooth Connectivity:

Established a connection between the Arduino Nano 33 BLE Sense board and a laptop using the integrated Bluetooth module. This connection facilitated real-time data transfer between the board and the laptop 42.1.

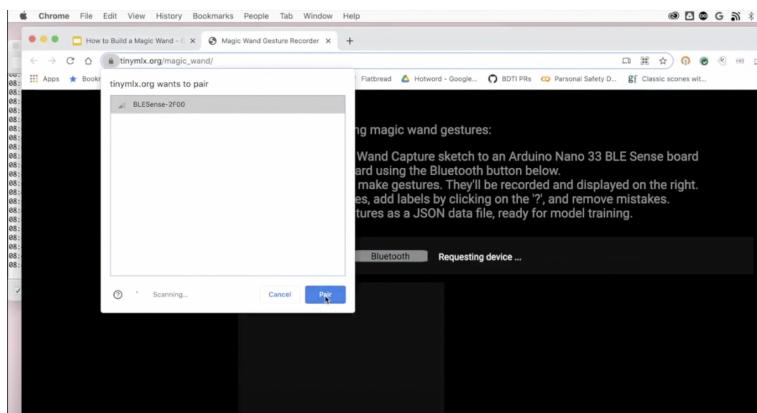


Figure 42.1.: Connecting the Arduino Nano BLE 33 Sensor through Bluetooth to create our own Dataset

3. Gesture Recording:

Waved the wand in distinct patterns to represent different gestures (W, O, and L). The accelerometer on the board captured the corresponding motion data in three dimensions (X, Y, Z) on the screen in real time.

The first set of data that we are trying to capture the movement is for the gesture W. The steps involved is mentioned below.



Figure 42.2.: Wing Gesture [WS20]

- The device is first moved down and to the right.
- Then the device is moved up and to the right.
- The device is then moved down and to the right.
- The device is then moved up and to the right again.

Shows a sample of real data captured during the "wing" gesture, measured in milli-Gs. The process involved in capturing a ring are as follows. [WS20]

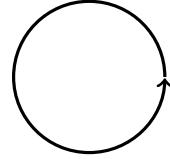


Figure 42.3.: Ring Gesture [WS20]

- Trace a clockwise circle using the wand.
- Aim again to take around a second to perform gesture.

The steps involved in waving the slope gesture is mentioned below [WS20]:.

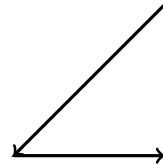


Figure 42.4.: Slope Gesture [WS20]

- The device is first moved down and to the left.
- Then the device is moved to the right.
- Finally you should get a corner of the triangle as shown in the figure.

The process is repeated until around 250 readings are captured by the wand for Data Preparation and Transformation.

In order to consider the unknown readings, we have decided that the gesture other than W, O and L are unknown. This would help the wand recognize any unknown gesture and gives the output stating that it is false. [WS20]

All readings that are recorded in a unique JSON file for each gesture. The files `.json` contains raw accelerometer data that would later be transformed in to machine learning language that should be interpreted by the compiler.

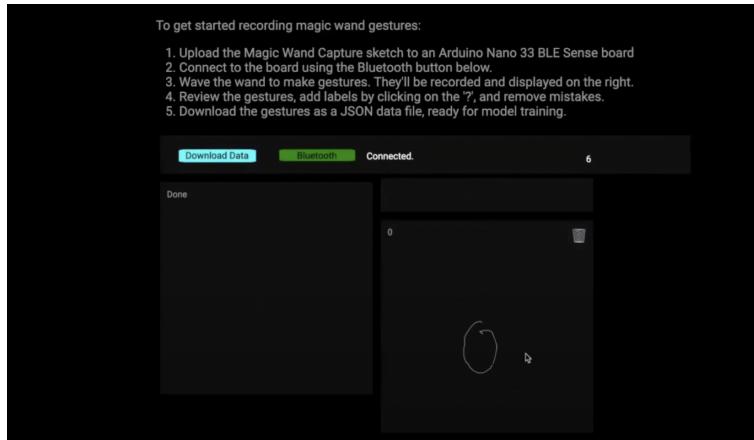


Figure 42.5.: Creating and collecting the data by moving the sensor to get correct gesture

4. Labeling Process:

Reviewed the recorded gestures on the laptop screen and labeled each gesture with the corresponding gesture name (W, O, or L) by clicking on the ? symbol in the user interface 42.5.

5. Error Correction:

Identified and corrected mistakes by removing any mislabeled or ambiguous instances, ensuring the dataset's quality.

6. Download as JSON:

Downloaded the labeled gestures as a JSON data file using the provided functionality on the project website. This file served as the foundation for model training.

42.4. Data Characteristics

1. Size:

The dataset consists of a total of 250 labeled instances, with over 70 instances for each gesture type (W, O, L).

2. Format:

The dataset is stored in JSON format, a versatile and widely used data interchange format. The JSON structure includes stroke indices, stroke points, and associated X-Y coordinates for each point.

3. Structure:

The JSON structure is organized into strokes, each containing an index and an array of stroke points with X-Y coordinates. This structured format is conducive to representing sequential information.

```
{
  "gesture_data": [
    {
      "label": "W",
      "sensor_readings": [
        {"acceleration_x": 0.23, "acceleration_y": 0.15, "acceleration_z": 0.05, "angle_x": 10, "angle_y": 20, "angle_z": 30}, {"acceleration_x": 0.21, "acceleration_y": 0.17, "acceleration_z": 0.07, "angle_x": 12, "angle_y": 22, "angle_z": 32}, ...
      ]
    },
    {
      "label": "O",
      "sensor_readings": [
        {"acceleration_x": 0.14, "acceleration_y": 0.12, "acceleration_z": 0.06, "angle_x": 5, "angle_y": 15, "angle_z": 25}, {"acceleration_x": 0.12, "acceleration_y": 0.13, "acceleration_z": 0.08, "angle_x": 6, "angle_y": 16, "angle_z": 26}, ...
      ]
    },
    {
      "label": "L",
      "sensor_readings": [
        {"acceleration_x": -0.10, "acceleration_y": -0.12, "acceleration_z": -0.05, "angle_x": -10, "angle_y": -20, "angle_z": -30}, {"acceleration_x": -0.12, "acceleration_y": -0.14, "acceleration_z": -0.07, "angle_x": -12, "angle_y": -22, "angle_z": -32}, ...
      ]
    },
    ...
  ]
}
```

4. Anomalies:

Emphasized clear and distinct wand movements during gesture performances to reduce anomalies. Applied manual review during the labeling process to identify and rectify any mislabeled or ambiguous instances.

5. Measurement and Screen Size:

- Accelerometer Measurements:

The accelerometer on the Arduino Nano 33 BLE Sense board captured motion data in three dimensions (X, Y, Z) during each gesture with the speed of 1 to 2 seconds for a gesture.

- Screen Size:

The laptop screen was utilized for real-time visualization that is 25X30 cm dimension and labeling of the recorded gestures. The user interface provided a platform to review, label, and correct the gestures.

6. Origin:

The origin of our Magic Wand project dataset lies in the convergence of real-world gestures and cutting-edge technology facilitated by the website. Through the collaborative efforts of our team, we employed the Magic Wand Capture sketch, uploading it onto the Arduino Nano 33 BLE Sense board. Leveraging the user-friendly interface provided by the website, we seamlessly recorded, reviewed, and labeled distinct gestures, ensuring a diverse dataset representative of various users and authentic real-world conditions. The Magic Wand prototype, in tandem with the Arduino Nano 33 BLE Sense board, served as the foundation for capturing nuanced variations in gestures. The website's structured approach played a pivotal role in refining our dataset, aligning with the core objectives of our Magic Wand project and enhancing its applicability to practical machine learning applications.

The advantage of creating a database with more inputs is that it allows the machine learning model to be more accurate and provide better and more efficient results. The exact process is repeated for the remaining gestures, namely ring, slope, and unknown. With the data available for the above gestures, we can use it to train a model. The available data is split into two: training data and test data.

The major challenge while preparing data can be the presence of outliers or anomalies with unexpected values. This can be overcome by feeding our model with more data so that the model is efficient. Here are some main types of outliers that may be encountered:[MO19]

- **Noise in Accelerometer Readings:**

Environmental noise or interference during data collection can introduce outliers in accelerometer readings. Sudden spikes or drops in sensor values that do not correspond to actual gestures may be considered noise outliers.

- **Abnormal Gesture Patterns:**

Outliers may occur when users perform gestures outside the expected range or in an unusual manner. These deviations from the standard gesture patterns can impact the model's training and recognition capabilities.

- **Sensor Malfunctions:**

Outliers might be caused by sensor malfunctions or inaccuracies in the Arduino Nano 33 BLE Sense device. Sudden jumps, constant offsets, or erratic behavior in sensor readings may indicate hardware issues.

- **Inconsistent Data Recording:**

Variability in the way users record gestures, such as differences in speed, amplitude, or duration, can introduce outliers. Harmonizing the dataset by addressing these inconsistencies is essential for training a robust and generalized gesture recognition model.

- **User-Specific Outliers:**

Each user may exhibit unique movement patterns or unintentional variations in how gestures are performed. Identifying and addressing user-specific outliers is crucial for creating a model that generalizes well across different individuals within the user group.

- **Data Transmission Errors:**

During data transmission from the Arduino Nano to the recording system, errors or interruptions can introduce outliers. Missing or corrupted data points in the dataset need to be identified and rectified to prevent negative impacts on model training.

42.5. Data Transformation and Data Mining

It is in the data mining step that the model is actually built. However before we start the process the algorithm that needs to be used has to be decided. As discussed in the previous section, the dataset is divided equally so that model testing and evaluation is also done to re-confirm the model works fine [WS20].

Training the Model

To commence our project, we need to modify one of the examples in the SparkFun Edge Board Support Package (BSP) to accommodate the input of the captured dataset. Initially, follow SparkFun's "Using SparkFun Edge Board with Ambiq Apollo3 SDK" guide to set up the Ambiq SDK and the SparkFun Edge BSP. Once the initial setup is complete, we can proceed to make the necessary code modifications [Lai22].

The program will then be ready to take the dataset as the input. We then need to build the example application and flash it to the device.

As described in the previous part, the three members of the group will perform the needed motions and construct the dataset in the following stage. To do so, open a terminal window and enter the following command:

`script output.txt`

In the next screen connect to “115200” Device. Post the measurements from the accelerometer will show up on the screen. The values will also be saved to a file `output.txt`. [WS20]

The text file will contain the data in accordance how it is required by the training set. In order to get a good quality dataset, the same gestures are repeated again as much as possible and then the program is exited. Then the next group member makes the similar file `output.txt`. We need to press the button marked 14 to stop logging the acceleration data.

We need to rename the `output.txt` folders as per the names of the person who performed the gestures, this helps to differentiate the datasets for testing and validation purposes. [WS20]

Data for unknown category also is then added, so that when a different gesture is shown, the model identifies it as an unknown gesture. The training then ramps up and the validation accuracy gets better with time. The following are the steps for training the model:

Table 42.1.: CNN sequence to classify gestures

Layer (type)	Output Shape	Param#
conv2d(Conv2D)	(None, 128, 3, 8)	104
max_pooling2d(MaxPooling2D)	(None, 42, 1, 8)	0
dropout(Dropout)	(None, 42, 1, 8)	0
conv2d_1(Conv2D)	(None, 42, 1, 16)	528
max_pooling2d_1(MaxPooling2D)	(None, 42, 1, 16)	0
dropout_1(Dropout)	(None, 42, 1, 16)	0
flatten(Flatten)	(None, 224)	0
dense(Dense)	(None, 16)	3600
dropout_2(Dropout)	(None, 16)	0
dense_1(Dense)	(None, 4)	68

- Loading the tensor board.
- Codes are run to begin the training on Pycharm.
- **data_augmentation** file is run to train the model better as it has changed values of acceleration as will it give the model more input values.
- We will then start to see output values appearing on the screen (These are the memory size of the model)

42.5.1. Model

In the model designed for this project, we transform a sequence of 128 three-axis accelerometer readings, which represents approximately five seconds of time, into an array of four probabilities: one for each gesture and one for "unknown." CNNs (Convolutional Neural Networks) are employed due to their effectiveness in capturing important information contained in the relationships between adjacent values [WS20]. A CNN with multiple layers has the capability to learn to recognize each gesture by analyzing its constituent parts. For instance, the network may learn to identify up-and-down motions and comprehend that combining two of these motions with appropriate z- and y-axis movements results in a "wing" gesture [WS20].

A CNN achieves this by learning a series of filters organized in layers, where each filter is trained to recognize a specific type of data feature. In the network's initial layer, a filter might learn to recognize a simple structure, such as a period of upward acceleration. Upon identifying such a feature, the filter communicates this high-level data to the subsequent layer of the network [WS20]. Outputs from earlier, more basic filters are amalgamated to construct larger structures in subsequent layers of filters. For example, the "W" shape in the "wing" gesture could be represented by a sequence of four alternating upward and downward accelerations.

This hierarchical learning process enables the CNN to progressively understand and recognize intricate patterns and gestures based on the input accelerometer readings, ultimately leading to robust and accurate gesture classification.

For data acquisition we are using IMU signals 42.6. IMU is the electronic component which contains the accelerometer. The IMU object comes from the Arduino LSM9DS1 library.

Convolutional layer directly receives network's input, which is a sequence of raw accelerometer data. The input's shape is provided in the input shape argument. It's set to (seqlength, 3, 1), where seqlength is the total number of accelerometer

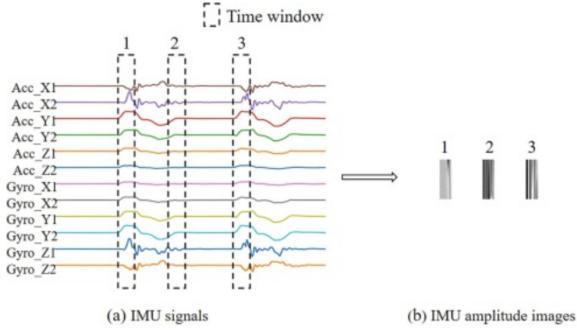


Figure 42.6.: IMU signals [Xu+22]

measurements that are passed (128 by default). Each measurement is composed of three values, representing the x, y, and z-axes [Xu+22] .

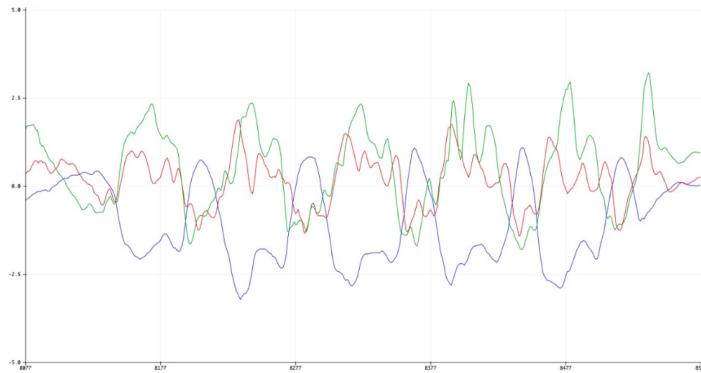


Figure 42.7.: IMU Accelerometer Graph

The job of the convolutional layer is to take this raw data and extract some basic features that can be interpreted by subsequent layers. The arguments to the Conv2D() function determine how many features will be extracted. Conv2D() is where we provide the dimensions of this window. In this case, it's (4, 3). This means that the features for which our filters are hunting span four consecutive accelerometer measurements and all three axes. Because the window spans four measurements, each filter analyses a small snapshot of time, meaning it can generate features that represent a change in acceleration over time. You can see how this works in 42.8.

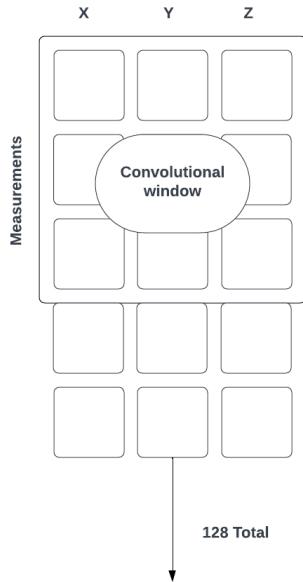


Figure 42.8.: A convolution window overlaid on the data

The padding argument determines how the window will be moved across the data. When padding is set to "same", the layer's output will have the same length (128) and width (3) as the input. Because every movement of the filter window results in a single output value, the "same" argument means the window must be moved three times across the data, and 128 times down it. As soon as the convolution window has moved across all the data, using each filter to create eight different feature maps, the output will be passed to our next layer, MaxPool2D. This MaxPool2D layer takes the output of the previous layer, a $(128, 3, 8)$ tensor, and shrinks it down to a $(42, 1, 8)$ tensor a third of its original size. It does this by looking at a window of input data and then selecting the largest value in the window and propagating only that value to the output. The process is then repeated with the next window of data. The argument provided to the MaxPool2D function, $(3, 3)$, specifies that a 3×3 window should be used. By default, the window is always moved so that it contains entirely new data. [WS20]

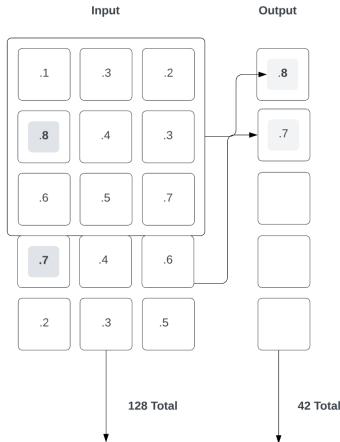


Figure 42.9.: Max Pooling

The goal of a CNN is to transform a big, complex input tensor into a small, simple output. The MaxPool2D layer helps make this happen. It boils down the output of our first convolutional layer into a concentrated, high-level representation of the relevant information that it contains. By concentrating the information, we begin to strip out things that aren't relevant to the task of identifying which gesture was contained within the input. Only the most significant features, which were maximally represented in the first convolutional layer's output, are preserved. After we've shrunk the data down, it goes through a Dropout layer. Dropout is a regularization technique; regularization is the process of improving machine learning models so that they are less likely to overfit their training data. Dropout is a simple but effective way to limit overfitting. By randomly removing some data between one layer and the next, we force the neural network to learn how to cope with unexpected noise and variation. Adding dropout between layers is a common and effective practice. The dropout layer is only active during training. During inference, it has no effect; all of the data is allowed through. [WS20]

This continues the process of distilling the original input down to a smaller, more manageable representation. The output, with a shape of (14, 1, 16), is a multidimensional tensor that symbolically represents only the most significant structures contained within the input data.

If we want to, we can continue the process of convolution and pooling. The number of layers in a CNN is just another hyperparameter that we can tune during model development. However, during the development of this model, we found that two convolutional layers was sufficient.42.10 shows the sequence.

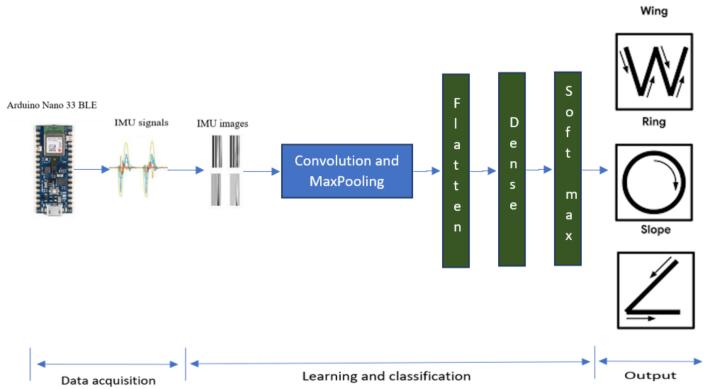


Figure 42.10.: CNN sequence to classify Wing,Ring and Slope

We flatten the data and feed it into a Dense layer (also known as a fully connected layer) to find out the major features within our input. The Flatten layer is used to transform a multidimensional tensor into one with a single dimension. In this case, our (14, 1, 16) tensor is squished down into a single dimension with shape (224). "It's then fed into a Dense layer with 16 neurons. This is one of the most basic tools in the deep learning toolbox: a layer where every input is connected to every neuron. By considering all of the data, all at once, this layer can learn the meanings of various combinations of inputs. The output of this Dense layer will be a set of 16 values representing the content of the original input in a highly compressed form".[WS20] Our final task is to shrink these 16 values down into 4 classes. This layer has four neurons; one representing each class of gesture. Each of them is connected to all 16 of the outputs from the previous layer. During training, each neuron will learn the combination of previous-layer activations that correspond to the gesture it represents. The layer is configured with a "softmax" activation function, which results in the layer's output being a set of probabilities that sum to 1. This output is what we see in the model's output tensor. "This type of model architecture—a combination of convolutional and fully connected layers—is very useful in classifying time-series sensor data like the measurements we obtain from our accelerometer. The model learns to identify the high-level features that represent the “gesture” of a particular class of input".[WS20]

Once we have a result from all the inferences in the model, in order to ensure there is no false positive the model's output tensor is given as input to a function called PredictGesture(). Two main actions are carried out by this function. It checks whether the gestures probability meets a minimum threshold and also checks if the gesture has been detected over a certain number of inferences.

The minimum number of inferences to confirm a gesture varies from the type of gesture as different gestures take different times to perform. The number of inferences required for each gesture is located in the file constants.cc. [WS20]

This function returns the predicted gesture by returning numeric values 0,1, 2,3 for Wing, Ring, Slope and unknown respectively.

Initially the "prediction scores" obtained from the main is passed on to the function in which a "maxPredictionScore" and "maxPrediction Index".

These values above are then compared "kNoGesture" and "kDetectionThrehsold" and then a value of found gesture variable is found which is equal to max prediction index.

This value is then returned to the main function and passed to output_handler() which then displays the recognised gesture. [WS20]

42.6. Evaluation and Verification

Once we get the model ready, it is important to test the model with a different dataset. This helps us confirm our model will work fine with gestures from other persons as well. To test the model we need to now call the Keras's `model.evaluate()` function.[WS20] We can also use a confusion matrix to check the performance of the model. An example of a confusion matrix according to the [WS20] is shown below which is calculated by `tf.math.confusion_matrix()` function:

$$\text{tf.Tensor} \left(\begin{bmatrix} 75 & 3 & 0 & 4 \\ 0 & 69 & 0 & 15 \\ 0 & 0 & 85 & 3 \\ 0 & 0 & 1 & 129 \end{bmatrix}, \text{shape} = (4, 4), \text{dtype} = \text{int32} \right)$$

The confusion matrix function tells us how much the predicted class of each input in the test dataset agrees to the actual value. It basically tells us the weak points of our model. We can then prepare the dataset again knowing the weak points and train the model once again.[WS20] which improves our model over time

42.7. Development Environment

The development environment for the Magic Wand project with Arduino Nano 33 BLE Sense encompasses a combination of hardware and software components tailored to facilitate the implementation of gesture recognition and other functionalities. This environment enables seamless integration between the Arduino Nano 33 BLE Sense board, machine learning algorithms, and communication protocols necessary for project execution. [Ard24a]

Hardware Components

1. **Arduino Nano 33 BLE Sense:** The core component of the project, featuring integrated BLE connectivity, motion sensors, environmental sensors, and a powerful microcontroller unit.

Software Components

1. **Arduino IDE:** The Arduino Integrated Development Environment (IDE) serves as the primary software platform for programming the Arduino Nano 33 BLE Sense board. It provides a user-friendly interface for writing, compiling, and uploading code to the board.
2. **ArduinoBLE Library:** This library facilitates Bluetooth Low Energy (BLE) communication and interaction with the Arduino Nano 33 BLE Sense board. It enables seamless integration of BLE functionalities into the project code.
3. **Machine Learning Frameworks:**
 - **TensorFlow:** A popular open-source machine learning framework used for developing and training neural network models. TensorFlow provides robust support for deep learning algorithms and is utilized for gesture recognition tasks in the project.
 - **Keras:** Built on top of TensorFlow, Keras offers a high-level neural networks API, enabling rapid prototyping and experimentation with deep learning models. It provides a user-friendly interface for designing and training neural networks, making it ideal for integrating machine learning capabilities into the project.

4. **Python:** As the primary programming language for machine learning tasks, Python is used extensively within the project environment. It facilitates interaction with TensorFlow, Keras, and other libraries, allowing for seamless development and deployment of machine learning models.
5. **Operating System:** The project development environment may be hosted on various operating systems, including Windows, macOS, or Linux distributions such as Ubuntu. Each operating system provides essential tools and utilities for software development and board communication.

By leveraging these hardware and software components within the development environment, the Magic Wand project aims to empower users to interact intuitively with the Arduino Nano 33 BLE Sense board through gesture recognition, opening avenues for diverse applications in IoT, robotics, and human-computer interaction.[Kur21c]

42.8. Versions

1. **Arduino IDE Version:** Specify the version of the Arduino IDE you are using. For example, Arduino IDE 1.8.19.
2. **ArduinoBLE Library Version:** Mention the version of the ArduinoBLE library you have installed. For instance, ArduinoBLE library version 1.2.0.
3. **TensorFlow Version:** If you're incorporating machine learning models for gesture recognition, specify the version of TensorFlow. For example, TensorFlow 2.7.0.[Ard24b]
4. **Keras Version:** Since you're using Keras for machine learning tasks, mention the version of Keras installed. For instance, Keras 2.8.0.
5. **Python Version:** Specify the version of Python installed on your system. For example, Python 3.10.0.[Fou24]
6. **Operating System:** Mention the operating system of your development environment, such as Windows 10, macOS Big Sur, or Ubuntu 20.04.
7. **Any other relevant library versions:** additional libraries like NumPy, SciPy, etc., mention their versions as well.

Providing detailed version information ensures reproducibility and compatibility for your project, allowing others to replicate your development environment accurately.

42.9. Description

It is an open source official Arduino software which used for editing, uploading and compiling codes in to the Arduino module. It is a cross-platform software which is available for Operating Systems like Windows, Linux, macOS. It runs on Java platform and supports a range of Arduino modules. It supports C and C++ languages. The microcontrollers present on the Arduino boards are programmed which accepts the information in the form of code. The program written in the IDE is called a sketch which will generate a Hex file which is then transferred and uploaded in the controller. The IDE environment is made up of two parts: an editor and a compiler. The editor is used to write the required code, while the compiler is used to compile and upload the code to the Arduino Module.[FAD18] The Menu bar has options such as File in which there are many options including Opening a new file or existing, Examples-in which we can find sketches for different applications like Blink, Fade etc. There is an error console at the bottom of the screen for displaying errors.

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



Figure 42.11.: **Menu button.**

- The check mark is used to verify your code. Click this once you have written your code.
- The arrow uploads your code to the Arduino to run.
- The dotted paper will create a new file.
- The upward arrow is used to open an existing Arduino project.
- The downward arrow is used to save the current file.
- The far right button is a serial monitor, which is useful for sending data from the Arduino to the PC for debugging purposes.

42.10. Installation

To install the Arduino IDE, we need to download the latest version from the Arduino webpage <https://www.arduino.cc/en/software>. We can select the version based on the operating system we are using. Here we are installing Arduino 1.8.15 for a Windows 10 operating system. The set up file name is arduino-1.8.15-windows.exe and the size of it is 1,174,470 KB. we can specify the path according to our needs. Here the path is set as **C:/Program Files (x86)/Arduino**.

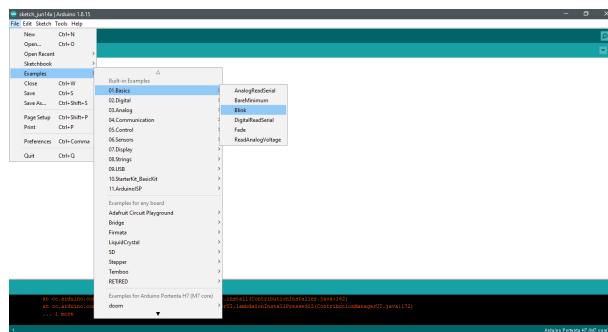


Figure 42.12.: **Menu bar options.**

After the download is done, open the setup file and proceed to install. Select all the components in the dialog box and click Next.

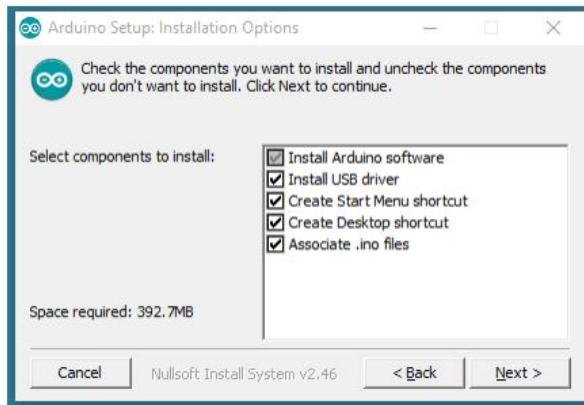


Figure 42.13.: **Arduino Setup Installation options.**

Select the destination folder and click Install

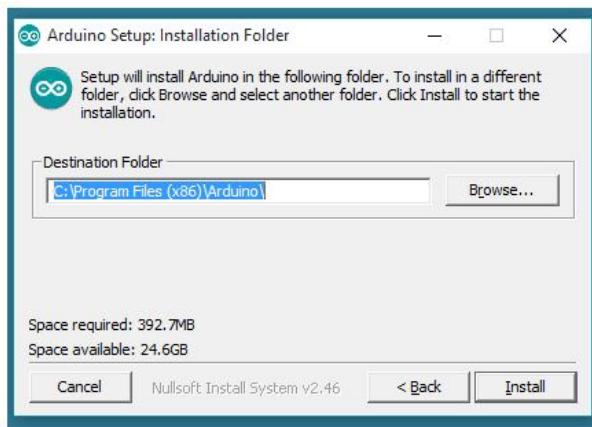


Figure 42.14.: Arduino Setup Installation Folder.

Once the installation is done, open the Arduino IDE and a default sketch appears on the screen as shows.

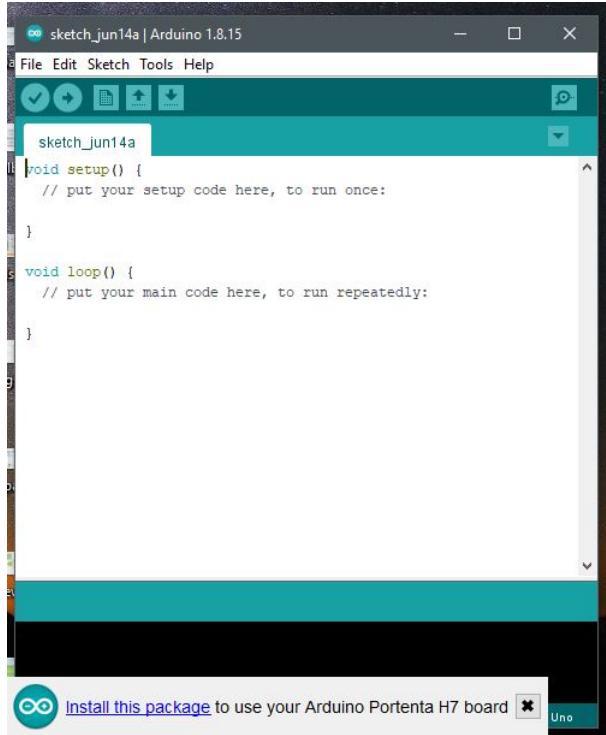


Figure 42.15.: Arduino Sketch.

It can be seen from the above figure that the basic arduino sketch has two parts. The first part is the function `void setup()` which returns void and we do the intiliaztion 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.

42.10.1. Arduino IDE on PC

Arduino Nano 33 BLE Sense uses the Arduino software integrated development environment (IDE) for programming, which is the most widely used and common (IDE) for all arduino boards that can be run online and offline. This is a open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. There are various version of software which is supported for each operating system (OS) e.g: mac, linux, and windows. Arduino community also provide us to start coding online and save our sketches in the cloud, this online arduino editor is most up-to-date version of the IDE includes all libraries and also supports new Arduino boards. For getting access to these software packages go to the following link <https://www.arduino.cc/en/software> and get more up to date inforamtion, because every single day there are some updates occurs which is available on the link mention above. These software can be used with any Arduino board, the most recent offline arduino IDE 1.8.15 can be seen in Figure,42.16. it is also supportive for all operating systems.



Figure 42.16.: Arduino Create Agent Installation.

42.11. Configuration

42.11.1. Configuration for the Arduino Nano 33 BLE Sense

To program the Arduino Nano 33 BLE Sense in offline state, we need to install one of the latest arduino IDE on our desktop. After installation, for getting access to the Arduino nano 33 ble sense board, we need to make configuration in our IDE. By opening the IDE, go to tool which can be seen on the uper left corner in IDE, in the tool there is an option for managed board. At this point we need to write our board name in the search which is Arduino Nano 33 BLE Sense as shown in figure,Select the Arduino Mbed OS Boards and install it. The Mbed OS nano board supports also other nano family boards including Arduino nano 33 ble sense, after installing simply connect the Arduino Nano 33 BLE Sense to the computer via USB cable.



Figure 42.17.: Arduino Mbed OS Nano Boards Installation.

42.12. First Steps

1. **Install Arduino IDE:** Begin by installing the Arduino Integrated Development Environment (IDE) on your computer. You can download the IDE from the official Arduino website download and follow the installation instructions for your operating system.[FAD18]
2. **Connect Arduino Nano 33 BLE Sense:** Use a USB cable to connect your Arduino Nano 33 BLE Sense board to your computer.
3. **Open Arduino IDE:** Launch the Arduino IDE application on your computer.
4. **Select Board:** In the Arduino IDE, navigate to the menu “Tools” and select “Board”. Choose “Arduino Nano 33 BLE” from the list of available boards.

5. **Select Port:** Still in the menu "Tools", go to the "Port" option and select the port to which your Arduino Nano 33 BLE Sense board is connected. On Windows, it will typically be something like "COMX", and on macOS, it will be "/dev/cu.usbmodemXXXX".
6. **Verify Connection:** Once the board and port are selected, verify that your Arduino Nano 33 BLE Sense board is successfully connected to the Arduino IDE. You can do this by checking the bottom right corner of the IDE window for a message indicating the board and port.
7. **Open Example Sketch:** To ensure that your setup is working correctly, you can open a built-in example sketch. Go to the "File" menu, select "Examples", then navigate to "01.Basics" and choose the "Blink" example. This sketch will blink the built-in LED on the Arduino Nano 33 BLE Sense board.
8. **Upload Example Sketch:** Click on the "Upload" button (right arrow icon) in the Arduino IDE toolbar to compile and upload the example sketch to your Arduino Nano 33 BLE Sense board. You should see the built-in LED blinking once the upload is complete.
9. **Verify Operation:** Verify that the LED on your Arduino Nano 33 BLE Sense board is blinking. This confirms that your development environment is set up correctly and your board is communicating with the Arduino IDE.
10. **Explore Further:** Now that you have successfully completed the initial setup and verification steps, you can explore more advanced features and functionalities of the Arduino Nano 33 BLE Sense board and the Arduino IDE. Consider experimenting with different example sketches, sensors, and communication protocols to familiarize yourself with the capabilities of your board.[Ard18]

Congratulations! You have completed the first steps in setting up the development environment for your Magic Wand project with Arduino Nano 33 BLE Sense. You're now ready to proceed with implementing your project's functionality and features.

42.13. Program "Hello World"

1. **Setup Arduino IDE:** Make sure you have the Arduino IDE installed on your development machine. You can download it from the official Arduino website and follow the installation instructions for your operating system.
2. **Connect Arduino Nano 33 BLE Sense:** Connect your Arduino Nano 33 BLE Sense board to your computer using a USB cable.[FAD18]
3. **Open Arduino IDE:** Launch the Arduino IDE on your computer.
4. **Select Board:** Go to the "Tools" menu, then select "Board", and choose "Arduino Nano 33 BLE" from the list of available boards.
5. **Select Port:** In the "Tools" menu, select the port to which your Arduino Nano 33 BLE Sense is connected. It will typically appear as something like "COMX" on Windows or "/dev/cu.usbmodemXXXX" on macOS.
6. **Open New Sketch:** Click on "File" > "New" to open a new sketch.
7. **Write Code:** In the new sketch window, write the following code:

```

void setup() {
  // Initialize serial communication
  Serial.begin(9600);

  // Wait for serial monitor to open
  while (!Serial);

  // Print "Hello, World!" to serial monitor
  Serial.println("Hello, World!");
}

void loop() {
  // Empty loop
}

```

8. **Upload Code:** Click on the "Upload" button (right arrow icon) to compile and upload the code to your Arduino Nano 33 BLE Sense board.
9. **View Output:** Once the upload is complete, open the Serial Monitor by clicking on "Tools" > "Serial Monitor" (or press Ctrl+Shift+M). You should see "Hello, World!" printed in the Serial Monitor at a baud rate of 9600.
10. **Verify Operation:** Verify that "Hello, World!" is successfully printed in the Serial Monitor. This confirms that your Arduino Nano 33 BLE Sense board is properly set up and functioning.

programmed a "Hello, World!" example for your Magic Wand project using Arduino Nano 33 BLE Sense. This serves as a basic test to ensure that your development environment is configured correctly and your hardware is working as expected.

42.13.1. 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 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.

Part V.

Add Ons

43. Bootloader

Ein Bootloader ist ein wichtiger Bestandteil von Computersystemen und Mikrocontrollern. Ein Bootloader, auch als Bootprogramm oder Bootstrap-Loader bekannt, ist eine spezielle Software, die nach dem Start in den Arbeitsspeicher eines Computers geladen wird. Seine Hauptaufgabe besteht darin, das eigentliche Betriebssystem oder die Anwendungssoftware zu laden und auszuführen.

Einige Mikrocontroller verfügen über wenig oder gar keinen internen Flash-Speicher. In solchen Fällen übernimmt der Bootloader das Kopieren von Code aus nicht ausführbarem Speicher (z. B. SPI-Flash) in einen ausführbaren Bereich wie den RAM. Der Bootloader ermöglicht die Trennung von kritischen Teilen des Programms (z. B. Firmware-Updates) von sich häufig änderndem Anwendungscode. Ein Bootloader kann eine kryptografische Signatur überprüfen, um sicherzustellen, dass die Anwendung nicht ersetzt oder manipuliert wurde.

Der Bootloader muss zwei Hauptaufgaben erfüllen:

1. Beim MCU-Start ausführen: Der Bootloader startet automatisch, wenn der Mikrocontroller eingeschaltet wird.
2. Zum Anwendungscode springen: Nach dem Start lädt der Bootloader die Anwendungssoftware und springt zu deren Ausführung.

Der Bootloader liest die Anwendungssoftware aus dem Speicher und startet sie.

43.1. Kopieren eines Arduinos

<https://docs.arduino.cc/tutorials/nano-33-ble/getting-started-omv/>
<https://support.arduino.cc/hc/en-us/articles/4841602539164-Burn-the-bootloader-on-an-Arduino-Nano-30/>
<https://www.instructables.com/How-To-Burn-a-Bootloader-to-Clone-Arduino-Nano-30/>
<https://forum.arduino.cc/t/how-to-re-flash-firmware-to-an-arduino-33-ble-sense/1053526>

44. Arduino's Science Journal

Das Arduino Science Journal ist eine kostenlose und benutzerfreundliche App, die speziell für den Schulunterricht im Bereich Naturwissenschaften entwickelt wurde. Hier sind die wichtigsten Punkte: []

1. **Zweck:** Das Arduino Science Journal ermöglicht Schülern, wie echte Wissenschaftler zu denken und mit Daten zu arbeiten.
2. **Einfach zu verwenden:** Die App ist einfach zu bedienen und kostenlos für Android- und iOS-Geräte verfügbar.
3. **Wissenschaftliche Methode:** Schüler lernen die wissenschaftliche Methode kennen und können Experimente durchführen.
4. **Sensoren nutzen:** Mit eingebauten Sensoren können Schüler ihre Umgebung erkunden und Daten zu Licht, Bewegung und Klang aufzeichnen.
5. **Experimente durchführen:** Die App bietet Dutzende von kostenlosen Experimenten zu verschiedenen Themen wie Physik, Mathematik, Chemie und Biologie.
6. **Daten aufzeichnen:** Schüler können Daten aufzeichnen, speichern, exportieren, Diagramme erstellen und Notizen hinzufügen.
7. **Teilen und vergleichen:** Experimente können mit anderen geteilt und verglichen werden.
8. **Lehrplanabstimmung:** Die App entspricht den Lehrplänen der Next Generation Science Standards (NGSS) in den USA und dem National Curriculum of England.
9. **Anpassbarkeit:** Lehrer können die App an verschiedene Lernsituationen anpassen.
10. **Lehrerplan:** Der Lehrerplan ist ein Abonnementdienst, der die Integration der App mit Google Classroom ermöglicht.
11. **Experimente erstellen:** Lehrer können Experimente direkt in der App erstellen und Aufgaben festlegen.
12. **Schülerengagement steigern:** Das neue Design macht die App für Schüler ansprechender.
13. **Effizienz:** Alles an einem Ort - keine Notwendigkeit, zwischen verschiedenen Plattformen zu wechseln.
14. **Unterrichtszeit optimieren:** Mehr Zeit für praktisches Lernen im Klassenzimmer.
15. **Benutzerbewertungen:** Lehrer und Schüler schätzen die App für ihre praktische Anwendung und Effektivität.

1. Zitieren von NGSS

2. Beispiele!

3. ...

<https://www.instructables.com/How-to-Use-the-Arduino-Science-Journal-With-the-N>

45. Edge Impulse with the Arduino Nano 33 BLE Sense

<https://docs.arduino.cc/tutorials/nano-33-ble/getting-started-omv/>

Part VI.

To Do

46. To Do

46.1. Task: TensorFlow Lite

46.1.1. Description

46.1.2. Installation

- Prerequisites

- Step-by-Step

- Results

- Directory

- Files

- Version

- License

46.1.3. Usage

- Options

- Input

- Output

46.1.4. Conversion/Optimization

What does TensorFlow do?

Quantization

Optimization

...

46.1.5. Filetypes

Filetype Tensorflow-Save

Filetype .tflite

Filetype .h

46.1.6. Restrictions

46.1.7. Tensorflow lite Micro

Arduino's Packages

Description

Class Diagram

Usage

Example Magic Wand

46.2. Arduino Nano 33 BLE Sense

Development environments for the Arduino Nano 33 BLE Sense

Automation of basic function tests

46.2.1. Arduino Web Editor

Configuration of the Arduino Web Editor

Description of the IDE

First Steps with the Arduino Nano 33 BLE Sense Lite using the Arduino Web Editor

Licence

Serial Monitor

Serial Plotter

Determining the basic functions of an Arduino Nano BLE 33 Sense

46.2.2. Arduino IDE 2.x.x

Configuration of the Arduino IDE 2.x.x

Description of the IDE

First Steps with the Arduino Nano 33 BLE Sense Lite using the Arduino IDE 2.x.x

Licence

Serial Monitor

Serial Plotter

Determining the basic functions of an Arduino Nano BLE 33 Sense

46.2.3. Command Line Interpreter CLI

Description of the CLI

Configuration of the CLI

Options

First Steps with the Arduino Nano 33 BLE Sense Lite using the CLI

Licence

Serial Monitor

Serial Plotter

Determining the basic functions of an Arduino Nano BLE 33 Sense

Test of the basic functions of an Arduino Nano BLE 33 Sense

Automation of the tests with a logging function

46.3. IMU

- General information about IMUs
- Special consideration of the IMU
- Calibration of an IMU
- Calibration of the special IMU
- Drift
- Algorithms for evaluating an IMU

- Evaluation of the special IMU
- Programming the IMU
- Generation of IMU data for gait patterns
- Description of the data

46.4. Camera module OV7675

- General information about camera modules
- Special consideration of the camera module
- General calibration of cameras
- Calibration of the camera module
- Calibration with the Arducam lens calibration tool, field of view (FoV) test diagram folding map
- Algorithms for evaluating a camera
- Evaluation of the special camera
- Programming the camera
- In addition to the kit, you will receive an Arducam lens calibration tool
- Creation of a database of images on the PC via the serial port without additional tool
- Description of the data

46.5. Light/Color Sensor

46.5.1. Light

46.5.2. Color

46.5.3. Color Models

RGB

HSV

...

46.5.4. Description of the light/color sensor

Description of the hardware

Description of the functionality

Description of the used algorithm

Description of use and evaluation

Example Programs

Description of the Application

Creation of a Database

Description of the Data

46.6. Speech Recognition with the Package

[**DSpotterSDK_Maker_33BLE**](#)

46.6.1. Description of Speech Recognition

46.6.2. Description of the library and its handling

46.6.3. Description of the functionality

46.6.4. Description of the algorithm used

46.6.5. Description of use and evaluation

46.6.6. Example programs

46.6.7. Description of the microphone

Description of the hardware

Description of the functionality

Description of use and evaluation

Example programs

46.7. KDD

KDD – Dateien, Bilder wie CRISP-DM

46.8. Creating a Template

Template HW

46.9. Tasks

46.9.1. Task “Jetson Orin”

There 4 tasks:

- Hardware Description "Jetson ORIN 4011" with 5 tikz pictures (board, interfaces,...)
- Description of TensorFlow Lite; consider the given file TensorFlowLite
- Package “mediapipe”
- Presentation “Stable Diffusion”, ca. 50 slides

46.9.2. Task “DepthAI OAK-D-Pro”

There 4 tasks:

- Hardware Description “Luxonis DepthAI OAK-D-Pro” with 5 tikz pictures
- Description of TensorFlow Lite: consider the given file TensorFlow Lite
- Package “BeautifulSoup”
- Presentation “Dall-e”, ca. 50 slides

46.9.3. DepthAI OAK-D-Pro auto focus

Key-Features

- 3D-Stereo Camera & OnBoard Image Processing
- Based on Intel Movidius Myriad X
- 2 x 1280 x 720 Stereo Camera & IR Projector
- Initial commissioning in < 30 seconds
- USB-C connectivity

[comprehensive e-manual](#)

Difference between OAK-D & OAK-D-Pro

The biggest difference between the two cameras is the IR projector of the OAK-D-Pro. This enables the creation of high-resolution point clouds even in dark environments and/or under poor lighting conditions.

The Luxonis DepthAI boards combine stereo or RGB cameras with the Intel Movidius Myriad X platform. The result is a small board that can be operated either stand-alone or with an additional device such as an NVIDIA Xavier Jetson, Raspberry Pi or many more. Due to the high performance Intel µC with a base clock of 700 MHz, this small & compact development board can calculate about 4.5 million operations in less than one second.

What is the concept behind the Luxonis boards?

The manufacturer wants to create a versatile platform with an extremely small form factor, on the basis of which anyone can execute a first basic script in under 30 seconds. The development platforms are available in different configurations, such as with WLAN or a Raspberry PI Compute Module. Should the first tests prove promising, Luxonis also offers only the pure bare PCBs, which can be integrated as an OEM into a third-party component.

What is DepthAI?

Every Luxonis development board has the so-called DepthAI Firmware, which comes with the following functionalities out of the box:

- Neural inference (e.g. object recognition, image classification, etc., including two-stage)
- Stereo depth (including median filtering)
- 3D object localisation (complementing 2D object detectors with 3D position in metres)
- Object tracking (also in 3D space)
- H.264 and H.265 encoding (HEVC, 1080p & 4K video)
- JPEG encoding
- MJPEG encoding
- Warp/Dewarp

Technical data

- 2 x OV9282 Stereo camera with 1280 x 720Px
- 1 x IMX378 fixed focus camera
- 1 x IR Projector & IR LED
- Intel Movidius Myriad X Plattform
- Intel Base clock: 700MHz
- Intel lithography: 16nm

Scope of delivery

1 x Luxonis DepthAI OAK-D Pro

Examples

- [comprehensive e-manual](#)
- [e-manual](#)
- [examples](#)
- [Introduction to OAK-D and DepthAI](#)
- [Introduction to OpenCV AI Kit \(OAK\)](#)

46.9.4. Task “Google Coral Micro”

There 4 tasks:

- Hardware description “Google Coral Micro” with 5 tikz pictures (board, interfaces,...)
- Description Edge Impulse
- Package “Tkinter”
- Presentation “Google Bard”, ca. 50 slides

46.9.5. Task “Google Dev Board”

There 4 tasks:

- Hardware description “Google Dev Board” with 5 tikz pictures
- Installation OS for Google Dev Board and First steps
- Package “Natural Language Toolkit (NLTK)”
- Presentation “LangChain”, ca. 50 slides

46.9.6. Task “Raspberry Pi 400”

There 4 tasks:

- Hardware description "Raspberry Pi 400" with 5 tikz pictures
- Description of the framework “tinygrad”
- Package “Seaborn”
- Presentation “Python code documentation with Sphinx”, ca. 50 slides

46.9.7. Task “ESP32”

There 4 tasks:

- Hardware description “ESP32” with tikz pictures (board, sensors on board, pins, interfaces, ...)
- Template for the KDD process with hardware
 - Creating all directories
 - all necessary LaTeX projects
 - contents and structure for all LaTeX projects
 - comments and hints for the LaTeX project
 - comments/hints for the evaluation sheet; consider the given file EvaluationHW.xlsx.
 - ...
- Package “PixelLib”
- Presentation “TinyML”, ca. 50 slides + additionally 40 articles

46.9.8. Task “Arduino Nano 33 BLE Sense”

There 4 tasks:

- Hardware description “Arduino Nano 33 BLE Sense” with tikz pictures (board, sensors on board, pins, interfaces, ...) and simple examples with code for the sensors “microphone”, “IMU”, “Arducam OV2640”, and “ADPS-9960”
- Template for the KDD process without hardware
 - Creating all directories
 - all necessary LaTeX projects
 - contents and structure for all LaTeX projects
 - comments and hints for the LaTeX project
 - comments/hints for the evaluation sheet; consider the given file EvaluationSW.xlsx.
 - ...
- Package “LineaPy”
- Presentation “Imagen”, ca. 50 slides

46.10. IndIn Student

1. QR-code - documentation and presentation
2. TensorFlow Lite - documentation and presentation
3. TensorFlow Lite micro - documentation and presentation
4. Camera
5. Camera calibration
 - Calibration
 - Camera errors
 - Grayscale
 - Color
 - Distortion
 - Focus
6. Saving pictures from the Arduino to the PC
7. BLE (optional)

46.11. Radio

Teufel Radio ONE

46.12. Yahboom

<https://www.amazon.de/Yahboom-Educational-Engineering-Electronics-Ubuntu20-04/dp/B0CWTYQGBT/>
<https://category.yahboom.net/products/large-tracking-map>

46.13. ESP32CAM

<https://www.hackster.io/mvtdesign/ip-camera-esp32cam-arduino-code-video-audio-e6>

46.14. Power BI

<https://powerbi.microsoft.com/de-de/desktop/>
<https://www.microsoft.com/de-de/power-platform/products/power-bi?market=de>

46.15. TinyML

<https://forum.arduino.cc/t/tinyml-tutorial-problems-with-the-arduino-nano-33-bluepill/1138100>
<https://github.com/tensorflow/tflite-micro/issues/2207>

46.16. Magic Wand

<https://www.instructables.com/Magic-Wand/>
<https://www.instructables.com/Making-Famous-Magic-Wand-33x-Faster/>
<https://github.com/googlecreativelab/astrowand/blob/main/README.md>

46.17. L^AT_EX

<https://github.com/xu-cheng/texlive-action>
<https://tex.stackexchange.com/questions/664809/compile-latex-with-github-actions>
<https://dev.to/mrturkmen/latex-with-github-actions-4580>

```

1000 % Hello
1001 import matplotlib
1002
1003 if (!APDS. begin ()) {
1004     Serial . println ( " Error initializ ing APDS9960 sensor . " );
1005 }
1006 // oder
1007 while (!APDS. colorAvailable ()) {
1008     delay (5);
1009 }
```

```

1000 % Hello
1001 import matplotlib
1002
1003 if (!APDS. begin ()) {
1004     Serial . println ( " Error initializ ing APDS9960 sensor . " );
1005 }
1006 // oder
1007 while (!APDS. colorAvailable ()) {
1008     delay (5);
1009 }
```

46.18. To Do

- Alles neu sortieren!
- lstlisting Farben definieren
- Verwendung von tikz
- Beschreibung des Serial Monitors
- Beschreibung des Serial Plotters
- Beschreibung des Motherboards
- Beschreibung ders LEDs
- Beschreibung des Tasters

46.19. French Student I

- Hardware description "Arduino Nano 33 BLE Sense"
 - tikz pictures board,
 - sensors on board,
 - pins,
 - interfaces,
 - ...
- simple examples with code for the sensors
 - microphone,
 - IMU,
 - Arducam OV2640,
 - ADPS-9960
 - ...
- TinyMLOverview
- TinyMLShort
- TinyMLFundamentals
- TinyMLApplications: Magic Wand
- TinyMLDeployment
- Battery
- TensorFlowLite
- MagicWandProblem
- MagicWandProject
- Nano33BLESense - Project
- BlueTooth
- Aufbau mit Anleitung

46.20. French Student II

- Hardware description "Raspberry Pi 400 with 5 tikz pictures
- Installation OS
- Description Camera Shutter
- Test Camera Shutter
- Camera Calibration - General
- LensCalibrationTool

46.21. Pixy CAM

- Hardware description with 5 tikz pictures
- Assembly
- Description of Camera
- Test Following
- Camera Calibration - General
- LensCalibrationTool

46.22. Potenzielle Aufgaben

1. Bluetooth
2. [GET STARTED WITH ARDUINO NANO 33 BLE](#)
3. Display Waveshare Pico-OLED-1.3
4. Kommandozeileninterpreter CLI: Automatisierung von Tests der Grundfunktionen
 - Arduino IDE
 - Konfiguration der Arduino IDE
 - Erste Schritte mit dem Arduino Nano 33 BLE sense mit Hilfe der Arduino
 - Serieller Monitor
 - Bestimmung der Grundfunktionen
 - Installation und Verwendung der CLI
 - Konfiguration der Arduino CLI
 - Erste Schritte mit dem Arduino Nano 33 BLE sense mit Hilfe der CLI
 - Serieller Monitor mittels der CLI
 - Test der Grundfunktionen
 - Automatisierung der Tests mit einer Loggingfunktion
5. IMU
 - Allgemeine Information über IMUs
 - Spezielle Betrachtung der IMU
 - Kalibrierung einer IMU
 - Kalibrierung der speziellen IMU

- Drift
- Algorithmen zur Auswertung einer IMU
- Auswertung der speziellen IMU
- Programmierung der IMU

6. Kameramodul OV7675

- Allgemeine Information über Kameramodule
- Spezielle Betrachtung des Kameramoduls
- Allgemeine Kalibrierung von Kameras
- Kalibrierung des Kameramoduls
- Kalibrierung mit dem Arducam Objektivkalibrierungswerkzeug, Sichtfeld (FoV) Testdiagramm Faltkarte
- Algorithmen zur Auswertung einer Kamera
- Auswertung der speziellen Kamera
- Programmierung der Kamera

7. Mikrophon

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

8. Feuchtigkeitssensor, Temperatur, Luftdruck,

9. Gesteuerung

10. Pulsweitenmodulation

11. Abstandssensor

12. Licht-/Farbsensor

13. Protokoll I²C

14. Protokoll SPI

15. Serielles Protokoll über UART

16. Steuerung eines Schrittmotors

17. Steuerung eines Servomotors

18. Erstellung eines EtherCAT-Slaves mit Hilfe von EasyCAT-Shield

- Beschreibung des EtherCAT-Protokolls
- Beschreibung der Hardware
- Weitere Informationen
 - <https://www.bausano.net/shop/en/home/1-arduino-ethercat.html>
 - <https://www.tinytronics.nl/shop/en/communication-and-signals/ethernet/modules/easycat-shield-arduino-ethercat-shield>

– <https://www.youtube.com/watch?v=5nkwYL-za5g>

19. Laufzeitverhalten

- Beschreibung, wie Programme abgearbeitet werden.
- Beschreibung des Laufzeitverhaltens einzelner Komponenten, z.B. der Sensoren
- Beschreibung, wie das Laufzeitverhalten von Programmen beeinflusst werden kann.
- Interne Messung des Laufzeitverhalten von Programmen
- Externe Messung des Laufzeitverhalten von Programmen
- Optimierung des Laufzeitverhalten von Programmen.

20. Optimierung der Speichernutzung

- Messung der Speicherbedarfs eines Programms
- Kriterien und Methoden zur Optimierung des Speicherbedarfs von Programmen.
- Exemplarische Durchführung der Optimierung an geeigneten Programmen.

21. Low Power

- Messung des Stromverbrauchs
- Wechsel in den Schlafmodus
- Aufwecken aus dem Schlafmodus
- Abschalten einzelner Sensoren
- Entwicklung von Maßnahmen zur Reduzierung der Stromaufnahme
- Entwicklung von Kriterien zur Wahl einer Stromquelle, z.B. Batterie oder Powerbank
- Wahl einer geeigneten Powerbank
- Versuchsaufbau zur Verifizierung der Maßnahmen und Berechnungen

46.23. 2.Teil

- [TinyTrainable](#)
- [Get Started With Machine Learning on Arduino – Arduino Documentation](#)
- [Arduino TinyML Kit Tutorial #4: IMU sensors, Barometer and Microphone on the Nano 33 BLE – YouTube](#)
- [Arduino TinyML Kit Tutorial #5: BLE Communication on the Nano 33 BLE – YouTube](#)
- [Arduino Nano 33 BLE OV7670 Camera Shield – Arduino Project Hub](#)
- [deepC – Arduino Reference](#)
- [EdgeML-Arduino – Arduino Reference](#)
- [Board Installation – Arduino Documentation](#)
- [DSpotterSDK_Maker_33BLE – Arduino Reference](#)
- [Fruit identification using Arduino and TensorFlow](#)

- Optimizing a low-cost camera for machine vision
- Simple machine learning with Arduino KNN
- Bücher;
 - [Kur21b]
 - [Kur21a]
 - [Sen19]
 - [Smy21]
- TinyML, Seite 236
- MLbib/tinyML prüfen
- <https://www.arduino.cc/en/Reference/Libraries>

47. First Steps with the Nano 33 BLE Sense

48. First Steps with the ArduCAM

49. Training a Custom Machine Learning Model for Nano 33 BLE Sense

50. Use TensorFlow Lite with Nano 33 BLE Sense

Die Firma Arduin stellt zur Verwendung von TensorFlow-Modellen eine Bibliothek zur Verfügung, die über den Bibliothek-Manager installiert wird.

Ebenso stellt Google über TensorFlow Beispiele, die zeigen, wie man TensorFlow-Modelle mit Mikrocontroller verwendet.

see [TensorFlow Lite for Microcontrollers](#)

The TensorFlow Lite Micro Library is no longer available in the Arduino Library Manager. This library will need to be manually downloaded, and included in your IDE.

50.1. GitHub

The officially supported TensorFlow Lite Micro library for Arduino resides in the GitHub repository [tflite-micro-arduino-examples](#). To install the in-development version of this library, you can use the latest version directly from the GitHub repository. This requires you clone the repo into the folder that holds libraries for the Arduino IDE. The location for this folder varies by operating system, but typically it's in `~/Arduino/libraries` on Linux, `~/Documents/Arduino/libraries/` on MacOS, and `My Documents/Arduino/Libraries` on Windows.

Once you're in that folder in the terminal, you can then grab the code using the git command line tool:

```
git clone https://github.com/tensorflow/tflite-micro-arduino-examples  
Arduino_TensorFlowLite
```

To update your clone of the repository to the latest code, use the following terminal commands:

```
cd Arduino_TensorFlowLite  
git pull
```

50.2. Checking your Installation

Once the library has been installed, you should then start the Arduino IDE. You will now see an [Arduino_TensorFlowLite](#) entry in the menu [File → Examples](#) of the Arduino IDE. This submenu contains a list of sample projects you can try out.

50.3. Compatibility

This library is designed for the board Arduino Nano 33 BLE Sense. The framework code for running machine learning models should be compatible with most Arm Cortex M-based boards, such as the Raspberry Pi Pico, but the code to access peripherals like microphones, cameras, and accelerometers is specific to the Nano 33 BLE Sense.

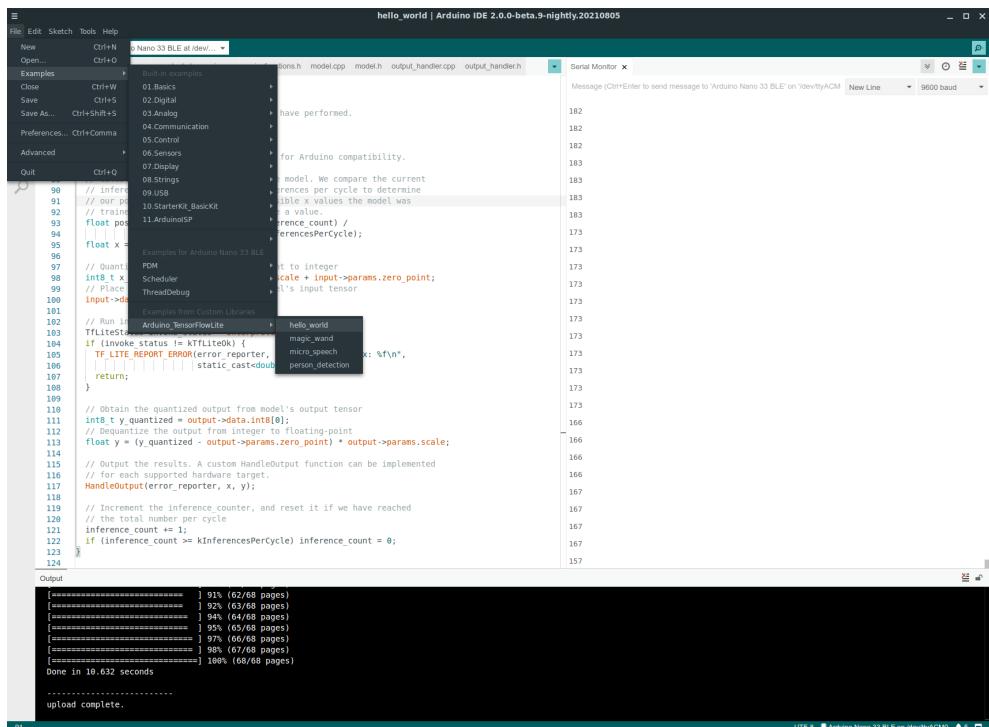


Figure 50.1.: Hello-World-Programm für TensorFlow Lite

50.4. License

This code is made available under the Apache 2 license.

50.5. Contributing

Forks of this library are welcome and encouraged. If you have bug reports or fixes to contribute, the source of this code is at [tflite-micro](#) and all issues and pull requests should be directed there.

The code here is created through an automatic project generation process and may differ from that source of truth, since it's cross-platform and needs to be modified to work within the Arduino IDE.

50.6. Gesture Recognition Model

Next, we'll introduce a more in-depth tutorial you can use to train your own custom gesture recognition model for Arduino using TensorFlow in Colab. This material is based on a practical workshop held by Sandeep Mistry and Don Coleman, an updated version of which is now online.

If you have previous experience with Arduino, you may be able to get these tutorials working within a couple of hours. If you're entirely new to microcontrollers, it may take a bit longer.

Note: The following projects are based on TensorFlow Lite for Microcontrollers which is currently experimental within the [TensorFlow repo](#). This is still a new and emerging field!

50.7. Goals

- Learn the fundamentals of TinyML implementation and training.
- Use the libraries BMI270_BMM150 and Arduino_TensorFlowLite

50.8. Hardware & Software Needed

- A board [Arduino Nano 33 BLE Sense Rev2](#)
- A Micro USB cable to connect the Arduino board to your desktop machine
- To program your board, you can use the [Arduino Web Editor](#) or install the [Arduino IDE](#). We'll give you more details on how to set these up in the following sections

The Arduino Nano 33 BLE Sense Rev2 has a variety of onboard sensors meaning potential for some cool TinyML applications:

- Voice - digital microphone
- Motion - 9-axis IMU (accelerometer, gyroscope, magnetometer)
- Environmental - temperature, humidity and pressure
- Light - brightness, color and object proximity

Unlike classic Arduino Uno, the board combines a microcontroller with onboard sensors which means you can address many use cases without additional hardware or wiring. The board is also small enough to be used in end applications like wearables. As the name suggests it has Bluetooth® Low Energy connectivity so you can send data (or inference results) to a laptop, mobile app or other Bluetooth® Low Energy boards and peripherals.

Tip: Sensors on a USB stick - Connecting the BLE Sense board over USB is an easy way to capture data and add multiple sensors to single board computers without the need for additional wiring or hardware - a nice addition to a Raspberry Pi, for example.

50.9. Microcontrollers and TinyML

Microcontrollers, such as those used on Arduino boards, are low-cost, single chip, self-contained computer systems. They're the invisible computers embedded inside billions of everyday gadgets like wearables, drones, 3D printers, toys, rice cookers, smart plugs, e-scooters, washing machines. The trend to connect these devices is part of what is referred to as the Internet of Things.

Arduino is an open-source platform and community focused on making microcontroller application development accessible to everyone. The board we're using here has an Arm Cortex-M4 microcontroller running at 64 MHz with 1 MB Flash memory and 256 KB of RAM. This is tiny in comparison to cloud, PC, or mobile but reasonable by microcontroller standards.

There are practical reasons you might want to squeeze ML on microcontrollers, including:

- Function - wanting a smart device to act quickly and locally (independent of the Internet).

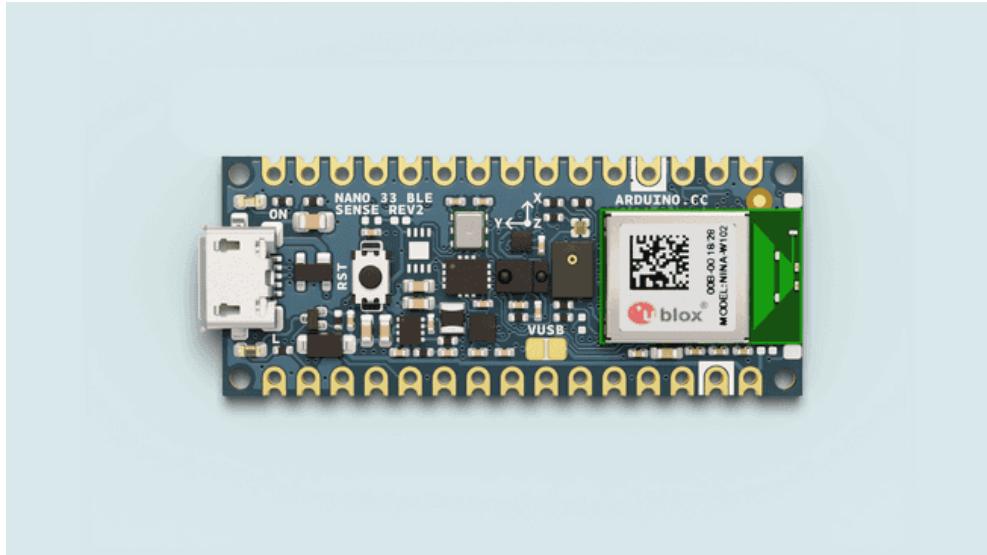


Figure 50.2.: Arduino Nano 33 BLE Sense Rev2 board is smaller than a stick of gum.

- Cost - accomplishing this with simple, lower cost hardware.
- Privacy - not wanting to share all sensor data externally.
- Efficiency - smaller device form-factor, energy-harvesting or longer battery life.

There's a final goal which we're building towards that is very important:

- Machine learning can make microcontrollers accessible to developers who don't have a background in embedded development

On the machine learning side, there are techniques you can use to fit neural network models into memory constrained devices like microcontrollers. One of the key steps is the quantization of the weights from floating point to 8-bit integers. This also has the effect of making inference quicker to calculate and more applicable to lower clock-rate devices.

TinyML is an emerging field and there is still work to do - but what's exciting is there's a vast unexplored application space out there. Billions of microcontrollers combined with all sorts of sensors in all sorts of places which can lead to some seriously creative and valuable TinyML applications in the future.

50.10. TensorFlow Lite for Microcontrollers Examples

The TensorFlow Lite examples are currently not compatible with the Arduino Nano BLE Sense Rev2 board.

The inference examples for TensorFlow Lite for Microcontrollers are now packaged and available through the Arduino Library Manager making it possible to include and run them on Arduino in a few clicks. In this section we'll show you how to run them. The examples are:

- micro_speech - speech recognition using the onboard microphone
- magic_wand - gesture recognition using the onboard IMU
- person_detection - person detection using an external ArduCam camera

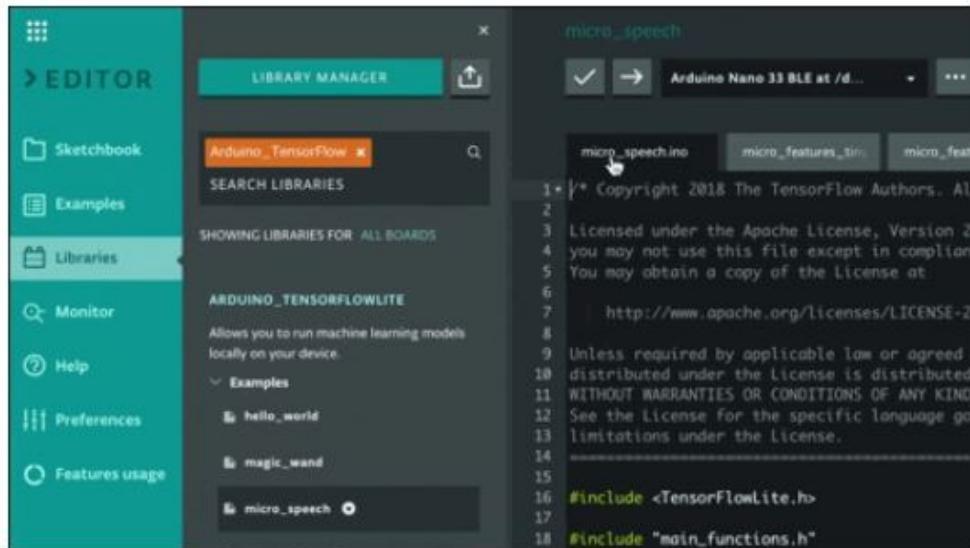


Figure 50.3.: Create lib

For more background on the examples you can take a look at the source in the [TensorFlow repository](#). The models in these examples were previously trained. The tutorials below show you how to deploy and run them on an Arduino. In the next section, we'll discuss training.

50.11. How to Run the Examples Using Arduino Create Web Editor.

Once you connect your Arduino Nano 33 BLE Sense Rev2 to your desktop machine with a USB cable you will be able to compile and run the following TensorFlow examples on the board by using the Arduino Create web editor:

50.12. Focus On The Speech Recognition Example

One of the first steps with an Arduino board is getting the LED to flash. Here, we'll do it with a twist by using TensorFlow Lite Micro to recognise voice keywords. It has a simple vocabulary of “yes” and “no”. Remember this model is running locally on a microcontroller with only 256 KB of RAM, so don't expect commercial ‘voice assistant’ level accuracy - it has no Internet connection and on the order of 2000x less local RAM available.

Note the board can be battery powered as well. As the Arduino can be connected to motors, actuators and more this offers the potential for voice-controlled projects.

50.13. How To Run The Examples Using the Arduino IDE

Alternatively you can use try the same inference examples using Arduino IDE application.

First, follow the instructions in the next section Setting up the Arduino IDE.

In the Arduino IDE, you will see the examples available via the menu [File > Examples > Arduino_TensorFlowLite](#) in the Arduino IDE.

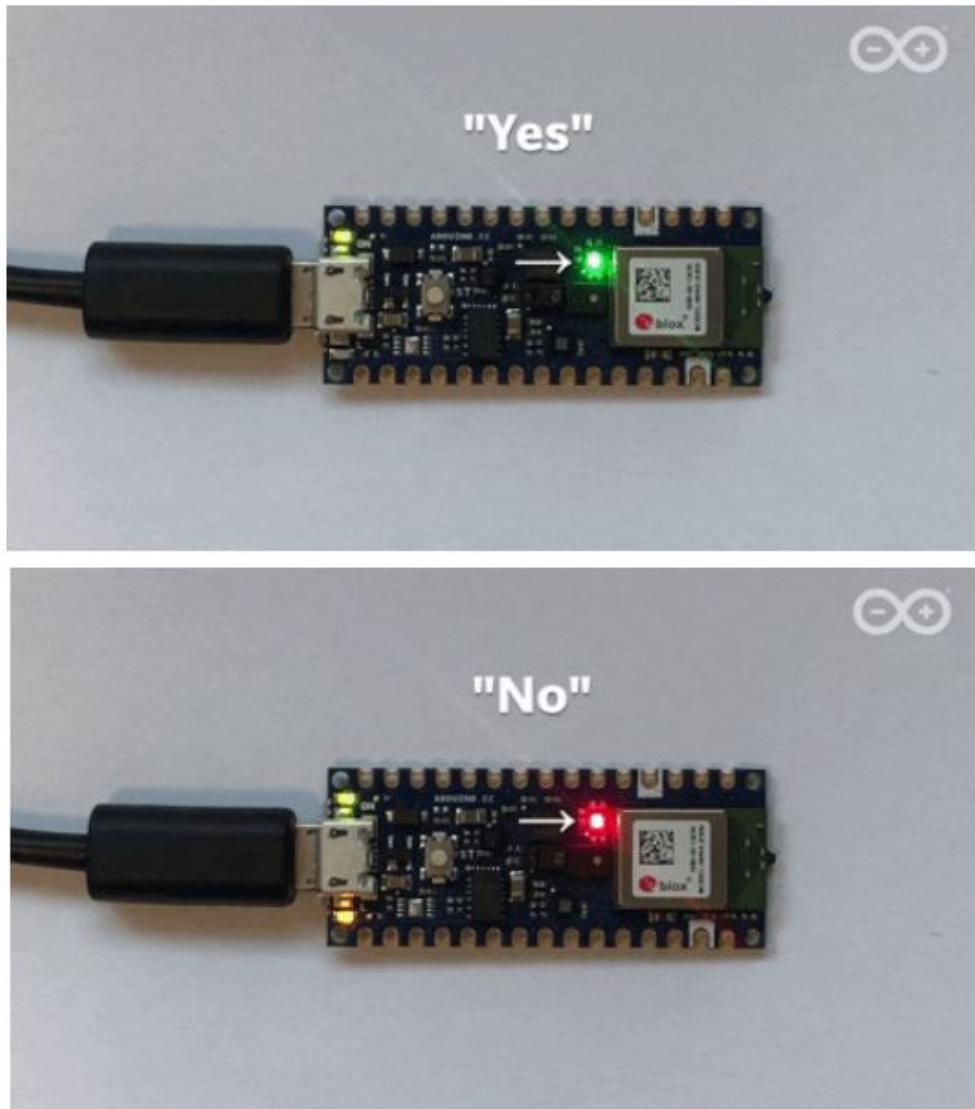


Figure 50.4.: LED red “No” – LED green “yes”



Figure 50.5.: upload

Select an example and the sketch will open. To compile, upload and run the examples on the board, and click the arrow icon:

50.14. Training a TensorFlow Lite Micro Model For Arduino

Next we will use ML to enable the Arduino board to recognise gestures. We'll capture motion data from the Arduino Nano 33 BLE Sense Rev2 board, import it into TensorFlow to train a model, and deploy the resulting classifier onto the board.

The idea for this tutorial was based on Charlie Gerard's awesome Play Street Fighter with [body movements using Arduino and Tensorflow.js](#). In Charlie's example, the board is streaming all sensor data from the Arduino to another machine which performs the gesture classification in Tensorflow.js. We take this further and "TinyML-ify" it by performing gesture classification on the Arduino board itself. This is made easier in our case as the Arduino Nano 33 BLE Sense Rev2 board we're using has a more powerful Arm Cortex-M4 processor, and an on-board IMU.

We've adapted the tutorial below, so no additional hardware is needed – the sampling starts on detecting movement of the board. The original version of the tutorial adds a breadboard and a hardware button to press to trigger sampling. If you want to get into a little hardware, you can follow that version instead.

50.15. IDE Setup

1. First, let's make sure we have the drivers for the Nano 33 BLE boards installed. If we are using the online IDE, there is no need to install anything, if you are using the offline IDE, we need to install it manually. This can be done by navigating to [Tools > Board > Board Manager...](#), search for **Arduino Mbed OS Nano Boards**, and install it.

2. Also, let's make sure we have all the libraries we need installed. If we are using the online IDE, there is no need to install anything. If we are using the offline IDE, this can be done by navigating to [Tools > Manage libraries...](#), search for **Arduino_TensorFlowLite** and **Arduino_BMI270_BMM150**, and install them both.

There are more detailed [Getting Started](#) and [Troubleshooting guides](#) on the Arduino site if you need help.

50.16. Streaming Sensor Data From the Arduino Board

First, we need to capture some training data. You can capture sensor data logs from the Arduino board over the same USB cable you use to program the board with your laptop or PC.

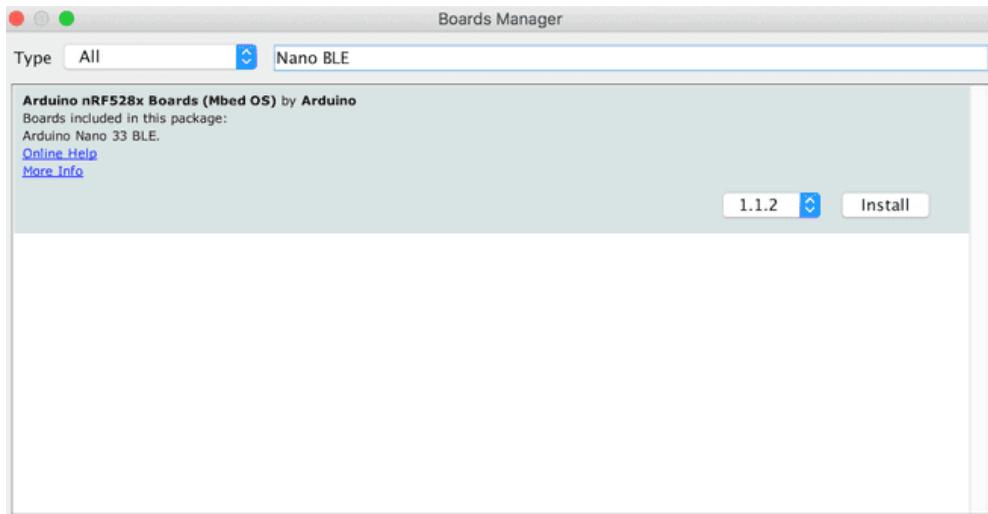


Figure 50.6.: Install Nano BLE board

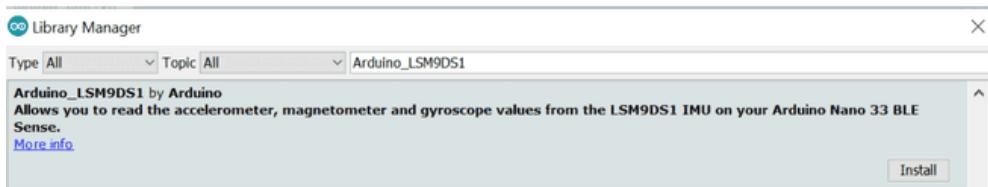


Figure 50.7.: Install the necessary libraries

Arduino boards run small applications (also called sketches) which are compiled from .ino format Arduino source code, and programmed onto the board using the Arduino IDE or Arduino Create.

With the sketch we are creating we will do the following:

- Monitor the board's accelerometer and gyroscope
- Trigger a sample window on detecting significant linear acceleration of the board
- Sample for one second at 119Hz, outputting CSV format data over USB
- Loop back and monitor for the next gesture

The sensors we choose to read from the board, the sample rate, the trigger threshold, and whether we stream data output as CSV, JSON, binary or some other format are all customizable in the sketch running on the Arduino. There is also scope to perform signal preprocessing and filtering on the device before the data is output to the log – this we can cover in another blog. For now, you can just upload the sketch and get sampling.

The complete sketch can be found below:

50.17. Visualizing Live Sensor Data Log From the Arduino Board

With that done we can now visualize the data coming off the board. We're not capturing data yet this is just to give you a feel for how the sensor data capture is

```

/*
  IMU Capture
  This example uses the on-board IMU to start reading acceleration and gyroscope
  data from on-board IMU and prints it to the Serial Monitor for one second
  when the significant motion is detected.
  You can also use the Serial Plotter to graph the data.
  The circuit:
  - Arduino Nano 33 BLE or Arduino Nano 33 BLE Sense Rev2 board.
  Created by Don Coleman, Sandeep Mistry
  Modified by Dominic Pajak, Sandeep Mistry
  This example code is in the public domain.
*/

#include <Arduino_BMI270_BMM150.h>

const float accelerationThreshold = 2.5; // threshold of significant in G's
const int numSamples = 119;

int samplesRead = numSamples;

void setup() {
  Serial.begin(9600);
  while (!Serial);

  if (!IMU.begin()) {
    Serial.println("Failed to initialize IMU!");
    while (1);
  }

  // print the header
  Serial.println("aX,aY,aZ,gX,gY,gZ");
}

void loop() {
  float aX, aY, aZ, gX, gY, gZ;

  // wait for significant motion
  while (samplesRead == numSamples) {
    if (IMU.accelerationAvailable()) {
      // read the acceleration data
      IMU.readAcceleration(aX, aY, aZ);

      // sum up the absolutes
      float aSum = fabs(aX) + fabs(aY) + fabs(aZ);

      // check if it's above the threshold
      if (aSum >= accelerationThreshold) {
        // reset the sample read count
        samplesRead = 0;
        break;
      }
    }
  }

  // check if all the required samples have been read since
  // the last time the significant motion was detected
  while (samplesRead < numSamples) {
    // check if both new acceleration and gyroscope data is
    // available
    if (IMU.accelerationAvailable() && IMU.gyroscopeAvailable()) {
      // read the acceleration and gyroscope data
      IMU.readAcceleration(aX, aY, aZ);
      IMU.readGyroscope(gX, gY, gZ);
    }
  }
}

```

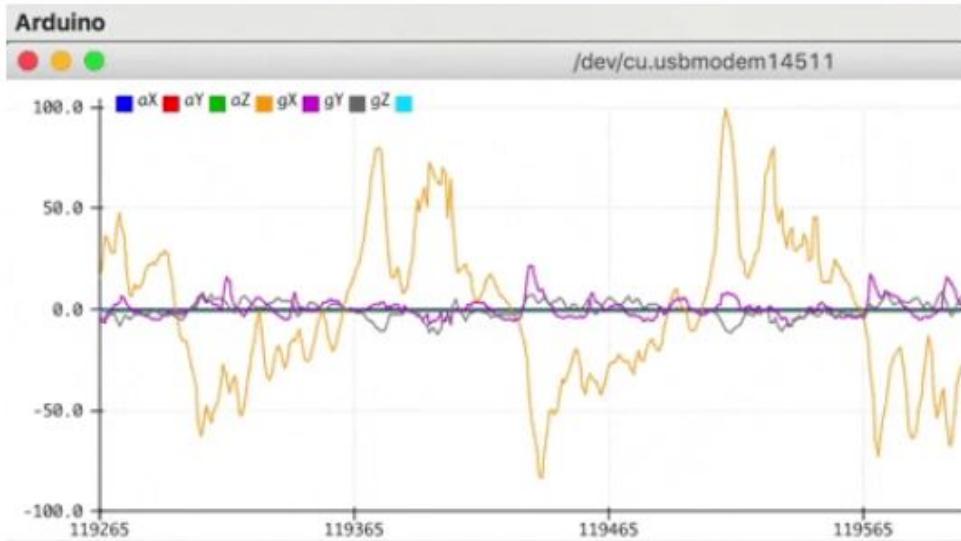


Figure 50.8.: Serial Plotter

triggered and how long a sample window is. This will help when it comes to collecting training samples.

- In the Arduino IDE, open the Serial Plotter [Tools > Serial Plotter](#)
- If you get an error that the board is not available, reselect the port: [Tools > Port > portname \(Arduino Nano 33 BLE\)](#)
- Pick up the board and practice your punch and flex gestures
- You'll see it only sample for a one second window, then wait for the next gesture
- You should see a live graph of the sensor data capture (see GIF below)

When you're done be sure to close the Serial Plotter window – this is important as the next step won't work otherwise.

50.18. Capturing Gesture Training Data

To capture data as a CSV log to upload to TensorFlow, you can use [Arduino IDE > Tools > Serial Monitor](#) to view the data and export it to your desktop machine:

- Reset the board by pressing the small white button on the top
- Pick up the board in one hand (picking it up later will trigger sampling)
- In the Arduino IDE, open the Serial Monitor [Tools > Serial Monitor](#)
- If you get an error that the board is not available, reselect the port:
- [Tools > Port > portname \(Arduino Nano 33 BLE\)](#)
- Make a punch gesture with the board in your hand (Be careful whilst doing this!)
- Make the outward punch quickly enough to trigger the capture
- Return to a neutral position slowly so as not to trigger the capture again

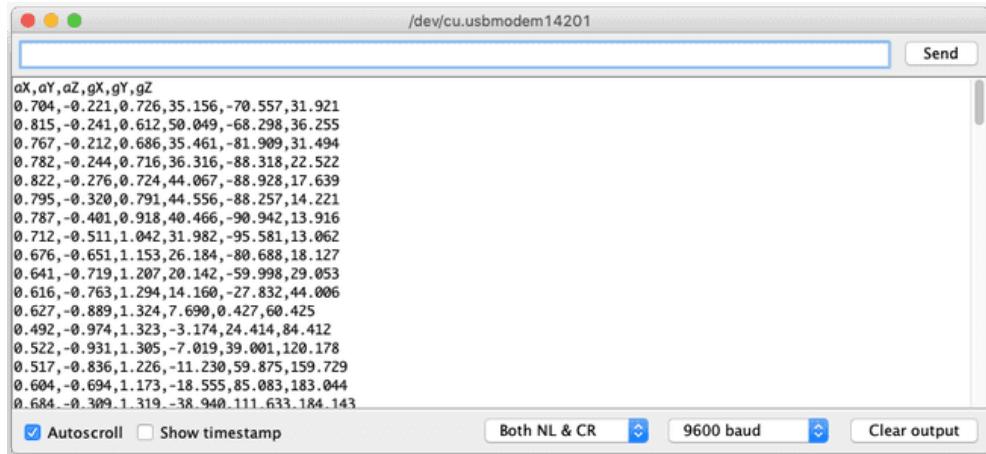


Figure 50.9.: Data recorded by your movements

- Repeat the gesture capture step 10 or more times to gather more data
- Copy and paste the data from the Serial Console to new text file called [punch.csv](#)
- Clear the console window output and repeat all the steps above, this time with a flex gesture in a file called [flex.csv](#)
- Make the inward flex fast enough to trigger capture returning slowly each time

Note: the first line of your two csv files should contain the fields aX,aY,aZ,gX,gY,gZ.
 Linux tip: *If you prefer you can redirect the sensor log output from the Arduino straight to .csv file on the command line. With the Serial Plotter / Serial Monitor windows close use:

```
$ cat /dev/cu.usbmodem[nnnnn] > sensorlog.csv
```

50.19. Training in TensorFlow

We're going to use [Google Colab](#) to train our machine learning model using the data we collected from the Arduino board in the previous section. Colab provides a Jupyter notebook that allows us to run our TensorFlow training in a web browser.

- Set up Python environment
- Upload the punch.csv and flex.csv data
- Parse and prepare the data
- Build and train the model
- Convert the trained model to TensorFlow Lite
- Encode the model in an Arduino header file

The final step of the colab is generates the model.h file to download and include in our Arduino IDE gesture classifier project in the next section:

Let's open the notebook in Colab and run through the steps in the cells - [arduino_tinyml_workshop.ipynb](#)

The screenshot shows a Google Colaboratory (Colab) interface titled "arduino tinyml workshop.ipynb". The left sidebar contains a "Table of contents" with sections like "Tiny ML on Arduino", "Setup Python Environment", "Upload Data", "Graph Data (optional)", "Train Neural Network", "Parse and prepare the data", "Build & Train the Model", "Verify", "Convert the Trained Model to Tensor Flow Lite", "Encode the Model in an Arduino Header File", and "Classifying IMU Data". The main content area features the Arduino logo and a section titled "Tiny ML on Arduino" which includes a "Gesture recognition tutorial" with two bullet points: "Sandeep Mistry - Arduino" and "Don Coleman - Chariot Solutions", and a link to "https://github.com/arduino/ArduinoTensorFlowLiteTutorials/". Below this is a section titled "Setup Python Environment" with a note about setting up dependencies. A code cell shows the command:

```
[ ] # Setup environment
!apt-get -qq install xxd
!pip install pandas numpy matplotlib
!pip install tensorflow==2.0.0-rc1
```

. The right sidebar shows RAM and Disk usage.

Figure 50.10.: Arduino gesture recognition training colab.

The screenshot shows a Google Colaboratory (Colab) interface titled "Encode the Model in an Arduino Header File". The left sidebar shows a file tree with "sample_data" containing "flex.csv", "gesture_model.tflite", and "punch.csv", and "model.h". The main content area displays a code cell with the following content:

```
[8] echo "const unsigned char model[] = {" > /content/model.h
!cat gesture_model.tflite | xxd -i      >> /content/model.h
echo "}";
import os
model_h_size = os.path.getsize("model.h")
print(f"Header file, model.h, is {model_h_size:,} bytes.")
print("\nOpen the side panel (refresh if needed). Double click model.h to download.
```

Below the code cell, a note says "Header file, model.h, is 911,246 bytes. Open the side panel (refresh if needed). Double click model.h to download."

Figure 50.11.: Gesture classifier

```
# -*- coding: utf-8 -*-
"""arduino_tinyml_workshop.ipynb
```

Automatically generated by Colaboratory .

Original file is located at

<https://colab.research.google.com/github/arduino/ArduinoTensorFlowLiteTutorials>

Tiny ML on Arduino

Gesture recognition tutorial

* Sandeep Mistry – Arduino

* Don Coleman – Chariot Solutions

[https://github.com/arduino/ArduinoTensorFlowLiteTutorials/](https://github.com/arduino/ArduinoTensorFlowLiteTutorials)

Setup Python Environment

The next cell sets up the dependencies in required for the notebook , run it .

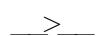
```
"""# Setup environment
```

```
!apt-getqqinstallxxd
```

```
!pip install pandasnumpymatplotlib
```

```
!pip install tensorflow==2.0.0-rc1
```

```
"""# Upload Data
```

1. Open the panel on the left side of Colab by clicking on the 

1. Select the files tab

1. Drag ‘punch.csv’ and ‘flex.csv’ files from your computer to the tab to upload

Graph Data (optional)

We'll graph the input files on two separate graphs , acceleration and gyroscope , a

```
import matplotlib.pyplot as plt
```

```
import numpy as np
```

```
import pandas as pd
```

```
filename = "punch.csv"
```

```
df = pd.read_csv("/content/" + filename)
```

```
index = range(1, len(df['aX']) + 1)
```

```
plt.rcParams["figure.figsize"] = (20, 10)
```

```
plt.plot(index, df['aX'], 'g.', label='x', linestyle='solid', marker=',')
```

```
plt.plot(index, df['aY'], 'b.', label='y', linestyle='solid', marker=',')
```

```
plt.plot(index, df['aZ'], 'r.', label='z', linestyle='solid', marker=',')
```

```
plt.title("Acceleration")
```

```
plt.xlabel("Sample #")
```

```
plt.ylabel("Acceleration (G) ")
```

```
plt.legend()
```

```
plt.show()
```

```
plt.plot(index, df['gX'], 'g.', label='x', linestyle='solid', marker=',')
```

```
plt.plot(index, df['gY'], 'b.', label='y', linestyle='solid', marker=',')
```

```
plt.plot(index, df['gZ'], 'r.', label='z', linestyle='solid', marker=',')
```

```
plt.title("Gyroscope")
```

```
plt.xlabel("Sample #")
```

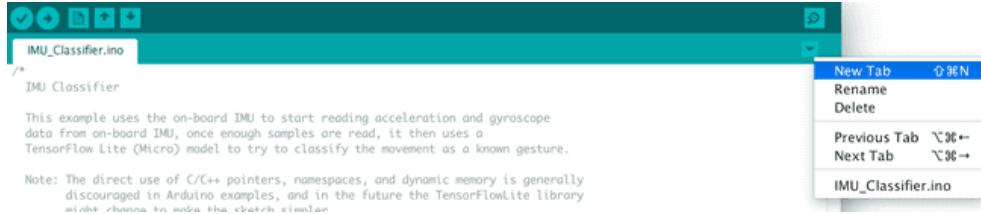


Figure 50.12.: Opening a new tab in your sketch

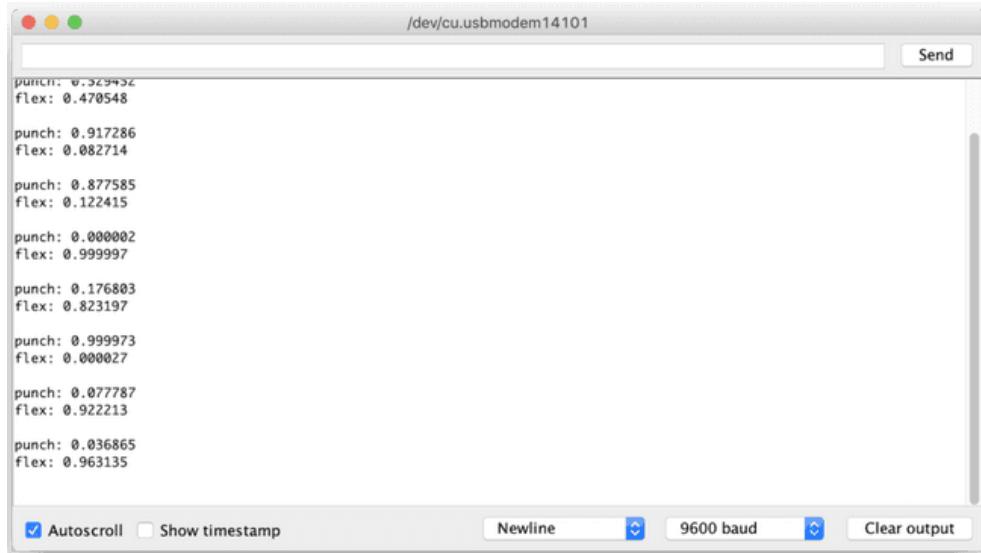


Figure 50.13.: Guessing the gesture with a confidence score

50.20. Classifying IMU Data

Next we will use file `model.h` we just trained and downloaded from Colab in the previous section in our Arduino IDE project:

We will be starting a new sketch, you will find the complete code below:

- Create a new tab in the IDE. When asked name it `model.h`
- Open the tab `model.h` and paste in the version you downloaded from Colab
- Upload the sketch: `Sketch > Upload`
- Open the Serial Monitor: `Tools > Serial Monitor`
- Perform some gestures
- The confidence of each gesture will be printed to the Serial Monitor (0 = low confidence, 1 = high confidence)
- Congratulations you've just trained your first ML application for Arduino!

For added fun the example `EmojiButton.ino` shows how to create a USB keyboard that prints an emoji character in Linux and macOS. Try combining the example `EmojiButton.ino` with the `IMUClassifier.ino` sketch to create a gesture controlled emoji keyboard.

```

/*
  IMU Classifier
  This example uses the on-board IMU to start reading acceleration and gyroscope
  data from on-board IMU, once enough samples are read, it then uses a
  TensorFlow Lite (Micro) model to try to classify the movement as a known gesture.
  Note: The direct use of C/C++ pointers, namespaces, and dynamic memory is gene
        discouraged in Arduino examples, and in the future the TensorFlowLite lib
        might change to make the sketch simpler.
  The circuit:
  - Arduino Nano 33 BLE or Arduino Nano 33 BLE Sense Rev2 board.
  Created by Don Coleman, Sandeep Mistry
  Modified by Dominic Pajak, Sandeep Mistry
  This example code is in the public domain.
*/

#include "Arduino_BMI270_BMM150.h"

#include <TensorFlowLite.h>
#include <tensorflow/lite/micro/all_ops_resolver.h>
#include <tensorflow/lite/micro/micro_error_reporter.h>
#include <tensorflow/lite/micro/micro_interpreter.h>
#include <tensorflow/lite/schema/schema_generated.h>
#include <tensorflow/lite/version.h>

#include "model.h"

const float accelerationThreshold = 2.5; // threshold of significant in G's
const int numSamples = 119;

int samplesRead = numSamples;

// global variables used for TensorFlow Lite (Micro)
tflite::MicroErrorReporter tflErrorReporter;

// pull in all the TFLM ops, you can remove this line and
// only pull in the TFLM ops you need, if would like to reduce
// the compiled size of the sketch.
tflite::AllOpsResolver tflOpsResolver;

const tflite::Model* tflModel = nullptr;
tflite::MicroInterpreter* tflInterpreter = nullptr;
TfLiteTensor* tflInputTensor = nullptr;
TfLiteTensor* tflOutputTensor = nullptr;

// Create a static memory buffer for TFLM, the size may need to
// be adjusted based on the model you are using
constexpr int tensorArenaSize = 8 * 1024;
byte tensorArena[tensorArenaSize] __attribute__((aligned(16)));

// array to map gesture index to a name
const char* GESTURES[] = {
    "punch",
    "flex"
};

#define NUM_GESTURES (sizeof(GESTURES) / sizeof(GESTURES[0]))

void setup() {
    Serial.begin(9600);
    while (!Serial);

    // initialize the IMU
    if (!IMU.begin()) {
        Serial.println("Failed to initialize IMU!");
    }
}

```

50.21. Conclusion

It's an exciting time with a lot to learn and explore in TinyML. We hope this blog has given you some idea of the potential and a starting point to start applying it in your own projects. Be sure to let us know what you build and share it with the Arduino community.

50.22. Nicht zu Rev2 kompatibel

50.22.1. Use the version of the library the tutorials were written for

1. Delete the folder that contains your current installation of the incompatible version of the library at this path:
`/Users/maxwelljohnson/Documents/Arduino/libraries/Arduino_TensorFlowLite`
2. Click the following link to download the version of the library the tutorial's sketches were written for:
[Arduino_TensorFlowLite-2.4.0-ALPHA.zip](#)
3. Wait for the download to finish.
4. Select `Sketch > Include library > Add .ZIP Library` from the Arduino IDE menus.
5. Select the downloaded file.
6. Click the Open button.

You should now be able to compile the tutorial sketches without any errors.

50.22.2. Update the tutorial sketches to work with the current version of the library

Although much more challenging than the alternative, the benefit of this approach is it will allow you to work with the modern maintained version of the Arduino library **TensorFlow Lite for Microcontrollers**.

You can learn how to use the modern version of the library by studying the examples that are included with the library. You'll find those under the menu `File > Examples > Arduino_TensorFlowLite` in Arduino IDE.

50.22.3. Comment

The error message you provided suggests that the file `tensorflow/lite/micro/micro_error_reporter.h` could not be found during the compilation process.

This error typically occurs when the specified file or directory is missing or not accessible to the compiler. The reason can be a missing TensorFlow Lite Micro library, an incorrect include path, or an incorrect file location. Here are some links that may help:

51. Links

- <https://www.hackster.io/search?i=projects&q=Arduino%20Nano%2033%20BLE>
- <https://www.hackster.io/gilbert-tanner/arduino-nano-33-ble-sense-overview-37>
- <https://www.hackster.io/ahmadradhy/nano-33-iot-with-mqtt-edge-impulse-studio>
- <https://www.hackster.io/sridhar-rajagopal/control-arduino-nano-ble-with-bluefruit-le-bridge>
- <https://courses.edx.org>
 - Fundamentals of TinyML
 - Applications of TinyML
 -
-
-
-

51.1. Arduino Projects

<https://itsourcecode.com/free-projects/arduino-projects/ultrasonic-sensor-in-arduino>
<https://itsourcecode.com/free-projects/arduino-projects/controlling-arduino-using-esp8266>
<https://itsourcecode.com/free-projects/arduino-projects/rfid-door-lock-arduino-project>
<https://itsourcecode.com/free-projects/arduino-projects/5v-stepper-motor-arduino-project>
<https://arduino.stackexchange.com/questions/73056/arduino-nano-33-battery-power-supply>

52. Sensor

Introduction

52.1. General

General description

cite books

52.2. Specific Sensor

cite board

52.3. Specification

- cite data sheet

- Circuit Diagram

52.4. Bibliothek

52.4.1. Description

52.4.2. Installation

52.4.3. Functions

52.4.4. Example - Manual

52.4.5. Example

52.4.6. Example - Code

52.4.7. Example - Files

52.5. Calibration

cite method

52.6. Simple Code**52.7. Simple Application****52.8. Tests****52.8.1. Simple Function Test****52.8.2. Test all Functions****52.9. Simple Application****52.10. Further Readings**

53. Package Example

53.1. Introduction

53.2. Description

53.3. Installation

`pip packageExample`

53.4. Example - Manual

53.5. Example

53.6. Example - Code

53.7. Example - Files

`PackageExample.py`

53.8. 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 VII.

Anhang

54. Materialliste

Anzahl	Bezeichnung	Link	Preis
1	ARD KIT TINYML Arduino - Lern-Kit Tiny Machine	 www.reichelt.de - ARD Kit TinyML	52,40 €
1	Arducam B0226 Lens Calibration Tool	 www.welectron.com - Arducam B0226 Lens Calibration Tool	1,90 €

Stand: 22.02.2023

55. Datenblätter

55.1. Datenübersicht

31/5/2019

https://store.arduino.cc/usa/datasheet/index?url_key/nano-33-ble-sense-reseller/



Arduino Nano 33 BLE Sense

Small, powerful, BT connected and with all the sensors you may need to design innovative applications.

SKU: ABX00031

Country of origin: IT

Taric: 85235210

ECCN: 5A992.c

HTS: 8542310001

Overview

This compact and reliable Nano board is built around the NINA B306 module for BLE and Bluetooth 5 communication; the 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 architecture, fully compatible with Arduino IDE Online and Offline, has a 9 axis Inertial Measurement Unit (IMU), temperature, pressure, humidity, light, color and even gestures sensors that are managed through our specialized libraries. Its reduced power consumption, compared to other same size boards, together with the NANO form factor opens up a wide range of applications.

This allows the design of wearable devices and gesture based projects that need to communicate to other devices at a close range. Arduino Nano 33 BLE Sense is ideal for interactive automation projects thanks to the multiprotocol BT 5.0 radio.

Tech Specs

This board is based on the [nRF52480](#) microcontroller.

Clock 64MHz

Flash 1MB

RAM 256KB

Please note: 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. Also, as opposed to Arduino Nano boards that support 5V operation, the 5V pin does NOT supply voltage but is rather connected, through a jumper, to the USB power input.

https://store.arduino.cc/usa/datasheet/index?url_key/nano-33-ble-sense-reseller/

1/3

31/5/2019 https://store.arduino.cc/usa/datasheet/index/index?url_key/nano-33-ble-sense-reseller/
The Bluetooth is managed by a [NINA B306](#) module.
The IMU is a [LSM9DS1](#) and it is managed through I2C.
The [LPS22HB](#) reads barometric pressure and environmental temperature.
The [HTS221](#) senses relative humidity.
The [ADPS-9900](#) is a digital proximity, ambient light, RGB and gesture sensor.
The [MP34DT05](#) is the digital microphone.

Crypto keys are managed by the ATECC608A crypto chip.

The board has a two 15 pins connectors - one on each side -, pin to pin compatible with the original Arduino Nano.

Pin	Funcion	Type	Description
1	D13	Digital	GPIO
2	+3V3	Power Out	Internally generated power output to external devices
3	AREF	Analog	Analog Reference; can be used as GPIO
4	A0/DAC0	Analog	ADC in/DAC out; can be used as GPIO
5	A1	Analog	ADC in; can be used as GPIO
6	A2	Analog	ADC in; can be used as GPIO
7	A3	Analog	ADC in; can be used as GPIO
8	A4/SDA	Analog	ADC in; I2C SDA; Can be used as GPIO (*)
9	A5/SCL	Analog	ADC in; I2C SCL; Can be used as GPIO(*)
10	A6	Analog	ADC in; can be used as GPIO
11	A7	Analog	ADC in; can be used as GPIO
12	V _{USB}	Power	Normally NC; can be connected to V _{USB} pin of the USB connector by shorting a jumper
13	RST	Digital In	Active low reset input (duplicate of pin 18)
14	GND	Power	Power Ground
15	VIN	Power In	Vin Power input
16	TX	Digital	USART TX; can be used as GPIO
17	RX	Digital	USART RX; can be used as GPIO
18	RST	Digital	Active low reset input (duplicate of pin 13)
19	GND	Power	Power Ground
20	D2	Digital	GPIO
21	D3/PWM	Digital	GPIO; can be used as PWM
22	D4	Digital	GPIO
23	D5/PWM	Digital	GPIO; can be used as PWM
24	D6/PWM	Digital	GPIO; can be used as PWM

https://store.arduino.cc/usa/datasheet/index/index?url_key/nano-33-ble-sense-reseller/

31/5/2019

https://store.arduino.cc/usa/datasheet/index/index?url_key/nano-33-ble-sense-reseller/

25	D7	Digital	GPIO
26	D8	Digital	GPIO
27	D9/PWM	Digital	GPIO; can be used as PWM
28	D10/PWM	Digital	GPIO; can be used as PWM
29	D11/MOSI	Digital	SPI MOSI; can be used as GPIO
30	D12/MISO	Digital	SPI MISO; can be used as GPIO

(*) As opposed to other Arduino Nano boards, pins A4 and A5 have an internal pull up and default to be used as an I²C Bus so usage as analog inputs is not recommended. opposed to Arduino Nano boards that support 5V operation, the 5V pin does NOT supply voltage but is rather connected, through a jumper, to the USB power input.

On the bottom side of the board, under the communication module, **debug signals** are arranged as 3x2 test pads with 100 mil pitch. Pin 1 is the bottom left one with the USB connector on the left and the test pads on the right.

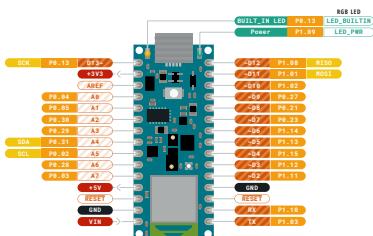
Pin	Function	Type	Description
1	+3V3	Power Out	Internally generated power output to be used as voltage reference
2	SWD	Digital	nRF52480 Single Wire Debug Data
3	SWCLK	Digital In	nRF52480 Single Wire Debug Clock
5	GND	Power	Power Ground
6	RST	Digital In	Active low reset input

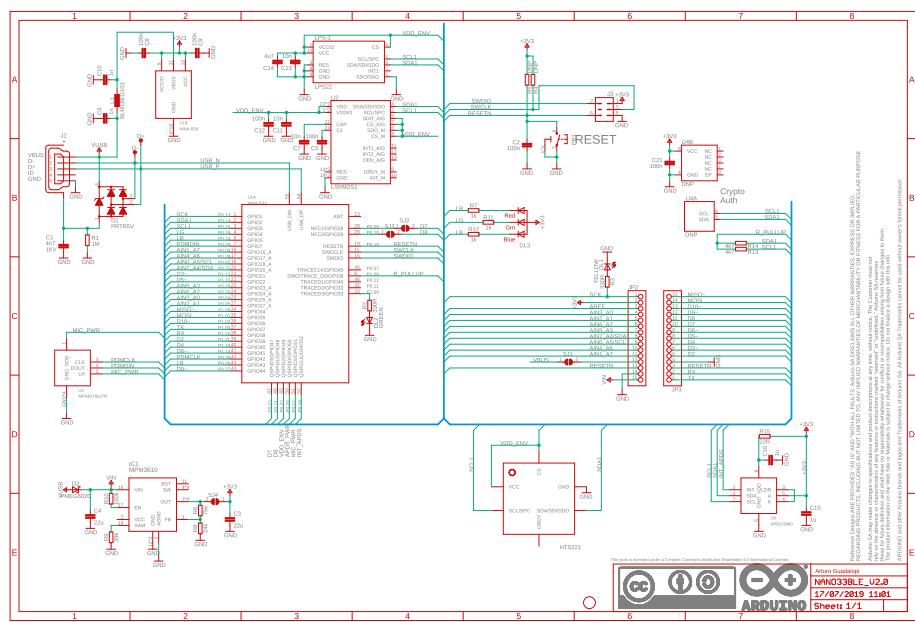
https://store.arduino.cc/usa/datasheet/index/index?url_key/nano-33-ble-sense-reseller/

3/3



ARDUINO
NANO 33 BLE SENSE
[STORE.ARDUINO.CC/NANO-33-BLUETOOTH-SENSE](https://store.arduino.cc/nano-33-bluetooth-sense)





56. Datenblätter Arduino Vision Shield

ArduCam
ArduCAM-M-2MP Camera Shield

2MP SPI Camera User Guide

Rev 1.0, Feb 2015



Table of Contents

1	Introduction	2
2	Application	2
3	Features	3
4	Key Specifications	3
5	Pin Definition	3
6	Block Diagram	4
7	Functions	4
7.1	Single Capture Mode	4
7.2	Multiple Capture Mode	4
7.3	JPEG Compression	4
7.4	Normal Read and Burst Read Operation	4
7.5	Rewind Read Operation	5
7.6	Low Power Mode	5
7.7	Image Sensor Control	5
8	Lens Options	6
9	Mechanical Dimension	7
10	Order Information	7

ArduCam**ArduCAM-M-2MP Camera User Guide**

1 Introduction

ArduCAM-M-2MP is optimized version of ArduCAM shield Rev.C, and is a high definition 2MP SPI camera, which reduce the complexity of the camera control interface. It integrates 2MP CMOS image sensor OV2640, and provides miniature size, as well as the easy to use hardware interface and open source code library. The ArduCAM mini can be used in any platforms like Arduino, Raspberry Pi, Maple, Chipkit, Beaglebone black, as long as they have SPI and I2C interface and can be well mated with standard Arduino boards. ArduCAM mini not only offers the capability to add a camera interface which doesn't have in some low cost microcontrollers, but also provides the capability to add multiple cameras to a single microcontroller.

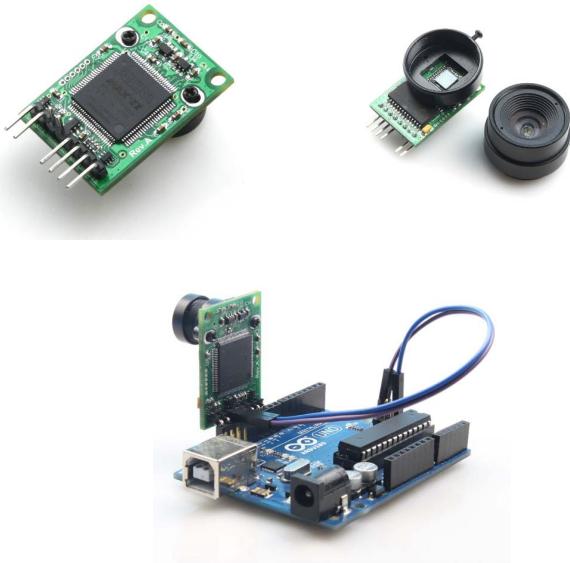


Figure 1 ArduCAM Mini Shield

2 Application

- IoT cameras
- Robot cameras
- Wildlife cameras
- Other battery-powered products
- Can be used in MCU, Raspberry Pi, ARM, DSP, FPGA platforms

3 Features

- 2MP image sensor OV2640
- M12 mount or CS mount lens holder with changeable lens options
- IR sensitive with proper lens combination
- I2C interface for the sensor configuration
- SPI interface for camera commands and data stream
- All IO ports are 5V/3.3V tolerant
- Support JPEG compression mode, single and multiple shoot mode, one time capture multiple read operation, burst read operation, low power mode and etc.
- Well mated with standard Arduino boards
- Provide open source code library for Arduino, STM32, Chipkit, Raspberry Pi, BeagleBone Black
- Small form of factor

4 Key Specifications

- | | |
|--|---|
| ■ Power supply
Normal :5V/70mA
Low power mode: 5V/20mA | ■ Active array size: 1600x1200 |
| ■ SPI speed: 8MHz | ■ Shutter: rolling shutter |
| ■ Frame buffer: 384KB | ■ Lens: 1/4 inch |
| ■ Size: 34 x 24 mm | ■ Resolution support:
UXGA, SVGA, VGA, QVGA, CIF, QCIF |
| ■ Weight: 20g | ■ Format support: RAW, YUV, RGB, JPEG |
| ■ Temperature: -10°C ~ +55°C | ■ Pixel Size: 2.2μm x 2.2μm |

5 Pin Definition

Table 1 ArduCAM-M-2MP Pin Definition

Pin No.	Pin Name	Type	Description
1	CS	Input	SPI slave chip select input
2	MOSI	Input	SPI master output slave input
3	MISO	Output	SPI master input slave output
4	SCLK	Input	SPI serial clock
5	GND	Ground	Power ground
6	+5V	POWER	5V Power supply
7	SDA	Bi-directional	Two-Wire Serial Interface Data I/O
8	SCL	Input	Two-Wire Serial Interface Clock

ArduCam

ArduCAM-M-2MP Camera User Guide

6 Block Diagram

Figure 2 shows the block diagram of ArduCAM mini shield which is composed by lens, image sensor and an ArduChip. The lens is changeable and can be mounted by S-mount (M12x0.5) or CS-mount lens holder. The image sensor is 2MP CMOS OV2640 from Omnipixel. The ArduChip uses ArduCAM proprietary third generation camera controller technology which handles the complex camera, memory and user interface hardware timing and provides a user friendly SPI interface.

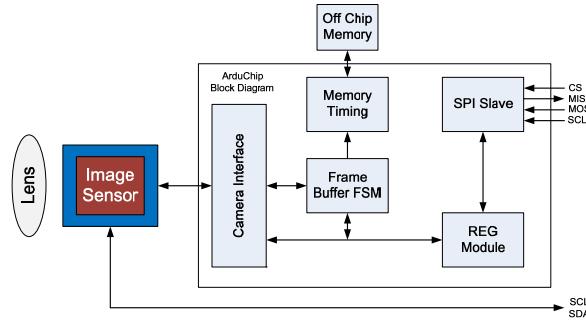


Figure 2 ArduCAM Mini Shield Block Diagram

7 Functions

7.1 Single Capture Mode

Single capture mode is the default capture mode of the camera. After issuing a capture command via SPI port, the ArduCAM will wait for a new frame and buffer the one entire image data to the frame buffer, and then assert the completion flag bit in the register. User only needs to poll the flag bit from the register to check out if the capture is done.

7.2 Multiple Capture Mode

Multiple capture mode is advanced capture mode. By setting the number of frames in the capture register, the ArduCAM will capture consequent frames after issuing capture command. Note that number of frames should be set properly and make sure do not exceed the maximum memory space.

7.3 JPEG Compression

The JPEG compression function is implemented in the image sensor. With proper register settings to the sensor, user can get different resolution with JPEG image stream output. It is recommended to use JPEG output to get higher resolution than RGB mode, due to the limitation of frame buffer.

7.4 Normal Read and Burst Read Operation

Normal read operation reads each image data by sending a read command in one SPI read operation cycle. While burst read operation only need to send a read command then read multiple image data in one SPI read operation cycle. It is recommended to use burst read operation to get better throughput performance.

7.5 Rewind Read Operation

Sometimes user wants to read the same frame of image data multiple times for processing, the rewind read operation is designed for this purpose. By resetting the read pointer to the beginning of the image data, user can read the same image data from the start point again.

7.6 Low Power Mode

Some battery power device need save power when in the idle status, the ArduCAM offers the low power mode to reduce power consumption, by shutdown the sensor and memory circuits.

7.7 Image Sensor Control

Image sensor control function is implemented in the image sensor. By setting proper set of register settings, user can control the exposure, white balance, brightness, contrast, color saturation and etc.

More technical information about ArduCAM mini shield, please read ArduCAM-M-2MP Hardware Application Note.pdf and ArduCAM-M-2MP Software Application Note.pdf for detail.

ArduCam**ArduCAM-M-2MP Camera User Guide****8 Lens Options**

The ArduCAM-M-2MP camera shield is shipped with default LS-4011 (S mount) or LS-6018 (CS mount), lenses specification list as follows. S mount lenses normally have build IR cut filter, while the CS mount lenses doesn't have build in IR cut filter.

Please contact us admin@arducam.com for more lens options.

LS-4011 Lens Specification**SPECIFICATION:**

1.Sensor Format:	max Ø5.3mm
2.EFL:	3.96 mm
3.F-number:	2.6
4.Construction:	4P+1IR
5.TV Distortion:	<1.2%
6.FOV:	56.8°
7.FBL:	1.29mm
8.IR:	645nm

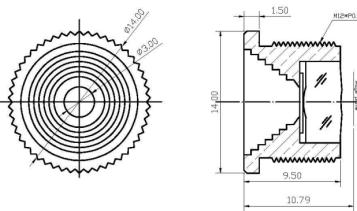


Figure 3 S Mount Lens Specification

LS-6018 Lens Specification**技术参数**

Technical parameters

型号 Model No.	LS-6018CS	视场角 Field of View	68°
焦距 Focal Length	6.0MM	外型尺寸 Dimensions	Φ28*24.2mm
通光口径 Aperture(F)	1.4	近摄距离 M.O.D(m)	0.1
接口 Mount	CS	净重 Weight(g)	29.0
靶面尺寸 Format	1/2.7"	备注 Remarks	Metal

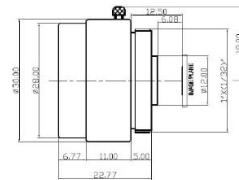
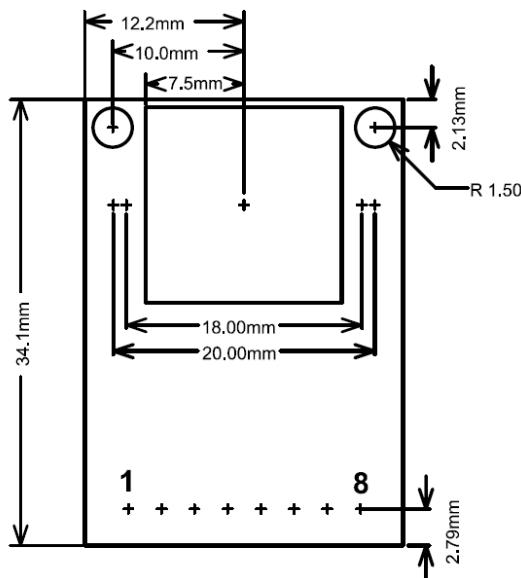


Figure 4 CS Mount Lens Specification

9 Mechanical Dimension



10 Order Information

Part Number	Description
ArduCAM-M-2MP-SM01	S Mount Preinstalled Pin Header
ArduCAM-M-2MP-SM02	S Mount Without Preinstalled Pin Header
ArduCAM-M-2MP-CSM01	CS Mount Preinstalled Pin Header
ArduCAM-M-2MP-CSM02	CS Mount Without Preinstalled Pin Header

ArduCam

ArduCAM-M-2MP Camera Shield

2MP SPI Camera Hardware Application Note

Rev 1.0, Mar 2015



Table of Contents

1	Introduction.....	2
2	Typical Wiring.....	2
2.1	Single Camera Wiring.....	2
2.2	Multi Cameras Wiring.....	2
3	I2C Interface.....	3
4	SPI Slave Interface.....	4
5	ArduChip Timing Diagram.....	4
5.1	SPI Bus Write Timing.....	4
5.2	SPI Bus Single Read Timing	4
5.3	SPI Bus Burst Read Timing	5
6	Registers Table.....	5

1 Introduction

This application note describes the detail hardware operation of ArduCAM-M-2MP camera shield.

2 Typical Wiring

2.1 Single Camera Wiring

The typical connection between ArduCAM shield and Arduino or etc platform is shown in the Figure 1. More typically the Figure 2 shows the wiring for Arduino UNO R3 board.

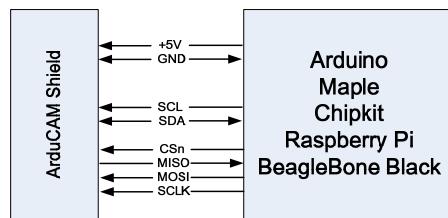


Figure 1 Typical Wiring

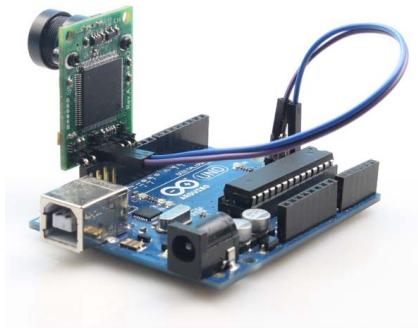


Figure 2 Wiring for Arduino UNO R3

2.2 Multi Cameras Wiring

The multi-cameras connection between ArduCAM shield and Arduino or etc platform is shown in the Figure 3. More typically the Figure 4 shows the multi-cameras wiring for Arduino UNO R3 board.

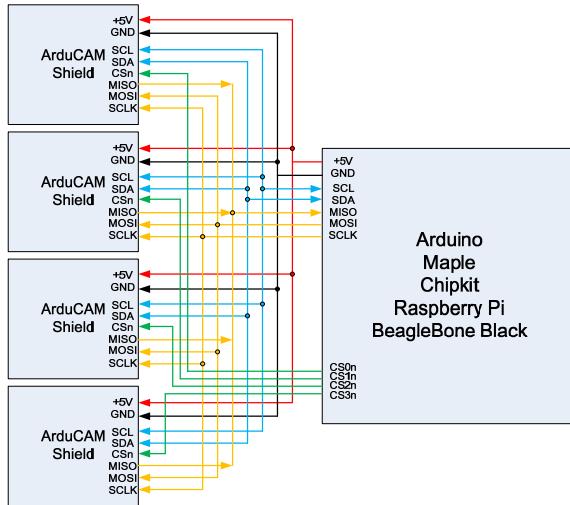


Figure 3 Multi-Cameras Wiring

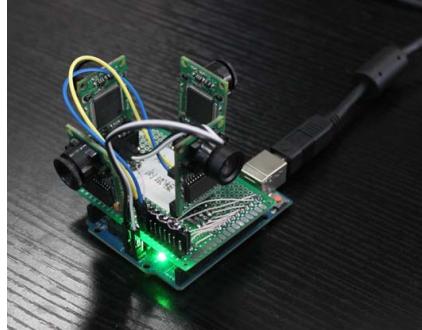


Figure 4 Mult-Cameras Wiring on Arduino UNO

3 I2C Interface

The I2C interface is directly connected to the image sensor OV2640. The OV2640 I2C slave address is 0x60 for write and 0x61 for read. User can use I2C master to read and write all the registers in the OV2640 sensor. For more information about the OV2640 register, please refer the OV2640 datasheet. The Figure 5 shows writing value 0x01 to the OV2640 register 0xFF. The Figure 6 shows reading value 0x26 from the OV2640 register 0x0A.

ArduCam

ArduCAM-M-2MP Hardware Application Note

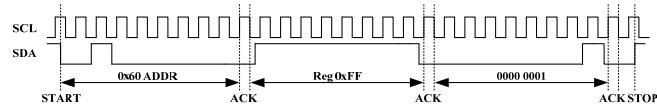


Figure 5 I2C Write Bus Timing

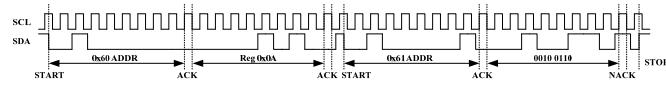


Figure 6 I2C Read Bus Timing

4 SPI Slave Interface

The ArduCAM SPI slave interface is fixed SPI mode 0 with $\text{POL} = 0$ and $\text{PHA} = 1$. The maximum speed of SCLK is designed for 8MHz, care should be taken not to over clock the maximum 8MHz. The SPI protocol is designed with a command phase with variable data phase. The chip select signal should always be asserted during the SPI read or write bus cycle.

The first bit[7] of the command phase is read/write byte, '0' is for read and '1' is for write, and the bit[6:0] is the address to be read or write in the data phase. ArduChip register table see Table 1.

5 ArduChip Timing Diagram

5.1 SPI Bus Write Timing

The SPI bus write timing composed of a command phase and a data phase during the assertion of the chip select signal CSn. The first 8 bits is command byte which is decoded as a register address, and the second 8 bits is data byte to be written to the ArduChip internal registers.

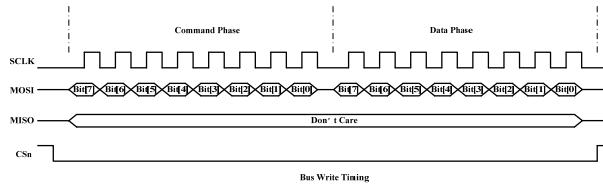


Figure 7 SPI Bus Write Timing

5.2 SPI Bus Single Read Timing

The SPI bus single read timing is for read operation of ArduChip internal registers and single FIFO read function. It is composed of a command phase and a data phase during the assertion of chip select signal CSn. The first 8 bits is command byte which is decoded as a register address, the second 8 bits is dummy byte written to the SPI bus MOSI signal, and the content read back from register is appeared on the SPI bus MISO signal.

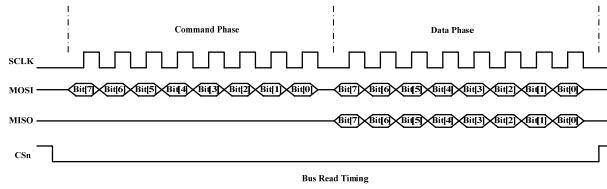


Figure 8 SPI Bus Single Read Timing

5.3 SPI Bus Burst Read Timing

The SPI bus burst read timing is only for burst FIFO read operation. It is composed of a burst read command phase and multiple data phases in order to get double throughput compared to the single FIFO read operation. The first byte read from the FIFO is a dummy byte, and the following bytes are valid bytes.

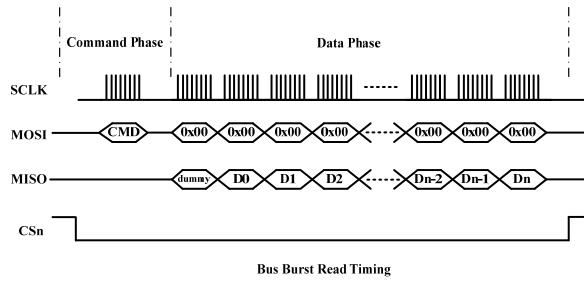


Figure 9 SPI Bus Burst Read Timing

6 Registers Table

Sensor and FIFO timing is controlled with a set of registers which is implemented in the ArduChip. User can send capture commands and read image data with a simple SPI slave interface. The detail description of registers' bits can be found in the software section in this document.

As mentioned earlier the first bit[7] of the command phase is read/write byte, '0' is for read and '1' is for write, and the bit[6:0] is the address to be read or write in the data phase. So user has to combine the 8 bits address according to the read or write commands they want to issue.

Table 1 ArduChip Register Table

Register Address bit[6:0]	Register Type	Description
0x00	RW	Test Register
0x01	RW	Capture Control Register Bit[2:0]: Number of frames to be captured
0x02	RW	Reserved
0x03	RW	Sensor Interface Timing Register Bit[0]: Sensor Hsync Polarity,

ArduCam**ArduCAM-M-2MP Hardware Application Note**

		0 = active high, 1 = active low Bit[1]: Sensor Vsync Polarity 0 = active high, 1 = active low Bit[3]: Sensor data delay 0 = no delay, 1= delay 1 PCLK Bit[4]: FIFO mode control 0 = FIFO mode disable, 1 = enable FIFO mode Bit[6]: low power mode control 0 = normal mode, 1 = low power mode
0x04	RW	FIFO control Register Bit[0]: write '1' to clear FIFO write done flag Bit[1]: write '1' to start capture Bit[4]: write '1' to reset FIFO write pointer Bit[5]: write '1' to reset FIFO read pointer
0x05	RW	GPIO Direction Register Bit[0]: Sensor reset IO direction Bit[1]: Sensor power down IO direction Bit[2]: Sensor power enable IO direction 0 = input, 1 = output
0x06	RW	GPIO Write Register Bit[0]: Sensor reset IO value Bit[1]: Sensor power down IO value Bit[1]: Sensor power enable IO value
0x3B	RO	Reserved
0x3C	RO	Burst FIFO read operation
0x3D	RO	Single FIFO read operation
0x3E	RO	Reserved
0x3F	RO	Reserved
0x40	RO	ArduChip version, constant value 0x40 for 2MP model Bit[7:4]: integer part of the revision number Bit[3:0]: decimal part of the revision number
0x41	RO	Bit[0]: camera vsync pin status Bit[3]: camera write FIFO done flag
0x42	RO	Camera write FIFO size[7:0]
0x43	RO	Camera write FIFO size[15:8]
0x44	RO	Camera write FIFO size[18:16]
0x45	RO	GPIO Read Register Bit[0]: Sensor reset IO value Bit[1]: Sensor power down IO value Bit[1]: Sensor power enable IO value

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 CLI 0.33*. 2018. URL: <https://arduino.github.io/arduino-cli/0.33/commands/arduino-cli/> (visited on 07/24/2023).
- [Ardc] *Arduino Libraries – LCD_I2C*. 2023. URL: https://www.arduino.cc/reference/en/libraries/lcd_i2c (visited on 07/09/2023).
- [Ardd] *Arduino Libraries – SSD1306Ascii*. 2023. URL: <https://reference.arduino.cc/reference/en/libraries/ssd1306ascii/> (visited on 06/26/2023).
- [Arde] *Arduino Libraries - PDM*. 2023. URL: <https://docs.arduino.cc/learn/built-in-libraries/pdm> (visited on 06/14/2023).
- [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).
- [Ard18] Arduino. *What is Arduino?* 2018. URL: <https://www.arduino.cc/en/Guide/Introduction>.
- [Ard19] Arduino, ed. *Arduino Library – Kalman*. 2019. URL: <https://www.arduino.cc/reference/en/libraries/kalman/> (visited on 07/09/2023).

-
- [Ard20] Arduino, ed. *Arduino CLI: An Introduction*. 2020. URL: <https://blog.arduino.cc/2020/03/13/arduino-cli-an-introduction>.
- [Ard21] Arducam, ed. *Featured Camera Modules Supported*. 2021. URL: <https://www.arducam.com/docs/usb-cameras/featured-camera-modules-supported/> (visited on 06/16/2021).
- [Ard21] Arduino, ed. *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>.
- [Ard22a] Arduino, ed. *Arduino CLI (Command Line Interface) Application*. 2022. URL: <https://github.com/arduino/arduino-cli>.
- [Ard22b] Arduino, ed. *Arduino CLI*. 2022. URL: <https://arduino.github.io/arduino-cli/0.22/>.
- [Ard22c] Arduino, ed. *arduino-cli*. 2022. URL: <https://www.arduino.cc/pro/cli>.
- [Ard24a] Arduino, ed. *Getting started with the Arduino Nano 33 BLE Sense*. 2024. URL: <https://wiki-content.arduino.cc/en/Guide/NANO33BLESense>.
- [Ard24b] Arduino, ed. *Software – Download*. 2024. URL: <https://www.arduino.cc/en/software>.
- [Ari21] D. Aristi. *LCD_I2C library on GitHub*. 2021. URL: https://github.com/blackhack/LCD_I2C (visited on 07/09/2023).
- [Ava15] Avago Technologies, ed. *APDS-9960 - Digital Proximity, Ambient Light, RGB and Gesture Sensor*. [pdf]. 2015. URL: <https://docs.broadcom.com/doc/AV02-4191EN>.
- [Az] 0,96 Zoll OLED Display - Datenblatt. [pdf]. AZ-Delivery, 2023. URL: <https://www.az-delivery.de/products/0-96zolldisplay>.
- [Bai18] J. Baichtal. *10 LED Projects fpr Geeks*. William Pollock, 2018. ISBN: 978-59327-825-0.
- [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.
- [Big21] Bigelow, Stephen. *What is edge computing? Everything you need to know*. accessed date 16.04.2022. 2021. URL: <https://www.techtarget.com/searchdatacenter/definition/edge-computing>.
- [Bosa] *BME280 – Combined humidity and pressure sensor*. [pdf]. Bosch, 2015. URL: https://cdn-reichelt.de/documents/datenblatt/B400/BST-BME280_DS001-10.pdf.
- [Bosb] *BME280 – Integrated Environmental Unit*. [pdf]. Bosch, 2021. URL: https://cdn-reichelt.de/documents/datenblatt/B400/BOSCH_SENSORTEC_FLYER_BME280.pdf.
- [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>.

- [Dej18] Dejan. *How to Control Servo Motors with Arduino – Complete Guide*. 2018.
- [Ene] *Energizer CR2032 – Product Datasheet*. [pdf]. 2018.
- [FAD18] M. Fezari and A. Al Dahoud. “Integrated Development Environment “IDE” For Arduino”. In: (Oct. 2018).
- [FD22] P. Filipi and Dozie. *A Difference between A N 33 BLE Sense vs. 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.
- [FPSS96] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. “From Data Mining to Knowledge Discovery in Databases”. In: 17.3 (1996). [pdf], p. 37. DOI: 10.1609/aimag.v17i3.1230. URL: <https://www.aaai.org/ojs/index.php/aimagazine/article/view/1230>.
- [Fet21] R. J. Fetick. *Kalman*. 2021.
- [Fou24] P. S. Foundation, ed. *Download Python*. 2024. URL: <https://www.python.org/downloads/>.
- [Fun4] *BME280 Luftdruck-, Luftfeuchtigkeits- und Temperatursensor*. 2023. URL: <http://funduino.de/nr-23-bme280-luftdruck-luftfeuchtigkeits-und-temperatursensor>.
- [Funb] *OLED Display SSD1306 128 × 64 / 128 × 32*. 2023. URL: <https://funduino.de/nr-42-oled-display-ssd1306-128x64-128x32>.
- [GW22] W. Gehrke and M. Winzker. *Digitaltechnik – Grundlagen, VHDL, FPGAs, Mikrocontroller*. Wiesbaden: Springer Vieweg, 2022. ISBN: 978-3-662-63953-5.
- [Goo11] Google, ed. *Coral: Build beneficial and privacy preserving AI*. 14/11/2019. URL: <https://coral.withgoogle.com/>.
- [Goo19a] Google, ed. *TensorFlow models on the Edge TPU*. 2019. URL: <https://coral.withgoogle.com/docs/edgetpu/models-intro/>.
- [Goo19b] Google, ed. *TensorFlow*. 2019. URL: <https://www.tensorflow.org/lite>.
- [Goo20] Google, ed. *Beginnen Sie mit TensorFlow Lite*. 2020. URL: https://www.tensorflow.org/lite/guide/get_started#1_choose_a_model.
- [Gup20] Gupta, Sakshi. *What is Tiny Machine Learning*. 2020. URL: <https://www.springboard.com/blog/data-science/tiny-machine-learning/>.
- [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.
- [HS23] E. Hering and G. Schönfelder. *Sensoren in Wissenschaft und Technik*. 3. [pdf]. Springer Vieweg, 2023. ISBN: 978-3-658-39490-5. DOI: 10.1007/978-3-658-39491-2978-3-662-62697-9.
- [HSH17] E. Hering and G. Schönfelder Hrsg. *Sensoren in Wissenschaft und Technik*. Wiesbaden: Springer Vieweg, 2017. ISBN: 978-3-658-12561-5.

- [Har23] S. Harris. *Inertial guidance: a brief history and overview*. Advanced Navigation. [\[pdf\]](#). Jan. 3, 2023. URL: <https://www.advancednavigation.com/tech-articles/inertial-guidance-a-brief-history-and-overview/> (visited on 12/09/2023).
- [Haz] *Event Driven Architecture*. accessed date 19.04.2022. 2022. URL: <https://hazelcast.com/glossary/event-driven-architecture/>.
- [Hui] *Technical Data Sheet & Specifications - 10003R1D-EHC-B*. [\[pdf\]](#). 2017.
- [Ibr18] D. Ibrahim. *Motorsteuerung mit Arduino & Raspberry Pi*. Aachen: Elektor, 2018. ISBN: 978-3-895-76336-6.
- [Iod23] G. M. Iodice. *TinyML Cookbook*. 2nd. Packt Publishing Ltd., 2023. ISBN: 978-1-83763-736-2.
- [Iot] *BME280 Temperatursensor – Arduino Sketch*. 2020. URL: <https://iot-space.dev/bme280-temperatursensor-arduino-sketch/> (visited on 06/23/0023).
- [Jam] *High End Micro*. 033212. Jamara e.K., 2018.
- [Kai09] B. Kainka. *Schnellstart LEDs*. 2. Franzis Verlag GmbH, 2009.
- [Kha20] Khandelwal, Renu. *A Basic Introduction to TensorFlow Lite*. accessed date 20.04.2022. 2020. URL: <https://towardsdatascience.com/a-basic-introduction-to-tensorflow-lite-59e480c57292>.
- [Kin] *T-1 3/4 (5mm) – Full Color LED Lamp*.
- [Kur21a] A. Kurniawan. *Beginning Arduino Nano 33 IoT: Step-By-Step Internet of Things Projects*. [\[pdf\]](#). Berkeley, CA: Apress, 2021. ISBN: 978-1-4842-6445-4. DOI: 10.1007/978-1-4842-6446-1. URL: <https://doi.org/10.1007/978-1-4842-6446-1>.
- [Kur21b] 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>.
- [Kur21c] A. Kurniawan. *IoT Projects with NVIDIA Jetson Nano: AI-Enabled Internet of Things Projects for Beginners*. Berkeley, CA: Apress, 2021. ISBN: 978-1-4842-6452-2. DOI: 10.1007/978-1-4842-6452-2_1. URL: https://doi.org/10.1007/978-1-4842-6452-2_1.
- [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.
- [Lai22] R. Laine. “Bluetooth Low Energy data transfer in energy harvester system: Comparison of platforms and boards”. MA thesis. 2022.
- [Las] *Interface BME280 Temperature, Humidity and Pressure Sensor with Arduino*. 2023. URL: <https://lastminuteengineers.com/bme280-arduino-tutorial/> (visited on 06/23/0023).
- [Lib21] A. Libraries, ed. *Arduino_LSM6DSOX Library for Arduino*. 2021. URL: https://github.com/arduino-libraries/Arduino_LSM6DSOX (visited on 07/09/2023).

- [MAB22] J. Martínez, D. Asiaín, 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>.
- [MO19] M. Munoz-Organero. “Outlier detection in wearable sensor data for human activity recognition (HAR) based on DRNNs”. In: *IEEE Access* 7 (2019), pp. 74422–74436.
- [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.
- [Qua] *Technical Data Sheet – Round LED 3mm*. [\[pdf\]](#). 2020.
- [RA20] c't Redaktion (Autor). “c't Python-Projekte”. In: (2020).
- [RPW21] V. J. Reddi, B. Plancher, and P. Warden. “Widening Access to Applied Machine Learning with TinyML”. In: *Journal of Optimization Theory and Applications* (2021). DOI: <https://doi.org/10.48550/arXiv.2106.04008>. URL: <https://docslib.org/doc/4278844/widening-access-to-applied-machine-learning-with-tinyml>.
- [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>.
- [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.
- [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.
- [Sen19] P. Seneviratne. *Beginning LoRa Radio Networks with Arduino - Build Long Range, Low Power Wireless IoT Networks*. [\[pdf\]](#). Berkeley, CA: Apress, 2019. ISBN: 978-1-4842-4356-5. DOI: 10.1007/978-1-4842-4357-2. URL: <https://doi.org/10.1007/978-1-4842-4357-2>.
- [Sima] *KY-016 – RGB LED Modul*. [\[pdf\]](#). 2020. URL: www.joy-it.net.
- [Simb] *SBC-OLED01 - Datenblatt*. [\[pdf\]](#). SIMAC Electronics GmbH, 2019. URL: https://cdn-reichelt.de/documents/datenblatt/A300/DEBO_OLED_0-96_DB-DE.pdf.

-
- [Smy21] 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>.
- [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).
- [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.
- [W3c] *Wire.h (I^2C)*. 2023. URL: <https://html.szaktilla.de/arduino/6.html>.
- [WS20] P. Warden and D. Situnayake. *TinyML – Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers*. [pdf]. O'Reilly Media, Incorporated, 2020, p. 504. ISBN: 978-1-492-05204-3.
- [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://doi.org/10.21227/z8bc-pt92>.
- [Xu+22] L. Xu et al. “Gesture recognition using dual-stream CNN based on fusion of sEMG energy kernel phase portrait and IMU amplitude image”. In: *Biomedical Signal Processing and Control* 73 (2022), p. 103364. ISSN: 1746-8094. DOI: <https://doi.org/10.1016/j.bspc.2021.103364>. URL: <https://www.sciencedirect.com/science/article/pii/S1746809421009617>.

Index

- File
 - .cli-config.yml, 54
 - .json, 313
 - .txt, 247
 - Adafruit-GFX.h, 200
 - Adafruit-SSD1306.h, 200
 - ArduCAM, 215
 - Arduino IDE > Tools > Serial Monitor, 372
 - arduino-cli-0.33.1.windows.exe, 53
 - arduino-cli.exe, 53
 - arduino-cli_0.33.0_Windows_64bit.zip, 53
 - Arduino_APDS9960, 107, 108
 - arduino_data, 54
 - ArduinoBLE, 228
 - ArduinoBLE.h, 226
 - cactus_io_BME280, 190
 - cactus_io_BME280_I2C, 190
 - cli_test, 57
 - Datei -> SSD1206Ascii -> HelloWorld-Wire, 191
 - EmojiButton.ino, 376
 - ESP8266, 216
 - Fade, 135, 329
 - File → Examples, 363
 - File > Examples > Arduino_Tensor-FlowLite, 367, 378
 - File/Examples/01.Basics, 135, 329
 - flex.csv, 373
 - Hello World, 206
 - HelloWorldWire, 199
 - host_app, 215
 - IMUClassifier.ino, 376
 - LSM6DSOXSensor.h, 169
 - LSM9DS1, 228
 - Mag_raw.txt, 74
 - memorysaver.h, 215
 - Mini_5MP_Plus, 215
 - model.h, 376
 - monitor_log.bat, 63
 - Nano33BLE_Led_A0.ino, 222
 - output.txt, 247, 316
 - PackageExample.py, 383
 - PDM.h, 121
 - pdm.h, 119, 120
 - punch.csv, 373
 - PySerial, 139
 - RaspberryPi, 215
 - README.md, 54
 - RevC, 215
 - Shield_V2, 215
 - SimpleAccelerometer, 152
 - Sketch -> Include Library -> Manage Library, 224
 - Sketch > Include library > Add .ZIP Library, 378
 - Sketch > Upload, 376
 - SSD1306Ascii.h, 153
 - tensorflow/lite/micro/micro_error_reporter.h, 378
 - TestAPDS9960Color.ino, 112
 - Tools > Board > Board Manager..., 369
 - Tools > Manage libraries..., 369
 - Tools > Port > portname (Arduino Nano 33 BLE), 372
 - Tools > Serial Monitor, 372, 376
 - Tools > Serial Plotter, 372
 - UTFT4ArduCAM_SPI, 215
 - Wire.h, 153, 200
 - Inertial Measurement Unit
 - see IMU, 17, 19, 23, 72
 - Acoustic Overload Point
 - see AOP, 23, 75
 - AOP, 23, 75
 - Arduino Nano 33 BLE sense, 53
 - Central Processing Unit
 - see CPU, 23, 241
 - CLI, 15, 23, 53, 54, 57, 58
 - CNN, 17–19, 23, 244, 248, 250–252, 317, 320, 321
 - Command Line Interface
 - see CCI, 15, 23, 53
 - Continuous Development, 53
 - Continuous Integration, 53
 - Convolutional Neural Network
 - see CNN, 17–19, 23, 244
 - CPU, 23, 241
 - Edge Impulse, 53
 - Example, 383

- Description, 383
- Installation, 383
- Introduction, 383
- General Purpose Input Output
 - see* GPIO, 23, 209
- GPIO, 23, 209
- GPU, 23, 241
- Graphics Processing Unit
 - see* GPU, 23, 241
- I²C, 23
- IDE, 23, 135, 243, 329
- IMU, 17, 19, 23, 72–74, 249, 250, 276, 317, 318
- Integrated Development Environment
 - see* IDE, 23, 135
- Inter-Integrated Circuit
 - see* I²C, 23
- KDD, 23, 244
- Knowledge Discovery in Databases
 - see* KDD, 23, 244
- LED, 23, 59, 85, 86, 88, 95, 97–101, 177–180, 183–186, 209, 243
 - Brightness, 100, 185
 - Built-in LED, 91
 - Built-in RGB-LED, 95
 - Colors, 98, 186
 - Power LED, 85
 - Standard LED, 179
 - Standard RGB LED, 183
- Light Emitting Diode
 - see* LED, 23
- mcd, 23, 96
- Millicandela
 - see* mcd, 23, 96
- Nicla Vision, 53
- PDM, 23, 75, 121
- Pin
 - Pin 11, 103
 - Pin 13, 91, 92
 - Pin 14, 105, 181
 - Pin 22, 23, 24, 97
 - Pin 22,23,24, 96
 - Pin 25, 86
- Portenta H7, 53
- Pulse Density Modulation
 - see* PDM, 23, 75
- Pulse with Modulation
 - see* PWM Signal, 23, 86
- Push Button
 - Built-in Push Button, 103
 - PWM Signal, 23, 86, 95, 98, 179, 183, 186
 - SCL, 23, 143
 - SDA, 23, 143
 - Serial Clock
 - see* SCL, 23, 143
 - Serial Data
 - see* SDA, 23, 143
- YAML, 54

Temporary page!

`LATEX` was unable to guess the total number of pages correctly. As there was some unprocessed data that should have been added to the final page this extra page has been added to receive it.

If you rerun the document (without altering it) this surplus page will go away, because `LATEX` now knows how many pages to expect for this document.