

Elmar Wings

Artificial Intelligence with an Arduino Nano 33 BLE Sense

Realtime Object Detection with the ArduCAM

Version: 1765
December 13, 2023

Contents

Contents	3
List of Figures	11
I. Machine Learning	17
1. Arduino IDE	19
1.1. Introduction	19
1.2. Arduino IDE Description	19
1.3. Installation	19
1.4. Arduino IDE on PC	22
1.4.1. Installation	22
1.4.2. Configuration	22
1.4.3. Test Example	23
1.5. Pre-requisite settings for Uploading the program	24
1.5.1. Select the Appropriate Port	24
1.5.2. Upload Code in Arduino Board	25
1.6. Output Window (Serial Monitor)	26
II. TinyML	29
2. Arduino Nano 33 BLE Sense	31
2.1. Arduino Nano 33 BLE Sense	31
2.2. On-Board Sensor Description	33
2.2.1. Gesture, Proximity, and Color Detection Sensor ADPS-9960 . .	33
2.2.2. Accelerometer, Gyroscope, and Magnetometre Sensor LSM9DS1 .	34
2.2.3. Pressure Sensor LPS22HB	36
2.2.4. Relative Humidity and Temperature Sensor HTS221	36
2.2.5. Digital Microphone MP34DT05-A	37
2.2.6. Bluetooth Module nRF52840	37
2.3. Arduino Nano 33 BLE Pin Configuration	38
3. Test of the Hardware	41
3.1. General Tests	41
3.2. Testing the On-Board Sensor	41
3.2.1. LSM9DS1 (Accelerometer, Gyroscope, and Magnetometre Sensor)	41
3.2.2. APDS-9960 (Gesture, Proximity, and Color Detection Sensor) .	43
3.2.3. LPS22HB (Pressure Sensor)	43
3.2.4. HTS221 (Humidity and Temperature Sensor)	44
3.2.5. MP34DT05 (Digital Microphone)	45
3.3. Test with Bluetooth Module Connection	46
3.4. Reset of the Arduino	47
4. Testspezifikation	49
4.1. Sketch “Hello World”	49

4.2. Test der Sensoren eines Arduino Nano 33 BLE Sense	50
4.3. APDS-9960 Modul	52
4.4. OLED-Display	53
4.5. RGB-LED	53
4.6. Mikrofon MP34DT05-A	54
4.6.1. Verwendete Bibliotheken	54
4.6.2. Pin-Konfiguration	54
4.6.3. Initialisierung und Setup	54
4.6.4. Messungssteuerung	54
4.6.5. Mikrofondatenverarbeitung	56
4.6.6. Anzeige der Ergebnisse	56
4.6.7. Umrechnung auf den Wert dB SPL	56
4.6.8. Decibels	57
4.6.9. Benutzeroberfläche	57
5. Protokoll I²C	59
5.1. Pin-Belegung für I ² C beim Arduino Nano 33 BLE Sense	60
6. Serial Peripheral Interface	61
7. Inertial Measurement Unit	63
7.1. Introduction	63
7.2. 6-Axis IMU LSM6DSOX	63
7.3. IMU LSM6DSOX Features	63
7.4. IMU LSM6DSOX Data	65
7.4.1. Accelerometer:	66
7.4.2. Gyroscope	66
7.5. Library setup in Arduino IDE	66
7.6. Applications	67
8. Software für eine IMU	69
8.1. Erste Schritte mit der Entwicklungsumgebung	69
8.2. Bibliotheken	69
8.2.1. Sensor LSM9DS1	69
8.3. Beispiele auf dem Mikrocontroller	70
8.3.1. Testen des Sensors LSM9DS1	70
8.4. Programmierung	71
8.4.1. Programmablaufplan	71
8.4.2. Programmcode und Dokumentation	71
8.4.3. Definition Echtzeit	76
8.5. Kalibrierung	77
8.6. Probleme	77
9. Sensormodul APDS9960	79
9.1. Der APDS-9960	79
9.2. OLED-Display DEBO OLED2 0,96	79
9.3. RGB-LED	79
9.4. Schaltplan	80
10. Softwarebeschreibung	81
10.1. Softwarestruktur	81
10.2. Einrichtung der Arduino IDE	82
10.2.1. Bibliotheken einfügen	82
10.2.2. Serielle Kommunikation	82
10.3. Header	83

Contents	5
10.4. <i>void setup{}</i>	83
10.5. <i>void loop{}</i>	84
11. Sensor BME280	89
11.1. Beschreibung der Hardware	89
11.2. Schaltplan des Aufbaus	90
11.2.1. Bibliothek <code>cactus_io_BME280</code> für den Sensor BME280	90
11.2.2. Testen des OLED-Displays	90
12. Servomotor JAMRA 033212	93
12.1. Datenblatt JAMRA 033212	94
12.2. Schaltplan	94
13. OLED-Display	97
13.1. OLED	97
13.2. Anschluss	97
13.3. Programmierung	97
13.3.1. <code>Wire.h</code>	97
13.3.2. OLED-Display	99
13.3.3. Testen des OLED-Displays	99
13.4. Software	99
13.4.1. Verwendete Bibliotheken	99
13.4.2. OLED-Display	100
13.4.3. Initialisierung und Setup	100
13.4.4. Messungssteuerung	101
13.4.5. Mikrofondatenverarbeitung	102
13.4.6. Anzeige der Ergebnisse	102
13.4.7. Umrechnung auf den Wert dB SPL	102
13.4.8. Benutzeroberfläche	102
13.5. Das OLED-Display SSD1306	102
13.6. Schaltplan des Aufbaus	103
13.7. Genutzte Bibliotheken	103
13.7.1. Bibliothek <code>Wire.h</code>	103
13.7.2. Bibliothek <code>SSD1306Ascii.h</code> für das Testprogramm	103
13.7.3. Testen des OLED-Displays	103
14. BlueTooth	111
14.1. Quick Start	111
14.2. Supplies	111
14.3. Step 1: Introduction	111
14.4. How pfodApp is optimised for short BLE style messages	113
14.5. Step 2: Creating the Custom Android Menus and Generating the Code	113
14.6. Arduino BLE Example – Explained Step by Step	114
14.6.1. Arduino BLE Example Code Explained	114
14.6.2. Arduino BLE – Bluetooth Low Energy Introduction	114
14.6.3. Arduino BLE Example 1 – Battery Level Indicator	115
14.6.4. Arduino BLE Tutorial Battery Level Indicator Code	115
14.6.5. Arduino Bluetooth Battery Level Indicator Code Explained	115
14.6.6. Installing the App for Android	118
14.6.7. Example 2 – Arduino BLE Accelerometer Tutorial	118
15. Power Source	125
15.1. Port	125
15.2. Power sources Available	126
15.2.1. Nickel-metal hydride (NiMH)	126

15.2.2. Lithium batteries	126
15.2.3. Powerbanks	128
15.2.4. How To Calculate Battery Run Time	129
16. Battery	131
16.1. Checking the Battery Voltage	131
17. Tiny Machine Learning Kit	133
17.1. Unterschied zwischen dem Arduino Nano 33 BLE Sense Rev1, dem Arduino Nano 33 BLE Sense Rev2 und dem Arduino Nano 33 BLE Sense Lite	133
17.2. Das Arduino Tiny Machine Learning Shield	134
17.3. Die OV7675 Kamera	135
17.4. USB-Micro-B- auf USB-A-Kabel	135
17.5. USB-Kabel mit USB-A- und USB-Micro-B-Stecker mit 90° Winkel	136
17.6. OLED-Display	137
17.7. Powerbank	138
17.8. Schaltplan	139
18. Tiny-ML on Arduino Nano 33 BLE Sense	141
18.1. Usefull Tiny-ML Model for Arduino Nano 33 BLE Sense	141
18.1.1. Voice Recognition	141
18.1.2. Custom Gesture Recognition	141
18.1.3. Color Detection	141
18.1.4. Person Detection	142
III. Arduino Nano 33 BLE Sense	143
19. Arduino	145
19.1. Handhabung der Ausgabe am Arduino	145
19.2. Ausführen des Beispiels	148
19.3. Differences in the Arduino Example Code	148
19.4. Eigene Änderungen vornehmen	151
20. Image Classification with the Arduino Nano 33 BLE Sense	155
20.1. Was wir bauen	155
20.2. Bemerkung	155
20.3. Kamera-Module	155
20.4. Anwendungsarchitektur	155
20.5. Einführung in unser Modell	156
20.6. Alle beweglichen Teile	157
20.7. Extra Dimensions	161
20.8. NOTE	161
20.9. The Image Provider	162
20.10. The Detection Responder	163
20.11. Detecting People	163
20.12. Einsatz auf Mikrocontrollern	167
20.13. Arduino	167
20.14. Welches Kameramodul Sie kaufen sollten	167
20.15. Aufnehmen von Bildern mit dem Arduino	168
20.16. Reagieren auf Erkennungen am Arduino	169
20.17. Ausführen des Beispiels	170
20.18. Tip	171
20.19. Bemerkung	171

20.20Bemerkung	172
20.21Tip	173
20.22Ausführung von Änderungen	176
20.23Laufen auf Arduino	176
20.24Installation anderer Bibliotheken	176
21.Sensor ov2640	179
21.1. Produktbeschreibung	179
21.2. ArduCAM Library Introduction	180
21.3. Libraries Structure	181
21.4. How to use	181
21.4.1. 1. Edit <code>memorysaver.h</code> file	182
21.4.2. 2. Choose correct CS pin for your camera	182
21.4.3. 3. Upload the examples	182
21.4.4. 4. How To Connect Bluetooth Module	182
22.ArduCAM ov2640	183
22.1. Indroduction	183
22.1.1. Pin Configuration of Arducam 0V2640 2MP Mini	183
22.2. ArduCAM Interface with Arduino	183
23.Lens Calibration Tool	185
23.1. Übersicht	185
23.2. Applications	185
23.3. Package Contents	186
IV. Project	187
24.TinyML and Edge Computing	189
24.1. TinyML	189
24.2. Real Time	189
24.3. Edge computing	189
25.TensorFlow Lite	191
25.1. TensorFlow Lite	191
25.2. What is TensorFlow Lite?	191
25.3. Procedure	191
25.4. TensorFlow Lite Converter	192
25.5. TensorFlow Lite Interpreter	193
26.Introduction	195
27.Project Magic Wand	197
27.1. Development	197
27.1.1. Database	197
27.1.2. Data Preparation	197
27.1.3. Data Transformation & Mining	199
27.1.4. Model	200
27.1.5. Evaluation and Verification	204
27.2. Deployment	205
27.2.1. Software	205
27.2.2. Program Flow Code	212

28. Magic Wand Program Workflow	215
28.1. Development	217
28.1.1. Introduction	217
28.1.2. System Requirements	217
28.1.3. Overview of the Program	217
28.2. Code Structure	217
28.2.1. Cases - ePendingMovement, eRecordingGesture	221
28.2.2. arduino_acceleometer_handler	222
28.2.3. Arduino_Output_Handler	223
28.2.4. main_functions	223
28.2.5. constants.h	224
28.2.6. Magic Wand Program Workflow	226
28.2.7. Frequently Asked Questions	227
28.3. Getting to Know Arduino Nano Magic Wand	227
28.3.1. Key Features	227
28.3.2. Functional Parts	228
28.4. Board Operation	228
28.5. LSM9DS1(IMU)	230
28.6. Features	230
28.7. Pin description LSM9DS1	232
28.8. Pin connections LSM9DS1	232
28.8.1. Module specifications	232
28.8.2. Block Diagram	232
28.8.3. Using Magic Wand	234
28.9. Connect with Computer	234
28.10Open Arduino IDE	234
28.11Select Port	235
28.12Run the application	235
28.13Open Serial Monitor	236
28.14Start waving magic wand	237
28.14.1.System Requirements	237
28.14.2.Precautions to be taken	237
28.14.3.FAQs	237
29. Project Implementation Steps	239
29.1. Edge Impulse	239
29.2. Gesture Detection	240
29.2.1. First Approach for Gesture Detection	240
29.3. Gesture Detection Test Using MediaPipe	240
29.3.1. Successful Approach of Gesture Detection	241
29.3.2. Gesture Detection on Arduino Nano 33 BLE Sense	242
29.3.3. Gesture Detection Results On Arduino Nano 33 BLE Sense	242
29.4. Gesture Detection Using LSTM Layer and Tensor Flow	243
29.4.1. Gesture Detection Results Using LSTM and TensorFlow using MediaPipe	243
29.5. Color Detection	244
29.6. Object Detection	244
29.6.1. Yolov3 and Yolov4 Object Detection Model	244
29.6.2. OpenCV Gpu support using Darknet YOLOV4	244
29.6.3. Deep Sort Object Tracking Model	245
30. Open Question and Possible Solution	247
30.1. Checkpoint 1 (Arduino IDE and Python supportive Packages)	247
30.1.1. Possible Solution	247

Contents	9
30.2. Checkpoint 2 (Arducam Mini 2MP OV2640 Replacement)	248
30.2.1. Possible solution	248
30.3. Checkpoint 3 (TensorFlow lite not supported complex object)	248
30.3.1. Possible solution	249
30.4. Object Detection Part Solution	249
30.5. Possible operating system options for Machine Learning Model	249
30.5.1. Nvidia GPU Installation requirement	250
31. Source Codes	251
31.0.1. Hand Landmark Tracking	251
31.0.2. Test Example for Counting figure Using Landmark detection	251
31.0.3. HandTracking Module	253
31.0.4. Detecting 6 Different Gesture Code Using Hand Landmarks	254
31.0.5. Module for Running code on Arduino	256
32. Datenblätter	259
32.1. Datenübersicht	259
33. Datenblätter Arduino Vision Shield	265
Literaturverzeichnis	281
Stichwortverzeichnis	286
Index	287

List of Figures

1.2. Menu butons	19
1.1. Menu bar options	20
1.3. Arduino Setup Installation options	20
1.4. Arduino Setup: Installation Folder	21
1.5. Arduino Sketch	21
1.6. Arduino Creat Agent Installation	22
1.7. Arduino Mbed OS Nano Boards Installation	23
1.8. LED-Example Test	23
1.9. Select the Connected board -here: Arduino Nano 33 BLE Sense	24
1.10. Arduino Nano 33 BLE Sense: Reset Button	24
1.11. Arduino Nano 33 BLE Sense: Orange LED Glow	25
1.12. Select Available Port for Uploading Arduino Sketch	25
1.13. Upload the Program in Arduino board	26
1.14. Setting the Port	26
1.15. Serial Monitor Icon	27
1.16. Output Window	27
2.1. Arduino Nano 33 BLE Sense, see Arduino Store	32
2.2. Components in Arduino Nano 33 BLE Sense [Raj19]	34
2.3. Circuit diagram microphone	37
2.4. Arduino Nano 33 BLE Pin Configuration	39
3.1. Power On Arduino Nano 33 BLE Sense	41
3.2. 9-Axis IMU Sensor	42
3.3. Results of IMU-Tests	42
3.4. APDS-9960 Gesture, Proximity, Color Sensor	43
3.5. Gesture, Proximity, Color Sensor Output Window	43
3.6. LPS22HB, Pressure and Temperature sensor	44
3.7. LPS22HB, Output	44
3.8. HTS221, Humidity and Temperature Sensor	44
3.9. HTS221, Output Window	45
3.10. MP34DT05, Digital Microphone	45
3.11. MP34DT05, Serial Plotter	46
3.12. Bluetooth Connection	46
3.13. The Reset Button of the Arduino	47
4.1. Code Einlesen/Zählen	54
4.2. Code Messungssteuerung	54
4.3. Code Überwachung	55
4.4. Code Initialisierung der Datenverarbeitung	56
4.5. Code Umrechnung Mikrofonsignal	56
4.6. Code OLED-Aktualisierung	57
5.1. Aufbau des Protokolls I ² C	59
5.2. I2C-Protokoll Quelle: [GW22]	60
7.1. <i>Axis Acceleration of Accelerometer Sensor</i>	66

7.2. <i>Angular Speed of Gyroscope Sensor</i>	66
7.3. Library setup in Arduino IDE	67
8.1. Beispiele in der LSM9DS1 Bibliothek	70
8.2. Output des Testprogramms	70
8.3. Der Programmablaufplan	72
8.4. Anzeige des Arduino OLED Displays sobald der Arduino eingeschaltet ist	77
8.5. Ausgabe von Messdaten	77
8.6. Der Reset Button des Arduino	78
9.1. Schaltplan Farberkennungsautomat	80
10.1. Software-Strukturplan	81
11.1. Sensor BME280	89
11.2. Stationärer Aufbau der Wetterstation	90
12.1. Aufbau eines Servomotors Quelle	94
12.2. PWM am Servomotor Quelle	95
12.3. Auszug aus Gebrauchsanleitung JAMRA 033212, Seite 1 [Jam]	95
12.4. Schaltplan zum Anschluss eines Servomotors JAMRA 033212	96
13.1. Gesamter Schaltplan	98
13.2. Beispiele in der OLED Bibliothek	99
13.3. Testausgabe des OLED Display	100
13.4. Code Einlesen/Zählen	101
13.5. Code Messungssteuerung	101
13.6. Code Überwachung	106
13.7. Code Initialisierung der Datenverarbeitung	106
13.8. Code Umrechnung Mikrofonsignal	107
13.9. Code OLED-Aktualisierung	107
13.10 Pins des OLED-Displays.	107
13.11 Stationärer Aufbau der Wetterstation	107
13.12 Pfad des Testprogramms.	108
13.13 Erste Ausgabe Display	109
14.1. BLE Devices – Peripheral and Central Devices	112
14.2. Step 2: Creating the Custom Android Menus and Generating the Code	113
14.3. BLE Devices – Peripheral and Central Devices	115
14.4. Battery app “nRF Connect”	118
14.5. nRF Connect Screen	122
14.6. Accelerometer Values Read from Arduino Using BLE	123
15.1. Li-ion battery composition [SSC16]	127
15.2. Damaged Lithium battery	128
17.1. Das Tiny Machine Learning Kit	133
17.2. Das Tiny Machine Learning Shield	134
17.3. Das OV7675 Kameramodul	135
17.4. Ein USB-A auf Micro-USB Kabel	136
17.5. USB-Kabel mit USB-A- und USB-Micro-B-Stecker mit 90° Winkel . .	137
17.6. Das Display zur Ausgabe der Messwerte	138
17.7. Die verwendete Powerbank	139
17.8. Schaltplan des Winkelmessers	139
18.1. IMU (Inertial Measurement Unit) Gesture Data	142

19.1. Arduino Nano 33 BLE Sense-Board mit hervorgehobener LED	145
19.2. Der serielle Plotter der Arduino-IDE	147
19.3. Der serielle Plotter der Arduino-IDE	149
19.4. Die Dropdown-Liste Board	150
19.5. Die Dropdown-Liste Port	150
19.6. Die Upload-Schaltfläche, ein nach rechts gerichteter Pfeil	151
19.7. Der Menüpunkt Serieller Plotter	151
19.8. Der Serienplotter, der den Wert grafisch darstellt	152
19.9. Der Serial Monitor zeigt Rohdaten an	152
20.1. Diagram of the components of our person detection application	157
20.2. Screenshot des Menüs 'Examples'	172
20.3. Screenshot of the 'Board' dropdown	174
20.4. Screenshot of the 'Port' dropdown	174
20.5. Screenshot of the upload button	175
21.1. Kamera IMX477 der Firma Arducam; [Ard21]	180
22.1. ArduCAM interface with Arduino	183
22.2. ArduCAM Pin Config	184
22.3. ArduCAM Interface with Arduin Mega 2560	184
23.1. Kalibrierungswerkzeug der Firma Arducam; [Ard21]	185
25.1. Tensorflow Lite workflow [Kha20]	192
25.2. The basic process for creating a model for an edge computer[Goo19a] .	192
26.1. Gestures used namely Slope, W and ring	195
27.1. Wing Gesture [WS20]	198
27.2. Ring Gesture [WS20]	198
27.3. Slope Gesture [WS20]	198
27.4. Convolutional Neural Network (CNN) sequence to classify gestures. Source : [WS20]	200
27.5. Inertial Measurement Unit (IMU) signals [Xu+22]	201
27.6. Inertial Measurement Unit (IMU) Accelerometer Graph [WS20]	202
27.7. A convolution window overlaid on the data. [WS20]	202
27.8. Max Pooling. [WS20]	203
27.9. Convolutional Neural Network (CNN) sequence to classify Wing,Ring and Slope.	204
27.10. Sample Confusion Matrix for the dataset [WS20]	205
27.11Arduino Nano website downloads page	205
27.12Arduino Nano software version number	206
27.13Arduino IDE initial software window	207
27.14Arduino IDE Boards Manager menu	207
27.15Arduino IDE Boards Manager list	208
27.16Selection of correct board for the program	208
27.17Selection of correct port for uploading the program	208
27.18Arduino Board Info	209
27.19Managing libraries in the Arduino IDE	209
27.20Function to Recognize gestures	210
27.21Code snippet that shows processing of the waved gestures . .	210
27.22Switch case statement, Pending Movement	210
27.23Code snippet that explains to train or to capture gestures .	211
27.24Code snippet displaying a Wing gesture	211
27.25The value of constants and constant thresholds in the program	211

27.26The displayed Wing gesture	212
27.27The displayed slope gesture	212
27.28The displayed ring gesture	212
27.29The unknown gesture	213
28.1. Code Snippet function Main header files()	218
28.2. Code snippet for isMoving() function	218
28.3. Code snippet that shows processing of the waved gestures	219
28.4. The pending switch case that checks if wand is moving or not	220
28.5. Code snippet checking if the wand moves or not	220
28.6. Code snippet to check if the wand is still	221
28.7. Code snippet to check if the wand is still	221
28.8. Code snippet showing capturing gesture function for training	222
28.9. Code snippet showing the loop() function in the program	223
28.10The accelerometer handler function	224
28.11Gesture Predictor function	225
28.12The code handling the output	225
28.13Code snippet showing the loop() function in the program	226
28.14Code displaying threshold values	226
28.15Magic Wand Program Workflow	226
28.16Components in Arduino Nano 33 BLE Sense [Raj19]	228
28.17How to connect with computer	229
28.18Pinout	230
28.19Schematic connection diagram	231
28.20Technical Specification	231
28.21Pin connections LSM9DS1	232
28.22Pin description LSM9DS1	233
28.23Sensor Characteristics	234
28.24Temperature Sensor Characteristics	234
28.25Absolute Maximum Temperature	235
28.26Accelerometer and gyroscope block diagram	235
28.27Magnetometer block diagram	236
28.28LSM9DS1 electrical connections	236
29.1. Edge Impulse Flow Chart	239
29.2. Counting Finger Using MediaPipe and OpenCV	240
29.3. Hand Landmarks using MediaPipe	241
29.4. Hand Landmarks using MediaPipe Function	241
29.5. Gesture Detection on Arduino	242
29.6. Gesture Detection Using LSTM, TensorFlow, and MediaPipe	243
29.7. Object detection Model Comparison	244
30.1. Pyfirmata Communication for Python on Arduino	248
30.2. Logitech Camera Replacement	249

Acronyms

- AOP** Acoustic Overload Point
- 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
- OLED** Organic Light Emitting Diode
- PDM** Pulse Density Modulation
- RGB** Rot-Grün-Blau
- SCL** Serial Clock
- SDA** Serial Data
- UART** Universal Asynchronous Receiver Transmitter

Part I.

Machine Learning

1. Arduino IDE

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 as follows



Figure 1.2.: Menu buttons

- 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

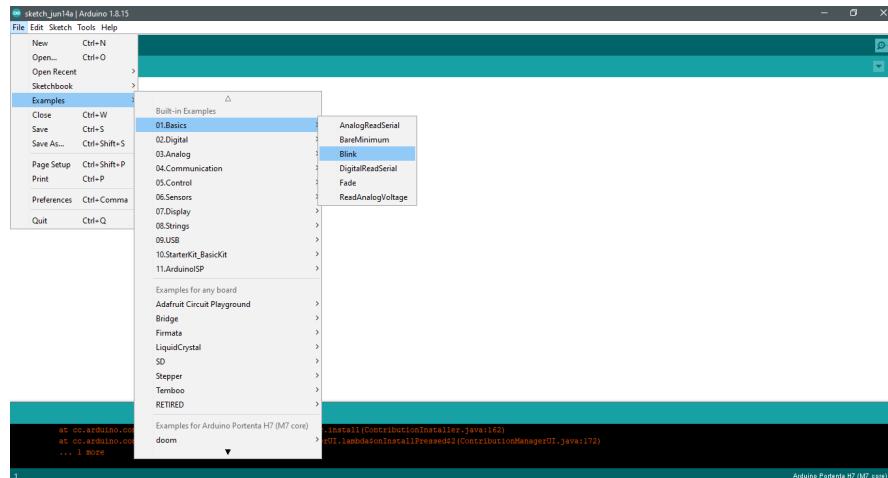


Figure 1.1.: Menu bar options

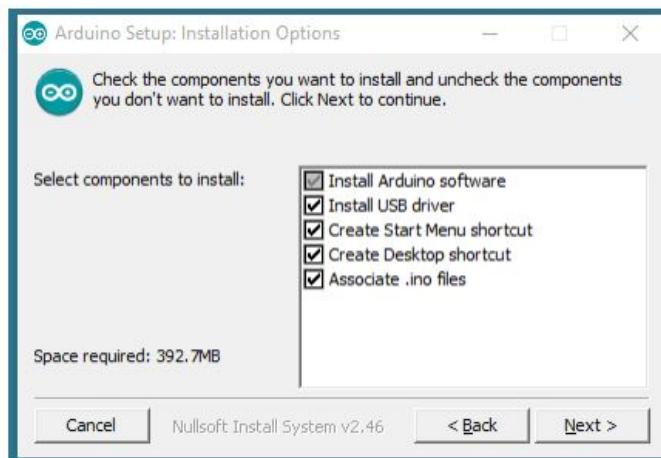


Figure 1.3.: Arduino Setup Installation options

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**.

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 1.5 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.

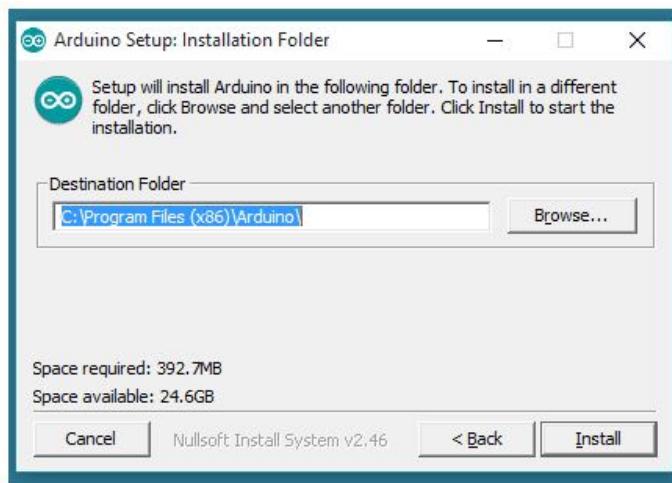


Figure 1.4.: Arduino Setup: Installation Folder

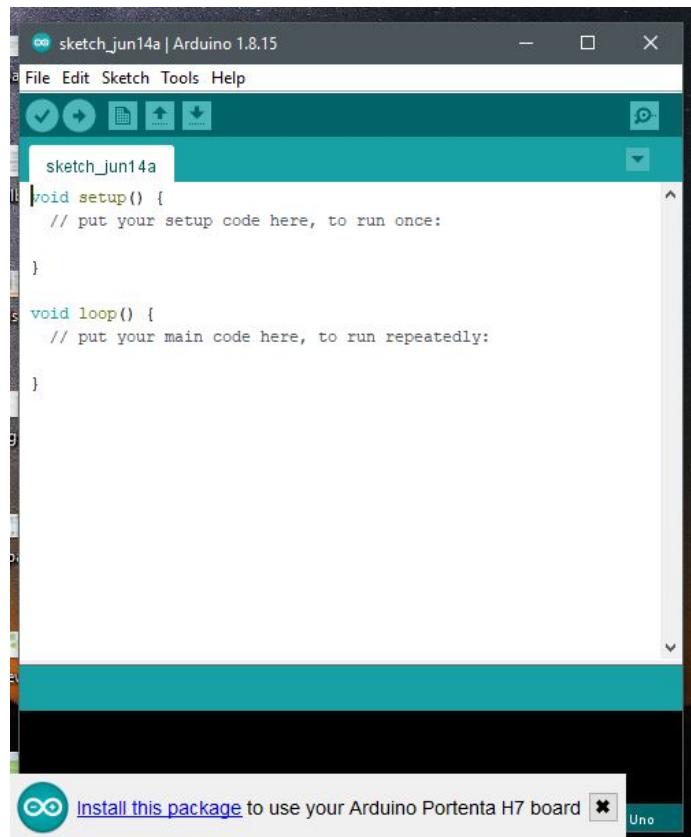


Figure 1.5.: Arduino Sketch

1.4. Arduino IDE on PC

1.4.1. Installation

Arduino Nano 33 BLE Sense can provide an energy-efficient and cost-effective solution for manufacturers who want to use Bluetooth Low Energy connectivity in their projects.

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 arduino IDE 1.8.15 can be seen in Figure, 1.6 it is also supportive for all operating systems.



Figure 1.6.: Arduino Creat Agent Installation

1.4.2. Configuration

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 upper 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, 1.7 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.

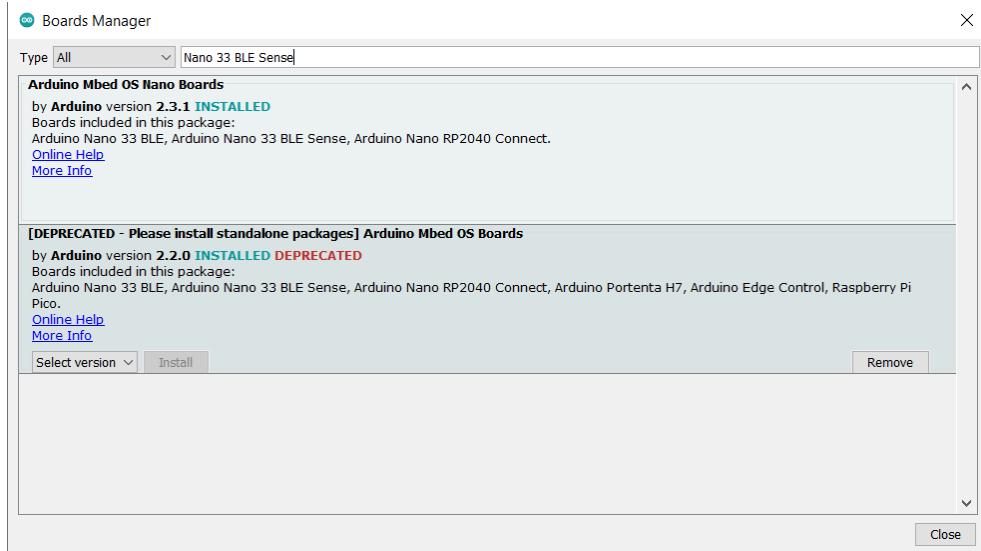


Figure 1.7.: Arduino Mbed OS Nano Boards Installation

1.4.3. Test Example

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. 1.8.

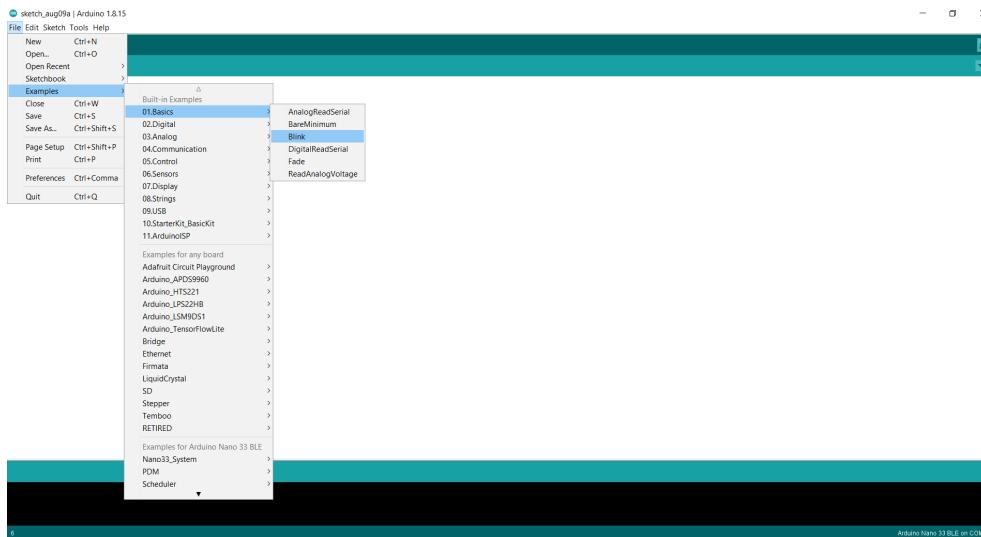


Figure 1.8.: LED-Example Test

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.

1.5. Pre-requisite settings for Uploading the program

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 1.9

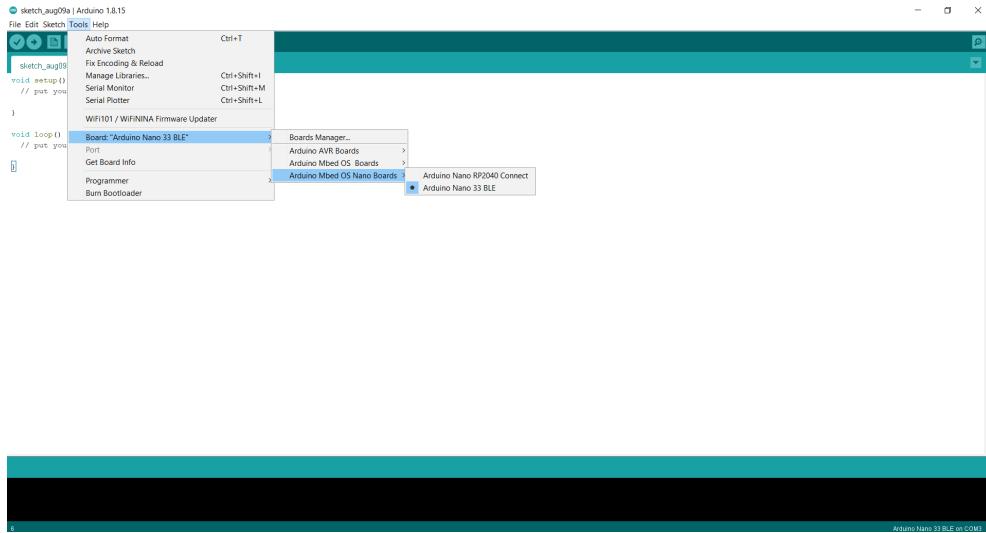


Figure 1.9.: Select the Connected board -here: Arduino Nano 33 BLE Sense

1.5.1. 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 figure1.10.

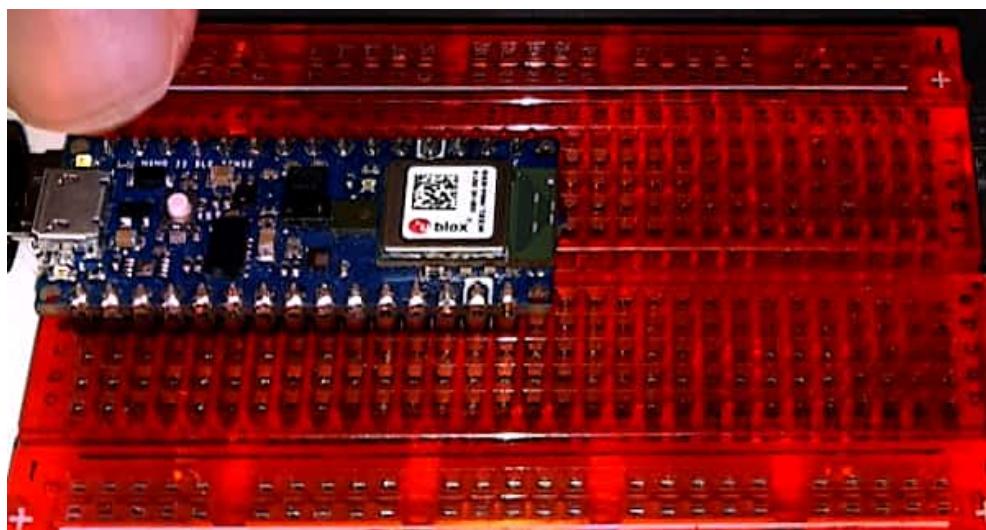


Figure 1.10.: Arduino Nano 33 BLE Sense: Reset Button

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 1.11.

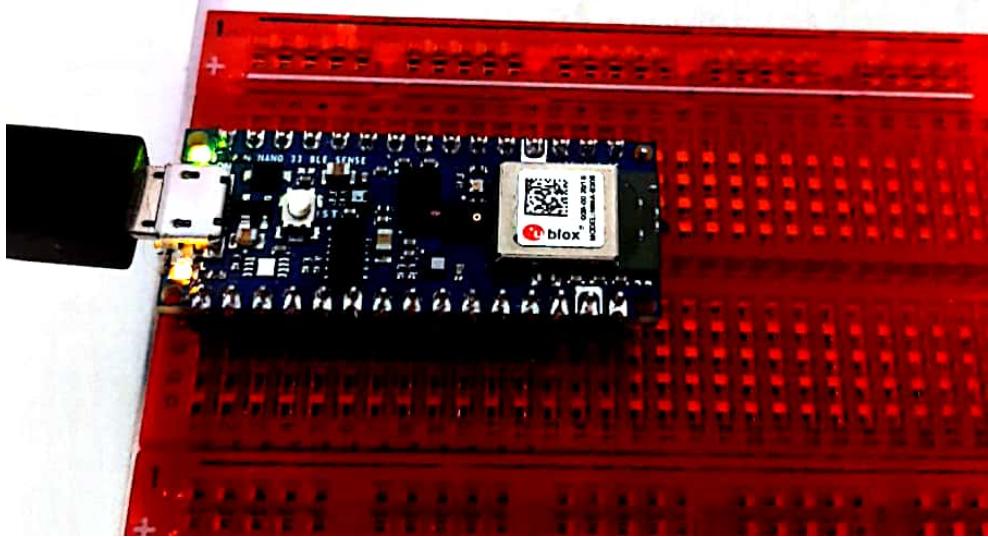


Figure 1.11.: Arduino Nano 33 BLE Sense: Orange LED Glow

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 1.12

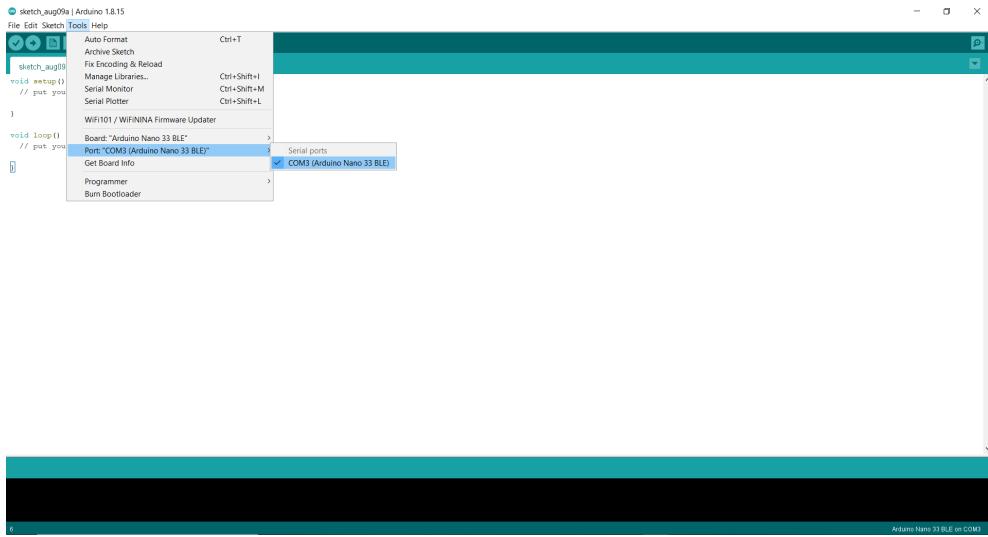


Figure 1.12.: Select Available Port for Uploading Arduino Sketch

1.5.2. 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 1.13.



Figure 1.13.: 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 1.14 by getting output in the serial monitor.

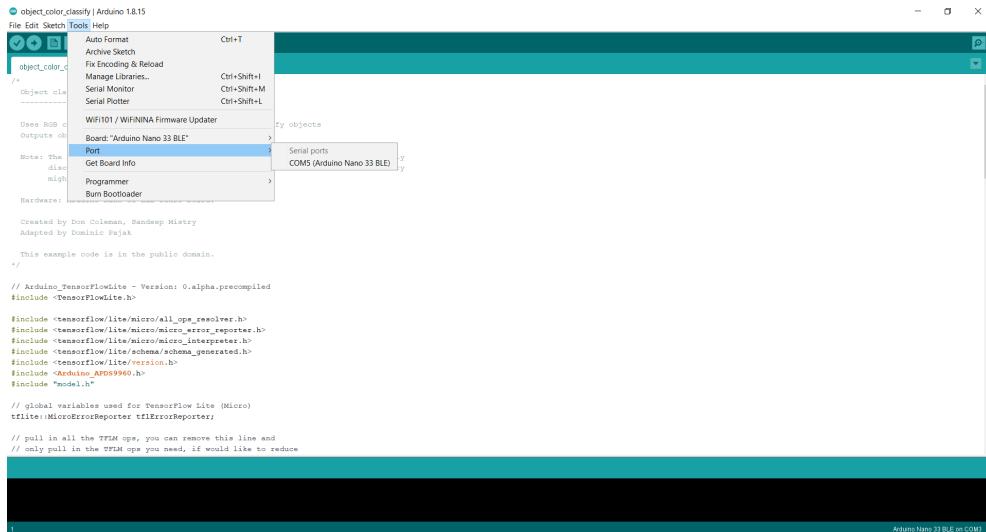


Figure 1.14.: Setting the Port

1.6. 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. 1.15

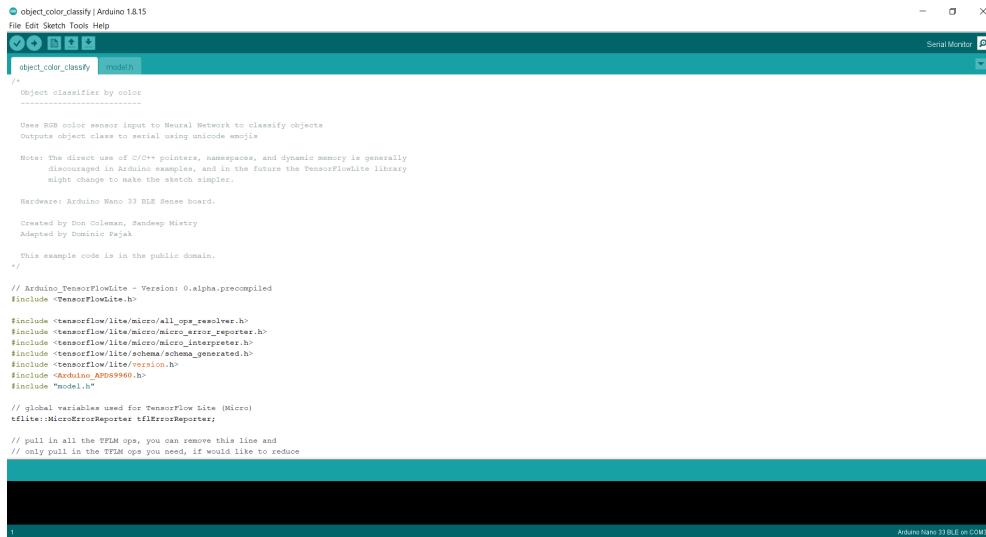


Figure 1.15.: Serial Monitor Icon

The Final results, all the variables, input, sensor values are shown in the serial monitor (Output Window) as shown in the figure 1.16 by clicking the serial monitor button.

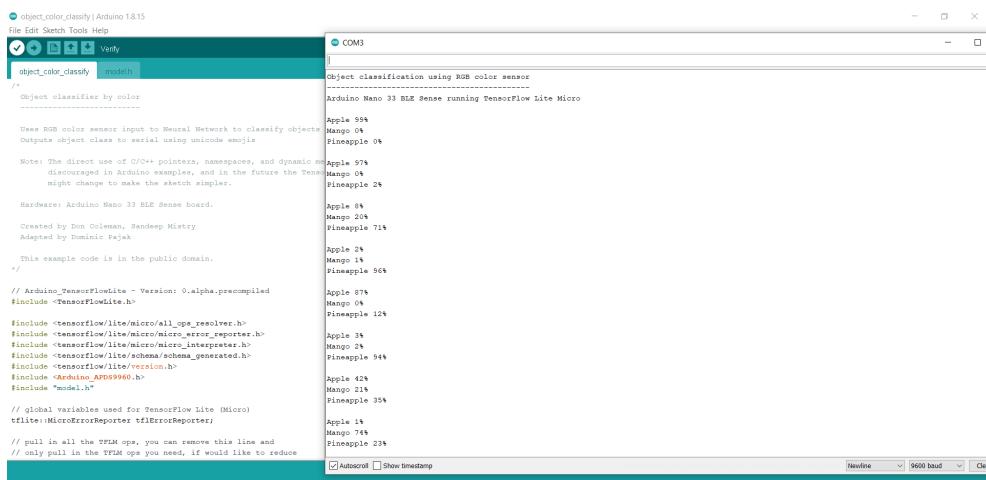


Figure 1.16.: Output Window

Part II.

TinyML

2. 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!

2.1. 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.

WS:Advertisement!
Rewrite more as an engineer!

The following figure 2.1 shows a Arduino Nano 33 BLE Sense, shows how compact it is and small in size.



Figure 2.1.: Arduino Nano 33 BLE Sense, see Arduino Store

The BLE (Bluetooth Low Energy) compact and reliable Nano board are built on NINA B306 module for BLE and Bluetooth 5 communication; the NINA B306 module based on Nordic nRF52480 processor that contains a powerful Cortex M4F CPU, Its architecture 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.

2.2. On-Board Sensor Description

Arduino Nano 33 BLE sense come up with the set of embed sensor on the board. The available embed sensors are commonly use for measuring both the analog and digital values around the sorrounding. Arduino Nano 33 BLE sense is very small 45mm × 18mm in size, which makes it very usefull for Internet of things (IOT) and Artificial intelligence (AI) application as a embed device where space is the main constrained issue. It is low power consumption board and operate normally on 3.3 V, we can say that this small size low power consumption board can operate on small batteries even for many months. Due to on-board available sensor, the low power consumption and mini architecture we can use this nano board anywhere. The Arduino Nano 33 BLE Sense is a completely new board on a well-known form factor. For getting detail information about each component of Arduino Nano 33 BLE Sense and data sheets of each sensor the following links give us a detail information [Ard21]. The short description of each sensor are as follow.

- The ADPS-9960 is a digital proximity, ambient light, RGB and gesture sensor. it can measure the proximity distance, light, color and gestures when moving close with the borad.
- The sensor LSM9DS1 is a 9 axis Inertial Measurement Unit (IMU) use as a accelerometre, gyroscope, and magnatometre, this 9 axis sensor is ideal for wearable devices.
- The sensor LPS22HB is a barometric pressure sensor, it measures the environmental pressure which is 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 2.2 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.

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.

2.2.1. 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

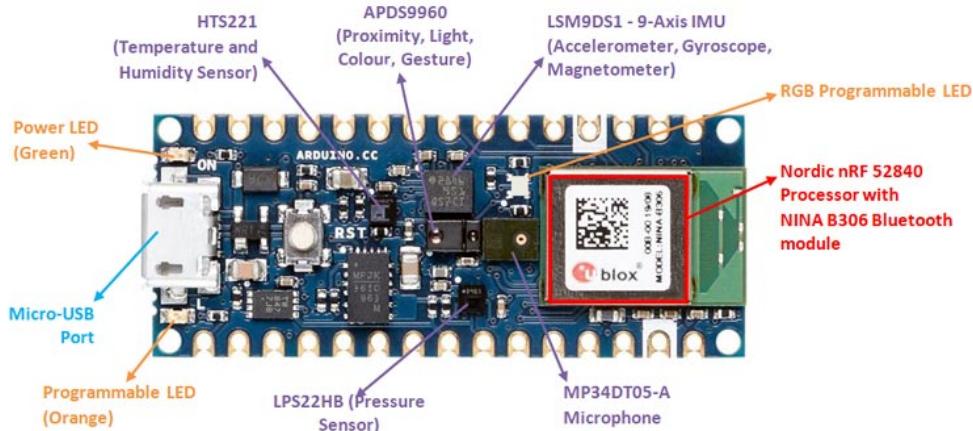


Figure 2.2.: Components in Arduino Nano 33 BLE Sense [Raj19]

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 2.1.: Technical Specifications of Arduino Nano 33 BLE Sense [Ard21]

- Color Sense
- Ambient Light Sensing
- Proximity Sensing

2.2.2. 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!

WS:What is calibration citations!

Calibration of an IMU

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

Low and High Limit Method

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

Magneto V1.2

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

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

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

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

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

- General infomrati about imus
- angle to roation matrix with example!

2.2.3. 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

2.2.4. 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 Temperaturmessung erfolgt mit einer Genauigkeit von $\pm 5^{\circ}\text{C}$. [STM23]

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

2.2.5. 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

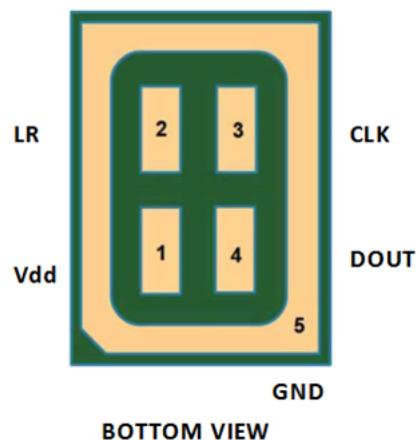


Table 1. Pin description

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 2.3.: Circuit diagram microphone [Stmc]

2.2.6. Bluetooth Module nRF52840

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

Applications

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

2.3. 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 2.4 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.

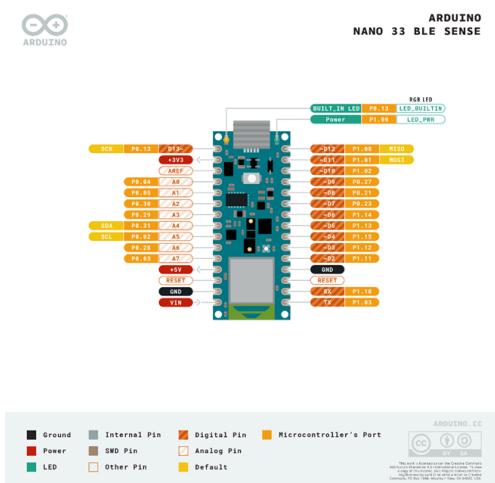


Figure 2.4.: Arduino Nano 33 BLE Pin Configuration

3. Test of the Hardware

3.1. 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 3.1, it shows the sign of Arduino board working.

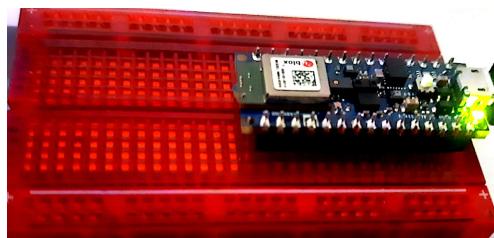


Figure 3.1.: Power On Arduino Nano 33 BLE Sense

3.2. 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.

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

The 9-axis inertial measurement unit (IMU) sensor is work as a accelerometre, 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. 3.2 Go to examples check LSM9DS1 sensor, it shows all the functionality of this sensor i.e: (Accelerometer, Gyroscope, and Magnetometer).

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

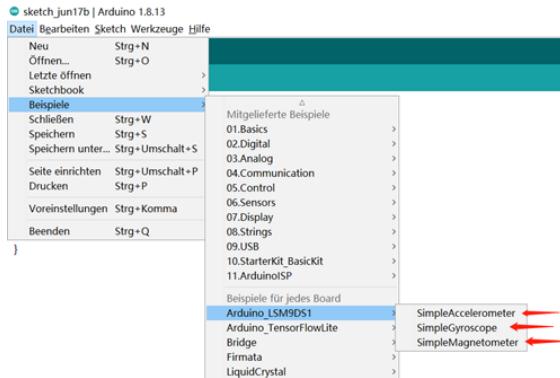


Figure 3.2.: 9-Axis IMU Sensor

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.3.3 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.

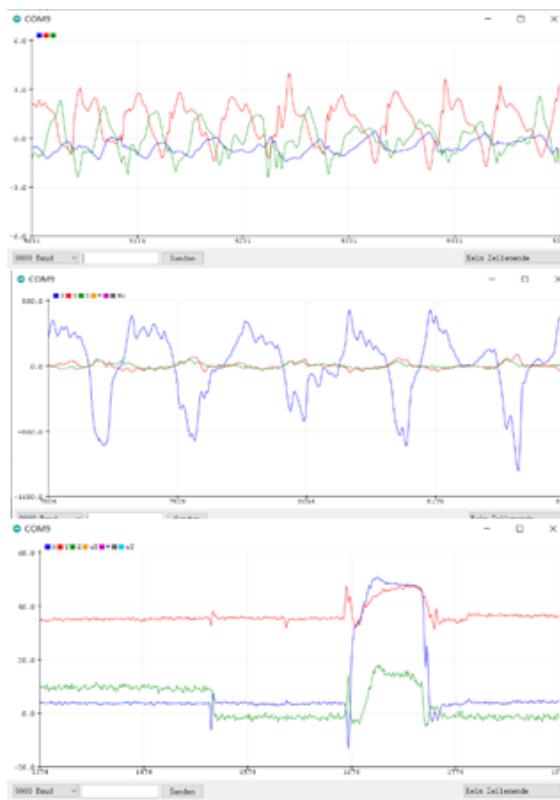


Figure 3.3.: Results of IMU-Tests

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.

3.2.2. APDS-9960 (Gesture, Proximity, and Color Detection Sensor)

APDS-9960 sensor has the functionality of proximity distance measure, RGB color detection, and Gesture detection too. It is also on-board embed sensor on Arduino nano 33 BLE Sense and has the same procedure for uploading and compiling the program as the 9-axis IMU sensor has. Go to example, check the APDS-9960 library and upload the complete program as shown in the figure. 3.4 Full example includes the RGB color detection code, gesture detection code, and also proximity distance measure code.

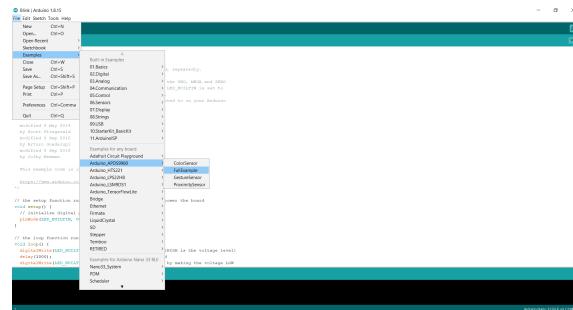


Figure 3.4.: APDS-9960 Gesture, Proximity, Color Sensor

Similarly, by following all the steps for uploading and compiling the program we can see the results of APDS-9960 Sensor 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. 3.5

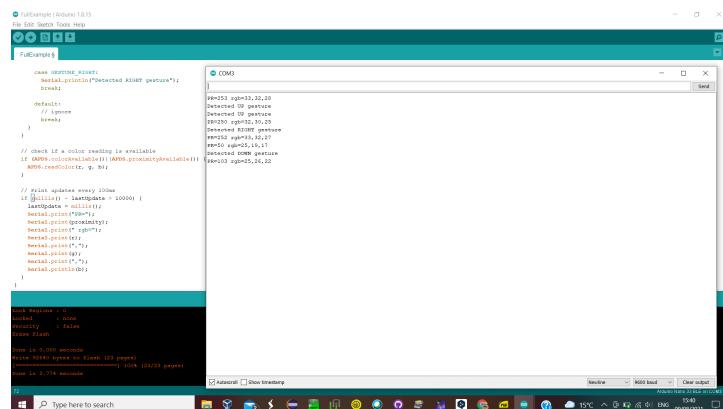


Figure 3.5.: Gesture, Proximity, Color Sensor Output Window

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.

3.2.3. 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 section from file, then go to LPS22HB and click on read or measuring pressure as shown in the figure. 3.6

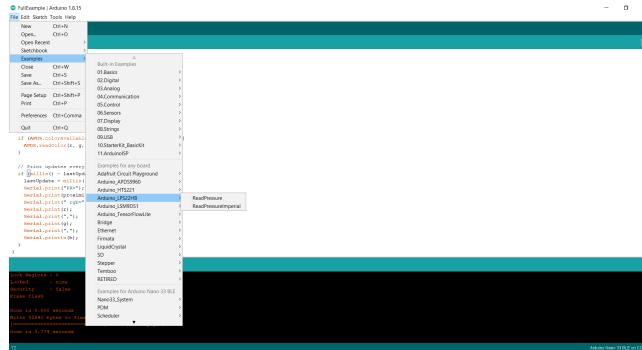


Figure 3.6.: LPS22HB, Pressure and Temperature sensor

The results for LPS22HB the environmental pressure also showed in the serial monitor as shown in the figure. 3.7 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.

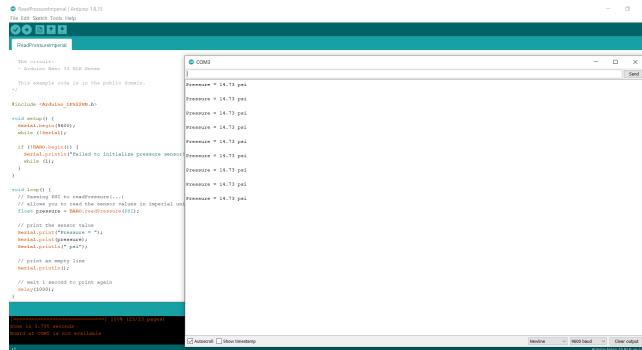


Figure 3.7.: LPS22HB, Output

3.2.4. HTS221 (Humidity and Temperature Sensor)

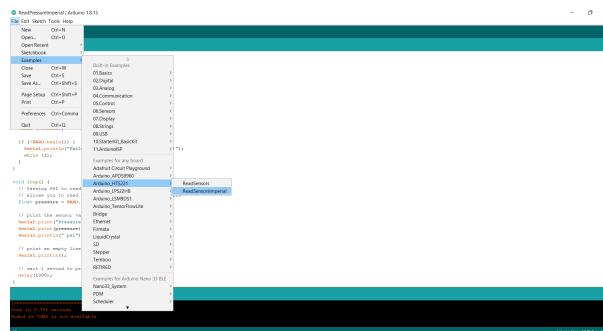


Figure 3.8.: HTS221, Humidity and Temperature Sensor

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. 3.8

After Uploading and compiling the program, the environmental humidity and temperature also display in the serial monitor as shown in the figure. 3.9 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.

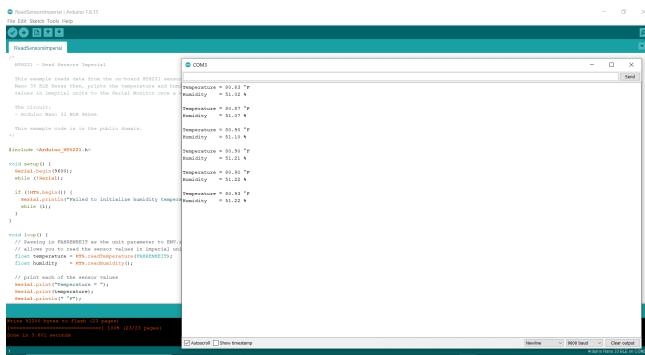


Figure 3.9.: HTS221, Output Window

3.2.5. 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. 3.10

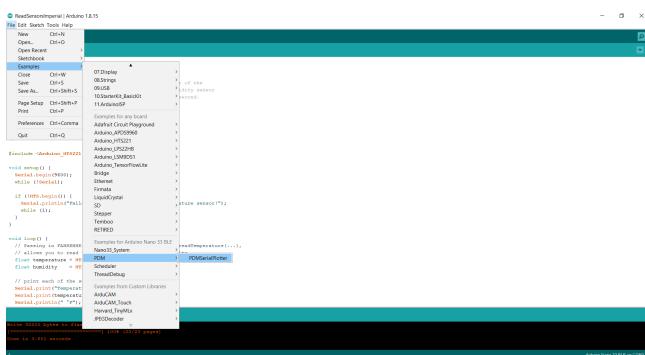


Figure 3.10.: MP34DT05, Digital Microphone

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. 3.11 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.

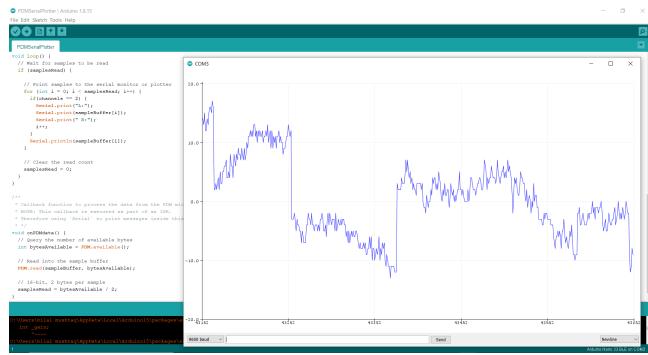


Figure 3.11.: MP34DT05, Serial Plotter

3.3. 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 3.12 shows the ArduinoBLE library in the example section of Arduino IDE.

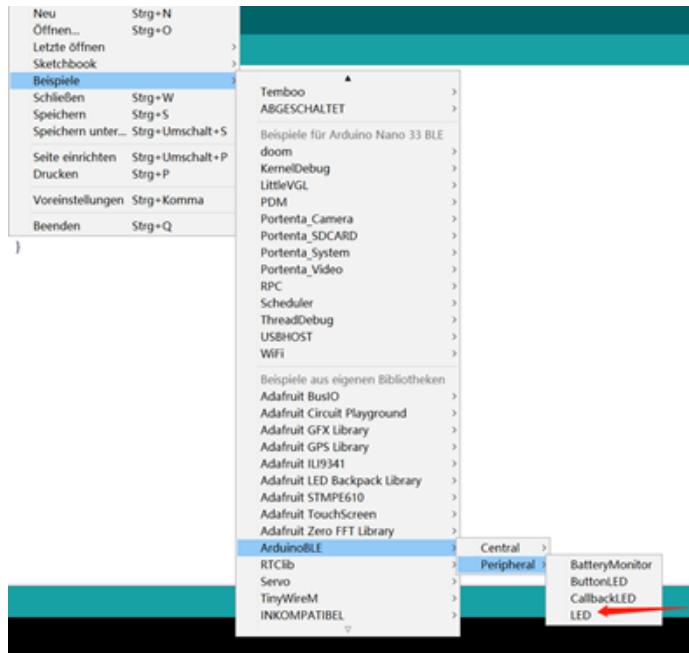


Figure 3.12.: Bluetooth Connection

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 recieving 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.

3.4. 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.

WS:No?

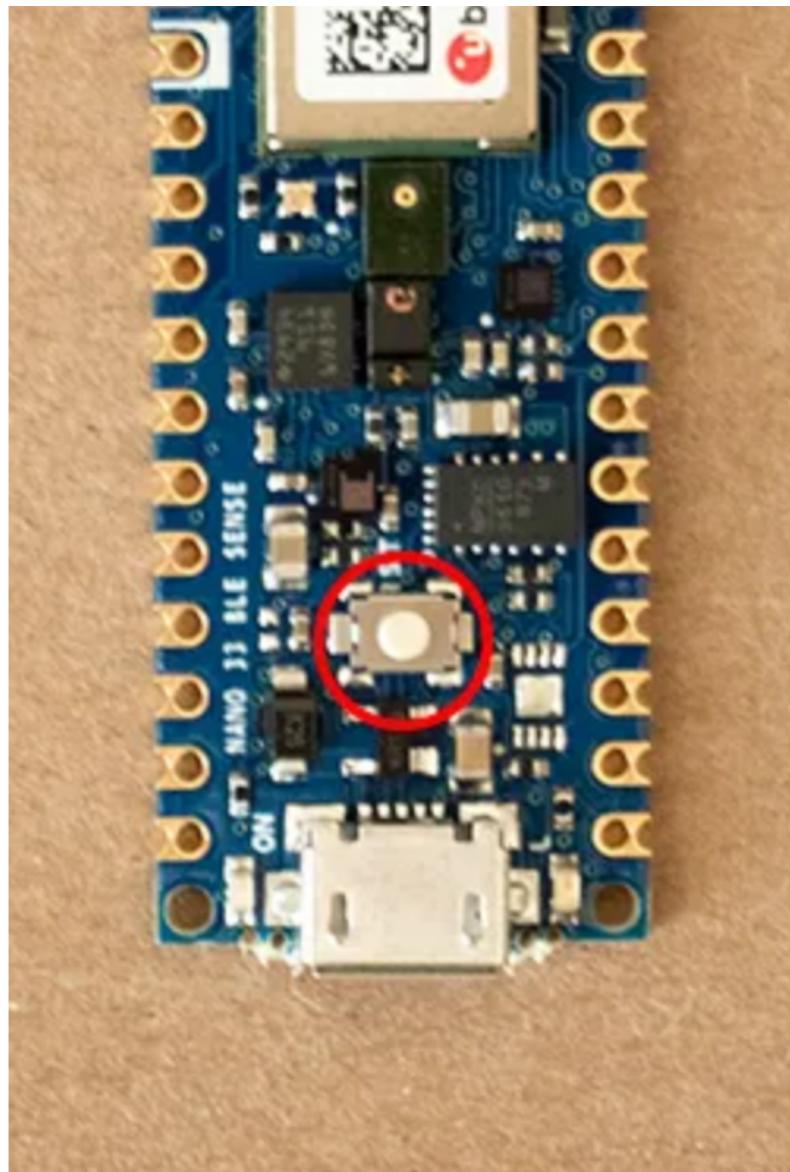


Figure 3.13.: The Reset Button of the Arduino

4. 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.

4.1. Sketch “Hello World”

Zur Prüfung der Entwicklungsumgebung und der grundsätzlichen Funktion der Hardware, bietet sich ein einfacher Beispieldatenkette 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 Beispieldatenketten ausgewählt werden. Hier wird das Beispiel [Fade](#) genutzt. Hierbei wird die eingebaute LED, welche eine RGB-LED ist, genutzt.

```
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);
}
```

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

4.2. 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.

```
#include <Arduino_LSM9DS1.h> // IMU
#include <Arduino_APDS9960.h> // Farbsensor
#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.

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

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

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

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

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 `.begin()` Funktion eine null als Rückgabewert, die zu einer eins negiert wird und es folgt die Mitteilung der fehlgeschlagenen Initialisierung an den seriellen Monitor.

```
void loop() {
    int c1 = 0; // Zähler für Verfügbarkeitsprüfung der Beschleunigungssensoren
    int c2 = 0; // Zähler für Verfügbarkeitsprüfung der Farbsensordaten

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

Die `loop()`-Funktion 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.

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

Serial.println();
Serial.println("Testwerte des LPS22HB-Sensors:");
Serial.println();
```

```

// Ausgabe der gemessenen Temperatur
Serial.print("Die Temperatur T = ");
Serial.print(temperatur);
Serial.println(" °C wurde gemessen.");

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

```

// Testen der IMU
Serial.println("Test der IMU-Einheit:");
Serial.println();
while (! IMU.accelerationAvailable()) {
    delay(5);
    c1++;
    if(c1 > 15)
    {
        break;
    }
}

```

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.

```

IMU.readAcceleration(x, y, z);

Serial.print("Die Beschleunigung in x-Richtung beträgt ");
Serial.print(x*100);
Serial.println(" % der Erdbeschleunigung.");
Serial.print("Die Beschleunigung in y-Richtung beträgt ");
Serial.print(y*100);
Serial.println(" % der Erdbeschleunigung.");
Serial.print("Die Beschleunigung in z-Richtung beträgt ");
Serial.print(z*100);
Serial.println(" % der Erdbeschleunigung.");
Serial.println();
Serial.print("Der Betrag der addierten Beschleunigungsvektoren beträgt ");
Serial.print(sqrt(x*x+y*y+z*z)*100);
Serial.println(" % der Erdbeschleunigung (Sollwert bei Ruhelage ca. 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-Rich-

tungen berechnet. Je nach geographischer Lage sollte der Betrag bei ca. 100 % der Erdbeschleunigung liegen.

```
while (! APDS.colorAvailable()) {
    delay(5);
    c2++;
    if(c2 > 15)
    {
        break;
    }
}
```

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

```
// Auslesen der Farbwerte
APDS.readColor(r, g, b);

// Ausgeben der Farbwerte
Serial.print("Der Rotwert r = ");
Serial.print(r);
Serial.println(" wurde gemessen.");
Serial.print("Der Grunwert g = ");
Serial.print(g);
Serial.println(" wurde gemessen.");
Serial.print("Der Blauwert b = ");
Serial.print(b);
Serial.println(" wurde gemessen.");
Serial.println();
```

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.

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

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.

4.3. APDS-9960 Modul

Der Farberkennungssensor APDS-9960 wird mittels einer Abfrage im *void setup{}-Programmteil* getestet.

```
#include <Arduino_APDS9960.h>

void setup(){
    Serial.begin(9600)

    if (!APDS.begin())
    {
        Serial.println("Error initializing APDS9960 sensor.")
    }
}
```

Hierfür muss zuerst die entsprechende Bibliothek, die zum Farbsensor APDS-9960 gehört, eingebunden werden. Über diese Bibliothek kann auf die Funktion *APDS.begin*

zugegriffen werden. Wird diese Abfrage mit einer *0* vom Sensor beantwortet, wird die Bedingung in der *if*- Abfrage durch das vorangestellte *!* negiert und somit zu *true*. Die Bedingung für die *if*- Verzweigung ist somit erfüllt und im seriellen Monitor erscheint der Text *Error initializing APDS9960 sensor*. Der Benutzer weiss nun, dass der Sensor nicht angesprochen werden kann und die Nutzung des Farbsensors nicht möglich ist. Im Regelfall sollte dieser Fehler nicht auftreten und das Programm springt in die *loop*-Funktion. Im *void loop{}-* Programmteil werden dann noch einmal der Farbsensor und der Abstandssensor direkt abgefragt. An dieser Stelle wird diese Abfrage allerdings in 5 ms Abständen wiederholt, bis der Sensor einen Wert zurück gibt.

```
while (!APDS.colorAvailable()) { // ueberpruefen ob
    //Farberkennung
    //verfuegbar ist ,
    //falls nicht 5 ms warten und Abfrage wiederholen
    delay(5);
}

while (!APDS.proximityAvailable()) { // ueberpruefen ob
    //Abstandsmessung
    //verfuegbar ist ,
    //falls nicht 5 ms warten und Abfrage wiederholen
    delay(5);
}
```

4.4. OLED-Display

Beim OLED-Display verhält es sich etwas anders. Auch hier wird im *void setup{}-* Programmteil die gewünschte Hardware, in diesem Fall das Display angesprochen. Allerdings wird direkt ein Befehl genutzt um den Text *READY!* auszugeben. Da die gesamte Kommunikation mit dem Benutzer über das Display läuft, wäre eine Benutzung im Falle eines Displayschadens grundsätzlich nicht möglich. In diesem Fall müsste der Arduino an einen PC angeschlossen und über den seriellen Monitor geschaut werden in wie weit der Arduino, das OLED-Display oder der APDS-9960 Sensor defekt ist.

```
Wire.begin();
Wire.setClock(400000L);
// Clock Frequenz in Hertz , 100.000 Standard ,
// 400.000 Fast Mode

oled.begin(&Adafruit128x64, I2C_ADDRESS);
// erst Bildschirmgroesse ,
// dann Referenz auf I2C Adresse

oled.setFont(System5x7);
oled.setCursor(0, 10);
oled.println("READY!\n");
```

4.5. RGB-LED

Die angeschlossene RGB-LED dient zur Vermeidung von schlechten Lichtverhältnissen, beispielsweise in der Dämmerung oder nachts. Softwareseitig wird die LED

nicht getestet. Nach dem Einschalten des Automaten wird die Initialisierungsphase eingeleitet. Diese wird über den OLED-Bildschirm angezeigt und begleitet durch ein dreimaliges Aufblitzen der LED. Sollte hier die LED nicht dreimal blinken ist dies ein Indiz für ein Problem an der LED. Ferner wird die LED auch während des Messvorgangs angesteuert und leuchtet. Auch hier kann optisch erkannt werden, falls es ein Problem mit der LED gibt.

4.6. Mikrophon MP34DT05-A

4.6.1. Verwendete Bibliotheken

Der Code verwendet die folgenden Bibliotheken:

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

4.6.2. Pin-Konfiguration

Die verwendeten Pins werden zu Beginn des Codes definiert:

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

4.6.3. Initialisierung und Setup

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

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

4.6.4. 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 4.1.: Code Einlesen/Zählen

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

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

Figure 4.2.: Code Messungssteuerung

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

```

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

```

Figure 4.3.: Code Überwachung

- 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.

4.6.5. Mikrofondatenverarbeitung

Die Funktion `StartSampling()` initialisiert die Datenverarbeitung:

```

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 4.4.: Code Initialisierung der Datenverarbeitung

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

4.6.6. 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.

4.6.7. Umrechnung auf den Wert dB SPL

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

```

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 4.5.: Code Umrechnung Mikrofonsignal

`MaxSampleVoltage` wird berechnet, indem der maximale Samplewert `maxSampleValue` mit der Referenzspannung des Mikrocontrollers („Vref = 3,3 V“) multipliziert und durch den maximalen Wert eines 16-Bit-Signed-Integers (32767) dividiert

wird. 32767 ist der maximale Wert eines 16-Bit-Signed-Integers, der der maximalen Amplitude des Audiosignals entspricht. Der dB SPL-Wert („dBspl“) wird mit der Formel „ $20 * \log_{10}(\text{Voltage} / \text{Vrms}) + \text{sensitivity}$ “ berechnet [Quelle der Formel: PDF Decibels Formula https://physicscourses.colorado.edu/phys3330/phys3330_sp19/resources/Decibels_Phys_3330.pdf University of Colorado System], wobei „Voltage“ der berechnete „maxSampleVoltage“ ist, und „sensitivity“ die Empfindlichkeit des Mikrofons in dBV (Dezibel-Volt) ist. „Vrms“ ist die RMS-Spannung (Root Mean Square), die einem Schalldruckpegel von 94 dB SPL entspricht. Dieser Wert wurde experimentell ermittelt.

4.6.8. 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 –

$$dB = 20 \log_{10} \left(\frac{V_2}{V_1} \right).$$

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 $220mVrms$.

4.6.9. Benutzeroberfläche

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

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

Figure 4.6.: Code OLED-Aktualisierung

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

5. 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 5.1[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[Ardd].

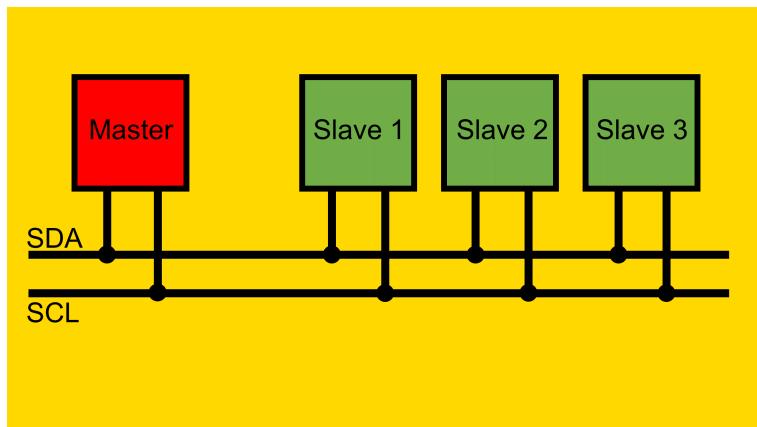


Figure 5.1.: Aufbau des Protokolls I²C [GW22]

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 5.2. [GW22]

WS:Grafik besser mit t
WS:unverständlich;
Beispiel fehlt

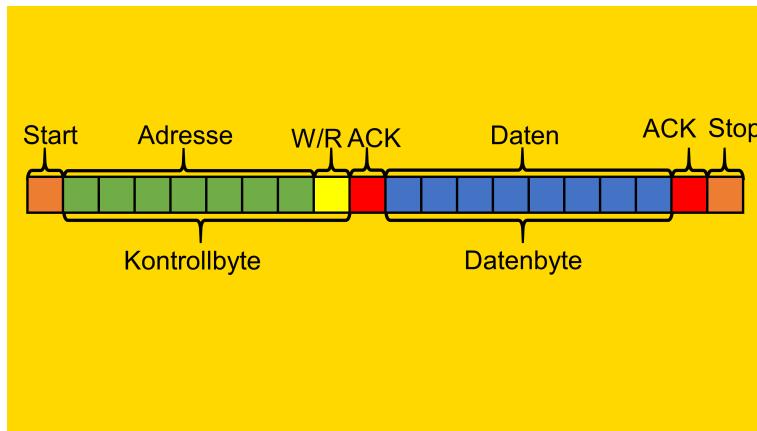


Figure 5.2.: I²C-Protokoll Quelle: [GW22]

S:Grafik besser mit tikz

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.

5.1. 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]

6. 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).

7. Inertial Measurement Unit

7.1. 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 accelerometers and gyroscopes that work together to provide information about an object's acceleration, velocity, orientation, and gravitational forces.[Sab11]

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]

7.2. 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.

7.3. 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.

7.4. 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 7.1

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 7.2

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.

7.4.1. Accelerometer:

[Chi+06]

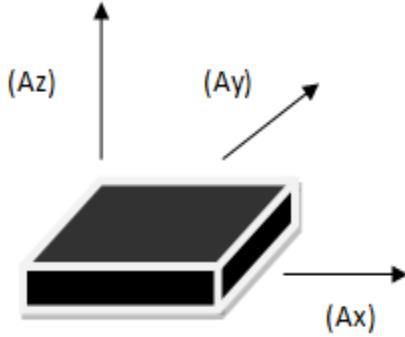


Figure 7.1.: Axis Acceleration of Accelerometer Sensor

- Acceleration in X-axis (Ax): meters per second squared (m/s^2)
- Acceleration in Y-axis (Ay): meters per second squared (m/s^2)
- Acceleration in Z-axis (Az): meters per second squared (m/s^2)

7.4.2. Gyroscope

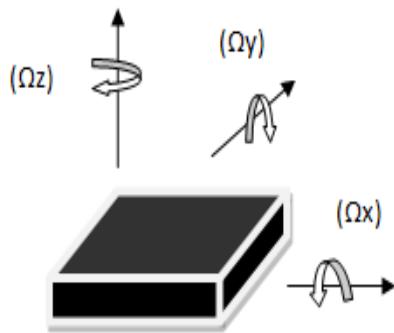


Figure 7.2.: Angular Speed of Gyroscope Sensor

- 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 ($^\circ/s$)

7.5. 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.

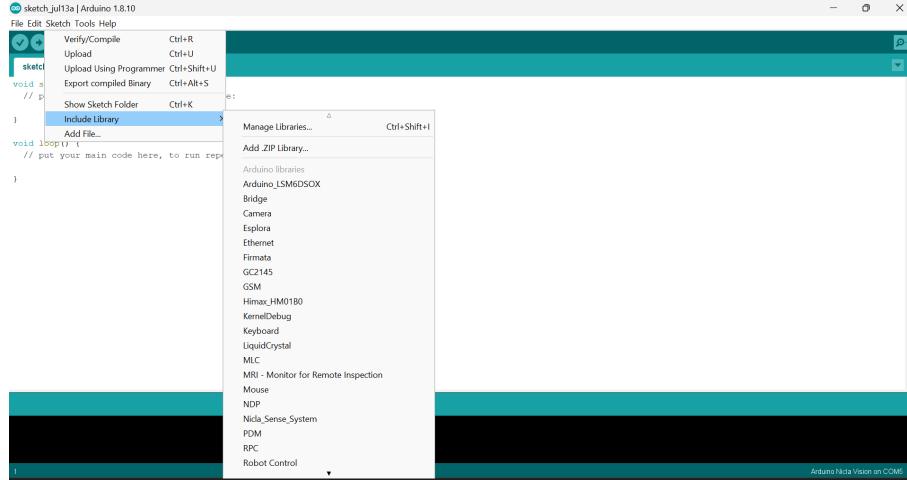


Figure 7.3.: Library setup in Arduino IDE

7.6. 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.

8. Software für eine IMU

8.1. Erste Schritte mit der Entwicklungsumgebung

Um den Arduino zu testen, wird er vorsichtig auf das Tiny Machine Learning Shield gesteckt. Über das mitgelieferte USB-A- auf USB-Micro-B-Kabel schließt man den Arduino an den Computer an. Um die Software einzurichten, wird zuallererst das Mbed OS Nano Board benötigt. Um den Arduino zu beschreiben, ist dieses Board essenziell. Darauffolgend muss ein Port gewählt werden. Bei einem Port handelt es sich um eine vom Computer selbst gewählte serielle Schnittstelle, auf der die Kommunikation zwischen dem Arduino und dem Computer erfolgt.

8.2. 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 des angesprochenen LSM9DS1 benötigt. In dieser Bibliothek sind die Funktionen enthalten, um die Bewegungen zu erfassen und in Winkel umzurechnen. Zusätzlich wird die SSD1306Ascii Bibliothek zur Zeichendarstellung für das OLED-Display benötigt.

8.2.1. Sensor LSM9DS1

Der Sensor LSM9DS1 besteht aus drei einzelnen Sensoren. Beschleunigungssensor, Gyroskop und Magnetometer. Kombiniert ergeben die drei Sensoren die IMU.

Die Bibliothek des Sensors LSM9DS1 enthält drei Beispiele, die es dem Anwender ermöglichen mit wenig Aufwand jeweils einen der Sensoren zu benutzen.

WS: Welche Bibliothek?

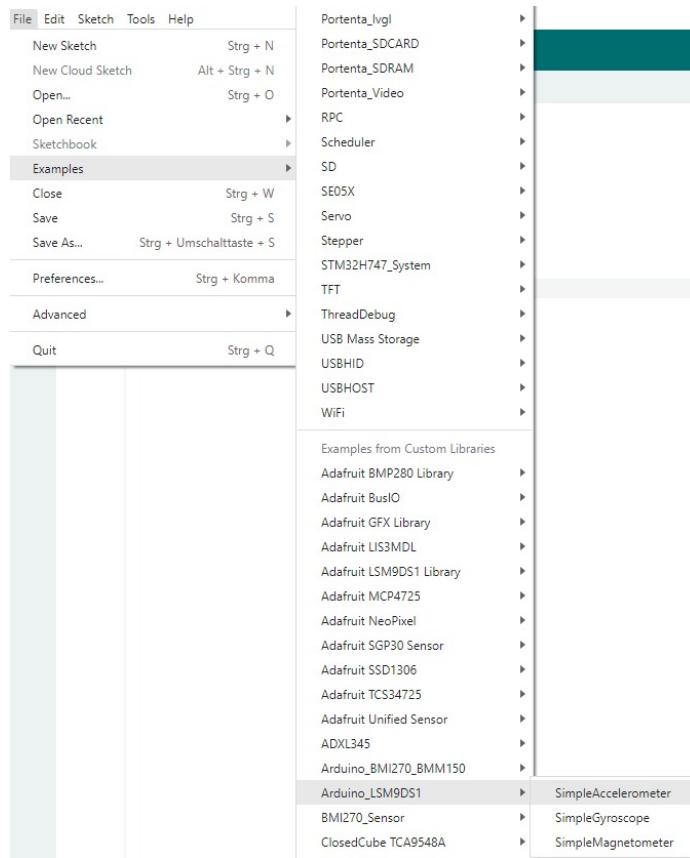


Figure 8.1.: Beispiele in der LSM9DS1 Bibliothek

8.3. Beispiele auf dem Mikrocontroller

8.3.1. Testen des Sensors LSM9DS1

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

```

Output Serial Monitor ×
Message (Enter to send message to 'Arduino Nano 33 BLE' on 'COM4')
No Line Ending 2000000 baud
Started
Accelerometer sample rate = 119.00 Hz

Acceleration in g's
X   Y   Z
0.02 -0.03 0.89
0.14 -0.08 1.41
0.01 -0.03 0.88
0.02 -0.03 0.97
0.02 -0.03 0.98
0.02 -0.03 0.98
0.02 -0.03 0.98
0.02 -0.03 0.98
0.02 -0.03 0.98
0.02 -0.03 0.98
0.02 -0.03 0.97
Ln 1, Col 1 Arduino Nano 33 BLE on COM4 🔍 2 ⌂

```

Figure 8.2.: Output des Testprogramms

8.4. Programmierung

8.4.1. Programmablaufplan

8.4.2. 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);
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);

```

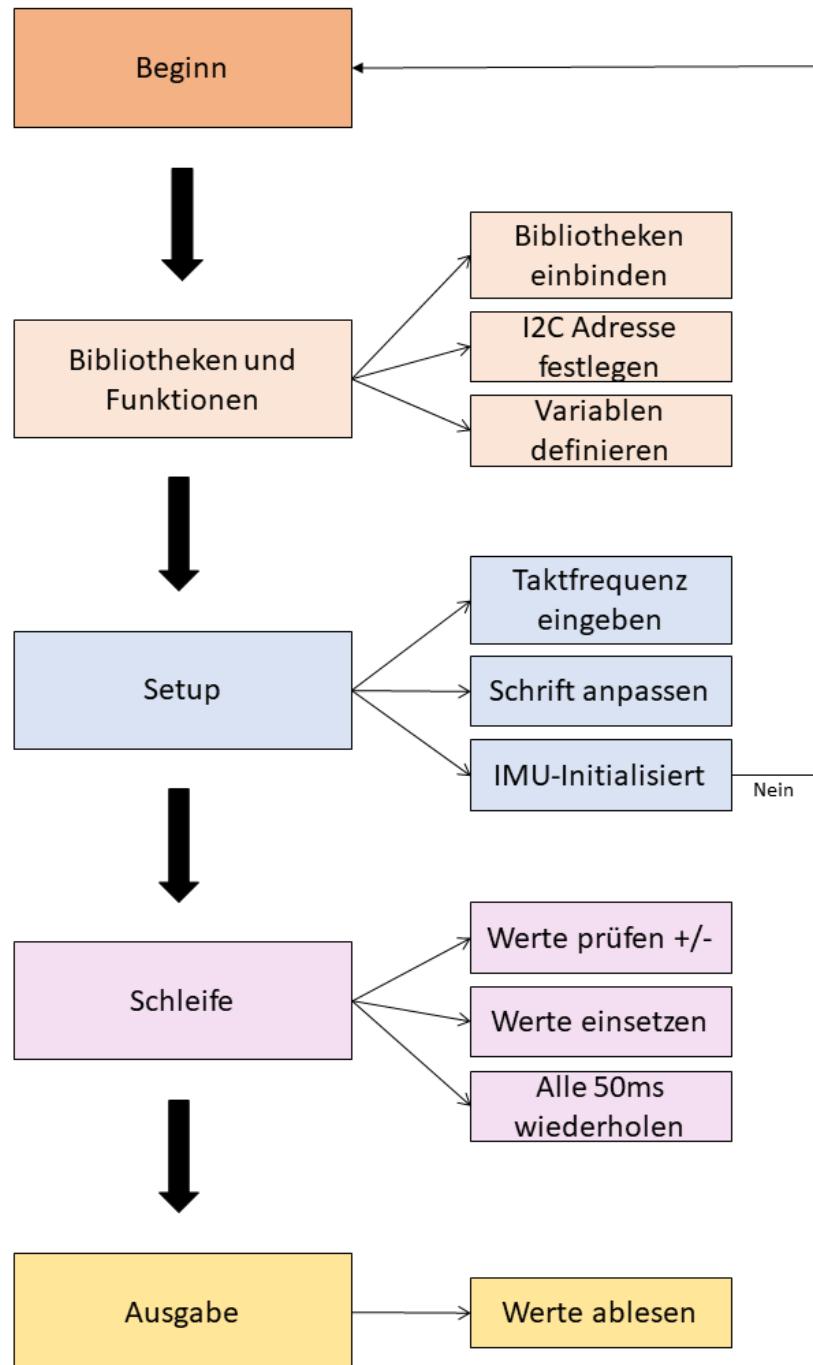


Figure 8.3.: Der Programmablaufplan

```

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(" X 0 Grad");
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("„Y„0„Grad");
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("„X„0„Grad");
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("„Y„0„Grad");
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("„X„0„Grad");
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("„Y„0„Grad");
25        }
26        else
27        {
28            oled.print("„Y„");
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("X0Grad");
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("Y0Grad");
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.

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

8.4.3. 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 8.4.: Anzeige des Arduino OLED Displays sobald der Arduino eingeschaltet ist

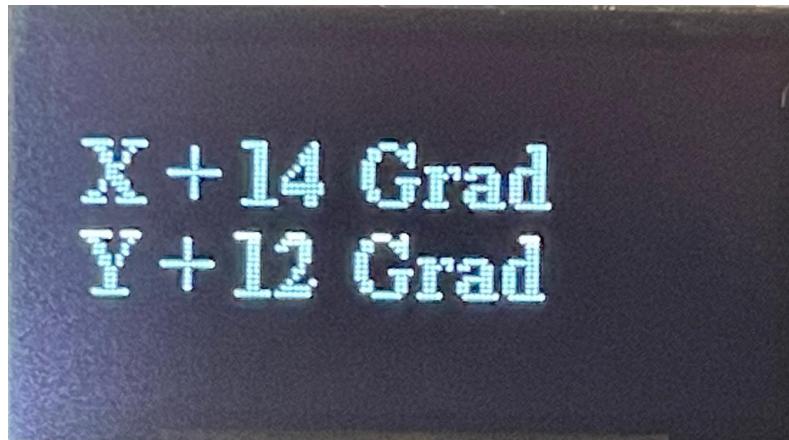


Figure 8.5.: Ausgabe von Messdaten

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.

8.5. 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.

8.6. 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.

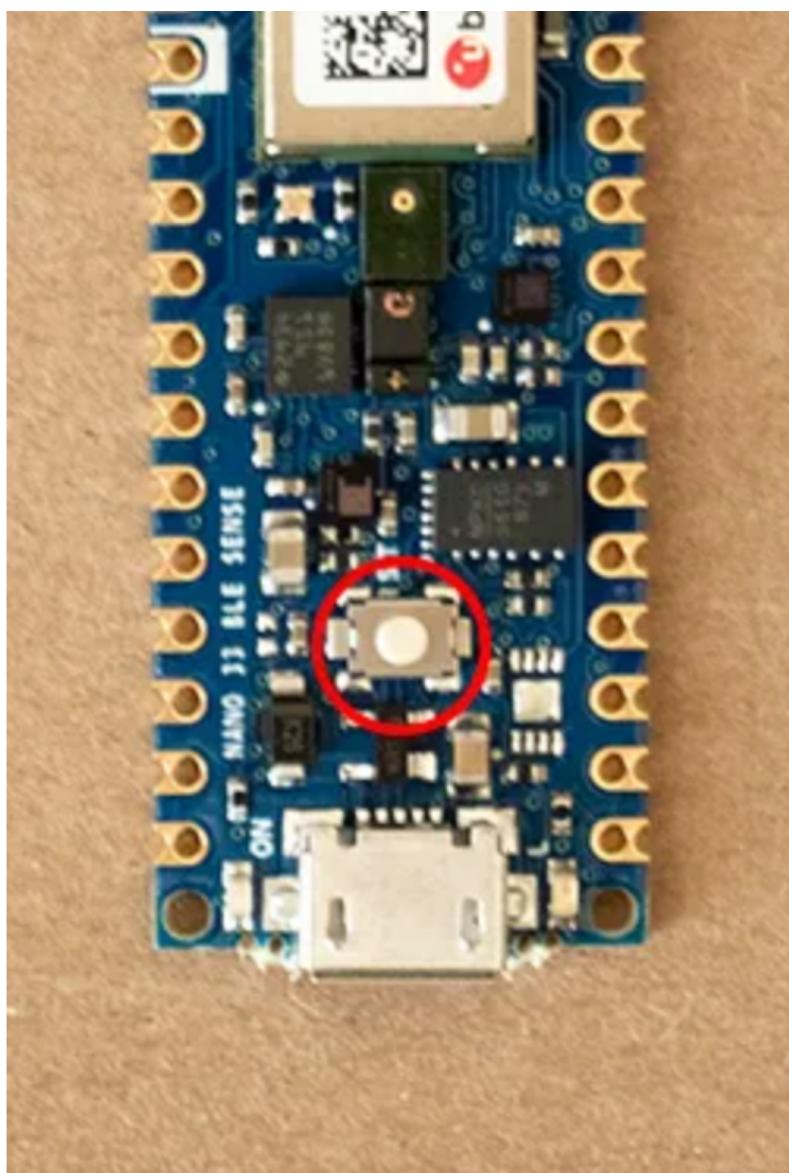


Figure 8.6.: Der Reset Button des Arduino

9. Sensormodul APDS9960

9.1. Der 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]. In diesem Projekt wurde nur mit dem Farbsensor gearbeitet. Daher findet auch die detaillierte Betrachtung des Sensors in Bezug auf die Anwendung als Farbsensor statt. Ein sehr bekannter Anwendungsfall für diesen Sensor ist das automatische Ausschalten des Smartphone-Displays beim Anlegen des Smartphones an das Ohr. Hier erkennt der Sensor mithilfe des Abstandssensors, dass etwas auf oder am Bildschirm anliegt. Bei der Farberkennung allerdings misst der Sensor die Intensität der roten, grünen und blauen (RGB) Farbe und gibt simultan einen 16-bit Wert zurück. Insgesamt kommt man damit 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. Die Infrarot (IR)-Light Emitting Diode (LED) zur Gestenerkennung und Abstandsmessung benötigt eine Spannung von 3,0 V - 4,5 V. Um die verarbeiteten Daten des APDS-9960 an den Arduino zu übergeben wird eine Inter-Integrated Circuit (I²C)-Schnittstelle genutzt [Ava15]. Auf die Funktionsweise dieser Kommunikation wird in einem späteren Abschnitt eingegangen.

9.2. OLED-Display DEBO OLED2 0,96

Als Ausgabe der Messwerte dient ein Organic Light Emitting Diode (OLED)-Display. Er besitzt eine Displaygröße von 0,96 Zoll und eine Modulgröße von 27 mm × 27 mm × 11 mm. Die Auflösung beträgt 128 × 64 Bildpunkte. Des Weiteren wird eine 3 V - 5 V Gleichstromspannung benötigt. Anzuschließen ist der Display über einen 4 Pol. Hierzu wird das I²C-Protokoll genutzt. Die Kommunikation findet über die PIN's SDA und SCL statt. Die benötigte Spannung liegt über dem PIN "Voltage at the comment collector" (VCC) und dem PIN "Ground" (GND) an. Mithilfe eines I²C-Scanners konnte die vorher angegebene I²C-Adresse 0x3C bestätigt werden. Als Controller/Treiber wird das Modul SSD1306 genutzt. Das Besondere an der OLED-Technik ist, dass keine Hintergrundbeleuchtung nötig ist. Jeder Bildpunkt leuchtet selbst. Somit ist stets ein kontrastreiches Bild gegeben und der Stromverbrauch bleibt niedrig [Sim].

9.3. RGB-LED

Die RGB-LED wurde während des laufenden Projekts integriert. Es war aufgefallen, dass der Farbsensor APDS-9960 in der Dämmerung oder nachts schlechte Messergeb-

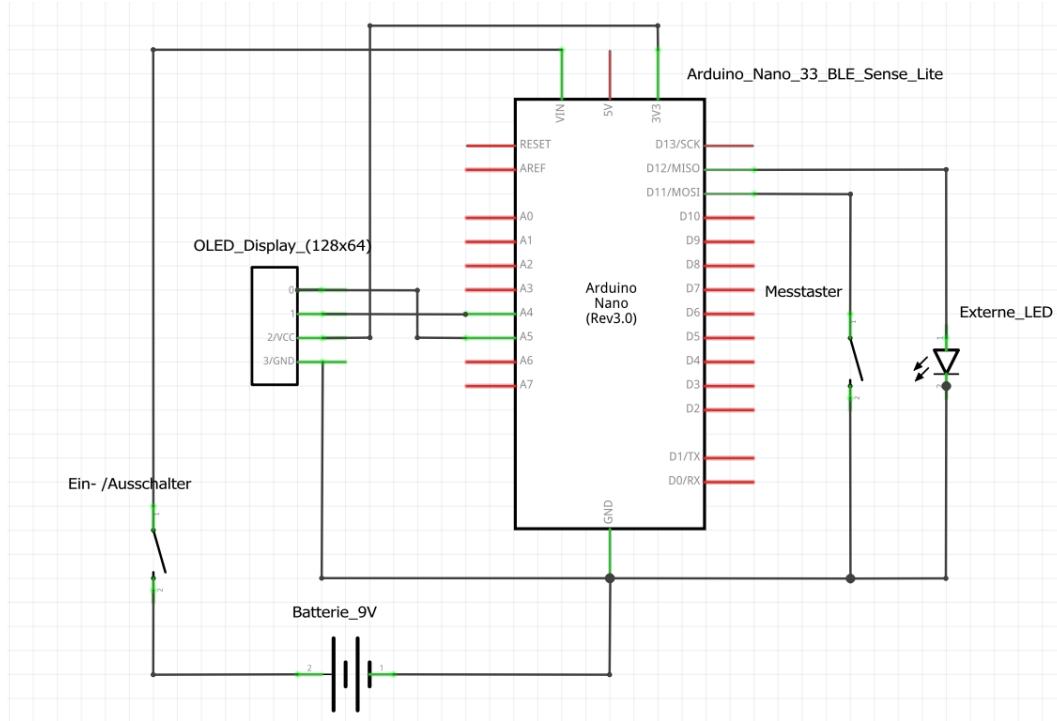


Figure 9.1.: Schaltplan Farberkennungsautomat

nisse liefert ohne zusätzliche Lichtquelle. Die RGB-LED ist eine Lichtquelle mit 3 verschiedenenfarbigen LED's, rot, blau und grün. Als Spannungsversorgung werden mindestens 3 V und maximal 3,6 V benötigt. Der Arduino Nano 33 BLE Sense Lite liefert 3,3 V, daher ist der Betrieb der LED ohne zusätzlichen Widerstand möglich. Die Helligkeit der LED variiert zwischen 5000-6000 Millicandela (mcd) bei einem Strom von 20 mA. Die Farbtemperatur ist steuerbar und kann in einem Bereich von 5000-6000 Kelvin eingestellt werden. Gelagert und benutzt werden sollte die LED zwischen -40°C und +85°C.

9.4. Schaltplan

10. Softwarebeschreibung

Die Software verarbeitet die eingehenden Werte des Farbsensors, erkennt das Drücken des Tasters, steuert das OLED-Display, schaltet die LED ein und erzeugt eine Grundlage des Debugging mithilfe des seriellen Monitors. Ferner kann das Debuggen allerdings auch im Source Code in der IDE stattfinden. Grundsätzlich wird als Programmiersprache für den Arduino Nano 33 BLE Sense Lite C oder C++ verwendet. Arduino bietet als Entwicklerumgebung kostenlos die Arduino IDE an. Möglich wäre allerdings auch eine Programmierung mit einer anderen Entwicklerumgebung wie beispielsweise Qt.

10.1. Softwarestruktur

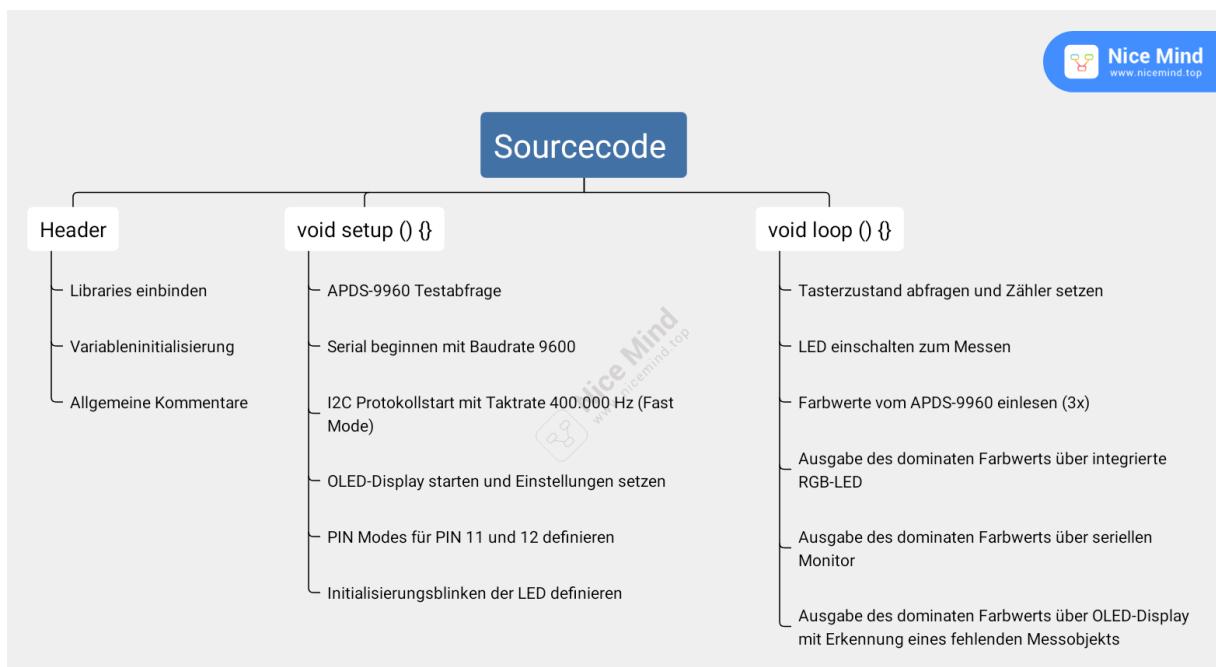


Figure 10.1.: Software-Strukturplan

Der in Abbildung 4.1 dargestellte Software-Strukturplan beschreibt die Struktur der Software. Grundsätzlich besitzt ein Arduino-Sketch drei Teile. Der erste Teil ist der Header. Hier werden Bibliotheken eingebunden, Variablen initialisiert und allgemeine Kommentare eingefügt. Der zweite Teil ist die `void setup()`-Funktion. Hier können auch Variablen erstellt und initialisiert werden. Außerdem können den Pins verschiedene Modi zugewiesen werden. Die `void setup()`-Funktion läuft nach dem Starten oder einem Reset einmal durch [Ardg]. Der nächste Abschnitt eines Arduino-Sketches wird `loop()` genannt und ist auch eine Funktion vom Datentyp `void`. Die `loop()`-Funktion wird konsequent wiederholt und ist die Schleife in der das eigentliche Programm geschrieben wird [Ardf].

10.2. Einrichtung der Arduino IDE

Bevor der eigene Sketch erstellt werden kann, muss die Arduino IDE eingerichtet werden. Während der Programmierung wurde eine Aktualisierung der Arduino IDE vorgenommen. Zum Ende des Projektes wurde dementsprechend mit der Version 2.1, statt der 2.0 gearbeitet. Der Download erfolgt über die offizielle Arduino-Website <https://www.arduino.cc/en/software>. Dort können verschiedene Versionen, Betriebssysteme und Dateiformate gewählt werden. Nach dem Installieren muss das richtige Board ausgewählt und die richtigen Bibliotheken installiert werden. Auf das Vorgehen und die Wahl des korrekten Board wurde bereits im Dokument "Hard- und Softwarebeschreibung mit Materialliste" eingegangen: "Um ein problemloses Kompilieren des erstellten Codes hinsichtlich des „Board Core“ (Arduino Nano, Arduino Uno, Arduino Micro, etc.) gewährleisten zu können, muss über den „Board Manager“ das passende Board installiert werden. In diesem Fall ist es das Board „Arduino Mbed OS Nano Boards“ von Arduino mit der aktuellen Version 4.0.2. Ausgewählt werden kann das passende Board dann unter „Tools-Board-Arduino Mbed OS Nano Boards“. Hier wird dann das Board „Arduino Nano BLE 33“ ausgewählt."

10.2.1. Bibliotheken einfügen

Bibliotheken stellen einen wichtigen Baustein in der Arduinoprogrammierung dar. Wie in anderen Programmiersprachen helfen Bibliotheken beim Aufrufen von häufig genutzten Funktionen [MW23]. Im Fall des genutzten APDS-9960 Sensors wären das Aufrufen der Farberkennung, des Abstandes oder der Gestenerkennung eben jene häufig genutzten Code-Blöcke. Nach dem Erzeugen eines Klassenobjektes können Funktionen der Klasse mit dem Objekt aufgerufen werden.

```
SSD1306AsciiWire oled ;  
  
oled.setFont(System5x7);  
oled.setCursor(0, 40);  
oled.println("INITIALISIERUNG!\n");
```

In diesem Beispiel wird zuerst das Objekt *oled*, der Klasse *SSD1306AsciiWire* erstellt und anschließend mit verschiedenen Befehlen *setFont*, *setCursor*, *println* auf die Funktionen der Klasse *SSD1306AsciiWire* zugegriffen. Dieses Beispiel bezieht sich nun auf die Ausgabe und Kommunikation mit dem OLED. Die wichtigere Bibliothek für dieses Projekt ist aber die APDS-9960 Bibliothek, *Arduino_APDS9960*. Diese erlaubt es dem Nutzer 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 APDS-9960 Modul über eine interne I2C-Schnittstelle statt [Ava15]. Als zweite Bibliothek wird *SSD1306AsciiWire* genutzt. Dabei bietet *SSD1306AsciiWire* die Möglichkeit wenig Speicherplatz für die Nutzung des OLED-Displays zu verbrauchen. Dies wird durch eine reine Textausgabe auf dem Display erzeugt. Grundsätzlich ist diese Bibliothek eine Möglichkeit mit einem SSD1306-Controller zu agieren. Ferner steht das *Wire* für die Verbindungsart zwischen dem Arduino und dem OLED-Display. In diesem Fall handelt es sich um eine Kabelverbindung mittels I²C-Protokoll. . Andere Bibliotheken die mit dem SSD1306-Controller arbeiten wie *U8g2* oder *Adafruit SSD1306* benötigen mehr Speicherplatz [Ardb].

10.2.2. Serielle Kommunikation

Als serielle Kommunikation oder serielle Datenübertragung wird zum Beispiel die Verbindung zwischen einem Mikrocontroller und dem PC oder anderen Peripheriegeräten bezeichnet. Der PC kommuniziert mit dem Mikrocontroller über den

sogenannten COM-Port. Dabei handelt es sich um das Universal Asynchronous Receiver Transmitter (UART)-Protokoll. Ein Computer oder auch ein Mikrocontroller versteht nur Ein und Aus bzw. 1 und 0. Im Fall des UART-Protokolls wird 3,3 V Signal (HIGH/ EIN/ 1) versendet [GW22]. Ein weiteres Protokoll, welches eine serielle Datenübertragung bereitstellt, ist das I²C-Protokoll. Die Funktionsweise dieses Protokolls wurde bereits im Abschnitt Hardwarebeschreibung behandelt.

10.3. Header

Der Header oder auch Kopfzeile des Sketchs beinhaltet die benötigten Bibliotheken. In diesem Fall sind das folgende:

```
#include <Arduino_APDS9960.h>
#include "SSD1306AsciiWire.h"
```

Es kann auf alle Funktionen durch das Aufrufen von *APDS*. und dem gewünschten Befehl aus der Bibliothek zugegriffen werden [Ava15].

```
if (!APDS.begin()) {
    Serial.println("Error initializing APDS9960 sensor.");
}
// oder
while (!APDS.colorAvailable()) {
    delay(5);
}
```

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.

```
#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.

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

10.4. 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.

```
Serial.begin(9600);
pinMode(buttonPin, INPUT);
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 [Ardh]. Als nächstes wird in das APDS-9960 Modul gestartet und geprüft ob eine Antwort zurückgegeben wird. Falls dies nicht der Fall ist, würde eine Fehlermeldung im seriellen Monitor erscheinen.

```
if (!APDS.begin()) {
    Serial.println("Error initializing APDS9960 sensor.");
}
```

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

```
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 [Ardb].

```
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();
```

10.5. void loop{}

Der Hauptteil des Programms liegt in der *loop()*-Funktion. Der erste Schritt ist das Ausgeben des Textes *READY!* um dem Benutzer zu sagen, dass er mit dem Messvorgang starten darf. Anschließend werden der Farb- und Abstandssensor abgefragt, ob beide jeweils einen Messwert generieren können.

```
oled.println("READY!");
```

```

while (!APDS.colorAvailable()) {
    delay(5);
}

while (!APDS.proximityAvailable()) {
    delay(5);
}

```

Nun werden die Variablen für die Farbwerte r , g , b und für den Taster $tasterCount$ als "integer" deklariert und die Tastervariable initialisiert. Zusätzlich wird die "boolean" Variable $zustandTaster$ deklariert und mit dem Wert der Funktion $digitalRead(TasterPin)$ initialisiert. $digitalRead$ gibt den Zustand des Tasters als HIGH(Nicht gedrückt) oder LOW(Gedrückt) zurück. Sollte der Taster gedrückt werden wird die Bedingung der **if**-Verzweigung im Anschluss wahr und die Variable $tasterCount$ erhöht sich um 1.

```

int r, g, b, tasterCount{0};

bool zustandTaster = digitalRead(TasterPin);

if (zustandTaster == 0) {
    tasterCount = 1;
}

```

Der nächste Schritt beinhaltet dann den Messvorgang und die Ausgabe des Messergebnisses bzw. den Hinweis an den Messenden eine Anpassung des Messvorgangs vorzunehmen, falls sich kein Objekt vor dem Sensor befindet oder der Abstand zum Objekt nicht korrekt ist. Starten tut dieser Vorgang mit der Bedingung, dass der Abstandswert ungleich -1 ist. -1 wäre in diesem Fall ein Fehler in der Abstandsmessung. Der Abstandswert wird dann im seriellen Monitor ausgegeben. Diese Ausgabe ist eine reine Service-/Debugg-Ausgabe. Ist der Abstand korrekt wird in die zweite **if**-Verzweigung gesprungen. Hier ist der erste Schritt das Löschen des Displays und das Einschalten der externen RGB-LED um optimale Lichtverhältnisse für die Farbmessung zu schaffen. Im Anschluss wird dreimal die Farbe gemessen und der Wert in den Variablen r , g und b gespeichert. Die erste Ausgabe des Ergebnisses erfolgt über die interne RGB-LED. Hier wird mit weiteren **if**-Verzweigungen geschaut welche der drei Farben rot, grün und blau dominiert. Diese Farbe wird mit der RGB-LED angezeigt. Die zweite Ausgabe erfolgt über den seriellen Monitor. Hier werden dann die gemessenen Farbwerte ausgegeben. Die dritte und für den Nutzer entscheidende Ausgabe erfolgt über das OLED-Display mit einem Text, welche Farbe die dominierende Farbe ist. Im Anschluss wird 4 s gewartet damit sich der Nutzer das Ergebnisse notieren oder merken kann. Falls der Abstand zu Beginn des Messvorgangs nicht korrekt war, wird ein Text auf dem Display ausgegeben der dem Nutzer diese Information mitteilt und Verbesserungsvorschläge macht.

```

if (tasterCount == 1) {

    int proximity = APDS.readProximity();

    if (proximity != -1) {
        Serial.print("Abstand: ");
        Serial.print(proximity);

        if (proximity < 20) {

            oled.clear();
        }
    }
}

```

```

digitalWrite(LedPin, HIGH);

for(int i=0;i<3;i++){
    APDS.readColor(r, g, b);
    delay(1500);
}

if (r > g & r > b) {
    digitalWrite(LED_R, LOW);
    digitalWrite(LED_G, HIGH);
    digitalWrite(LED_B, HIGH);
} else if (g > r & g > b) {
    digitalWrite(LED_G, LOW);
    digitalWrite(LED_R, HIGH);
    digitalWrite(LED_B, HIGH);
} else if (b > g & b > r) {
    digitalWrite(LED_B, LOW);
    digitalWrite(LED_R, HIGH);
    digitalWrite(LED_G, HIGH);
} else {
    digitalWrite(LED_R, HIGH);
    digitalWrite(LED_G, HIGH);
    digitalWrite(LED_B, HIGH);
}

Serial.print("Red_light = ");
Serial.println(r);
Serial.print("Green_light = ");
Serial.println(g);
Serial.print("Blue_light = ");
Serial.println(b);
Serial.println();

if (r > g & r > b) {
    oled.println("Das Objekt ist rot!");
} else if (g > r & g > b) {
    oled.println("Das Objekt ist gruen!");
} else if (b > g & b > r) {
    oled.println("Das Objekt ist blau!");
}

delay(4000);
oled.clear();
}

else {
    oled.clear();
    oled.println("Kein Objekt vorhanden!\n");
}

```

```

        oled . println ( " Halten \u00e4 Sie \u00e4 ein \n Objekt \u00e4 vor
den \n Sensor \u00e4 \u00f6 noder \u00e4 veraendern \u00e4 Sie \u00e4 ndie \u00e4 Position ! "
);
        digitalWrite (LEDR, HIGH);
        digitalWrite (LEDG, HIGH);
        digitalWrite (LEDB, HIGH);
        delay (4000);
        oled . clear ();
    }
} else {
    oled . println ( " Abstandsmessung \u00e4 fehlgeschlagen ! " );
}
}
}

```

Falls der Taster nicht gedrückt wird, bleiben die externe und interne RGB-LED ausgeschaltet. Als letzte Aktion in der *loop()*-Funktion wird die Variable *zustandTaster* auf 0 zurückgesetzt.

```

else {
    digitalWrite (LedPin , LOW);
    oled . println ( " READY! " );
    digitalWrite (LEDR, HIGH);
    digitalWrite (LEDG, HIGH);
    digitalWrite (LEDB, HIGH);
}
zustandTaster = 0;
}

```

Der gesamte Programmcode befindet sich im Verzeichnis *Programmcode*.

11. Sensor BME280

11.1. 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 (Abbildung 11.1). [Funak]

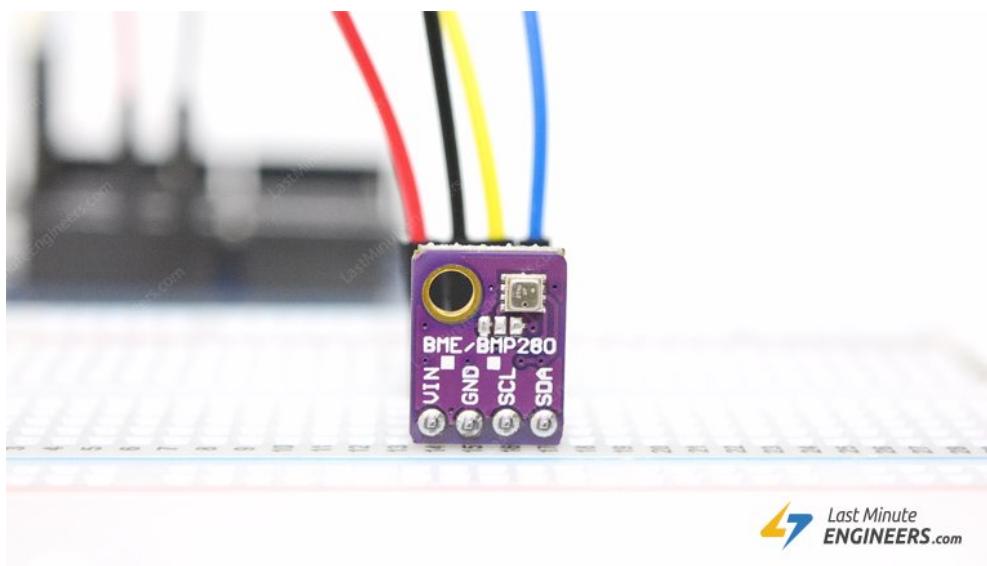


Figure 11.1.: Sensor BME280
[Las]

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 +/- 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 µA für die Messung von Feuchtigkeit und Temperatur, 2,8 µA für die Messung von Luftdruck und Temperatur und 3,6 µA 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]

11.2. 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 13.11 abgebildet.

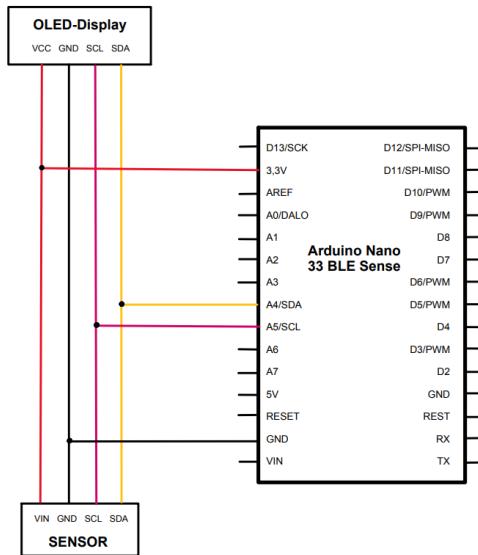


Figure 11.2.: Stationärer Aufbau der Wetterstation
[Arda]

11.2.1. 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]

11.2.2. 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 \enquote{Hello

World}, welcher unter Datei -> SSD1206Ascii -> HelloWorldWire zu finden ist (siehe Abbildung \ref{fig:TestprogrammDisplay}).

```
\begin{figure} \centering \includegraphics[width=1\textwidth] {Images/Testprogramm Display} \captionof{figure}{Pfad des Testprogramms.} \label{fig:TestprogrammDisplay}
```

Das Testprogramm \enquote{Hello World} (siehe Seite \pageref{Test1}) wird dafür benutzt, um den Text Hello World auf dem Display auszugeben (siehe Abbildung \ref{fig:ErsteAusgabeDisplay}). 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.

```
\bigskip \newpage \noindent\rule[1ex]{\textwidth}{1pt}
```

```
\lstset{numbers=left, numberstyle=\tiny, numbersep=3pt} \lstset{language=Perl}
```

```
\begin{lstlisting} #include <Wire.h> #include "SSD1306Ascii.h" #include "SSD1306AsciiWire.h"
```

```
#define I2C_ADDRESS 0x3c
```

```
SSD1306AsciiWire oled;
```

```
void setup() { Wire.begin(); Wire.setClock(400000L); oled.begin(&Adafruit128x64,
```

```
I2C_ADDRESS); }
```

```
void loop() { oled.setFont(System5x7); oled.clear(); oled.println("Viel"); oled.print("Erfolg!!!");
```

```
delay(2000); } \end{lstlisting}\label{Test1} \noindent\rule[1ex]{\textwidth}{1pt}
```

Nachdem der Quellcode kompiliert und an den Arduino geschickt wurde, wurde auf dem Display der Text Viel Erfolg ausgegeben (siehe Abbildung \ref{fig:ErsteAusgabeDisplay}). Hiermit ist sichergestellt worden, dass die Entwicklungsumgebung und der Compiler korrekt funktionieren. \cite{Funduino:2023}

```
\begin{figure} \centering \includegraphics[width=0.5\textwidth] {Images/Erste Ausgabe Display} \captionof{figure}{Erste Ausgabe Display} \label{fig:ErsteAusgabeDisplay}
```

```
\cite{Funduino:2023} \end{figure}
```


12. Servomotor JAMRA 033212

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

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

WS:Quelle fehlt!

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

Motor: Als Aktuator ist im 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

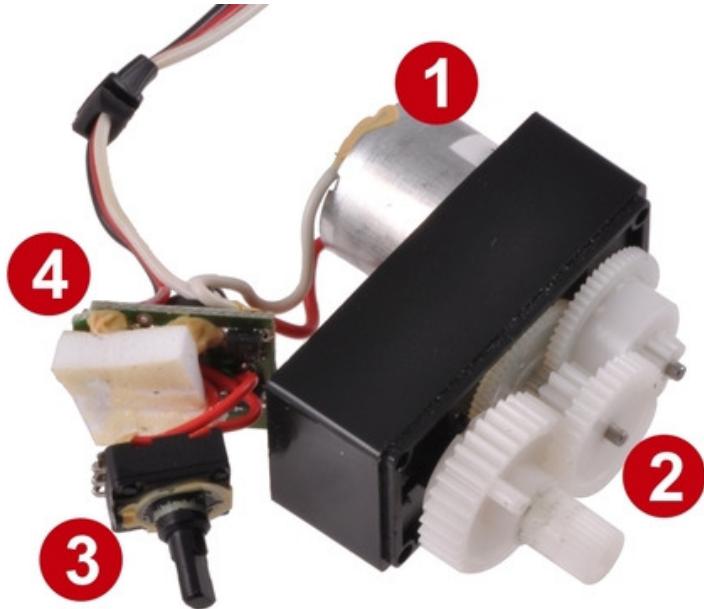


Figure 12.1.: Aufbau eines Servomotors Quelle

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.

12.1. 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.

12.2. Schaltplan

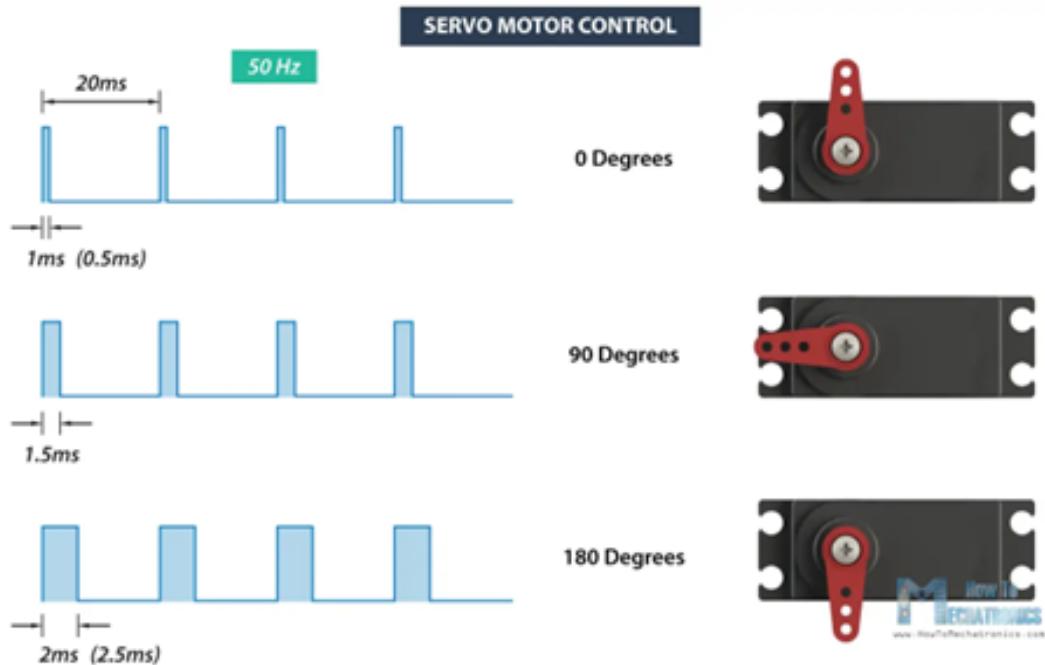


Figure 12.2.: PWM am Servomotor Quelle

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

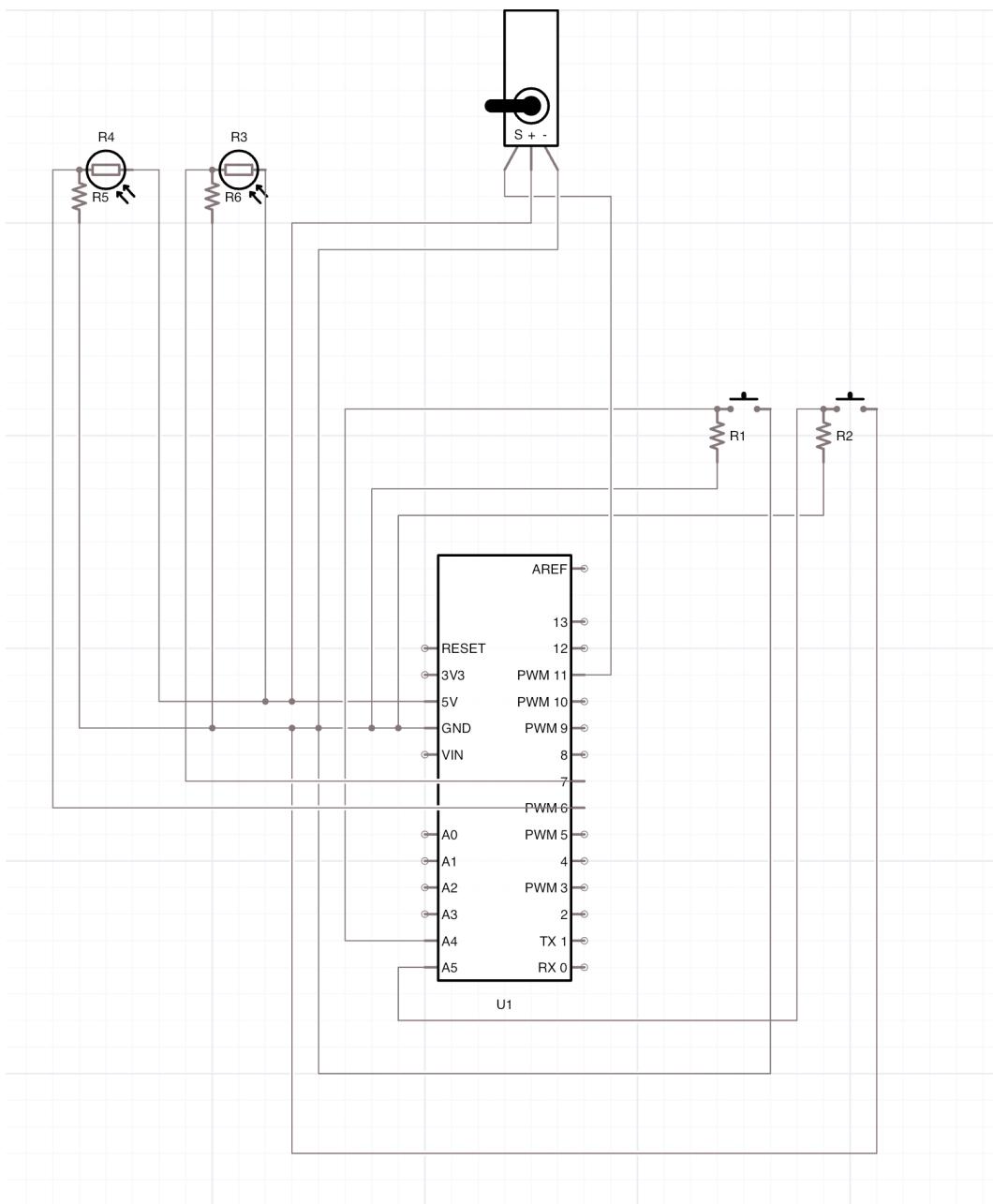


Figure 12.4.: Schaltplan zum Anschluss eines Servomotors JAMRA 033212

13. OLED-Display

13.1. OLED

Im Folgenden wird das Display DEBO OLED2 0.96 beschrieben. Dieses kleine Display mit einem schwarzen Hintergrund und blauer Anzeigefarbe lässt sich mit I²C ansteuern. Die hohe Auflösung bietet ein scharfes Bild.[Sim]

Technische Daten:

- Auflösung: 128 × 64 Pixel
- Hintergrundfarbe: Schwarz
- Anzeigefarbe: blau
- Maße: 27 mm × 27 mm × 11 mm
- Anschluss: 4-polig Anschluss
- Interface: I²C
- SSD-Controller: SSD1306
- Spannungsversorgung: 3,3 - 5 V

13.2. Anschluss

In der Abbildung 13.1 ist einer Schaltplan zu sehen, in dem das Display verwendet wird. Die Masse des OLED-Displays ist in der Farbe schwarz gekennzeichnet und ist über das Arduino Shield an den Masse Port des Arduino verbunden. Die Spannungsversorgung mit 3,3V für das OLED-Display ist in rot gekennzeichnet. Die beiden Pins für die Datenübertragung und für das OLED-Display gehen in die vorgesehenen SDA und SCL Pins am Arduino.

WS:Farbe und Numme
WS:Nur Display
darstellen

13.3. Programmierung

13.3.1. [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.

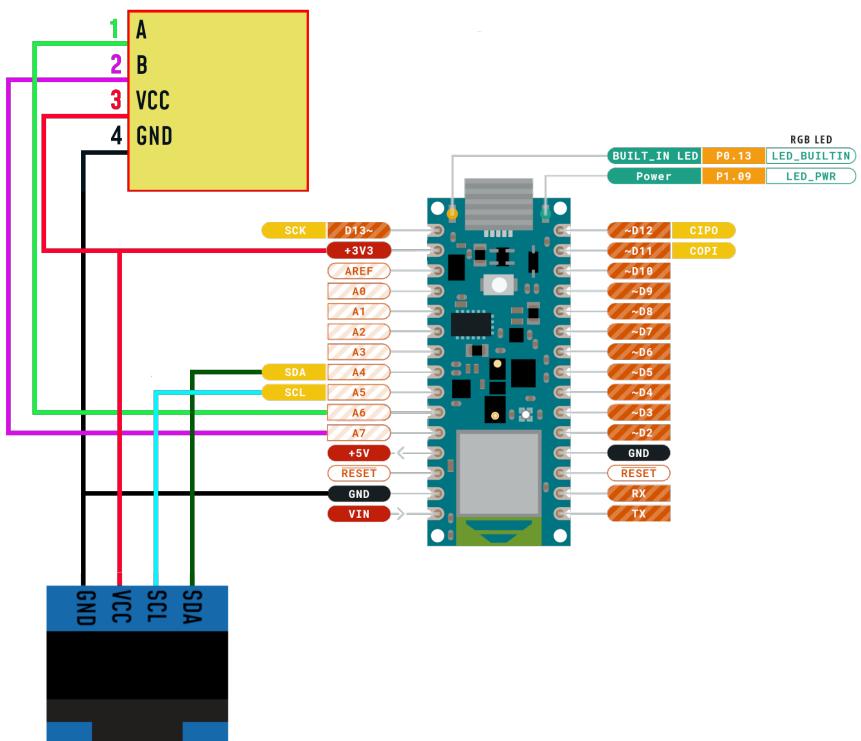


Figure 13.1.: Gesamter Schaltplan

13.3.2. 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.

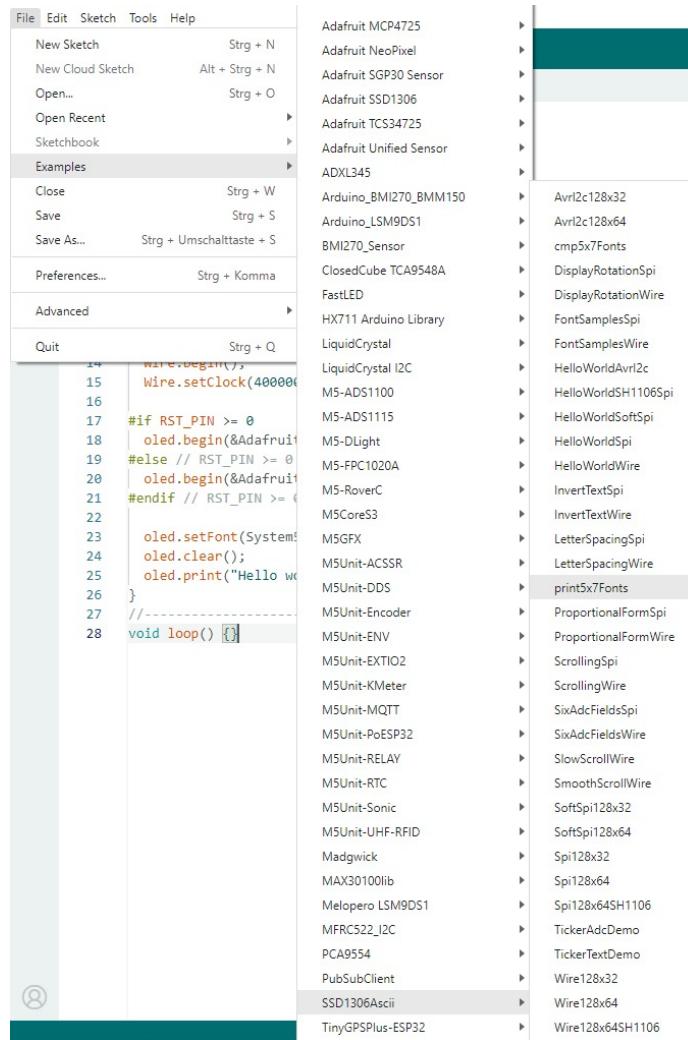


Figure 13.2.: Beispiele in der OLED Bibliothek

13.3.3. 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.

13.4. Software

13.4.1. Verwendete Bibliotheken

Zur Ansteuerung des Displays werden folgende Bibliotheken verwendet:

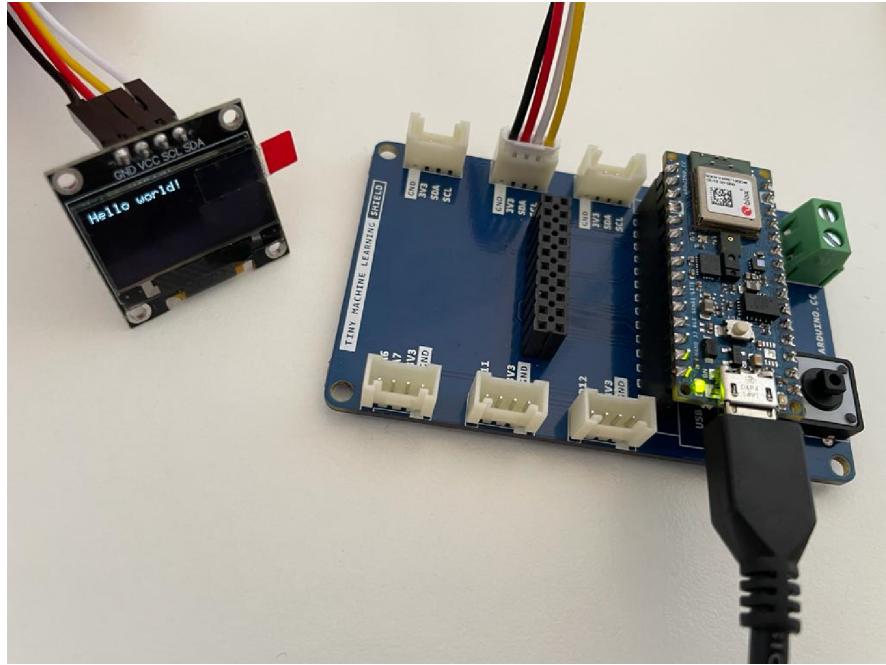


Figure 13.3.: Testausgabe des OLED Display

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

VS:Ist dies kompatibel?
Was ist damit möglich?

13.4.2. 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
```

13.4.3. 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

13.4.4. 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 13.4.: Code Einlesen/Zählen

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

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

Figure 13.5.: Code Messungssteuerung

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

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

13.4.5. 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.

13.4.6. 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.

13.4.7. 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 „ $20 * \log_{10}(\text{Voltage} / \text{Vrms}) + \text{sensitivity}$ “ berechnet [Quelle der Formel: PDF Decibels Formula https://physicscourses.colorado.edu/phys3330/phys3330_sp19/resources/Decibels_Phys_3330.pdf University of Colorado System], wobei „Voltage“ der berechnete „maxSampleVoltage“ ist, und „sensitivity“ die Empfindlichkeit des Mikrofons in dBV (Dezibel-Volt) ist. „Vrms“ ist die RMS-Spannung (Root Mean Square), die einem Schalldruckpegel von 94 dB SPL entspricht. Dieser Wert wurde experimentell ermittelt.

13.4.8. 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.

13.5. 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 Das Display verfügt über vier Pins, welche in der Abbildung 13.10 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]

13.6. 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 13.11 abgebildet.

13.7. 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.

13.7.1. Bibliothek Wire.h

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

13.7.2. Bibliothek SSD1306Ascii.h für das Testprogramm

Bei der Bibliothek SSD1306Ascii.h handelt es sich um eine benutzerdefinierte Bibliothek, die ähnlich zu der Adafruit Bibliothek SSD1306 ist. Beide sind für die Ansteuerung von OLED-Displays notwendig und basieren auf den Controller-Chip SSD1306. Sie ermöglichen das einfache Schreiben von den Text, Zeichen von Formen und Anzeigen von Bitmap-Bildern auf dem Display. Um die Bibliothek SSD1306Ascii.h zu verwenden, muss sie erst als ZIP-Datei heruntergeladen und anschließend von dem Arduino Library Manager installiert werden. Mit Hilfe von `#include<SSD1306Ascii.h>` kann man sie in dem Arduino-Code einbinden. [Fun]

13.7.3. Testen des OLED-Displays

Für die Programmierung des Displays gibt es viele Möglichkeiten. Da es viele Libraries gibt, muss zuerst einmal überlegt werden, welche verwendet werden sollen. Um einen ersten Eindruck über die Programmierung des Displays zu bekommen, wurde zunächst ein Beispieldsketch getestet. Es handelt sich hier um den Sketch [Hello World](#), welcher unter Datei -> SSD1206Ascii -> HelloWorldWire zu finden ist (siehe Abbildung 13.12).

Das Testprogramm `Hello World` (siehe Seite 105) wird dafür benutzt, um den Text Hello World auf dem Display auszugeben (siehe Abbildung 13.13). 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.

```
1 #include <Wire.h>
2 #include "SSD1306Ascii.h"
3 #include "SSD1306AsciiWire.h"
4 #define I2C_ADDRESS 0x3c
5
6
7 SSD1306AsciiWire oled;
8
9 void setup() {
10     Wire.begin();
11     Wire.setClock(400000L);
12     oled.begin(&Adafruit128x64, I2C_ADDRESS);
13 }
14
15 void loop()
16 {
17     oled.setFont(System5x7);
18     oled.clear();
19     oled.println("Viel");
20     oled.print("Erfolg !!! ");
21     delay(2000);
22 }
```

Nachdem der Quellcode kompiliert und an den Arduino geschickt wurde, wurde auf dem Display der Text Viel Erfolg ausgegeben (siehe Abbildung 13.13). Hiermit ist sichergestellt worden, dass die Entwicklungsumgebung und der Compiler korrekt funktionieren. [Fun]

```

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 13.6.: 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 13.7.: 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 13.8.: Code Umrechnung Mikrofonsignal

```

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

```

Figure 13.9.: Code OLED-Aktualisierung

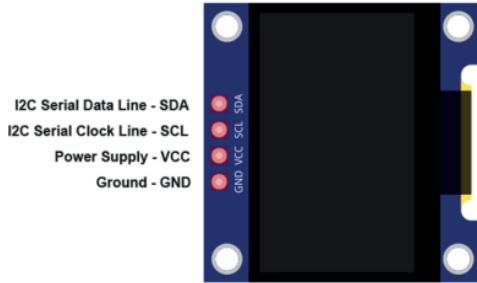
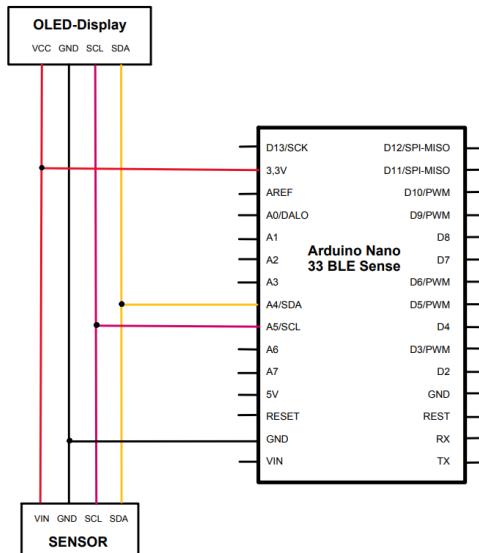


Figure 13.10.: Pins des OLED-Displays.

Figure 13.11.: Stationärer Aufbau der Wetterstation
[Arda]

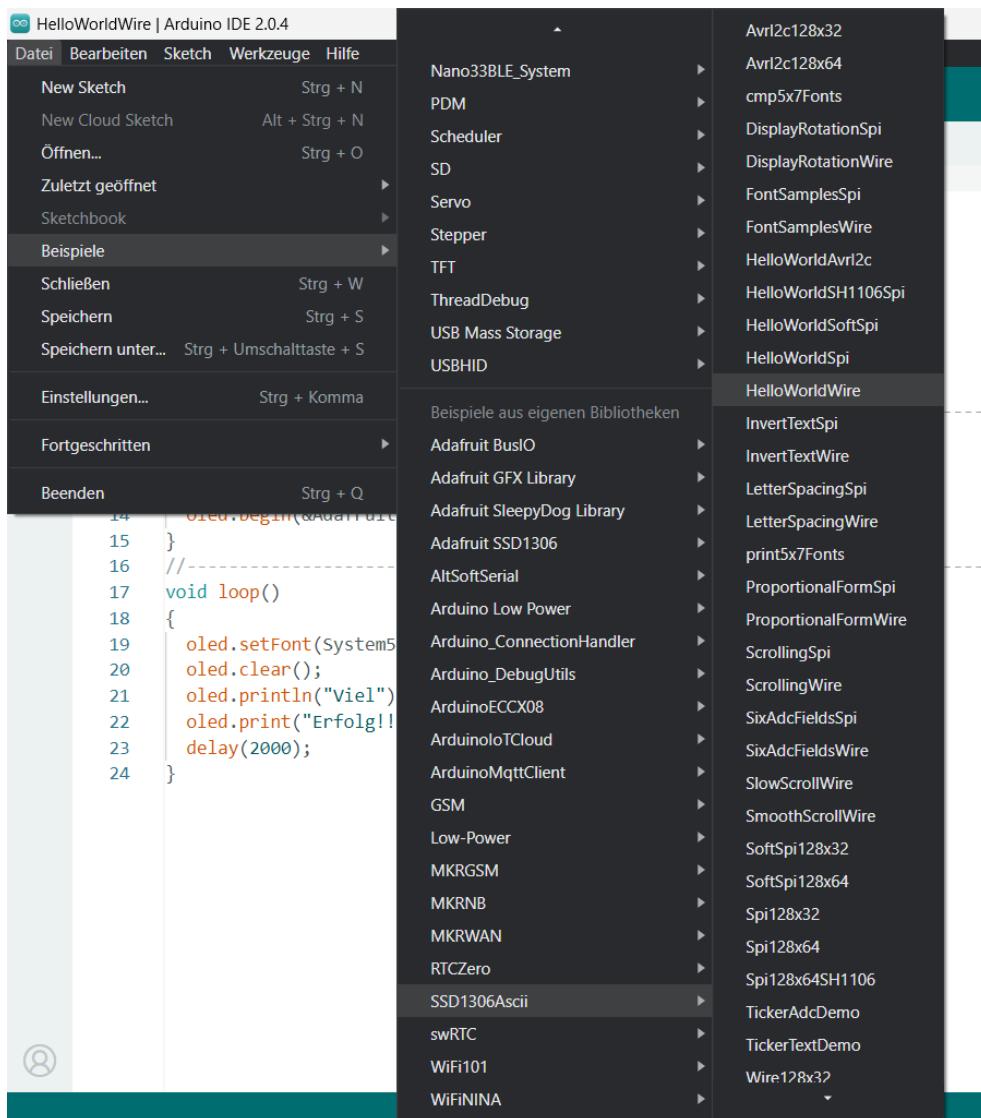


Figure 13.12.: Pfad des Testprogramms.



Figure 13.13.: Erste Ausgabe Display
[Funb]

14. 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/>

14.1. 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.

14.2. Supplies

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

14.3. 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

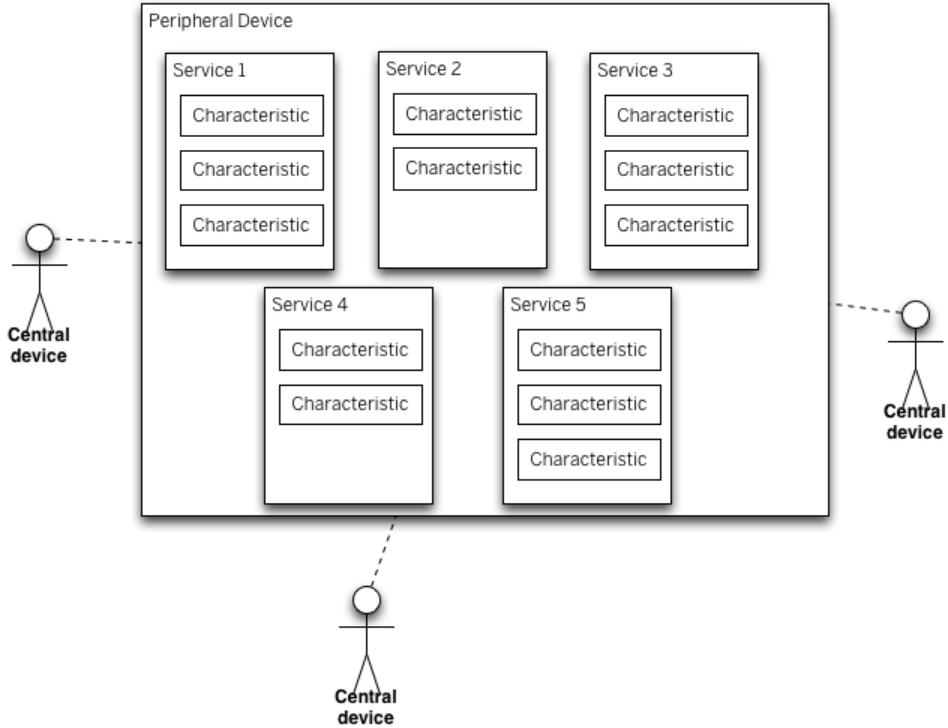


Figure 14.1.: BLE Devices – Peripheral and Central Devices

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 of a wide range of BLE boards see Bluetooth Low Energy (BLE) made simple with pfodApp. Here we will be using a BLE connection for NANO 33 BLE, Sense and IoT and a WiFi connection for the IoT.

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

pfodDesigner generates simple menus and charts, however you can also program custom graphical interfaces in your Arduino sketch. Above is an example of slider control 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.

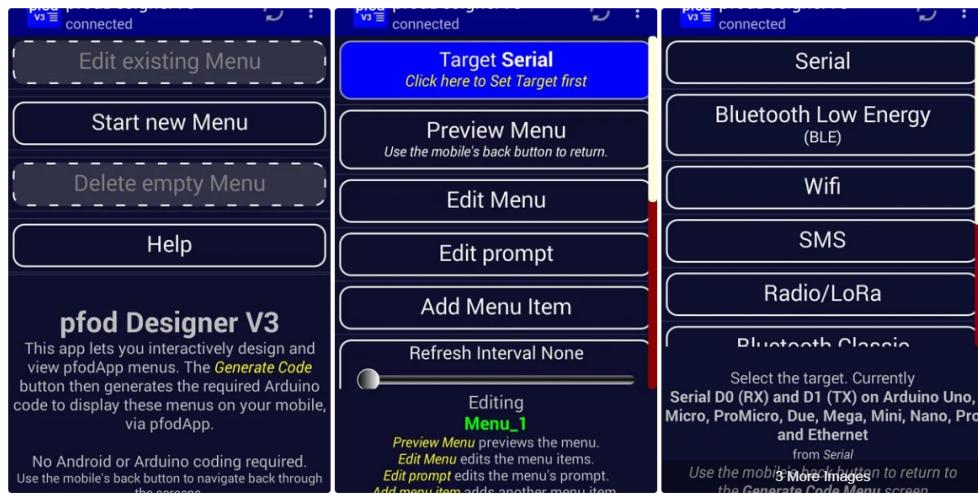


Figure 14.2.: Step 2: Creating the Custom Android Menus and Generating the Code

14.4. 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.

14.5. 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](#)

14.6. Arduino BLE Example – Explained Step by Step

14.6.1. 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.

14.6.2. Arduino BLE – Bluetooth Low Energy Introduction

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

WS:cite Arduino have a wonderful introduction to BLE but here in this post, I will give you a brief introduction for you to get started with BLE communication.

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

- The Peripheral Device
- The Central Device

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

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

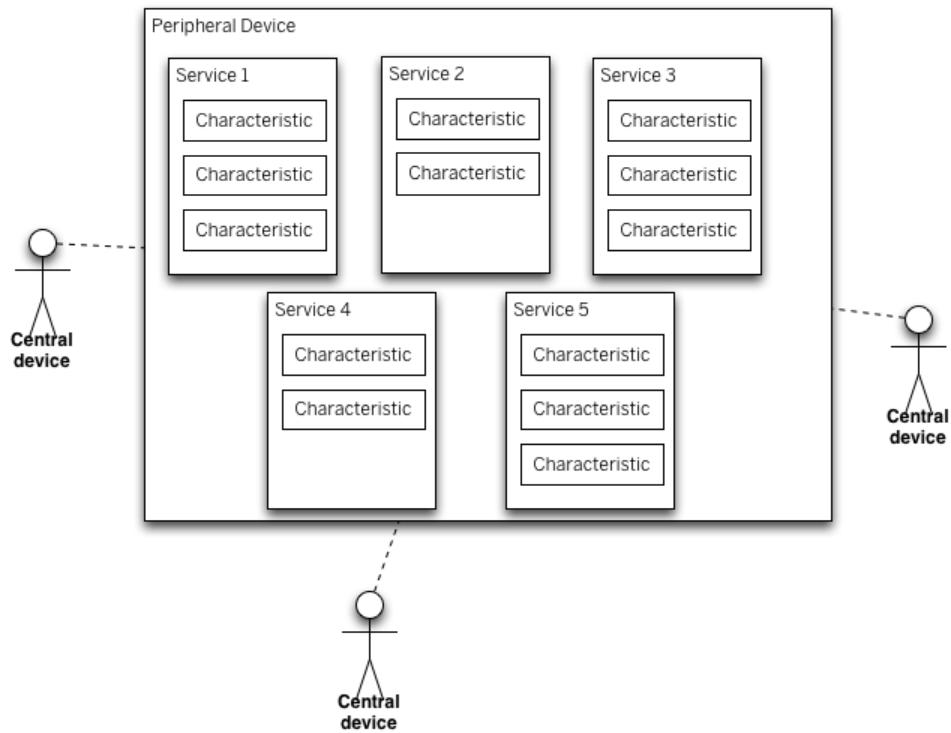


Figure 14.3.: BLE Devices – Peripheral and Central Devices

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.

14.6.3. 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.

14.6.4. Arduino BLE Tutorial Battery Level Indicator Code

14.6.5. Arduino Bluetooth Battery Level Indicator Code Explained

```

1 #include <ArduinoBLE.h>
2 BLEService batteryService("1101");
3 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`.

```

1 #include <ArduinoBLE.h>
2 BLEService batteryService("1101");
3 BLEUnsignedCharCharacteristic batteryLevelChar("2101", BLERead | BLENotify);
4
5 void setup() {
6     Serial.begin(9600);
7     while (!Serial);
8
9     pinMode(LED_BUILTIN, OUTPUT);
10    if (!BLE.begin())
11    {
12        Serial.println("starting BLE failed!");
13        while (1);
14    }
15
16    BLE.setLocalName("BatteryMonitor");
17    BLE.setAdvertisedService(batteryService);
18    batteryService.addCharacteristic(batteryLevelChar);
19    BLE.addService(batteryService);
20
21    BLE.advertise();
22    Serial.println("Bluetooth device active, waiting for connections ... ");
23 }
24
25 void loop()
26 {
27     BLEDevice central = BLE.central();
28
29     if (central)
30     {
31         Serial.print("Connected to central: ");
32         Serial.println(central.address());
33         digitalWrite(LED_BUILTIN, HIGH);
34
35         while (central.connected()) {
36
37             int battery = analogRead(A0);
38             int batteryLevel = map(battery, 0, 1023, 0, 100);
39             Serial.print("Battery Level % is now: ");
40             Serial.println(batteryLevel);
41             batteryLevelChar.writeValue(batteryLevel);
42             delay(200);
43
44         }
45     }
46     digitalWrite(LED_BUILTIN, LOW);
47     Serial.print("Disconnected from central: ");
48     Serial.println(central.address());
49 }
```

Listing 14.1.: Arduino BLE Tutorial Battery Level Indicator Code

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

Now we will jump on to the Setup function.

```

1 Serial.begin(9600);
2 while (!Serial);
3 pinMode(LED_BUILTIN, OUTPUT);
4 if (!BLE.begin()) {
5     Serial.println("starting BLE failed!");
6     while (1);
7 }
```

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.

```

1 BLE.setLocalName("BatteryMonitor");
2 BLE.setAdvertisedService(batteryService);
3 batteryService.addCharacteristic(batteryLevelChar);
4 BLE.addService(batteryService);
```

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

```

1 BLE.advertise();
2 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.

```

1 BLEDevice central = BLE.central();
2 if (central) {
3     Serial.print("Connected to central: ");
4     Serial.println(central.address());
5     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.

```

1 while (central.connected()) {
2     int battery = analogRead(A0);
3     int batteryLevel = map(battery, 0, 1023, 0, 100);
4     Serial.print("Battery Level % is now: ");
5     Serial.println(batteryLevel);
6     batteryLevelChar.writeValue(batteryLevel);
7     delay(200);
8 }
```

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.

```

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

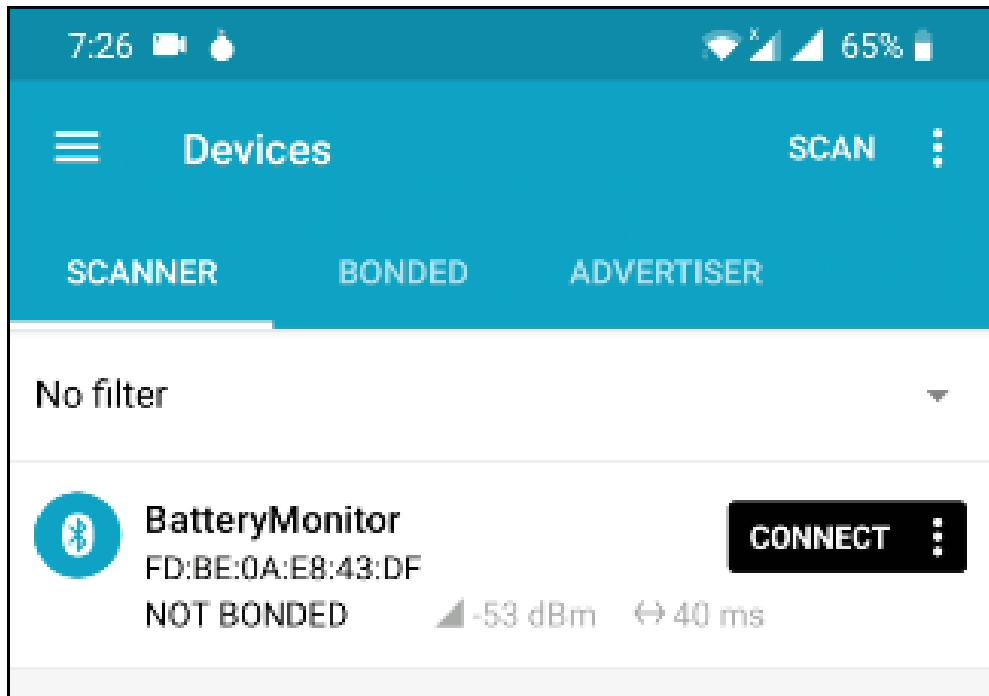


Figure 14.4.: Battery app “nRF Connect”

14.6.6. 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 see the battery levels being read from the Arduino.

14.6.7. 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.

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.

```

1 #include <Arduino_LSM9DS1.h>
2
3 void setup() {
4     Serial.begin(9600);
5     while (!Serial);
6     Serial.println("Started");
7
8     if (!IMU.begin()) {
9         Serial.println("Failed to initialize IMU!");
10        while (1);
11    }
12
13    Serial.print("Accelerometer sample rate = ");
14    Serial.print(IMU.accelerationSampleRate());
15    Serial.println(" Hz");
16    Serial.println();
17    Serial.println("Acceleration in G's");
18    Serial.println("XtYtZ");
19 }
20
21 void loop() {
22     float x, y, z;
23
24     if (IMU.accelerationAvailable()) {
25         IMU.readAcceleration(x, y, z);
26
27         Serial.print(x);
28         Serial.print('t');
29         Serial.print(y);
30         Serial.print('t');
31         Serial.println(z);
32     }
33 }
```

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.

```

1 #include <ArduinoBLE.h>
2 #include <Arduino_LSM9DS1.h>
3
4 int accelX=1;
5 int accelY=1;
6 float x, y, z;
7
8 BLEService customService("1101");
9 BLEUnsignedIntCharacteristic customXChar("2101", BLERead | BLENotify);
10 BLEUnsignedIntCharacteristic customYChar("2102", BLERead | BLENotify);
11
12 void setup() {
13     IMU.begin();
14     Serial.begin(9600);
```

```

15     while (!Serial);
16
17     pinMode(LED_BUILTIN, OUTPUT);
18
19     if (!BLE.begin()) {
20         Serial.println("BLE failed to initiate");
21         delay(500);
22         while (1);
23     }
24
25     BLE.setLocalName("Arduino Accelerometer");
26     BLE.setAdvertisedService(customService);
27     customService.addCharacteristic(customXChar);
28     customService.addCharacteristic(customYChar);
29     BLE.addService(customService);
30     customXChar.writeValue(accelX);
31     customYChar.writeValue(accelY);
32
33     BLE.advertise();
34
35     Serial.println("Bluetooth device is now active, waiting for connections... ");
36 }
37
38
39 void loop() {
40
41     BLEDevice central = BLE.central();
42     if (central) {
43         Serial.print("Connected to central: ");
44         Serial.println(central.address());
45         digitalWrite(LED_BUILTIN, HIGH);
46         while (central.connected()) {
47             delay(200);
48             read_Accel();
49
50             customXChar.writeValue(accelX);
51             customYChar.writeValue(accelY);
52
53             Serial.print("At Main Function");
54             Serial.println(" ");
55             Serial.print(accelX);
56             Serial.print(" ");
57             Serial.println(accelY);
58             Serial.println(" ");
59             Serial.println(" ");
60         }
61     }
62     digitalWrite(LED_BUILTIN, LOW);
63     Serial.print("Disconnected from central: ");
64     Serial.println(central.address());
65 }
66
67 void read_Accel() {
68

```

```
69     if (IMU.accelerationAvailable()) {  
70         IMU.readAcceleration(x, y, z);  
71         accelX = (1+x)*100;  
72         accelY = (1+y)*100;  
73     }  
74 }  
75 }
```

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.

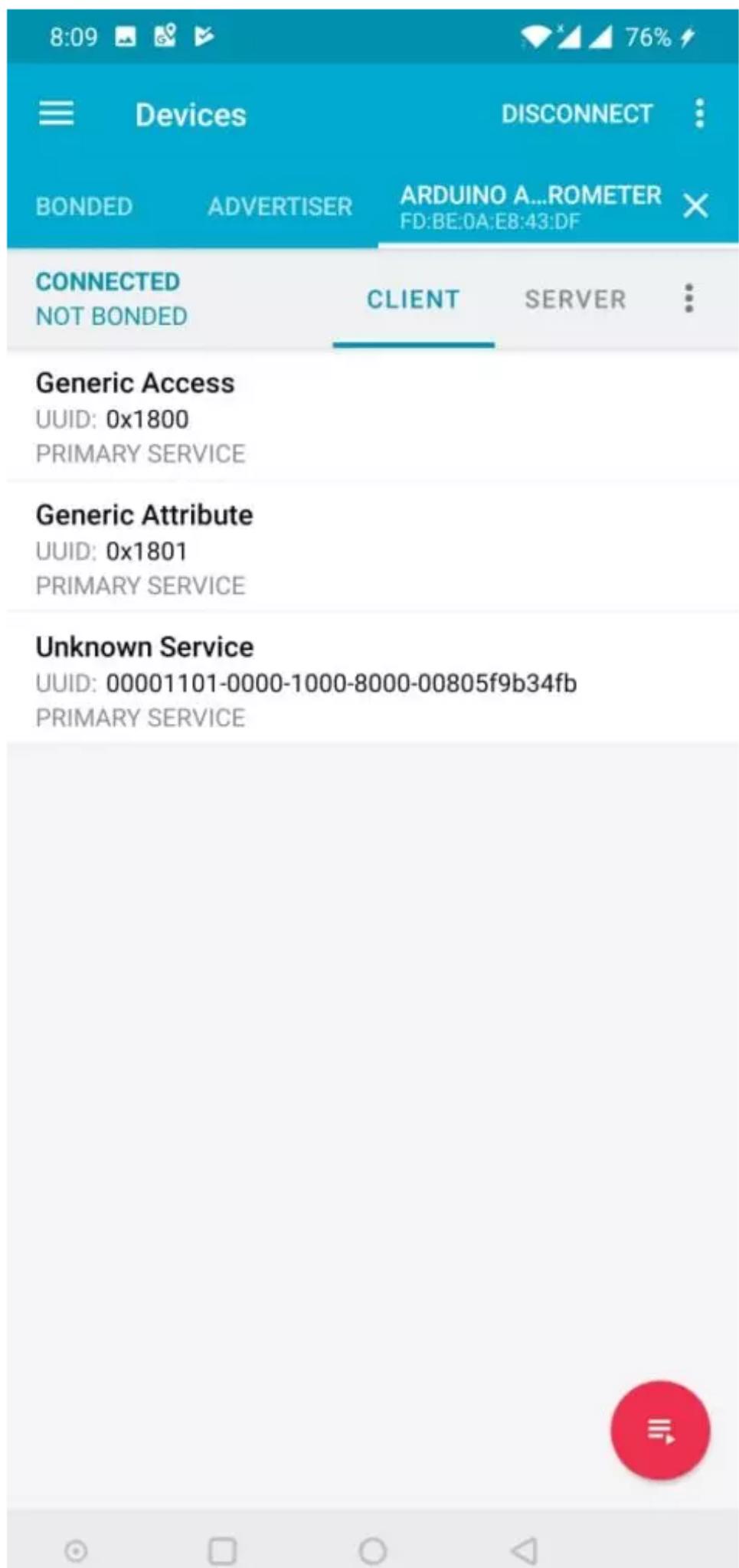
Tap on Unknown Service and you will see the accelerometer values being read from the Arduino.

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.

The screenshot shows a mobile application interface for managing nearby Bluetooth devices. At the top, there is a header bar with the time (8:09), signal strength, battery level (76%), and a 'SCAN' button. Below the header, the title 'Devices' is displayed, along with filter options: 'SCANNER', 'BONDED', 'ADVERTISER', and the device entry 'ARDUINO A FD:BE:0A:E8:'. A dropdown menu labeled 'No filter' is visible. The main content area lists two devices:

- Arduino Accelerometer**
FD:BE:0A:E8:43:DF
NOT BONDED -63 dBm 101 ms
Actions: OPEN TAB (button)
- COLLECTOR**
NOT BONDED -64 dBm 552 ms
Actions: CONNECT (button) More (three-dot menu)

A large, semi-transparent light gray rectangular overlay covers the bottom half of the screen, obscuring the footer area.



15. Power Source

For some application, using the computer to power the device - although most common - would not work. For better results in recording subjects' everyday moves this device is designed; therefore, it requires a light-weighted, long-term portability powering system, and is easily carried in a pocket.

Commercially available with various power densities and voltages, batteries have gained popularity [Dew+14]. Considering various aspects can make choosing the right type of battery a challenging task.

Understanding your device's power needs is essential to selecting an appropriate battery. To fulfill their requirements for higher outputs of electricity, devices such as EVs must rely on batteries of greater energy density and more powerful outputs.

The dimensions and mass of the battery hinge on those of the device it powers. Smartphones and laptops require batteries that are both lightweight and compact.

How often the device is used will dictate the necessary battery life. The duration between charges will also play a role. Devices that require frequent use should have batteries with extended cycle lives.

Factors like temperature and humidity, which fall under environmental conditions, may impact how effectively the battery operates. Special considerations should be made when planning on using batteries in areas that have extreme temperatures or humid conditions.

Selecting an appropriate battery requires consideration of potential safety risks. Certain types like lithium-ion have known concerns and need to be taken into account. Prioritizing safety concerns when making decisions is important.

The power supply for the device should be lightweight, portable, and long-lasting. The power supply should also not add excessive weight to the device. So the lighter the power supply is the better. [Sch22]

15.1. Port

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 power supplies for the Arduino Nano 33 BLE Sense board are connected via:

- MicroUSB port
- Pins: Vin, GND

As a result, there are two ways to power up the Arduino Nano 33 BLE Sense board. The first option is to use the Pins: Pin 15 (Vin), with an input voltage range of 6.2V to 20V (optimal range of 7V to 12V) for the positive (+) connection and Pin 14 as GND for the negative connection (-) (See 2.4 for the PIN Configuration). This option allows the USB port to remain free.

The second option is to use the MicroUSB port with either a lithium battery or a power bank (depending on the powerbank, some changes may need to be made). This method requires a USB-B cable.

The easiest way to power the Arduino Nano 33 BLE Sense board for simple testing is through the MicroUSB connection. The board can be powered by connecting a USB cable to a computer or USB power supply, which is the most common method for powering the device during development and testing.

15.2. Power sources Available

(introduction, compatibility with our board, how long does it last, advantages disadvantages)

Several options are available for powering the Arduino 33 BLE Sense board. To narrow our focus, we will study only the portable ones. This chapter covers three potential sources. Lithium batterie, Nickel-Metal Hydride (NiMH) batterie, and Power banks. Choosing the right type is vital as they all have unique advantages and disadvantages. Now, let's take a look at the three possible power sources:

15.2.1. Nickel-metal hydride (NiMH)

Nickel-metal hydride (NiMH) batteries are a popular choice because of their rechargeability; the most common is a nominal voltage of 1.2 volts per cell, which means you will have to use several of them.

NiMH batteries offer reliability and cost-effectiveness for various applications including digital cameras, flashlights, remote controls, and handheld gaming consoles thanks to their long cycle life.

NiMH batteries are a versatile choice for numerous users, especially in devices with low power consumption that require extended battery life. However, they may not be the optimum decision for high-power, high-drain devices. They might lack enough energy to keep the item running for extended durations. Moreover, NiMH batteries come with certain drawbacks. As an illustration, their charge diminishes over time even when not being used since they tend to self-discharge. Additionally, they are not as powerful and have lesser capacity compared to other battery kinds, like lithium-ion.[Bab+23] Notwithstanding these drawbacks, NiMH batteries remain a top choice for specific applications. As a case in point, they can be charged rapidly which proves useful when the battery is quickly depleted as seen with portable electronics and communications. So, if you require a battery that can be charged rapidly and is appropriate for low-drain devices, NiMH batteries are a suitable choice. However, if your needs include a battery for high-power and high-drain devices, it might be worthwhile to explore better choices. Take lithium-ion batteries as an example, they are more powerful and have a higher capacity.[Bab+23]

The bigger version of them has been used in "prototypes" of hybrid cars, the Toyota Prius, models II and III came with a fast, heavy NiMH battery with around 200 V and around 6.5 Ah stocked. These batteries achieve enormous lifespans and cycle numbers in such cars. However, for plug-in hybrids and newer hybrid models, Toyota and other manufacturers have switched to lithium to save weight, after apparently solving the problems associated with this technology. [Sch22]

NiMH batteries are considered a safer option for safety-sensitive applications due to the inherent non-flammability of their aqueous electrolyte. Additionally, they also perform better at low temperatures compared to lithium-ion batteries because ions are more mobile in aqueous electrolytes than in organic electrolytes. [Bab+23]

15.2.2. Lithium batteries

Highly preferred for numerous applications owing to their notable features such as extended lifespan, a higher power level, and greater energy intensity are lithium batteries. In other words, this denotes their capacity to store ample amounts of energy in a tiny space while delivering high-powered outputs swiftly and operating for prolonged periods without needing battery replacements [Bab+23].

Energy densities up to 141 Wh kg^{-1} and power densities up to $20,600 \text{ W kg}^{-1}$ can be achieved from an aqueous lithium-ion battery with a single LiVPO_4F material as both cathode and anode when using a “water-in-salt” gel polymer electrolyte. This

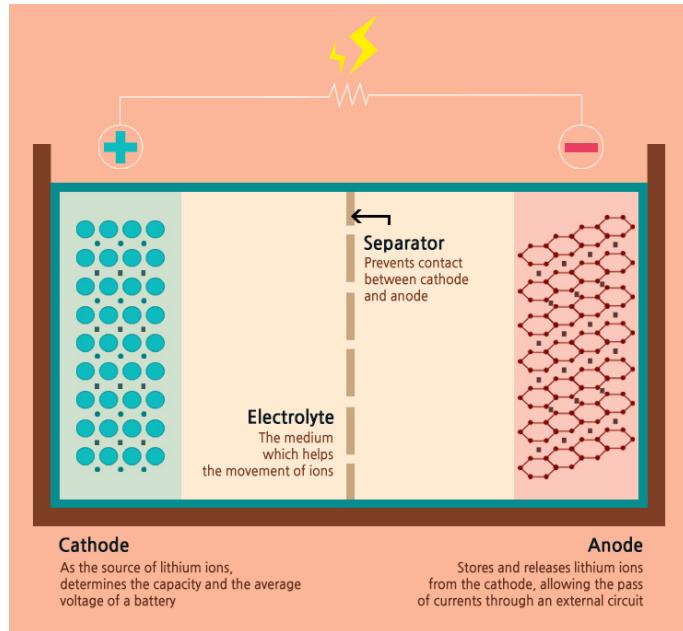


Figure 15.1.: Li-ion battery composition [SSC16]

battery also remains flexible during >4000 cycles. This exceeds reported aqueous energy storage devices by far at the same power level [Yan+17].

If the cell is not fully discharged and has a residual voltage of around 2.5V, it is charged using a CC/CV approach - first constant current with a charging voltage and ending the charging process when a defined voltage is reached. [Sch22]

A lithium-ion battery consists of four main components: cathode, anode, electrolyte, and separator. The cathode determines the capacity and voltage of the battery by using lithium oxide as an active material. The anode allows the flow of electric current through the external circuit while enabling reversible absorption/emission of lithium ions released from the cathode. The electrolyte enables the movement of only lithium ions between the cathode and anode while the separator functions as a physical barrier to prevent the direct flow of electrons and carefully allows only the ions to pass through[SSC16; Li+21]. These components work together to generate electricity through chemical reactions of lithium, which results in the movement of lithium ions and electrons generating electricity. (See 15.1 for a better understanding) Considering the potential impact of moisture on battery safety and performance is crucial. High humidity may cause corrosion and damage to the nail penetration of high energy-density lithium-ion batteries resulting in compromised battery safety. Their life cycle may deteriorate due to several reasons such as rate flow, current, number and sequence while charging/discharging, temperature variations, and storage methods. The recommended temperature range for Li-ion operation is between 15°C and 35°C [European].

Several effects can result from exposing lithium-ion batteries to low temperatures including the loss of both capacity and power as well as life degradation. Furthermore, safety hazards may arise alongside charging difficulties or an unbalanced charge distribution and supplementary expenditure. The study of correlations between these effects is ongoing, along with the exploration of potential solutions to enhance the low-temperature operating scenarios of lithium-ion batteries [Vid+19].

Compared to other types of batteries, Li-ion batteries exhibit lower thermal stability, which has resulted in numerous incidents of battery fires in various applications, such as mobile phones, electric vehicles, and airplanes, in recent years. [Kon+18; Ouy+19;



Figure 15.2.: Damaged Lithium battery

Sun+20a]

To prevent damage to lithium batteries due to overheating during charging, high-quality chargers and battery packs often include thermal monitoring, typically in the form of a thermal resistor. It's safer for us to use high-quality cells because overheating can cause a cell to expand or even become hot enough to ignite. Figure 15.2 shows damaged lithium batteries that have expanded from their original flat shape to an almost round shape. [Sch22]

To operate the Arduino Nano 33 BLE sense, a minimum input voltage of 4.8V is required. However, most Lithium batteries available in the market only provide a maximum of 4.2V and a minimum of 3.7V. To address this issue, two potential solutions are available.

The first solution is to connect two batteries in series, which would enable us to meet the minimum input requirement and will as a bonus result in longer battery life for the device. Although, it should be noted that this solution would increase the weight, cost, and size of the device.

The second solution entails utilizing a boost converter to attain the appropriate voltage, which is crucial in ensuring that the board is powered effectively. This approach would enable us to achieve the necessary voltage without significant additional weight or volume to the device.

15.2.3. Powerbanks

There are colorful types of power banks available in the request, differing in size and power capacity. Some are small and movable, ideal for charging smartphones on-the-go, while others are larger and can charge multiple biases contemporaneously or indeed power laptops. The power capacity of power banks also varies, with some offering a few thousand mAh while others can go up to tens of thousands of mAh. It's important to consider your charging requirements and device comity when choosing a power bank.

Power banks are an accessible and movable way to charge electronic bias on the go, they're movable and can be fluently carried around, extending the operating time of mobile bias without the need to charge them through a bowl that's connected to the mains, and they can serve as a provisory power source for electronic bias in situations where there's no access to a power outlet or during power outages. They're also useful for trial and educational purposes [HP20].

The size of a power bank is generally commensurable to its capacity, meaning that power banks with advanced capacities tend to be larger in size, utmost power banks use lithium-ion batteries, which can hold a charge for days. Still, the charging miracle that power banks use is time-consuming, which can be a debit for druggies who need to charge their bias snappily [Sun+20b].

Luc Lemmens emphasized in a laboratory trial featured in the Elektor special magazine 2022 that power banks differ in the quantum of energy they can store, and that the energy that a power bank can give should be of interest from a physical point of

view. The typical affair voltage and amperage of a marketable power bank can vary depending on the specific model and capacity.

One issue that may arise when using a power bank is that if the connected device has a too low current draw, the power bank may be unfit to serve duly or will shut itself down. Power banks are designed to switch off after many seconds as soon as the smartphone is completely charged, and this could only be detected from the affair current. However, it would mean that the phone is charged, if the affair current falls below a certain threshold.

Given our circumstances, it's likely that the power bank will automatically shut off after a brief period if we don't make any custom variations to it. This is because we're exercising a low-power board [Sun+20b]. To ensure safety when using power banks, it is important to avoid leaving them in places with high temperatures, such as cars parked under the sun. Charging a power bank while it's hot is also not recommended, so be sure to let it cool down before charging. Taking these precautions can help prevent any potential accidents or damage to your power bank.

Power banks are generally safe to use, but users should be aware of safety concerns such as the safety concerns when testing large station battery banks. Users should also use power banks with caution and according to the manufacturer's instructions to avoid overheating, overcharging, and other safety hazards.

15.2.4. How To Calculate Battery Run Time

The theoretical formula for calculating battery life is:

$$\text{Time (h)} = \text{Capacity (Ah)} / \text{Current (A)}$$

where time is in hours, capacity is given in Ah and current is in A. so a 10 (Ah) battery delivering 1(A) should last for 10 hours, if the current of the powered device is 10A then it would last for 1 hour, and if it is 5A would last for 2 hours.

Most batteries, especially those with circuits, will not work down to 0 volts and will stop working at a set voltage before the battery is fully drained, usually around 80-90% of the capacity.

therefore, depending of the battery's specifications, the calculation will need to times 0.8-0.9:

$$\text{Hours Time (h)} = \text{Capacity (mAh)} / \text{Current (mA)} * 0,9$$

If we assume that the current consumption of the Arduino board is 20mA and that we are using 4 AA Nickel-Metallhydrid batteries with a capacity of 2600mAh each then:

$$H= 4*2600\text{mAh}/50\text{mA} * 0,9 = 187,2 \text{ hrs or } 7,8 \text{ days (24h)}$$

To calculate battery life in hours when only Watts are given, it is possible to use the formula:

$$\text{Discharging Time} = \text{Battery Capacity} * \text{Battery Volt} / \text{Device Watt}.$$

For example, a 5AH battery with 3V attached to a 10W bulb $5\text{AH} * 3,7\text{V} / 10\text{W}$ would last for 1.85 hours in theory, and 1.66 hours in reality with 90% power efficiency for Li-ion/LiPo batteries.

16. Battery

16.1. Checking the Battery Voltage

We use an analog input pin to read the voltage. As we are running from a 3.7V volt battery, we need to adjust the reference voltage used by the pin as otherwise it would be comparing the voltage to itself. The statement `analogReference(INTERNAL)` sets the pin to compare the input voltage to a regulated 1.1V. We therefore need to reduce the voltage on the input pin to less than 1.1V for this to work. This is done by dividing the voltage using 2 resistors, 1m and 330k ohms. This divides the voltage by approximately 4 so when the battery is fully charged, which is 4.2V, the voltage at the pin input is $4.2/4 = 1.05V$.

```
1 // Battery Monitor
2 #define MONITOR_PIN A0           // Pin used to monitor supply voltage
3 const float voltageDivider = 4.0; // Used to calculate the actual voltage fRom
4                                         // Using 1m and 330k ohm resistors divids th
voltage by approx 4                  // You may wany to substitute
5                                         // actual values of resistors in an equation (R1 + R2)/R2
6                                         // E.g. (1000 + 330)/330 = 4.03
7                                         // Alternatively take the voltage reading a
8                                         // the 2 resistors to ground and divide one
9
10 // Read the monitor pin and calculate the voltage
11 float BatteryVoltage()
12 {
13     float reading = analogRead(MONITOR_PIN);
14     // Calculate voltage - reference voltage is 1.1v
15     return 1.1 * (reading/1023) * voltageDivider;
16 }
```

The function `BatteryVoltage()`, reads the analog pin, which will range from 0 for 0V to 1,023 for 1.1V and using this reading calculates the actual voltage coming form the battery.

The function `DrawScreenSave()` function calls this then selects the appropriate bitmap to display based on the following:

- If voltage is greater then 3.6V - full
- Voltage between 3.5 and 3.6V - 3/4
- Voltage between 3.4 and 3.5V - half
- Voltage between 3.3 and 3.4V - 1/4
- Voltage < 3.3V - empty

17. Tiny Machine Learning Kit

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

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 17.1.: Das Tiny Machine Learning Kit [Ardi]

17.1. 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 Drehratensor BMI270 und das Magentometer 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]

17.2. 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.

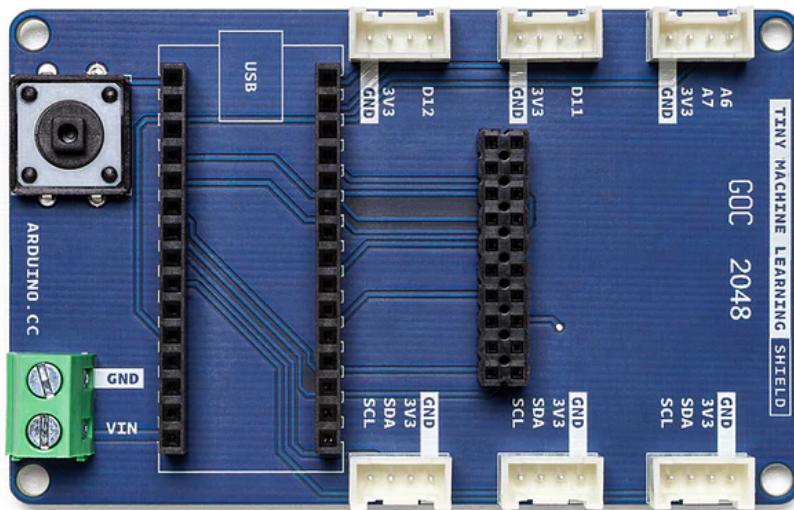


Figure 17.2.: Das Tiny Machine Learning Shield [Ardi]

17.3. 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.

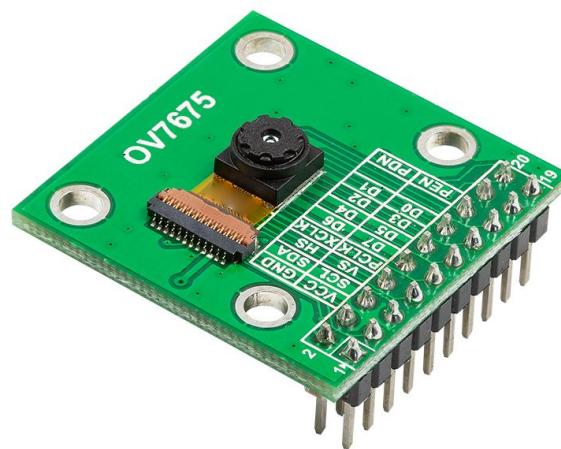


Figure 17.3.: Das OV7675 Kameramodul [Ardi]

17.4. 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 17.4.: Ein USB-A auf Micro-USB Kabel

WS:Quelle fehlt

17.5. USB-Kabel mit USB-A- und USB-Micro-B-Stecker mit 90° Winkel

Aus Platzgründen haben wir uns dazu entschieden, ein abgewinkeltes USB-Micro-B-auf USB-A-Kabel zu verwenden, da so das Gehäuse kleiner ausgelegt werden kann.



Figure 17.5.: USB-Kabel mit USB-A- und USB-Micro-B-Stecker mit 90° Winkel

WS:Quelle fehlt

17.6. OLED-Display

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.

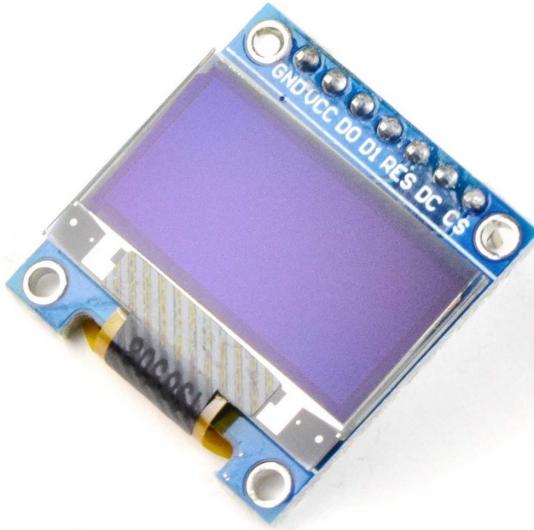


Figure 17.6.: Das Display zur Ausgabe der Messwerte

WS:Quelle fehlt

17.7. Powerbank

WS:text fehlt

Die Spannungsversorgung erfolgt über die Powerbank. Sie verfügt über einen Taster, mit dem sie ein- und ausgeschaltet werden kann. Zum Einschalten muss der Taster für eine Sekunden gedrückt werden. Bei kurzer Betätigung wird dem Nutzer der Ladezustand angezeigt. Zum Ausschalten muss der Taster drei Sekunden lang gedrückt werden.



Figure 17.7.: Die verwendete Powerbank

WS:Quelle fehlt

17.8. Schaltplan

WS:text fehlt

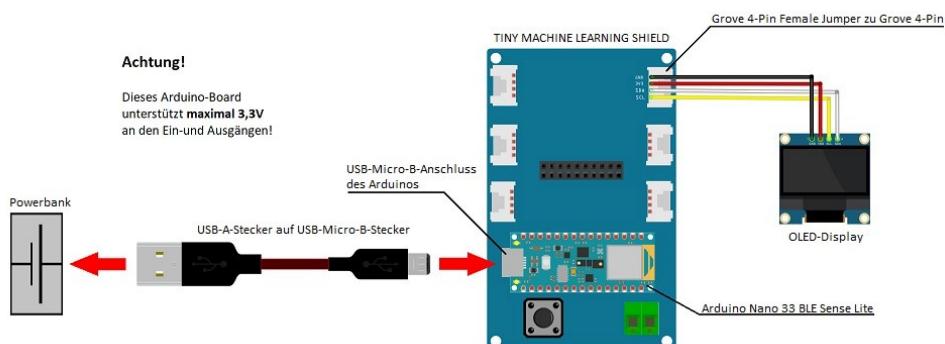


Figure 17.8.: Schaltplan des Winkelmessers

WS:Quelle fehlt

18. Tiny-ML on Arduino Nano 33 BLE Sense

Machine learning (ML) and Artificial intelligence (AI) are everywhere, its application, use cases and importance make it very useful for every process. In the same sense, the tiny Arduino nano 33 BLE Sense board give us a way to make Tiny-ML application, where the model are not as big as previously made for high and big processing computer. By having a tiny machine learning model we can deploy it on low power devices, one of these is Arduino nano 33 BLE Sense. Due to its small size, the available set of sensors and low power consumption, it is easy to use anywhere by deploying the Machine learning algorithm and run the Internet of things (IOT) and Artificial intelligence (AI) application for any process.[DMM20]

18.1. Usefull Tiny-ML Model for Arduino Nano 33 BLE Sense

18.1.1. Voice Recognition

Although, being a low power processing device it is not supported the big model and very big data. One of the usefull Tiny-ML model we can make on Arduino Nano 33 BLE Sense is to detect the different voices from the surrounding. There is a on-board embed sensor on arduino for detecting voices is MP34DT05 the digital microphone. By using the MP34DT05, we can make a data set for keyword voice recognition model. Initially we train a model on Google colab and then convert it into TensorFlow lite for low power devices.[Waq+21] For using the Voice recognition functionality of arduino nano 33 ble board, we need the on-board sensor MP34DT05 (Microphone), which is use for capturing, recognizing and detecting the voice. The supporting library for activating this sensor is PDM which will be discussed in the later chapter.

18.1.2. Custom Gesture Recognition

For capturing the gesture data using the on-board 9-axis Inertial measurement unit (IMU), we can make different types of gestures by rotating and changing the position by holding the arduino board in our hand . By doing this, the 9-axis IMU changes the accelerometre, and gyroscope value of the sensor. The limitation for training this model is to hold the board in our hand all the time. But, for the testing purpose it changes the value as per the gesture as shown in the figure below 18.1 for training the Tiny-ML model.[Ard21]

The another possibility for detecting gesture is to use the APDS9960 sensor, it can also detect the gesture by moving the hand in front of it. It has also some limitation, it will not detect any gesture above 15 mm distance from the sensor.

18.1.3. Color Detection

The same above mentioned sensor APDS9960 is use for (Red, Green, Blue) RGB color detection too. We can also use the RGB functionality of this sensor and train the Tiny-ML model for detecting the different color product, which make the differentiation among the product on the basis of RGB color.

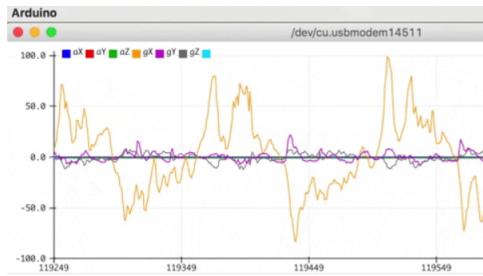


Figure 18.1.: IMU (Inertial Measurement Unit) Gesture Data

18.1.4. Person Detection

One of the advantages of using a small devices such as the Arduino Nano 33 BLE Sense with TinyML is that it could be used as a remote low powered sensor to detect movement or even if there is a person in the area or not. APDS9960 sensor working as a proximity sensor, it can start to detect person at the certain distance. For doing this, we need external camera which can detect the moving person in the sorrounding or not. The supporting camera can be a Arducam Mini 2MP, which will be discussed in the hardware setup chapter later. Some usefull libraries e.g; JPEGDecoder library to decode JPEG-encoded images, the Arducam Library need to be installed to make the compatibility between Arducam and edge computer (Arduino Nano 33 BLE Sense) by setting the hardware parametre in the library folder. The most necessary setting will be discussed in the later chapters too. Person Detection

Part III.

Arduino Nano 33 BLE Sense

19. Arduino

Es gibt eine große Auswahl an Arduino-Boards, alle mit unterschiedlichen Fähigkeiten. Nicht auf allen läuft TensorFlow Lite für Mikrocontroller. Das Board, das nun verwendet wird, ist das Arduino Nano 33 BLE Sense. Es ist nicht nur mit TensorFlow Lite kompatibel, sondern enthält auch ein Mikrofon und einen Beschleunigungssensor. Wir empfehlen, die Version des Boards mit Headern zu kaufen, die es einfacher macht, andere Komponenten ohne Löten anzuschließen. Die meisten Arduino-Boards haben eine eingebaute LED, und diese werden wir verwenden, um unsere Sinuswerte visuell auszugeben. Abbildung 19.1 zeigt ein Arduino Nano 33 BLE Sense-Board mit hervorgehobener LED.

19.1. Handhabung der Ausgabe am Arduino

Da wir nur mit einer LED arbeiten können, müssen wir kreativ denken. Eine Möglichkeit ist, die Helligkeit der LED basierend auf dem zuletzt vorhergesagten Sinuswert zu variieren. Da der Wert von -1 bis 1 reicht, könnten wir 0 mit einer vollständig ausgeschalteten LED, -1 und 1 mit einer vollständig leuchtenden LED und alle Zwischenwerte mit einer teilweise gedimmten LED darstellen. Während das Programm in einer Schleife Schlussfolgerungen ausführt, wird die LED wiederholt eingeschaltet und ausgeschaltet.

Mit der Konstante `kInferencesPerCycle` kann die Anzahl der Inferenzen, die über einen vollen Sinuszyklus durchgeführt werden, variiert werden. Da eine Inferenz eine bestimmte Zeit in Anspruch nimmt, kann durch die Einstellung der Konstante `kInferencesPerCycle`, die in `constants.cc` definiert ist, eingestellt werden, wie schnell die LED ausgeblendet wird.

Es gibt eine Arduino-spezifische Version dieser Datei in `hello_world/arduino/-constants.cc`. Die Datei hat den gleichen Namen wie `hello_world/constants.cc`, so dass sie anstelle der ursprünglichen Implementierung verwendet wird, wenn die Anwendung für Arduino erstellt wird. Um unsere eingebaute LED zu dimmen, können wir eine Technik namens Pulsweitenmodulation (PWM) verwenden. Wenn wir einen Ausgangspin extrem schnell ein- und ausschalten, wird die Ausgangsspannung des Pins

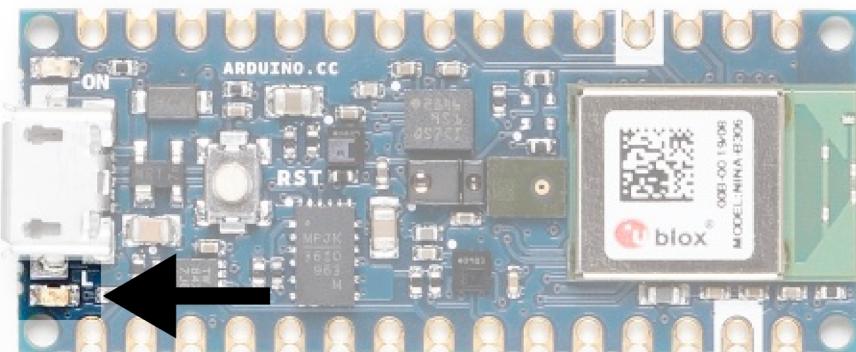


Figure 19.1.: Arduino Nano 33 BLE Sense-Board mit hervorgehobener LED

zu einem Faktor des Verhältnisses zwischen der Zeit, die im Aus- und Ein-Zustand verbracht wird. Wenn der Pin 50 % der Zeit in jedem Zustand verbringt, wird seine Ausgangsspannung 50 % seines Maximums betragen. Wenn er 75 % im Ein-Zustand und 25 % im Aus-Zustand verbringt, wird seine Spannung 75 % seines Maximums betragen.

PWM ist nur an bestimmten Pins bestimmter Arduino-Geräte verfügbar, aber es ist sehr einfach zu verwenden: Wir rufen einfach eine Funktion auf, die unseren gewünschten Ausgangspiegel für den Pin festlegt.

Der Code, der die Ausgabebehandlung für Arduino implementiert, befindet sich in `hello_world/arduino/output_handler.cc`, die anstelle der Originaldatei `hello_world/output_handler.cc`.

Gehen wir den Quellcode durch:

```
1 #include "tensorflow/lite/micro/examples/hello_world/output_handler.h"
2 #include "Arduino.h"
3 #include "tensorflow/lite/micro/examples/hello_world/constants.h"
```

Zunächst binden wir einige Header-Dateien ein. Unsere `output_handler.h` spezifiziert die Schnittstelle für diese Datei. `Arduino.h` stellt die Schnittstelle für die Arduino-Plattform bereit; wir verwenden diese, um Steuerung des Boards. Da wir Zugriff auf `kInferencesPerCycle` benötigen, binden wir auch `constants.h`.

Als nächstes definieren wir die Funktion und weisen sie an, was sie bei der ersten Ausführung tun soll:

```
1 // Adjusts brightness of an LED to represent the current y value
2 void HandleOutput(tflite::ErrorReporter* error_reporter, float x_value,
3 float y_value) {
4     // Track whether the function has run at least once
5     static bool is_initialized = false;
6     // Do this only once
7     if (!is_initialized) {
8         // Set the LED pin to output
9         pinMode(LED_BUILTIN, OUTPUT);
10        is_initialized = true;
11    }
}
```

In C++ behält eine Variable, die innerhalb einer Funktion als statisch deklariert ist, ihren Wert über mehrere Durchläufe der Funktion hinweg. Hier verwenden wir die Variable `is_initialized`, um zu verfolgen, ob der Code in dem folgenden Block `if (!is_initialized)` jemals zuvor ausgeführt wurde.

Der Initialisierungsblock ruft die Funktion `pinMode()` von Arduino auf, die dem Mikrocontroller anzeigt, ob ein bestimmter Pin im Eingangs- oder Ausgangsmodus sein soll. Dies ist notwendig, bevor ein Pin verwendet wird. Die Funktion wird mit zwei von der Arduino-Plattform definierten Konstanten aufgerufen: `LED_BUILTIN` und `OUTPUT`. `LED_BUILTIN` steht für den Pin, der mit der eingebauten LED des Boards verbunden ist, und `OUTPUT` steht für den Ausgangsmodus.

Nachdem Sie den Pin der eingebauten LED auf den Ausgabemodus konfiguriert haben, setzen Sie `is_initialized` auf `true`, damit dieser Blockcode nicht mehr ausgeführt wird.

Als Nächstes berechnen wir die gewünschte Helligkeit der LED:

```
1 // Calculate the brightness of the LED such that y=-1 is fully off
2 // and y=1 is fully on. The LED's brightness can range from 0-255.
3 int brightness = (int)(127.5f * (y_value + 1));
```

Der Arduino erlaubt uns, den Pegel eines PWM-Ausgangs als Zahl von 0 bis 255 einzustellen, wobei 0 ganz aus und 255 ganz an bedeutet. Unser `y_value` ist eine Zahl zwischen -1 und 1. Der vorhergehende Code ordnet `y_value` dem Bereich 0 bis

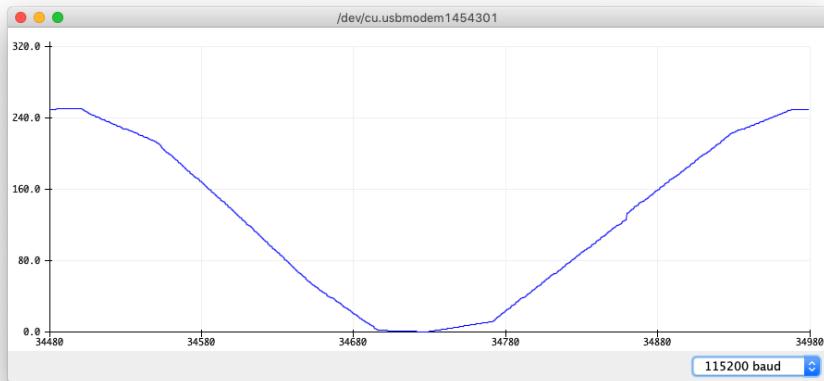


Figure 19.2.: Der serielle Plotter der Arduino-IDE

255 zu, so dass bei $y = -1$ die LED ganz aus ist, bei $y = 0$ die LED halb leuchtet und bei $y = 1$ die LED ganz leuchtet.

Der nächste Schritt besteht darin, die Helligkeit der LED tatsächlich einzustellen:

```

1 // Set the brightness of the LED. If the specified pin does not support PWM
2 // this will result in the LED being on when y > 127, off otherwise.
3 analogWrite(LED_BUILTIN, brightness);
```

Die Funktion `analogWrite()` der Arduino-Plattform nimmt eine Pin-Nummer (wir geben `LED_BUILTIN` an) und einen Wert zwischen 0 und 255. Wir geben unsere Helligkeit an, die in der vorherigen Zeile berechnet wurde. Wenn diese Funktion aufgerufen wird, leuchtet die LED mit dieser Helligkeit.

Leider ist bei einigen Modellen von Arduino-Boards der Pin, an dem die eingebaute LED angeschlossen ist, nicht PWM-fähig. Das bedeutet, dass unsere Aufrufe von `analogWrite()` seine Helligkeit nicht verändern werden. Stattdessen wird die LED eingeschaltet, wenn der in `analogWrite()` übergebene Wert über 127 liegt, und ausgeschaltet, wenn er 126 oder weniger beträgt.

Das bedeutet, dass die LED blinkt und ausschaltet, anstatt zu verblassen. Nicht ganz so cool, aber es demonstriert immer noch unsere Sinuswellenvorhersage.

Schließlich verwenden wir die `ErrorReporter`-Instanz, um den Helligkeitswert zu protokollieren:

```

1 // Log the current brightness value for display in the Arduino plotter
2 error_reporter->Report( "%d\n", brightness );
```

Auf der Arduino-Plattform ist der `ErrorReporter` so eingerichtet, dass er Daten über eine serielle Schnittstelle protokolliert. Die serielle Schnittstelle ist eine sehr verbreitete Art der Kommunikation zwischen Mikrocontrollern und Host-Computern und wird häufig für die Fehlersuche verwendet. Es ist ein Kommunikationsprotokoll, bei dem Daten bitweise durch Ein- und Ausschalten eines Ausgangspins übertragen werden. Wir können es verwenden, um alles zu senden und zu empfangen, von binären Rohdaten bis hin zu Text und Zahlen.

Die Arduino-IDE enthält Werkzeuge zum Erfassen und Anzeigen von Daten, die über eine serielle Schnittstelle empfangen werden. Eines der Werkzeuge, der Serial Plotter, kann ein Diagramm der Werte anzeigen, die er über die serielle Schnittstelle empfängt. Indem wir einen Strom von Helligkeitswerten aus unserem Code ausgeben, können wir sie grafisch darstellen. Abbildung 19.2 zeigt dies in Aktion.

Eine Anleitung zur Verwendung des seriellen Plotters finden Sie weiter unten in diesem Abschnitt.

Sie fragen sich vielleicht, wie der [ErrorReporter](#) in der Lage ist, Daten über die serielle Schnittstelle des Arduino auszugeben. Sie finden die Code-Implementierung in [micro/arduino/debug_log.cc](#). Sie ersetzt die ursprüngliche Implementierung in [micro/debug_log.cc](#).

Genauso wie [output_handler.cc](#) überschrieben wird, können wir plattformspezifische Implementierungen von jeder Quelldatei in TensorFlow Lite für Mikrocontroller bereitstellen, indem wir sie in ein Verzeichnis mit dem Namen der Plattform hinzufügen.

19.2. Ausführen des Beispiels

Unsere nächste Aufgabe besteht darin, das Projekt für Arduino zu erstellen und es auf einem Gerät einzusetzen. Es besteht immer die Möglichkeit, dass sich der Erstellungsprozess geändert hat, seit dieses Buch geschrieben wurde, daher sollten Sie in README.md nach den neuesten Anweisungen suchen.

Hier ist alles, was wir brauchen werden:

- Ein unterstütztes Arduino-Board (wir empfehlen das Arduino Nano 33 BLE Sense)
- Das passende USB-Kabel
- Die Arduino IDE (Sie müssen diese herunterladen und installieren, bevor Sie fortfahren)

Die Projekte in diesem Buch sind als Beispielcode in der TensorFlow Lite Arduino-Bibliothek verfügbar, die Sie einfach über die Arduino-IDE und die Auswahl von Manage Libraries aus dem Tools-Menü installieren können. Suchen Sie im erscheinenden Fenster nach der Bibliothek mit dem Namen [Arduino_TensorFlowLite](#) und installieren Sie sie. Sie sollten in der Lage sein, die neueste Version zu verwenden, aber wenn Sie auf Probleme stoßen, ist die Version, die mit diesem Buch getestet wurde, 1.14-ALPHA.

Sie können die Bibliothek auch aus einer .zip-Datei installieren, die Sie entweder vom TensorFlow Lite Team download oder mit dem TensorFlow Lite for Microcontrollers Makefile selbst erzeugen können. Wenn Sie dies bevorzugen, lesen Sie bitte Anhang A. Nachdem Sie die Bibliothek installiert haben, wird das Beispiel im Menü File unter [Examples→Arduino_TensorFlowLite_](#) angezeigt, wie in Abbildung 19.3 gezeigt. Klicken Sie auf "hello_world", um das Beispiel zu laden. Es wird als neues Fenster mit einer Registerkarte für jede der Quelldateien angezeigt. Die Datei in der ersten Registerkarte, [hello_world](#), entspricht der [main_functions.cc](#), die wir vorhin durchgegangen sind.

19.3. Differences in the Arduino Example Code

Wenn die Arduino-Bibliothek generiert wird, werden einige kleinere Änderungen am Code vorgenommen, damit er gut mit der Arduino-IDE funktioniert. Das bedeutet, dass es einige subtile Unterschiede zwischen dem Code in unserem Arduino-Beispiel und im TensorFlow GitHub Repository gibt. Zum Beispiel werden in der Datei [hello_world](#) die Funktionen `setup()` und `loop()` automatisch von der Arduino Umgebung aufgerufen, so dass die Datei `main.cc` und ihre Funktion `main()` nicht benötigt werden.

Die Arduino-IDE erwartet außerdem, dass die Quelldateien die Erweiterung .cpp anstelle von .cc haben. Da die Arduino-IDE keine Unterordner unterstützt, wird außerdem jedem Dateinamen im Arduino-Beispiel der ursprüngliche Name des Unterordners vorangestellt. Zum Beispiel entspricht [arduino_constants.cpp](#) der ursprünglich benannten Datei [arduino/constants.cc](#).

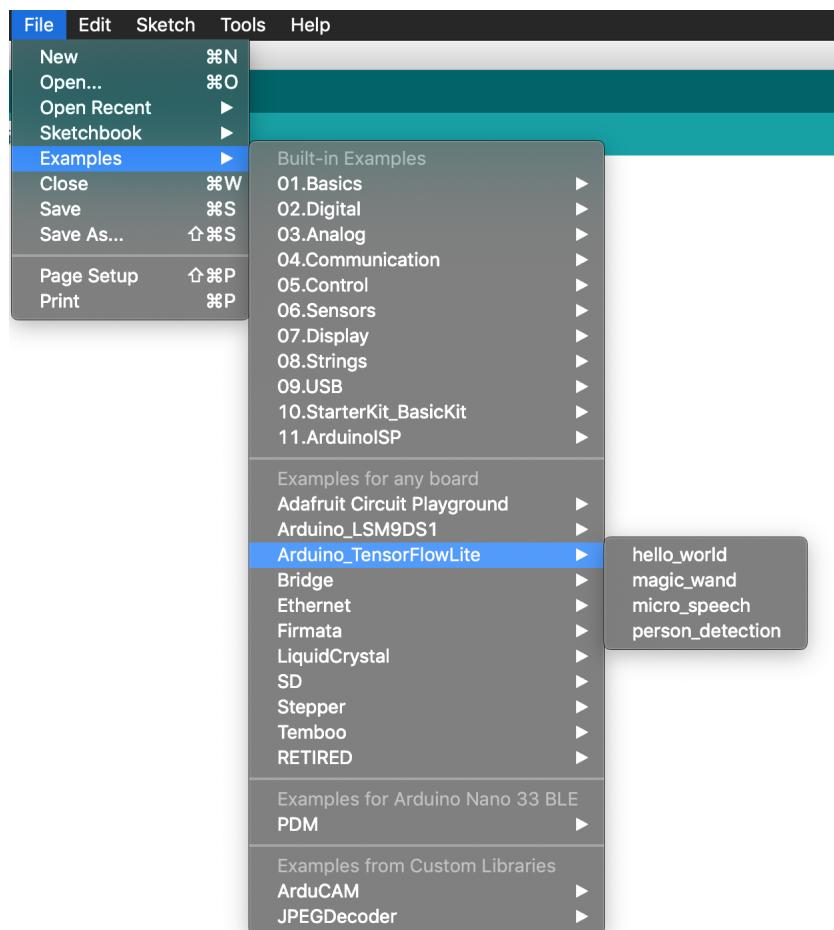


Figure 19.3.: Der serielle Plotter der Arduino-IDE

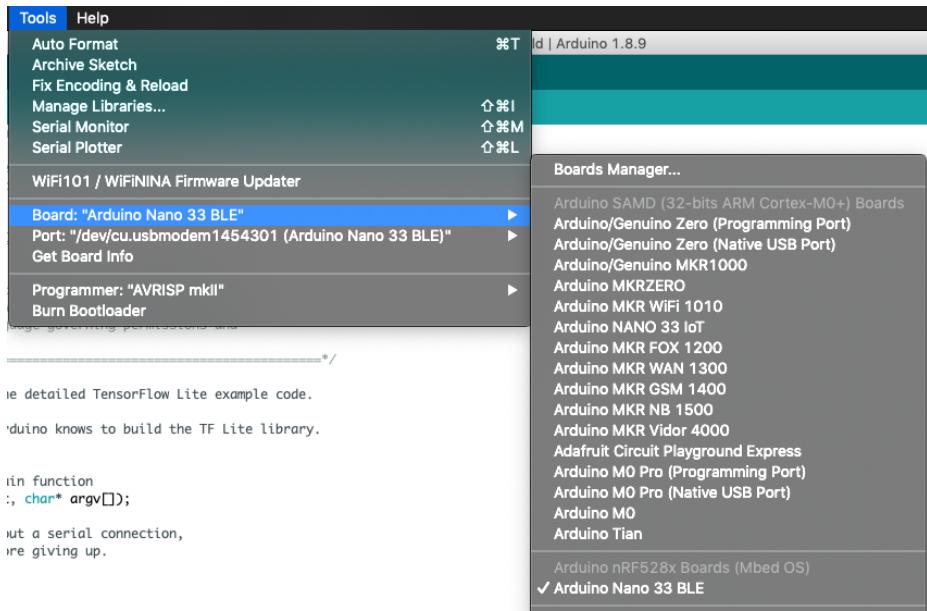


Figure 19.4.: Die Dropdown-Liste Board

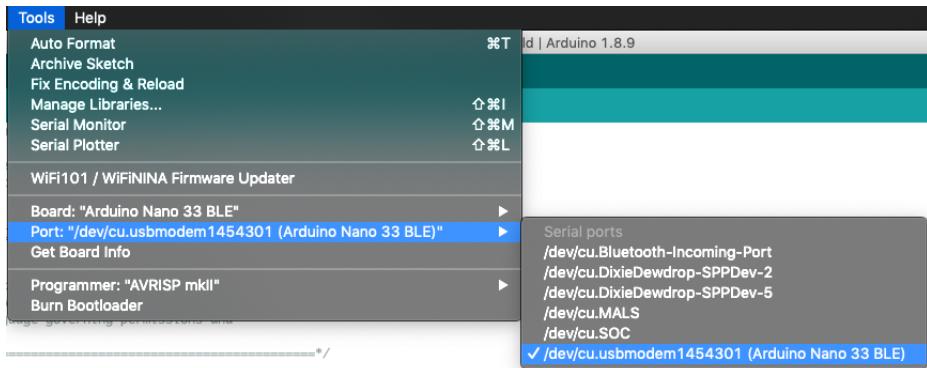


Figure 19.5.: Die Dropdown-Liste Port

Abgesehen von ein paar kleinen Unterschieden bleibt der Code jedoch weitgehend unverändert.

Um das Beispiel auszuführen, schließen Sie Ihr Arduino-Gerät über USB an. Stellen Sie sicher, dass der richtige Gerätetyp in der Dropdown-Liste "Board" im Menü "Tools" ausgewählt ist, wie in Abbildung 19.4 gezeigt.

Wenn der Name Ihres Geräts nicht in der Liste erscheint, müssen Sie sein Support-Paket installieren. Klicken Sie dazu auf Boards Manager. Suchen Sie in dem daraufhin angezeigten Fenster nach Ihrem Gerät und installieren Sie die neueste Version des entsprechenden Support-Pakets.

Stellen Sie anschließend sicher, dass der Anschluss des Geräts in der Dropdown-Liste "Anschluss" ausgewählt ist, die sich ebenfalls im Menü "Tools" befindet (siehe Abbildung 19.5).

Klicken Sie schließlich im Arduino-Fenster auf die Schaltfläche Upload (in Abbildung 19.6 weiß hervorgehoben), um den Code zu kompilieren und auf Ihr Arduino-Gerät hochzuladen.

Nachdem der Upload erfolgreich abgeschlossen wurde, sollten Sie sehen, dass die LED auf Ihrem Arduino-Board beginnt, entweder ein- und auszublenden oder ein- und auszublinken, je nachdem, ob der Pin, an den sie angeschlossen ist, PWM unterstützt.

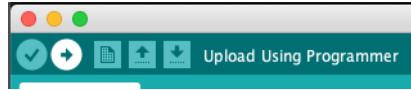


Figure 19.6.: Die Upload-Schaltfläche, ein nach rechts gerichteter Pfeil

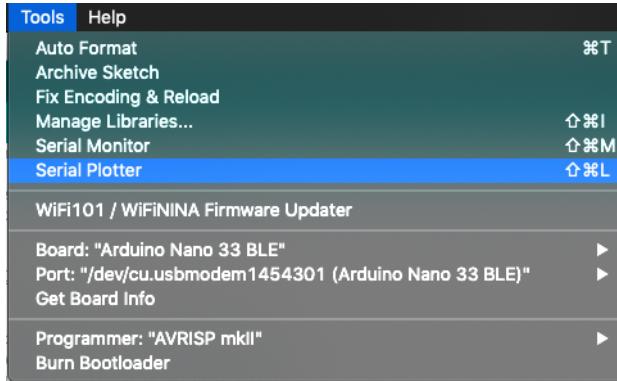


Figure 19.7.: Der Menüpunkt Serieller Plotter

Glückwunsch: Sie führen ML auf dem Gerät aus!

Verschiedene Modelle von Arduino-Boards haben unterschiedliche Hardware und führen die Inferenz mit unterschiedlichen Geschwindigkeiten aus. Wenn Ihre LED entweder flackert oder komplett an bleibt, müssen Sie möglicherweise die Anzahl der Inferenzen pro Zyklus erhöhen. Sie können dies über die Konstante `kInferencesPerCycle` in `arduino_constants.cpp` tun.

"Eigene Änderungen vornehmen" auf Seite 111 zeigt Ihnen, wie Sie den Code des Beispiels bearbeiten können.

Sie können den Helligkeitswert auch in einem Diagramm darstellen. Öffnen Sie dazu den seriellen Plotter der Arduino-IDE, indem Sie ihn im Menü "Tools" auswählen, wie in Abbildung 19.7 gezeigt.

Der Plotter zeigt den Wert an, wie er sich über die Zeit ändert, wie in Abbildung 19.8 gezeigt.

Um die Rohdaten zu betrachten, die von der seriellen Schnittstelle des Arduinos empfangen werden, öffnen Sie den Serial Monitor aus dem Menü Tools. Sie sehen dann einen Strom von Zahlen vorbeifliegen, wie in Abbildung 19.9.

19.4. Eigene Änderungen vornehmen

Jetzt, wo Sie die Anwendung bereitgestellt haben, macht es vielleicht Spaß, herumzuspielen und einige Änderungen am Code vorzunehmen. Sie können die Quelldateien in der Arduino-IDE bearbeiten. Wenn Sie speichern, werden Sie aufgefordert, das Beispiel an einem neuen Ort zu speichern. Wenn Sie mit den Änderungen fertig sind, können Sie in der Arduino-IDE auf die Schaltfläche "Hochladen" klicken, um das Beispiel zu erstellen und bereitzustellen.

Um mit dem Vornehmen von Änderungen zu beginnen, können Sie einige Experimente durchführen:

- Lassen Sie die LED langsamer oder schneller blinken, indem Sie die Anzahl der Inferenzen pro Zyklus anpassen.
- Ändern Sie `output_handler.cc`, um eine textbasierte Animation an der seriellen Schnittstelle zu protokollieren.

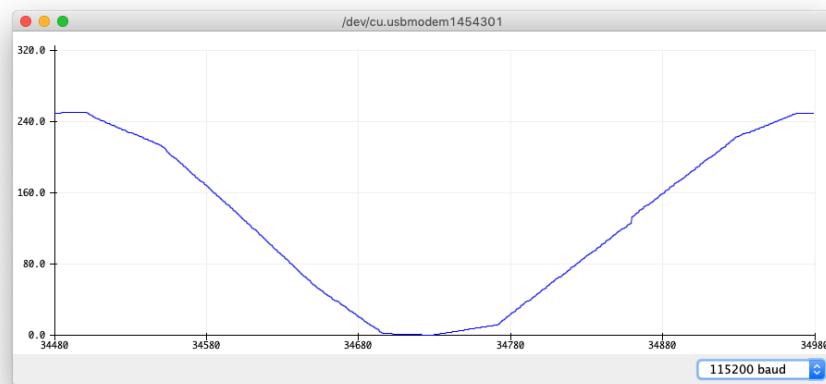


Figure 19.8.: Der Serienplotter, der den Wert grafisch darstellt

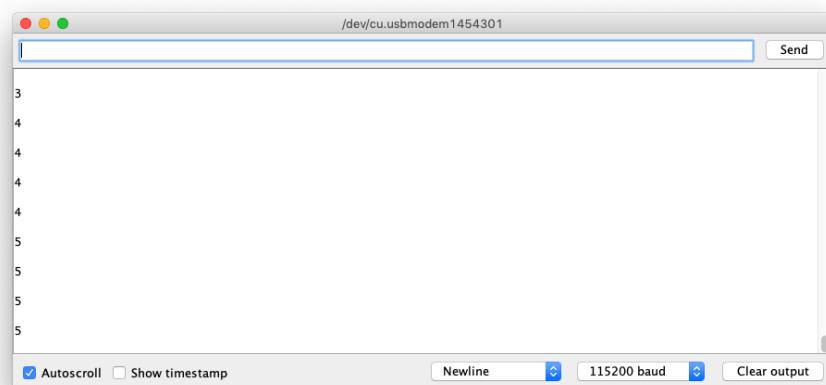


Figure 19.9.: Der Serial Monitor zeigt Rohdaten an

- Verwenden Sie die Sinuswelle, um andere Komponenten zu steuern, z. B. zusätzliche LEDs oder Tongeneratoren.

20. Image Classification with the Arduino Nano 33 BLE Sense

20.1. Was wir bauen

Wir werden eine eingebettete Anwendung erstellen, die ein Modell zur Klassifizierung von Bildern verwendet, die von einer Kamera aufgenommen wurden. Das Modell wird so trainiert, dass es erkennt, wenn eine Person in der Kameraeingabe vorhanden ist. Das bedeutet, dass unsere Anwendung in der Lage sein wird, die An- oder Abwesenheit einer Person zu erkennen und eine entsprechende Ausgabe zu erzeugen.

Dies ist im Wesentlichen der intelligente Vision-Sensor, den wir etwas früher beschrieben haben. Wenn eine Person erkannt wird, lässt unser Beispielcode eine LED aufleuchten - aber Sie können ihn erweitern, um alle möglichen Projekte zu steuern.

https://github.com/tensorflow/tensorflow/tree/master/tensorflow/lite/micro/examples/person_detection

20.2. Bemerkung

TensorFlow Lite fügt regelmäßig Unterstützung für neue Geräte hinzu. Wenn also das Gerät, das Sie verwenden möchten, hier nicht aufgeführt ist, lohnt es sich, in der [README.md](#) des Beispiels nachzusehen. Sie können dort auch nach aktualisierten Deployment-Anweisungen suchen, wenn Sie bei der Ausführung dieser Schritte auf Probleme stoßen.

Anders als bei den vorherigen Kapiteln benötigen Sie für diese Anwendung zusätzliche Hardware. Da keines der beiden Boards eine integrierte Kamera hat, empfehlen wir den Kauf eines Kameramoduls. Diese Informationen finden Sie im Abschnitt über das jeweilige Gerät.

20.3. Kamera-Module

Kameramodule sind elektronische Bauteile auf Basis von Bildsensoren, die Bilddaten digital erfassen. Der Bildsensor wird mit einem Objektiv und einer Steuerelektronik kombiniert und das Modul wird in einer Form hergestellt, die einfach an ein Elektronikprojekt angebracht werden kann.

Lassen Sie uns zunächst die Struktur unserer Anwendung durchgehen. Es ist viel einfacher, als Sie vielleicht erwarten.

20.4. Anwendungsarchitektur

Inzwischen haben wir festgestellt, dass eingebettete Anwendungen für maschinelles Lernen die folgende Abfolge von Dingen tun:

- Erhalten Sie eine Eingabe.
- Verarbeiten Sie die Eingabe vor, um Merkmale zu extrahieren, die für die Eingabe in ein Modell geeignet sind.

- Ausführen der Inferenz auf die verarbeitete Eingabe.
- Die Ausgabe des Modells nachverarbeiten, um sie sinnvoll zu nutzen.
- Verwenden Sie die resultierenden Informationen, um etwas zu bewirken.

In Kapitel 7 haben wir gesehen, wie dies auf die Wake-Word-Erkennung angewendet wird, die Audio als Eingabe verwendet. Dieses Mal werden wir als Eingabe Bilddaten verwenden. Das hört sich vielleicht komplizierter an, ist aber in Wirklichkeit viel einfacher zu handhaben als Audio.

Bilddaten werden üblicherweise als ein Array von Pixelwerten dargestellt. Wir werden unsere Bilddaten von eingebetteten Kameramodulen beziehen, die alle Daten in diesem Format bereitstellen. Unser Modell erwartet auch, dass seine Eingabe ein Array von Pixelwerten ist. Aus diesem Grund müssen wir nicht viel Vorarbeit leisten, bevor wir die Daten in unser Modell einspeisen.

Angesichts der Tatsache, dass wir nicht viel Vorverarbeitung machen müssen, wird unsere Anwendung ziemlich einfach sein. Sie nimmt einen Schnapschuss der Daten von einer Kamera auf, speist ihn in ein Modell ein und bestimmt, welche Ausgabeklasse erkannt wurde. Anschließend zeigt sie das Ergebnis auf einfache Weise an.

Bevor wir weitermachen, wollen wir ein wenig mehr über das Modell erfahren, das wir verwenden werden.

20.5. Einführung in unser Modell

In Kapitel 7 haben wir gelernt, dass faltungsneuronale Netze neuronale Netze sind, die für die Arbeit mit mehrdimensionalen Tensoren ausgelegt sind, bei denen die Informationen in den Beziehungen zwischen Gruppen benachbarter Werte enthalten sind. Sie sind besonders gut für die Arbeit mit Bilddaten geeignet.

Unser Personenerkennungsmodell ist ein neuronales Faltungsnetzwerk, das auf dem Visual Wake Words-Datensatz trainiert wurde. Dieser Datensatz besteht aus 115.000 Bildern, von denen jedes mit der Angabe versehen ist, ob es eine Person enthält oder nicht.

Das Modell ist 250 KB groß und damit deutlich größer als unser Sprachmodell. Diese zusätzliche Größe belegt nicht nur mehr Speicher, sondern bedeutet auch, dass es viel länger dauert, eine einzelne Inferenz auszuführen.

Das Modell akzeptiert 96×96 -Pixel-Graustufenbilder als Eingabe. Jedes Bild wird als 3D-Tensor mit der Form $(96, 96, 1)$ bereitgestellt, wobei die letzte Dimension einen 8-Bit-Wert enthält, der ein einzelnes Pixel repräsentiert. Der Wert gibt den Farbton des Pixels an, der von 0 (vollständig schwarz) bis 255 (vollständig weiß) reicht.

Unsere Kameramodule können Bilder in verschiedenen Auflösungen zurückgeben, daher müssen wir sicherstellen, dass sie auf 96×96 Pixel verkleinert werden. Außerdem müssen wir vollfarbige Bilder in Graustufen umwandeln, damit sie mit dem Modell funktionieren.

<https://arxiv.org/abs/1906.05721>

Sie denken vielleicht, dass 96×96 Pixel eine winzige Auflösung ist, aber sie ist mehr als ausreichend, um eine Person in jedem Bild zu erkennen. Modelle, die mit Bildern arbeiten, akzeptieren oft erstaunlich kleine Auflösungen. Eine Erhöhung der Eingabegröße eines Modells führt zu abnehmenden Erträgen, und die Komplexität des Netzwerks steigt stark an, wenn die Größe der Eingabe skaliert. Aus diesem Grund arbeiten selbst moderne Bildklassifizierungsmodelle in der Regel mit einer maximalen Auflösung von 320×320 Pixeln.

Das Modell gibt zwei Wahrscheinlichkeiten aus: eine, die die Wahrscheinlichkeit angibt, dass eine Person in der Eingabe vorhanden war, und eine andere, die die Wahrscheinlichkeit angibt, dass niemand dort war. Die Wahrscheinlichkeiten reichen von 0 bis 255.

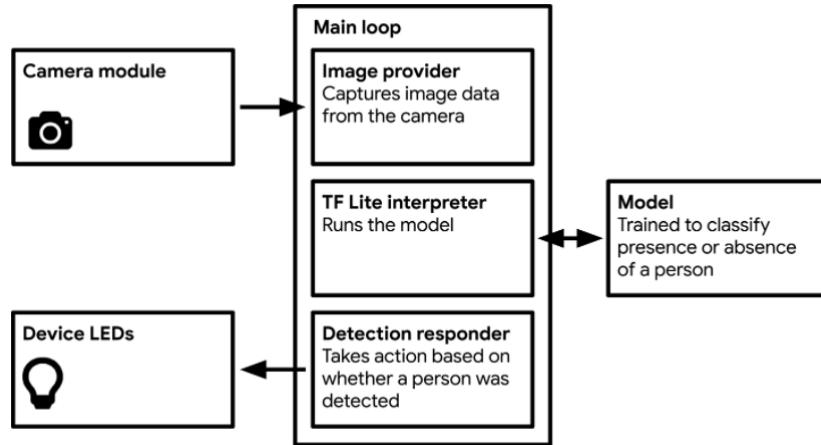


Figure 20.1.: Diagram of the components of our person detection application

Unser Personenerkennungsmodell verwendet die MobileNet-Architektur, eine bekannte und erprobte Architektur, die für die Bildklassifizierung auf Geräten wie Mobiltelefonen entwickelt wurde. In Kapitel 10 werden Sie erfahren, wie dieses Modell an Mikrocontroller angepasst wurde und wie Sie Ihr eigenes Modell trainieren können. Lassen Sie uns nun weiter erkunden, wie unsere Anwendung funktioniert.

20.6. Alle beweglichen Teile

Figure 20.1 shows the structure of our person detection application.

Wie wir bereits erwähnt haben, ist dies viel einfacher als die Wake-Word-Anwendung, da wir die Bilddaten direkt in das Modell übergeben können - es ist keine Vorverarbeitung erforderlich.

Ein weiterer Aspekt, der die Dinge einfach hält, ist, dass wir die Ausgabe des Modells nicht mitteln. Unser Wake-Word-Modell wurde mehrmals pro Sekunde ausgeführt, so dass wir seine Ausgabe mitteln mussten, um ein stabiles Ergebnis zu erhalten. Unser Personenerkennungsmodell ist viel größer und braucht viel länger, um die Inferenz auszuführen. Das bedeutet, dass es keine Notwendigkeit gibt, seine Ausgabe zu mitteln. Der Code besteht aus fünf Hauptteilen:

Main loop

Wie die anderen Beispiele läuft auch unsere Anwendung in einer Endlosschleife. Da unser Modell jedoch viel größer und komplexer ist, dauert es länger, bis die Inferenz ausgeführt wird. Je nach Gerät können wir eine Inferenz alle paar Sekunden erwarten, anstatt mehrere Inferenzen pro Sekunde.

Image provider

Diese Komponente erfasst Bilddaten von der Kamera und schreibt sie in den Eingangstensor. Die Methoden zur Erfassung von Bildern variieren von Gerät zu Gerät, daher kann diese Komponente überschrieben und angepasst werden.

TensorFlow Lite interpreter

Der Interpreter führt das TensorFlow Lite Modell aus und transformiert das Eingabebild in einen Satz von Wahrscheinlichkeiten.

Modell

Das Modell wird als Datenarray eingebunden und vom Interpreter ausgeführt. Mit 250 KB ist dieses Modell unangemessen groß, um es an das TensorFlow GitHub Repository zu übergeben. Aus diesem Grund wird es vom Makefile heruntergeladen, wenn das Projekt gebaut wird. Wenn Sie einen Blick darauf werfen wollen, können Sie es selbst

```
1 namespace tflite {
2     namespace ops {
3         namespace micro {
4             TfLiteRegistration* Register_DEPTHWISE_CONV_2D();
5             TfLiteRegistration* Register_CONV_2D();
6             TfLiteRegistration* Register_AVERAGE_POOL_2D();
7         } // namespace micro
8     } // namespace ops
9 } // namespace tflite
```



```
1 const int tensor_arena_size = 70 * 1024;
2 uint8_t tensor_arena[tensor_arena_size];
```

unter [tf-lite-micro-person-data-grayscale.zip](#) herunterladen.

Detection responder

Der Erkennungsresponder nimmt die vom Modell ausgegebenen Wahrscheinlichkeiten und verwendet die Ausgabefähigkeiten des Geräts, um sie anzuzeigen. Wir können ihn für verschiedene Gerätetypen überschreiben. In unserem Beispielcode leuchtet eine LED auf, aber Sie können ihn so erweitern, dass er so ziemlich alles tun kann. Um ein Gefühl dafür zu bekommen, wie diese Teile zusammenpassen, werfen wir einen Blick auf ihre Tests.

Durch die Tests gehen

Diese Anwendung ist schön einfach, da es nur ein paar Tests zu durchlaufen gibt. Sie können sie alle im GitHub-Repository finden:

[person_detection_test.cc](#)

Zeigt, wie eine Inferenz auf ein Array, das ein einzelnes Bild repräsentiert, ausgeführt werden kann

[image_provider_test.cc](#)

Zeigt, wie Sie den Bildanbieter verwenden, um ein Bild aufzunehmen

[detection_responder_test.cc](#)

Zeigt, wie der Erkennungsresponder verwendet wird, um die Ergebnisse der Erkennung auszugeben.

Beginnen wir mit der Untersuchung von [person_detection_test.cc](#), um zu sehen, wie die Inferenz auf Bilddaten ausgeführt wird. Da dies das dritte Beispiel ist, das wir durchlaufen haben, sollte Ihnen dieser Code ziemlich vertraut vorkommen. Sie sind auf dem besten Weg, ein eingebetteter ML-Entwickler zu werden!

Der Grundablauf

Beginnen wir mit der Datei [Person_detection_test.cc](#). Wir beginnen damit, dass wir die Ops hineinziehen, die unser Modell benötigen wird:

Als nächstes definieren wir eine Tensor-Arena, die für das Modell angemessen groß ist. Wie üblich wurde diese Zahl durch Versuch und Irrtum ermittelt:

Anschließend führen wir die typischen Einrichtungsarbeiten durch, um den Interpreter einsatzbereit zu machen, wozu auch die Registrierung der erforderlichen Ops mit dem [MicroMutableOpResolver](#) gehört:

Unser nächster Schritt ist die Überprüfung des Eingabetensors. Wir prüfen, ob er die erwartete Anzahl von Dimensionen hat und ob seine Dimensionen angemessen dimensioniert sind:

Daraus können wir erkennen, dass die Eingabe technisch gesehen ein 5D-Tensor ist. Die erste Dimension ist nur ein Wrapper, der ein einzelnes Element enthält. Die folgenden zwei Dimensionen stellen die Zeilen und Spalten der Pixel des Bildes dar. Die letzte Dimension enthält die Anzahl der Farbkanäle, die zur Darstellung jedes Pixels verwendet werden.

```

1 // Set up logging.
2 tflite::MicroErrorReporter micro_error_reporter;
3 tflite::ErrorReporter* error_reporter = &micro_error_reporter;
4
5 // Map the model into a usable data structure. This doesn't involve any
6 // copying or parsing, it's a very lightweight operation.
7 const tflite::Model* model = ::tflite::GetModel(g_person_detect_model_data);
8 if (model->version() != TFLITE_SCHEMA_VERSION) {
9     error_reporter->Report(
10         "Model provided is schema version %d not equal to "
11         "supported version %d.\n",
12         model->version(), TFLITE_SCHEMA_VERSION);
13 }
14
15 // Pull in only the operation implementations we need.
16 tflite::MicroMutableOpResolver micro Mutable_op_resolver;
17 micro Mutable_op_resolver.AddBuiltin(
18     tflite::BuiltinOperator_DEPTHWISE_CONV_2D,
19     tflite::ops::micro::Register_DEPTHWISE_CONV_2D());
20 micro Mutable_op_resolver.AddBuiltin(tflite::BuiltinOperator_CONV_2D,
21     tflite::ops::micro::Register_CONV_2D());
22 micro Mutable_op_resolver.AddBuiltin(
23     tflite::BuiltinOperator_AVERAGE_POOL_2D,
24     tflite::ops::micro::Register_AVERAGE_POOL_2D());
25
26 // Build an interpreter to run the model with.
27 tflite::MicroInterpreter interpreter(model, micro Mutable_op_resolver,
28     tensor_arena, tensor_arena_size,
29     error_reporter);
30 interpreter.AllocateTensors();

```

```

1 // Get information about the memory area to use for the model's input.
2 TfLiteTensor* input = interpreter.input(0);
3
4 // Make sure the input has the properties we expect.
5 TF_LITE_MICRO_EXPECT_NE(nullptr, input);
6 TF_LITE_MICRO_EXPECT_EQ(4, input->dims->size);
7 TF_LITE_MICRO_EXPECT_EQ(1, input->dims->data[0]);
8 TF_LITE_MICRO_EXPECT_EQ(kNumRows, input->dims->data[1]);
9 TF_LITE_MICRO_EXPECT_EQ(kNumCols, input->dims->data[2]);
10 TF_LITE_MICRO_EXPECT_EQ(kNumChannels, input->dims->data[3]);
11 TF_LITE_MICRO_EXPECT_EQ(kTfLiteUInt8, input->type);
12 //
```

```

1 constexpr int kNumCols = 96;
2 constexpr int kNumRows = 96;
3 constexpr int kNumChannels = 1;

1 // Copy an image with a person into the memory area used for the input.
2 const uint8_t* person_data = g_person_data;
3 for (int i = 0; i < input->bytes; ++i) {
4     input->data.uint8[i] = person_data[i];
5 }
```

Die Konstanten, die die erwarteten Dimensionen angeben, `kNumRows`, `kNumCols` und `kNumChannels`, sind in `model_settings.h` definiert. Sie sehen folgendermaßen aus:

Wie Sie sehen können, wird erwartet, dass das Modell eine Bitmap mit 96×96 Pixeln akzeptiert. Das Bild wird in Graustufen dargestellt, mit einem Farbkanal für jedes Pixel.

Als nächstes im Code kopieren wir ein Testbild in den Eingangstensor mit einer einfachen for-Schleife:

Die Variable, die die Bilddaten speichert, `g_person_data`, wird in der Datei `person_image_data.h` definiert. Um zu vermeiden, dass dem Repository weitere große Dateien hinzugefügt werden, werden die Daten selbst zusammen mit dem Modell heruntergeladen, als Teil von `tf_lite_micro_person_data_grayscale.zip`, wenn die Tests zum ersten Mal ausgeführt werden.

Nachdem wir den Eingabetensor aufgefüllt haben, führen wir die Inferenz durch. Es ist so einfach wie immer:

Wir überprüfen nun den Ausgangstensor, um sicherzustellen, dass er die erwartete Größe und Form hat:

Die Ausgabe des Modells hat vier Dimensionen. Die ersten drei sind nur Umhüllungen für die vierte, die ein Element für jede Kategorie enthält, für die das Modell trainiert wurde.

Die Gesamtzahl der Kategorien ist als Konstante `kCategoryCount` verfügbar, die sich zusammen mit einigen anderen hilfreichen Werten in `model_settings.h` befindet:

Wie `kCategoryCount` zeigt, gibt es drei Kategorien in der Ausgabe. Die erste ist zufällig eine unbenutzte Kategorie, die wir ignorieren können. Die Kategorie "Person" kommt an zweiter Stelle, wie wir an ihrem Index erkennen können, der in der Konstante `kPersonIndex` gespeichert ist. An dritter Stelle steht die Kategorie "keine Person", deren Index durch `kNotAPersonIndex` angezeigt wird.

Es gibt auch ein Array von Kategoriebezeichnungen, `kCategoryLabels`, das in `model_settings.cc` implementiert ist:

```

1 // Run the model on this input and make sure it succeeds.
2 TfLiteStatus invoke_status = interpreter.Invoke();
3 if (invoke_status != kTfLiteOk) {
4     error_reporter->Report("Invoke failed\n");
5 }
6 TF_LITE_MICRO_EXPECT_EQ(kTfLiteOk, invoke_status);
```

```

1 TfLiteTensor* output = interpreter.output(0);
2 TF_LITE_MICRO_EXPECT_EQ(4, output->dims->size);
3 TF_LITE_MICRO_EXPECT_EQ(1, output->dims->data[0]);
4 TF_LITE_MICRO_EXPECT_EQ(1, output->dims->data[1]);
5 TF_LITE_MICRO_EXPECT_EQ(1, output->dims->data[2]);
6 TF_LITE_MICRO_EXPECT_EQ(kCategoryCount, output->dims->data[3]);
7 TF_LITE_MICRO_EXPECT_EQ(kTfLiteUInt8, output->type);

1 constexpr int kCategoryCount = 3;
2 constexpr int kPersonIndex = 1;
3 constexpr int kNotAPersonIndex = 2;
4 extern const char* kCategoryLabels[kCategoryCount];

```

20.7. Extra Dimensions

Wie `kCategoryCount` zeigt, gibt es drei Kategorien in der Ausgabe. Die erste ist zufällig eine unbenutzte Kategorie, die wir ignorieren können. Die Kategorie "Person" kommt an zweiter Stelle, wie wir an ihrem Index erkennen können, der in der Konstante `kPersonIndex` gespeichert ist. An dritter Stelle steht die Kategorie "keine Person", deren Index durch `kNotAPersonIndex` angezeigt wird.

Es gibt auch ein Array von Kategoriebezeichnungen, `kCategoryLabels`, das in `model_settings.cc` implementiert ist: Wie `kCategoryCount` zeigt, gibt es drei Kategorien in der Ausgabe. Die erste ist zufällig eine unbenutzte Kategorie, die wir ignorieren können. Die Kategorie "Person" kommt an zweiter Stelle, wie wir an ihrem Index erkennen können, der in der Konstante `kPersonIndex` gespeichert ist. An dritter Stelle steht die Kategorie "keine Person", deren Index durch `kNotAPersonIndex` angezeigt wird.

Es gibt auch ein Array von Kategoriebezeichnungen, `kCategoryLabels`, das in `model_settings.cc` implementiert ist:

Da der einzige Dateninhalt des Ausgabetensors die drei `uint8`-Werte sind, die die Klassenwerte darstellen, wobei der erste unbenutzt ist, können wir direkt auf die Werte zugreifen, indem wir `output->data.uint8[kPersonIndex]` und `output->data.uint8[kNotAPersonIndex]` verwenden. Als `uint8`-Typen haben sie einen Minimalwert von 0 und einen Maximalwert von 255.

20.8. NOTE

Wenn die Werte für "Person" und "keine Person" ähnlich sind, kann dies bedeuten, dass das Modell nicht sehr sicher in seiner Vorhersage ist. In diesem Fall können Sie das Ergebnis als nicht schlüssig betrachten.

Als nächstes testen wir auf ein Bild ohne Person, das von `g_no_person_data` gehalten wird:

Nachdem die Inferenz gelaufen ist, behaupten wir dann, dass der Wert "keine Person" höher ist:

```

1 const char* kCategoryLabels[kCategoryCount] = {
2     "unused",
3     "person",
4     "notperson",
5 };

```

```

1 uint8_t person_score = output->data.uint8[kPersonIndex];
2 uint8_t no_person_score = output->data.uint8[kNotAPersonIndex];
3 error_reporter->Report(
4 "person_data.person_score:%d,no_person_score:%d\n", person_score,
5 no_person_score);
6 TF_LITE_MICRO_EXPECT_GT(person_score, no_person_score);

1 const uint8_t* no_person_data = g_no_person_data;
2 for (int i = 0; i < input->bytes; ++i) {
3     input->data.uint8[i] = no_person_data[i];
4 }
```

Wie Sie sehen können, geht hier nichts Ausgefallenes vor sich. Wir geben zwar Bilder anstelle von Skalaren oder Spektrogrammen ein, aber der Prozess der Inferenz ist ähnlich zu dem, was wir zuvor gesehen haben.

Das Ausführen des Tests ist ähnlich einfach. Geben Sie einfach den folgenden Befehl aus dem Stammverzeichnis des TensorFlow-Repositorys ein:

Wenn der Test das erste Mal ausgeführt wird, werden das Modell und die Bilddaten heruntergeladen. Wenn Sie einen Blick auf die heruntergeladenen Dateien werfen möchten, finden Sie diese in [tensorflow/lite/micro/tools/make/-downloads/person_model_grayscale](#). Siehe [../Code/Nano33BLESense/person_detection_int8](#). Achtung ist eine C-Datei.

Als Nächstes schauen wir uns die Schnittstelle für den Bildanbieter an.

20.9. The Image Provider

Der Image-Provider ist dafür verantwortlich, Daten von der Kamera abzugreifen und sie in einem Format zurückzugeben, das für das Schreiben in den Eingabetensor des Modells geeignet ist. Die Datei `image_provider.h` definiert seine Schnittstelle: Da die eigentliche Implementierung plattformspezifisch ist, gibt es eine Referenzimplementierung in `person_detection/image_provider.cc`, die Dummy-Daten zurückgibt.

Der Test in `image_provider_test.cc` ruft diese Referenzimplementierung auf, um zu zeigen, wie sie verwendet wird. Als erstes müssen wir ein Array erstellen, das die Bilddaten enthält. Dies geschieht in der folgenden Zeile:

Die Konstante `kMaxImageSize` stammt aus unserem alten Freund, `model_settings.h`.

Nachdem wir dieses Array eingerichtet haben, können wir die Funktion `GetImage()` aufrufen, um ein Bild von der Kamera zu erfassen:

Wir rufen sie mit einer ErrorReporter-Instanz auf, mit der Anzahl der gewünschten Spalten, Zeilen und Kanäle und mit einem Zeiger auf unser Array `image_data`. Die Funktion wird die Bilddaten in dieses Array schreiben. Wir können den Rückgabewert

```

1 person_score = output->data.uint8[kPersonIndex];
2 no_person_score = output->data.uint8[kNotAPersonIndex];
3 error_reporter->Report(
4 "no_person_data.person_score:%d,no_person_score:%d\n", person_score,
5 no_person_score);
6 TF_LITE_MICRO_EXPECT_GT(no_person_score, person_score);
```

```
1 make -f tensorflow/lite/micro/tools/make/Makefile \
2 test_person_detection_test
```

```
1 TfLiteStatus GetImage( tflite::ErrorReporter* error_reporter, int image_width,
2 int image_height, int channels, uint8_t* image_data);
```

der Funktion überprüfen, um festzustellen, ob der Erfassungsprozess erfolgreich war; er wird auf `kTfLiteError` gesetzt, wenn es ein Problem gibt, oder andernfalls auf `kTfLiteOk`.

Schließlich geht der Test durch die zurückgegebenen Daten, um zu zeigen, dass alle Speicherplätze lesbar sind. Auch wenn das Bild technisch gesehen Zeilen, Spalten und Kanäle hat, werden die Daten in der Praxis in ein 1D-Array gepresst:

Um diesen Test auszuführen, verwenden Sie den folgenden Befehl:

Wir werden die gerätespezifischen Implementierungen von `image_provider.cc` später im Kapitel untersuchen; für den Moment wollen wir uns die Schnittstelle des Erkennungsresponders ansehen.

20.10. The Detection Responder

Unser letzter Test zeigt, wie der Erkennungsresponder verwendet wird. Dies ist der Code, der für die Übermittlung der Ergebnisse der Inferenz verantwortlich ist. Seine Schnittstelle ist in `detection_responder.h` definiert, und der Test befindet sich in `detection_responder_test.cc`.

Die Schnittstelle ist ziemlich einfach:

Wir rufen es einfach mit den Werten für die beiden Kategorien "Person" und "keine Person" auf, und es entscheidet, was von da an zu tun ist.

Die Referenzimplementierung in `detection_responder.cc` protokolliert nur diese Werte. Der Test in `detection_responder_test.cc` ruft die Funktion ein paar Mal auf:

Um den Test auszuführen und die Ausgabe zu sehen, verwenden Sie den folgenden Befehl:

Wir haben alle Tests und die Schnittstellen, die sie ausüben, erkundet. Lassen Sie uns nun das Programm selbst durchgehen.

20.11. Detecting People

Die Kernfunktionen der Anwendung befinden sich in `DATEImain_functions.cc`. Sie sind kurz und bündig, und wir haben viel von ihrer Logik in den Tests gesehen.

Zuerst holen wir uns alle Funktionen, die unser Modell benötigt:

Als nächstes deklarieren wir eine Reihe von Variablen, um die wichtigen beweglichen Teile zu halten:

Danach weisen wir etwas Arbeitsspeicher für Tensoroperationen zu:

In der Funktion `function`, die ausgeführt wird, bevor irgendetwas anderes passiert, erstellen wir einen Fehlerreporter, laden unser Modell, richten eine Interpreterinstanz ein und holen uns eine Referenz auf den Eingabetensor des Modells:

```
1 uint8_t image_data[kMaxImageSize];
```

```

1 TfLiteStatus get_status =
2 GetImage(error_reporter, kNumCols, kNumRows, kNumChannels, image_data);
3 TF_LITE_MICRO_EXPECT_EQ(kTfLiteOk, get_status);
4 TF_LITE_MICRO_EXPECT_NE(image_data, nullptr);

1 uint32_t total = 0;
2 for (int i = 0; i < kMaxImageSize; ++i) {
3     total += image_data[i];
4 }

1 make -f tensorflow/lite/micro/tools/make/Makefile \
2 test_image_provider_test

1 void RespondToDetection(tflite::ErrorReporter* error_reporter,
2 uint8_t person_score, uint8_t no_person_score);

1 RespondToDetection(error_reporter, 100, 200);
2 RespondToDetection(error_reporter, 200, 100);

1 make -f tensorflow/lite/micro/tools/make/Makefile \
2 test_detection_responder_test

1 namespace tflite {
2     namespace ops {
3         namespace micro {
4             TfliteRegistration* Register_DEPTHWISE_CONV_2D();
5             TfliteRegistration* Register_CONV_2D();
6             TfliteRegistration* Register_AVERAGE_POOL_2D();
7         } // namespace micro
8     } // namespace ops
9 } // namespace tflite

1 tflite::ErrorReporter* g_error_reporter = nullptr;
2 const tflite::Model* g_model = nullptr;
3 tflite::MicroInterpreter* g_interpreter = nullptr;
4 TfLiteTensor* g_input = nullptr;

1 constexpr int g_tensor_arena_size = 70 * 1024;
2 static uint8_t tensor_arena[kTensorArenaSize];

```

```
1 void setup() {
2     // Set up logging.
3     static tflite::MicroErrorReporter micro_error_reporter;
4     g_error_reporter = &micro_error_reporter;
5
6     // Map the model into a usable data structure. This doesn't involve any
7     // copying or parsing, it's a very lightweight operation.
8     g_model = tflite::GetModel(g_person_detect_model_data);
9     if (g_model->version() != TFLITE_SCHEMA_VERSION) {
10         g_error_reporter->Report(
11             "Model provided is schema version %d not equal to supported version %d.",
12             g_model->version(), TFLITE_SCHEMA_VERSION);
13         return;
14     }
15
16     // Pull in only the operation implementations we need.
17     static tflite::MicroMutableOpResolver micro_mutable_op_resolver;
18     micro_mutable_op_resolver.AddBuiltin(
19         tflite::BuiltinOperator_DEPTHWISE_CONV_2D,
20         tflite::ops::micro::Register_DEPTHWISE_CONV_2D());
21     micro_mutable_op_resolver.AddBuiltin(tflite::BuiltinOperator_CONV_2D,
22                                         tflite::ops::micro::Register_CONV_2D());
23     micro_mutable_op_resolver.AddBuiltin(
24         tflite::BuiltinOperator_AVERAGE_POOL_2D,
25         tflite::ops::micro::Register_AVERAGE_POOL_2D());
26
27     // Build an interpreter to run the model with.
28     static tflite::MicroInterpreter static_interpreter(
29         model, micro_mutable_op_resolver, tensor_arena, kTensorArenaSize,
30         error_reporter);
31     interpreter = &static_interpreter;
32
33     // Allocate memory from the tensor_arena for the model's tensors.
34     TfLiteStatus allocate_status = interpreter->AllocateTensors();
35     if (allocate_status != kTfLiteOk) {
36         error_reporter->Report("AllocateTensors() failed");
37     }
38     return;
39 }
40
41 // Get information about the memory area to use for the model's input.
42     input = interpreter->input(0);
43 }
```

```

1 void loop() {
2     // Get image from provider.
3     if (kTfLiteOk != GetImage(g_error_reporter, kNumCols, kNumRows, kNumChannels,
4     g_input->data.uint8)) {
5         g_error_reporter->Report("Image capture failed.");
6     }
7
8     // Run the model on this input and make sure it succeeds.
9     if (kTfLiteOk != g_interpreter->Invoke()) {
10         g_error_reporter->Report("Invoke failed.");
11     }
12
13     TfLiteTensor* output = g_interpreter->output(0);
14
15     // Process the inference results.
16     uint8_t person_score = output->data.uint8[kPersonIndex];
17     uint8_t no_person_score = output->data.uint8[kNotAPersonIndex];
18     RespondToDetection(g_error_reporter, person_score, no_person_score);
19 }
```

Der nächste Teil des Codes wird kontinuierlich in der Hauptschleife des Programms aufgerufen. Er holt sich zunächst ein Bild über den Image-Provider und übergibt eine Referenz auf den Input-Tensor, damit das Bild direkt dorthin geschrieben wird: Dann wird die Inferenz ausgeführt, der Ausgabetensor abgerufen und die "Person"- und "Keine Person"-Bewertungen daraus gelesen. Diese Scores werden an die Funktion `RespondToDetection()` des Erkennungsresponders übergeben:

Nachdem `RespondToDetection()` die Ausgabe der Ergebnisse beendet hat, kehrt die Funktion `loop()` zurück und ist bereit, von der Hauptschleife des Programms erneut aufgerufen zu werden.

Die Schleife selbst wird in der Funktion `main()` des Programms definiert, die sich in `main.cc` befindet. Sie ruft die Funktion `setup()` einmal auf und ruft dann die Funktion `loop()` wiederholt und unendlich oft auf:

Und das ist das gesamte Programm! Dieses Beispiel ist großartig, weil es zeigt, dass die Arbeit mit anspruchsvollen maschinellen Lernmodellen überraschend einfach sein kann. Das Modell enthält die gesamte Komplexität, und wir müssen es nur mit Daten füttern.

Bevor wir weitermachen, können Sie das Programm lokal ausführen, um es auszuprobieren. Die Referenzimplementierung des Bildanbieters gibt nur Dummy-Daten zurück, so dass Sie keine aussagekräftigen Erkennungsergebnisse erhalten werden, aber Sie werden zumindest den Code bei der Arbeit sehen.

Verwenden Sie zunächst diesen Befehl, um das Programm zu erstellen:

Sobald der Build abgeschlossen ist, können Sie das Beispiel mit dem folgenden Befehl ausführen:

```

1 int main(int argc, char* argv[]) {
2     setup();
3     while (true) {
4         loop();
5     }
6 }
```

```

1 make -f tensorflow/lite/micro/tools/make/Makefile person_detection

1 tensorflow/lite/micro/tools/make/gen/osx_x86_64/bin/ \
2 person_detection

```

Sie werden sehen, wie die Ausgabe des Programms vorbeiläuft, bis Sie es mit Strg-C beenden:

Im nächsten Abschnitt gehen wir den gerätespezifischen Code durch, der Kamerabilder erfasst und die Ergebnisse auf jeder Plattform ausgibt. Wir zeigen auch, wie Sie diesen Code einsetzen und ausführen.

20.12. Einsatz auf Mikrocontrollern

In diesem Abschnitt stellen wir unseren Code auf zwei bekannten Geräten bereit:

- Arduino Nano 33 BLE Sense
- SparkFun Edge

Diesmal gibt es einen großen Unterschied: Da keines dieser Geräte eine eingebaute Kamera hat, empfehlen wir Ihnen, ein Kameramodul für das jeweilige Gerät zu kaufen. Jedes Gerät hat seine eigene Implementierung von `image_provider.cc`, die mit dem Kameramodul zusammenarbeitet, um Bilder zu erfassen. Es gibt auch einen gerätespezifischen Ausgabecode in `detection_responder.cc`.

Dies ist schön und einfach, so dass es eine ausgezeichnete Vorlage für die Erstellung Ihrer eigenen bildverarbeitungsbasierten ML-Anwendungen ist.

Beginnen wir mit der Erkundung der Arduino-Implementierung.

20.13. Arduino

Als Arduino-Board hat das Arduino Nano 33 BLE Sense Zugriff auf ein riesiges Ökosystem kompatibler Hardware und Bibliotheken von Drittanbietern. Wir verwenden ein Kameramodul eines Drittanbieters, das für die Zusammenarbeit mit Arduino entwickelt wurde, zusammen mit einigen Arduino-Bibliotheken, die eine Schnittstelle zu unserem Kameramodul bilden und die von diesem ausgegebenen Daten sinnvoll nutzen.

20.14. Welches Kameramodul Sie kaufen sollten

Dieses Beispiel verwendet das Arducam Mini 2MP Plus Kameramodul. Es lässt sich einfach an den Arduino Nano 33 BLE Sense anschließen und kann über die Stromversorgung des Arduino-Boards mit Strom versorgt werden. Es hat ein großes Objektiv

```

1 person score:129 no person score 202
2 person score:129 no person score 202
3 person score:129 no person score 202
4 person score:129 no person score 202
5 person score:129 no person score 202
6 person score:129 no person score 202

```

```
1 static bool g_is_camera_initialized = false;
2 if (!g_is_camera_initialized) {
3     TfLiteStatus init_status = InitCamera(error_reporter);
4     if (init_status != kTfLiteOk) {
5         error_reporter->Report("InitCamera failed");
6         return init_status;
7     }
8     g_is_camera_initialized = true;
9 }
```

und kann qualitativ hochwertige 2-Megapixel-Bilder aufnehmen - allerdings werden wir die integrierte Bildskalierungsfunktion verwenden, um eine kleinere Auflösung zu erhalten. Sie ist nicht besonders stromsparend, aber ihre hohe Bildqualität macht sie ideal für den Aufbau von Bildaufnahme-Applikationen, z. B. für die Aufnahme von Wildtieren.

20.15. Aufnehmen von Bildern mit dem Arduino

Wir verbinden das Arducam-Modul über eine Reihe von Pins mit dem Arduino-Board. Um Bilddaten zu erhalten, senden wir einen Befehl vom Arduino-Board an die Arducam, der sie anweist, ein Bild aufzunehmen. Das Arducam wird dies tun und das Bild in seinem internen Datenpuffer speichern. Anschließend senden wir weitere Befehle, die es uns ermöglichen, die Bilddaten aus dem internen Puffer der Arducam zu lesen und sie im Speicher des Arduino zu speichern. Um all dies zu tun, verwenden wir die offizielle Arducam-Bibliothek.

Das Arducam-Kameramodul hat einen 2-Megapixel-Bildsensor, mit einer Auflösung von 1920×1080 . Unser Personenerkennungsmodell hat eine Eingabegröße von nur 96×96 , so dass wir nicht alle diese Daten benötigen. Tatsächlich hat der Arduino selbst nicht genug Speicher, um ein 2-Megapixel-Bild zu speichern, das mehrere Megabyte groß wäre.

Glücklicherweise ist die Arducam-Hardware in der Lage, ihre Ausgabe auf eine viel kleinere Auflösung von 160×120 Pixel zu verkleinern. Wir können dies in unserem Code leicht auf 96×96 verkleinern, indem wir nur die zentralen 96×96 Pixel beibehalten. Erschwerend kommt hinzu, dass die verkleinerte Ausgabe der Arducam mit JPEG kodiert ist, einem gängigen Kompressionsformat für Bilder. Unser Modell benötigt ein Array von Pixeln, kein JPEG-codiertes Bild, was bedeutet, dass wir die Ausgabe der Arducam dekodieren müssen, bevor wir sie verwenden. Wir können dies mit einer Open-Source-Bibliothek tun.

Unsere letzte Aufgabe besteht darin, die Farbbildausgabe der Arducam in Graustufen zu konvertieren, was unser Modell zur Personenerkennung erwartet. Wir werden die Graustufendaten in den Eingabetensor unseres Modells schreiben.

Der Image-Provider ist in [arduino/image_provider.cc](#) implementiert. Wir werden hier nicht jedes Detail erklären, da der Code spezifisch für das Arducam-Kameramodul ist. Gehen wir stattdessen Schritt für Schritt durch, was auf hoher Ebene passiert.

Die Funktion [GetImage\(\)](#) ist die Schnittstelle des Bildanbieters zur Außenwelt. Sie wird in der Hauptschleife unserer Anwendung aufgerufen, um einen Frame mit Bilddaten zu erhalten. Wenn sie das erste Mal aufgerufen wird, müssen wir die Kamera initialisieren. Dies geschieht mit einem Aufruf der Funktion [InitCamera\(\)](#), wie folgt:

Die Funktion [InitCamera\(\)](#) ist weiter oben in [image_provider.cc](#) definiert. Wir werden sie hier nicht durchgehen, da sie sehr gerätespezifisch ist, und wenn

```
1 TfLiteStatus capture_status = PerformCapture(error_reporter);

1 TfLiteStatus read_data_status = ReadData(error_reporter);
```

Sie sie in Ihrem eigenen Code verwenden möchten, können Sie sie einfach kopieren und einfügen. Es konfiguriert die Arduino-Hardware für die Kommunikation mit der Arducam und bestätigt dann, dass die Kommunikation funktioniert. Schließlich weist sie die Arducam an, 160×120 -Pixel-JPEG-Bilder auszugeben.

Die nächste Funktion, die von `GetImage()` aufgerufen wird, ist `PerformCapture()`:

Wir werden auch hier nicht ins Detail gehen. Es sendet lediglich einen Befehl an das Kameramodul, der es anweist, ein Bild aufzunehmen und die Bilddaten in seinem internen Puffer zu speichern. Dann wartet es auf die Bestätigung, dass ein Bild aufgenommen wurde. Zu diesem Zeitpunkt warten die Bilddaten im internen Puffer der Arducam, aber es gibt noch keine Bilddaten auf dem Arduino selbst.

Die nächste Funktion, die wir aufrufen, ist `ReadData()`:

Die Funktion `ReadData()` verwendet weitere Befehle, um die Bilddaten von der Arducam zu holen. Nachdem die Funktion ausgeführt wurde, wird die globale Variable `jpeg_buffer` mit den von der Kamera abgerufenen JPEG-kodierten Bilddaten gefüllt.

Wenn wir das JPEG-kodierte Bild haben, ist unser nächster Schritt, es in rohe Bilddaten zu dekodieren. Dies geschieht mit der Funktion `DecodeAndProcessImage()`:

Die Funktion verwendet eine Bibliothek namens `JPEGDecoder`, um die JPEG-Daten zu dekodieren und direkt in den Eingangstensor des Modells zu schreiben. Dabei beschneidet sie das Bild und verwirft einige der 160×120 Daten, so dass nur 96×96 Pixel übrig bleiben, etwa in der Mitte des Bildes. Außerdem wird die 16-Bit-Farbdarstellung des Bildes auf 8-Bit-Graustufen reduziert.

Nachdem das Bild erfasst und im Eingabetensor gespeichert wurde, können wir die Inferenz ausführen. Als nächstes zeigen wir, wie die Ausgabe des Modells angezeigt wird

20.16. Reagieren auf Erkennungen am Arduino

Der Arduino Nano 33 BLE Sense verfügt über eine eingebaute RGB-LED, d. h. eine einzelne Komponente, die unterschiedliche rote, grüne und blaue LEDs enthält, die Sie separat steuern können. Bei der Implementierung des Erkennungsresponders blinkt die blaue LED jedes Mal, wenn eine Inferenz ausgeführt wird. Wenn eine Person erkannt wird, leuchtet die grüne LED; wenn eine Person nicht erkannt wird, leuchtet die rote LED.

Die Implementierung befindet sich in `arduino/detection_responder.cc`. Lassen Sie uns kurz durchgehen.

Die Funktion `RespondToDetection()` akzeptiert zwei Wertungen, eine für die Kategorie "Person" und die andere für "keine Person". Beim ersten Aufruf werden die blauen, grünen und gelben LEDs für die Ausgabe eingerichtet:

Um anzuzeigen, dass eine Inferenz gerade abgeschlossen wurde, schalten wir als nächstes alle LEDs aus und lassen dann die blaue LED ganz kurz aufblinken:

```
1 TfLiteStatus decode_status = DecodeAndProcessImage(
2 error_reporter, image_width, image_height, image_data);
```

```

1 void RespondToDetection( tflite :: ErrorReporter* error_reporter ,
2   uint8_t person_score , uint8_t no_person_score ) {
3     static bool is_initialized = false ;
4     if ( !is_initialized ) {
5       pinMode(led_green , OUTPUT) ;
6       pinMode(led_blue , OUTPUT) ;
7       is_initialized = true ;
8     }
9
10 // Note: The RGB LEDs on the Arduino Nano 33 BLE
11 // Sense are on when the pin is LOW, off when HIGH .
12
13 // Switch the person/not person LEDs off
14 digitalWrite(led_green , HIGH) ;
15 digitalWrite(led_red , HIGH) ;
16
17 // Flash the blue LED after every inference .
18 digitalWrite(led_blue , LOW) ;
19 delay(100) ;
20 digitalWrite(led_blue , HIGH) ;

```

Sie werden feststellen, dass diese LEDs im Gegensatz zu den eingebauten LEDs des Arduinos mit LOW eingeschaltet und mit HIGH ausgeschaltet werden. Das liegt einfach daran, wie die LEDs mit der Platine verbunden sind.

Als Nächstes schalten wir die entsprechenden LEDs ein und aus, je nachdem, welcher Kategorienwert höher ist:

Schließlich verwenden wir die Instanz `error_reporter`, um die Ergebnisse auf der seriellen Schnittstelle auszugeben:

Und das war's! Der Kern der Funktion ist eine einfache if-Anweisung, und Sie könnten leicht eine ähnliche Logik verwenden, um andere Arten von Ausgaben zu steuern. Es hat etwas sehr Spannendes, wenn eine so komplexe visuelle Eingabe in eine einzige boolesche Ausgabe umgewandelt wird: "Person" oder "keine Person".

20.17. Ausführen des Beispiels

Die Ausführung dieses Beispiels ist ein wenig komplexer als unsere anderen Arduino-Beispiele, da wir die Arducam mit dem Arduino-Board verbinden müssen. Außerdem müssen wir die Bibliotheken installieren und konfigurieren, die die Schnittstelle zur Arducam bilden und deren JPEG-Ausgabe dekodieren. Aber keine Sorge, es ist trotzdem sehr einfach!

```

1 // Switch on the green LED when a person is detected ,
2 // the red when no person is detected
3 if ( person_score > no_person_score ) {
4   digitalWrite(led_green , LOW) ;
5   digitalWrite(led_red , HIGH) ;
6 } else {
7   digitalWrite(led_green , HIGH) ;
8   digitalWrite(led_red , LOW) ;
9 }

```

```

1     error_reporter->Report( "Person(score : %d)No(score : %d)", person_score,
2     no_person_score );
3 }
```

Arducam pin	Arduino pin
CS	D7 (unlabeled, immediately to the right of D6)
MOSI	D11
MISO	D12
SCK	D13
GND	GND (either pin marked GND is fine)
VCC	3.3 V
SDA	A4
SCL	A5

Table 20.1.: Arducam Mini 2MP Plus to Arduino Nano 33 BLE Sense connections

Um dieses Beispiel zu implementieren, benötigen wir Folgendes:

- An Arduino Nano 33 BLE Sense board
- An Arducam Mini 2MP Plus
- Jumper cables (and optionally a breadboard)
- A micro-USB cable
- The Arduino IDE

Unsere erste Aufgabe besteht darin, das Arducam mit Hilfe von Jumper-Kabeln an den Arduino anzuschließen. Dies ist kein Elektronikbuch, daher werden wir nicht auf die Details der Verwendung der Kabel eingehen. Stattdessen zeigt Tabelle 20.1, wie die Pins angeschlossen werden sollten. Die Pins sind auf jedem Gerät beschriftet. Nachdem Sie die Hardware eingerichtet haben, können Sie mit der Installation der Software fortfahren.

20.18. Tip

Es besteht immer die Möglichkeit, dass sich der Erstellungsprozess geändert hat, seit dieses Buch geschrieben wurde, also überprüfen Sie [README.md](#) auf die neuesten Anweisungen.

Die Projekte in diesem Buch sind als Beispielcode in der TensorFlow Lite Arduino-Bibliothek verfügbar. Wenn Sie die Bibliothek noch nicht installiert haben, öffnen Sie die Arduino IDE und wählen Sie Bibliotheken verwalten aus dem Menü Werkzeuge. Suchen Sie im erscheinenden Fenster nach der Bibliothek mit dem Namen [Arduino_TensorFlowLite](#) und installieren Sie sie. Sie sollten in der Lage sein, die neueste Version zu verwenden, aber wenn Sie auf Probleme stoßen, ist die Version, die mit diesem Buch getestet wurde, 1.14-ALPHA.

20.19. Bemerkung

Sie können die Bibliothek auch aus einer .zip-Datei installieren, die Sie entweder vom TensorFlow Lite Team herunterladen oder mit dem TensorFlow Lite for Microcontrollers Makefile selbst erzeugen können. Wenn Sie letzteres bevorzugen, lesen Sie Anhang A.

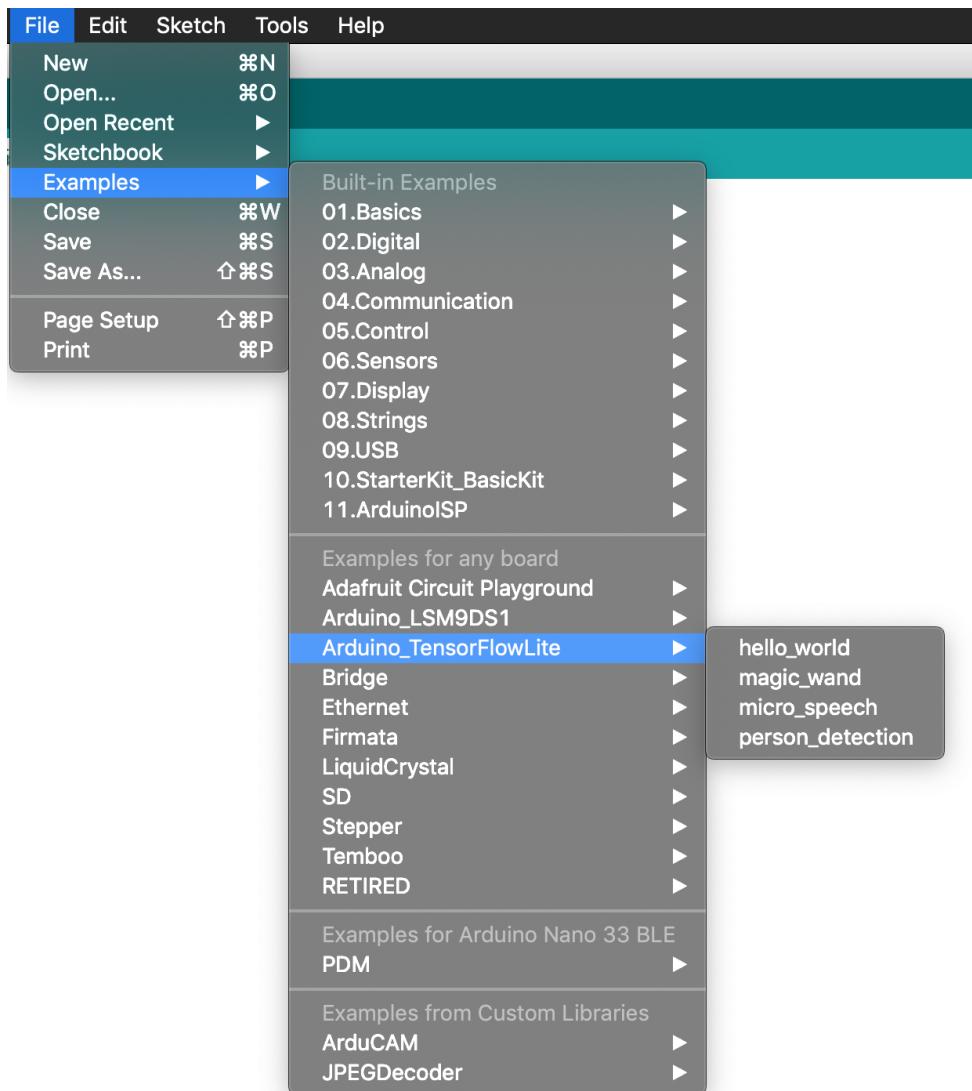


Figure 20.2.: Screenshot des Menüs 'Examples'

Nachdem Sie die Bibliothek installiert haben, erscheint das Beispiel `person_detection` im Menü File unter Examples->Arduino_TensorFlowLite, wie in Abbildung 20.2 gezeigt.

Klicken Sie auf „`person_detection`“, um das Beispiel zu laden. Es erscheint ein neues Fenster mit einer Registerkarte für jede der Quelldateien. Die Datei in der ersten Registerkarte, `person_detection`, entspricht der Datei `main_functions.cc`, die wir zuvor durchgelesen haben.

20.20. Bemerkung

"Das Ausführen des Beispiels" hat bereits die Struktur des Arduino-Beispiels erklärt, daher werden wir es hier nicht noch einmal behandeln.

Zusätzlich zur TensorFlow-Bibliothek müssen wir zwei weitere Bibliotheken installieren:

1. Die Bibliothek `Arducam`, damit unser Code mit der Hardware interagieren kann

```

1 //Step 1: select the hardware platform , only one at a time
2 //#define OV2640_MINI_2MP
3 //#define OV3640_MINI_3MP
4 //#define OV5642_MINI_5MP
5 //#define OV5642_MINI_5MP_BIT_ROTATION_FIXED
6 #define OV2640_MINI_2MP_PLUS
7 //#define OV5642_MINI_5MP_PLUS
8 //#define OV5640_MINI_5MP_PLUS

1 // Comment out the next #defines if you are not using an SD Card to store
2 // the JPEGs
3 // Commenting out the line is NOT essential but will save some FLASH space if
4 // SD Card access is not needed. Note: use of SdFat is currently untested!
5
6 //#define LOAD_SD_LIBRARY // Default SD Card library
7 //#define LOAD_SDFAT_LIBRARY // Use SdFat library instead , so SD Card SPI can
8 // be bit bashed

```

2. Die Bibliothek [JPEGDecoder](#), damit wir JPEG-kodierte Bilder dekodieren können

Die Arducam-Arduino-Bibliothek ist auf GitHub verfügbar. Um sie zu installieren, laden Sie das Repository herunter oder klonen Sie es. Kopieren Sie anschließend das Unterverzeichnis [ArduCAM](#) in Ihr Verzeichnis [Arduino/libraries](#). Um das Bibliotheksverzeichnis auf Ihrem Rechner zu finden, überprüfen Sie den Sketchbook-Speicherort im Voreinstellungsfenster der Arduino IDE.

Nachdem Sie die Bibliothek heruntergeladen haben, müssen Sie eine ihrer Dateien bearbeiten, um sicherzustellen, dass sie für die Arducam Mini 2MP Plus konfiguriert ist. Öffnen Sie dazu [Arduino/libraries/ArduCAM/memoriesaver.h](#).

Sie sollten eine Reihe von **#define**-Anweisungen aufgelistet sehen. Stellen Sie sicher, dass sie alle auskommentiert sind, außer **#define OV2640_MINI_2MP_PLUS**, wie hier gezeigt:

Nachdem Sie die Datei gespeichert haben, sind Sie mit der Konfiguration der Arducam-Bibliothek fertig.

20.21. Tip

Das Beispiel wurde mit [Commit #e216049](#) der Arducam-Bibliothek entwickelt. Wenn Sie Probleme mit der Bibliothek haben, können Sie versuchen, diesen speziellen Commit herunterzuladen, um sicherzustellen, dass Sie genau denselben Code verwenden.

Der nächste Schritt ist die Installation der Bibliothek [JPEGDecoder](#). Sie können dies aus der Arduino-IDE heraus tun. Wählen Sie im Menü "Tools" die Option "Manage Libraries" und suchen Sie nach [JPEGDecoder](#). Sie sollten die Version 1.8.0 der Bibliothek installieren.

Nachdem Sie die Bibliothek installiert haben, müssen Sie sie konfigurieren, um einige optionale Komponenten zu deaktivieren, die nicht mit dem Arduino Nano 33 BLE Sense kompatibel sind. Öffnen Sie [Arduino/libraries/JPEGDecoder/src/User_Config.h](#) und stellen Sie sicher, dass sowohl **#define LOAD_SD_LIBRARY** als auch **#define LOAD_SDFAT_LIBRARY** auskommentiert sind, wie in diesem Auszug aus der Datei gezeigt:

Nachdem Sie die Datei gespeichert haben, ist die Installation der Bibliotheken abgeschlossen. Sie sind nun bereit, die Anwendung zur Personenerkennung auszuführen!

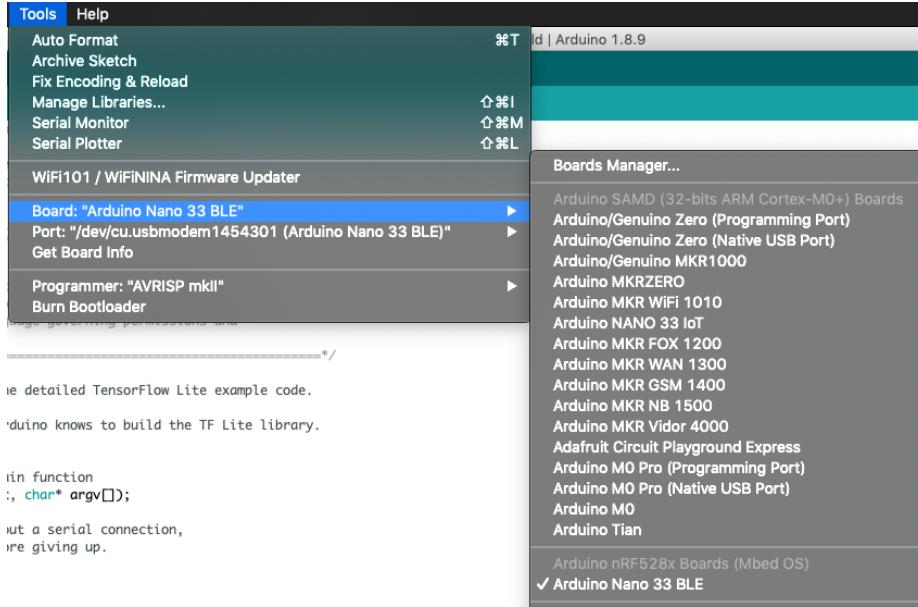


Figure 20.3.: Screenshot of the 'Board' dropdown

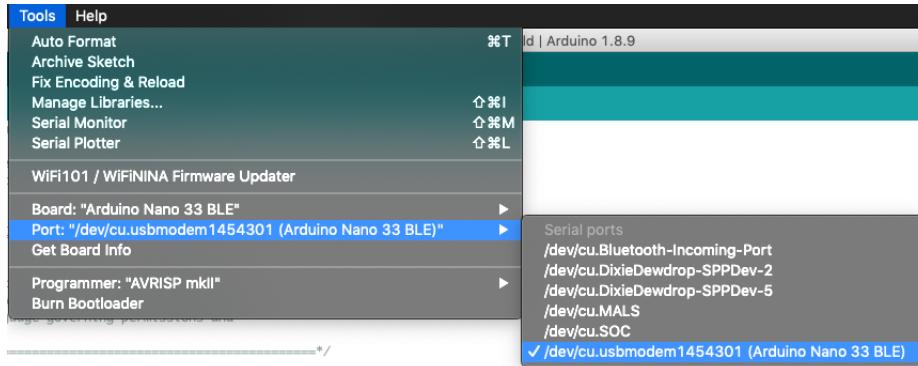


Figure 20.4.: Screenshot of the 'Port' dropdown

Schließen Sie zunächst Ihr Arduino-Gerät über USB an. Vergewissern Sie sich, dass der richtige Gerätetyp in der Dropdown-Liste "Board" im Menü "Tools" ausgewählt ist, wie in Abbildung 20.3 gezeigt.

Wenn der Name Ihres Geräts nicht in der Liste erscheint, müssen Sie sein Support-Paket installieren. Klicken Sie dazu auf Boards Manager. Suchen Sie in dem daraufhin angezeigten Fenster nach Ihrem Gerät und installieren Sie die neueste Version des entsprechenden Support-Pakets.

Vergewissern Sie sich auch im Menü Tools, dass der Anschluss des Geräts in der Dropdown-Liste Port ausgewählt ist, wie in Abbildung 20.5 gezeigt.

Klicken Sie schließlich im Arduino-Fenster auf die Schaltfläche "Hochladen" (in Abbildung 20.5 weiß hervorgehoben), um den Code zu kompilieren und auf Ihr Arduino-Gerät hochzuladen.

Sobald der Upload erfolgreich abgeschlossen ist, wird das Programm ausgeführt.

Um es zu testen, beginnen Sie damit, die Kamera des Geräts auf etwas zu richten, das definitiv keine Person ist oder nur das Objektiv verdeckt. Wenn die blaue LED das nächste Mal blinkt, nimmt das Gerät ein Bild von der Kamera auf und beginnt, die Inferenz auszuführen. Da das Bildverarbeitungsmodell, das wir für die Personenerkennung verwenden, relativ groß ist, wird die Inferenz sehr lange

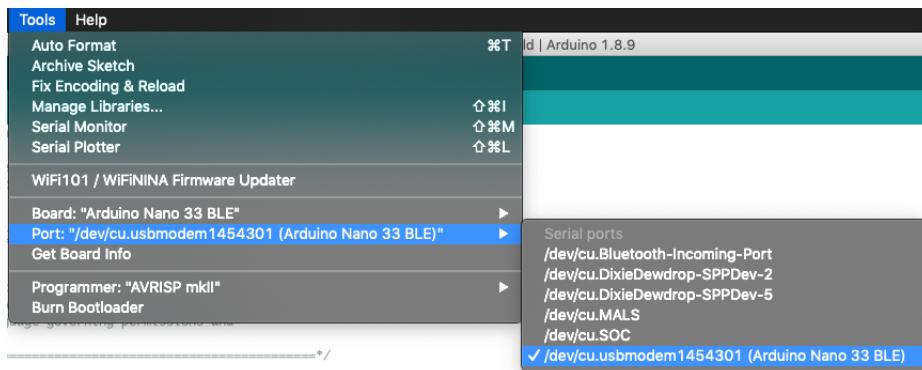


Figure 20.5.: Screenshot of the upload button

```

1 14:17:50.714 -> Starting capture
2 14:17:50.714 -> Image captured
3 14:17:50.784 -> Reading 3080 bytes from ArduCAM
4 14:17:50.887 -> Finished reading
5 14:17:50.887 -> Decoding JPEG and converting to greyscale
6 14:17:51.074 -> Image decoded and processed
7 14:18:09.710 -> Person score: 246 No person score: 66

```

dauern - etwa 19 Sekunden zum Zeitpunkt des Schreibens, obwohl es möglich ist, dass TensorFlow Lite seitdem schneller geworden ist.

Wenn die Inferenz abgeschlossen ist, wird das Ergebnis in eine weitere leuchtende LED übersetzt. Sie haben die Kamera auf etwas gerichtet, das keine Person ist, also sollte die rote LED aufleuchten.

Versuchen Sie nun, die Kamera des Geräts auf sich selbst zu richten! Wenn die blaue LED das nächste Mal blinkt, nimmt das Gerät ein weiteres Bild auf und beginnt, die Inferenz durchzuführen. Nach etwa 19 Sekunden sollte die grüne LED aufleuchten.

Denken Sie daran, dass die Bilddaten vor jeder Inferenz als Schnappschuss aufgenommen werden, wenn die blaue LED blinks. Das, worauf die Kamera in diesem Moment gerichtet ist, wird in das Modell eingespeist. Es spielt keine Rolle, worauf die Kamera gerichtet ist, bis das nächste Mal ein Bild aufgenommen wird, wenn die blaue LED wieder blinks.

Wenn Sie scheinbar falsche Ergebnisse erhalten, stellen Sie sicher, dass Sie sich in einer Umgebung mit guter Beleuchtung befinden. Sie sollten auch sicherstellen, dass die Kamera richtig ausgerichtet ist, mit den Stiften nach unten, so dass die Bilder, die sie aufnimmt, richtig herum sind - das Modell wurde nicht darauf trainiert, auf dem Kopf stehende Personen zu erkennen. Außerdem sollten Sie bedenken, dass es sich um ein winziges Modell handelt, bei dem die Genauigkeit gegen die geringe Größe eingetauscht wird. Es funktioniert sehr gut, aber es ist nicht immer zu 100%

Sie können die Ergebnisse der Inferenz auch über den Arduino Serial Monitor sehen. Öffnen Sie dazu im Menü "Tools" den "Serial Monitor". Sie werden ein detailliertes Protokoll sehen, das zeigt, was passiert, während die Anwendung läuft. Es ist auch interessant, das Kästchen "Show timestamp" (Zeitstempel anzeigen) zu aktivieren, damit Sie sehen können, wie lange jeder Teil des Prozesses dauert:

Aus diesem Protokoll ist ersichtlich, dass das Erfassen und Lesen der Bilddaten vom Kameramodul ca. 170 ms, das Dekodieren des JPEGs und die Umwandlung in Graustufen 180 ms und die Durchführung der Inferenz 18,6 Sekunden dauerte.

20.22. Ausführung von Änderungen

Jetzt, wo Sie die Basisanwendung bereitgestellt haben, können Sie ein wenig herumspielen und einige Änderungen am Code vornehmen. Bearbeiten Sie einfach die Dateien in der Arduino-IDE und speichern Sie sie, und wiederholen Sie dann die vorherigen Anweisungen, um den geänderten Code auf dem Gerät einzusetzen.

Hier sind ein paar Dinge, die Sie ausprobieren können:

- Ändern Sie den Erkennungsresponder so, dass er mehrdeutige Eingaben ignoriert, bei denen es keinen großen Unterschied zwischen den Bewertungen "Person" und "keine Person" gibt.
- Verwenden Sie die Ergebnisse der Personenerkennung, um andere Komponenten zu steuern, z. B. zusätzliche LEDs oder Servos.
- Bauen Sie eine intelligente Sicherheitskamera, indem Sie Bilder speichern oder übertragen - aber nur solche, die eine Person enthalten.

[Cho+19]

https://github.com/tensorflow/tensorflow/tree/master/tensorflow/lite/micro/examples/person_detection

20.23. Laufen auf Arduino

Die folgenden Anweisungen helfen Ihnen, dieses Beispiel zu erstellen und auf Arduino-Geräten einzusetzen.

Das Beispiel wurde mit dem folgenden Gerät getestet:

Arduino Nano 33 BLE Sense Sie benötigen außerdem das folgende Kameramodul:

Arducam Mini 2MP Plus

Hardware Schließen Sie die Pins der Arducam wie folgt an:

Arducam pin name	Arduino pin name
CS	D7 (unlabelled, immediately to the right of D6)
MOSI	D11
MISO	D12
SCK	D13
GND	GND (either pin marked GND is fine)
VCC	3.3 V
SDA	A4
SCL	A5

Installieren Sie die Bibliothek [Arduino_TensorFlowLite](#)

Laden Sie den aktuellen nächtlichen Build der Bibliothek herunter: [person_detection.zip](#)

Diese Beispielanwendung ist als Teil der offiziellen TensorFlow Lite Arduino-Bibliothek enthalten. Um sie zu installieren, öffnen Sie den Arduino Bibliotheksmanager in Tools -> Manage Libraries... und suchen Sie nach [Arduino_TensorFlowLite](#).

20.24. Installation anderer Bibliotheken

Zusätzlich zur TensorFlow-Bibliothek müssen Sie noch zwei weitere Bibliotheken installieren:

Die Arducam-Bibliothek, damit unser Code mit der Hardware kommunizieren kann

Die JPEGDecoder-Bibliothek, so dass wir JPEG-kodierte Bilder dekodieren können

Die Arducam-Arduino-Bibliothek ist auf GitHub unter <https://github.com/ArduCAM/Arduino> verfügbar. Um sie zu installieren, laden Sie das Repository herunter oder klonen Sie es. Kopieren Sie anschließend das Unterverzeichnis ArduCAM

```

1 //Step 1: select the hardware platform , only one at a time
2 //#define OV2640_MINI_2MP
3 //#define OV3640_MINI_3MP
4 //#define OV5642_MINI_5MP
5 //#define OV5642_MINI_5MP_BIT_ROTATION_FIXED
6 #define OV2640_MINI_2MP_PLUS
7 //#define OV5642_MINI_5MP_PLUS
8 //#define OV5640_MINI_5MP_PLUS

1 // Comment out the next #defines if you are not using an SD Card to store the JPEG
2 // Commenting out the line is NOT essential but will save some FLASH space if
3 // SD Card access is not needed. Note: use of SdFat is currently untested!
4
5 //#define LOAD_SD_LIBRARY // Default SD Card library
6 //#define LOAD_SDFAT_LIBRARY // Use SdFat library instead , so SD Card SPI can be

```

in Ihr Arduino/libraries-Verzeichnis. Um dieses Verzeichnis auf Ihrem Rechner zu finden, überprüfen Sie den Sketchbook-Speicherort im Fenster "Einstellungen" der Arduino IDE.

Nachdem Sie die Bibliothek heruntergeladen haben, müssen Sie eine ihrer Dateien bearbeiten, um sicherzustellen, dass sie für die Arducam Mini 2MP Plus konfiguriert ist. Öffnen Sie dazu die folgende Datei:

[Arduino/libraries/ArduCAM/memorysaver.h](#)

You'll see a bunch of **#define** statements listed. Make sure that they are all commented out, except for **#define OV2640_MINI_2MP_PLUS**, as so:

Sobald Sie die Datei gespeichert haben, sind wir mit der Konfiguration der Arducam-Bibliothek fertig.

Unser nächster Schritt ist die Installation der JPEGDecoder-Bibliothek. Wir können dies aus der Arduino-IDE heraus tun. Gehen Sie zunächst auf die Option "Manage Libraries..." im Menü "Tools" und suchen Sie nach JPEGDecoder. Sie sollten die Version 1.8.0 der Bibliothek installieren.

Sobald die Bibliothek installiert ist, müssen wir sie konfigurieren, um einige optionale Komponenten zu deaktivieren, die nicht mit dem Arduino Nano 33 BLE Sense kompatibel sind. Öffnen Sie die folgende Datei:

[Arduino/libraries/JPEGDecoder/src/User_Config.h](#)

Stellen Sie sicher, dass sowohl **#define LOAD_SD_LIBRARY** als auch **#define LOAD_SDFAT_LIBRARY** auskommentiert sind, wie in diesem Auszug aus der Datei gezeigt:

Um die Kamera zu testen, beginnen Sie damit, die Kamera des Geräts auf etwas zu richten, das definitiv keine Person ist, oder sie einfach zu verdecken. Wenn die blaue LED das nächste Mal blinkt, erfasst das Gerät ein Bild von der Kamera und beginnt mit der Inferenz. Das Bildverarbeitungsmodell, das wir für die Personenerkennung verwenden, ist relativ groß, aber mit den cmsis-nn-Optimierungen dauert es nur etwa 800 ms, um das Modell auszuführen.

Nach einem Moment wird das Ergebnis der Inferenz in das Aufleuchten einer weiteren LED umgesetzt. Da Sie die Kamera auf etwas gerichtet haben, das keine Person ist, sollte die rote LED aufleuchten.

Versuchen Sie nun, die Kamera des Geräts auf sich selbst zu richten! Wenn die blaue LED das nächste Mal blinkt, nimmt das Gerät ein weiteres Bild auf und beginnt, die Inferenz durchzuführen. Nach einer kurzen Pause sollte die grüne LED aufleuchten! Denken Sie daran, dass die Bilddaten vor jeder Inferenz als Schnappschuss aufgenommen werden, wenn die blaue LED blinkt. Das, worauf die Kamera in diesem Moment

```
1 14:17:50.714 -> Starting capture
2 14:17:50.714 -> Image captured
3 14:17:50.784 -> Reading 3080 bytes from ArduCAM
4 14:17:50.887 -> Finished reading
5 14:17:50.887 -> Decoding JPEG and converting to greyscale
6 14:17:51.074 -> Image decoded and processed
7 14:18:09.710 -> Person score: 246 No person score: 66
```

gerichtet ist, wird in das Modell eingespeist. Es spielt keine Rolle, wohin die Kamera gerichtet ist, bis das nächste Mal ein Bild aufgenommen wird, wenn die blaue LED wieder blinkt.

Wenn Sie scheinbar falsche Ergebnisse erhalten, stellen Sie sicher, dass Sie sich in einer Umgebung mit guter Beleuchtung befinden. Sie sollten auch sicherstellen, dass die Kamera richtig ausgerichtet ist, mit den Stiften nach unten, so dass die Bilder, die sie aufnimmt, richtig herum sind - das Modell wurde nicht darauf trainiert, auf dem Kopf stehende Personen zu erkennen! Außerdem sollte man bedenken, dass es sich um ein winziges Modell handelt, bei dem die Genauigkeit gegen die geringe Größe eingetauscht wird. Es funktioniert sehr gut, aber es ist nicht immer zu 100

Wir können die Ergebnisse der Inferenz auch über den Arduino Serial Monitor sehen. Dazu öffnen Sie den Serial Monitor aus dem Menü Tools. Sie werden ein detailliertes Protokoll dessen sehen, was passiert, während unsere Anwendung läuft. Es ist auch interessant, das Kontrollkästchen Zeitstempel anzeigen zu aktivieren, so dass Sie sehen können, wie lange jeder Teil des Prozesses dauert:

Aus dem Protokoll geht hervor, dass etwa 170 ms für die Erfassung und das Lesen der Bilddaten vom Kameramodul, 180 ms für die Dekodierung des JPEGs und die Umwandlung in Graustufen und 18,6 Sekunden für die Inferenz benötigt wurden.

21. Sensor ov2640

<https://www.arducam.com/ov2640/>
<https://www.arducam.com/focal-length-calculator/>

21.1. Produktbeschreibung

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

Anwendung:

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

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

Eigenschaften:

- 2-Megapixel-Bildsensor OV2640.
- M12-Mount- oder CS-Mount-Objektivhalter mit wechselbaren Objektivoptionen.
- IR-empfindlich mit entsprechender Objektivkombination.
- I2C-Schnittstelle für die Sensorkonfiguration.
- SPI-Schnittstelle für Kamera-Befehle und Datenstrom.
- Alle E/A-Anschlüsse sind für 5 V/3,3 V geeignet.
- Unterstützt JPEG-Komprimierungsmodus, Einzel- und Mehrfachaufnahmemodus, einmaliges Erfassen mehrerer Lesevorgänge, Burst-Lese-Operation, Niedrige-Energie-Modus usw..
- Kann mit Standard-Arduino-Boards verbunden werden.
- Open-Source-Code-Bibliothek für Arduino, STM32, Chipkit, Raspberry Pi, BeagleBone Black.
- Schlanke Form.



Figure 21.1.: Kamera IMX477 der Firma Arducam; [Ard21]

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%A4M%C3%A4%C3%96%C3%A4;%C3%A4T%C3%A4%C3%96&dchild=1&keywords=arducam&qid=1622358684&sr=8-17-spons&psc=1&spLa=ZW5jcnlwdGVkUXVhbGlmaWVyPUEyWUdNSVhJSFNUQUtLJmVuY3J5cHR1ZE1kPUEwNjI4NT

Raspberry Pi Kamerakabel, iUniker 15-poliges Flachbandkabel, Pi Kamera Flex Kabel, Flex CSI Kabel 50 cm/1 m/2 m für Raspberry Pi 3B+, 3B, 2B (nicht für Pi Zero)

21.2. 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.

21.3. 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.

21.4. How to use

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

21.4.1. 1. Edit `memoriesaver.h` file

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

21.4.2. 2. Choose correct CS pin for your camera

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

21.4.3. 3. Upload the examples

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

1. Der Ordner `Mini_5MP_Plus` ist für ArduCAM-Mini-5MP-Plus (OV5640/OV5642) Module.
2. Der Ordner `RevC` ist für ArduCAM-Shield-RevC oder ArduCAM-Shield-RevC+ Shields.
3. Der Ordner `Shield_V2` ist für das ArduCAM-Shield-V2 Schild.
4. Der Ordner `host_app` ist die Host-Erfassungs- und Anzeigeanwendung für alle ArduCAM-Module.
5. Der Ordner `RaspberryPi` ist eine Beispielanwendung für die Raspberry Pi-Plattform, siehe weitere Anleitung.
6. Der Ordner `ESP8266` ist für ArduCAM-ESP8266-UNO-Board-Beispiele für Bibliothekskompatibilität. Bitte versuchen Sie stattdessen, ESP8266 mit dem Skript `josn board manager` zu repositoryn.

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

21.4.4. 4. How To Connect Bluetooth Module

Mit dieser Demo

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

So laden Sie den Host V2:

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

22. ArduCAM ov2640

22.1. Introduction

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 22.1 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]



Figure 22.1.: ArduCAM interface with Arduino

22.1.1. 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 22.2 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.

22.2. ArduCAM Interface with Arduino

ArduCAM OV2640 needs the SPI and I2C connection with the arduino boards. It will be connecting untill these two connections are make sure. The figure 22.3 shows

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 22.2.: ArduCAM Pin Config

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

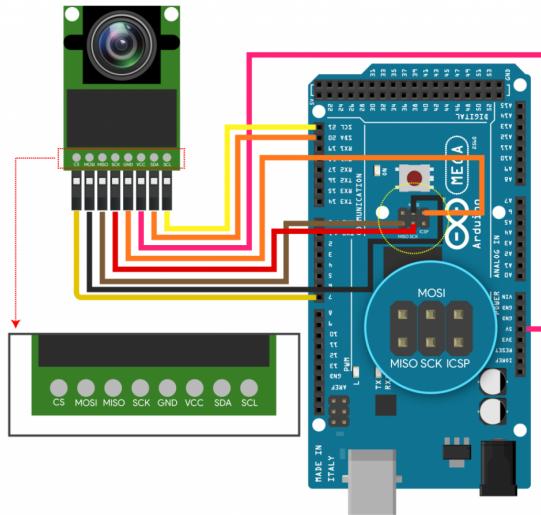


Figure 22.3.: ArduCAM Interface with Arduino Mega 2560

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.

Application

- IOT Cameras
- Robot Cameras
- Wildlife Cameras

23. 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%A5%C3%8D&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/>

23.1. 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.

23.2. Applications

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



Figure 23.1.: Kalibrierungswerkzeug der Firma Arducam; [Ard21]

23.3. Package Contents

2*Foldable Lens Calibration Card

Part IV.

Project

24. TinyML and Edge Computing

24.1. 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.

24.2. 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]

24.3. 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.

25. TensorFlow Lite

25.1. 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.

25.2. 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.

WS:Aufbau des Kapite

WS:Einführung von
TensorFlow
Lite-Interpreter und
-Konverter?

25.3. 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.

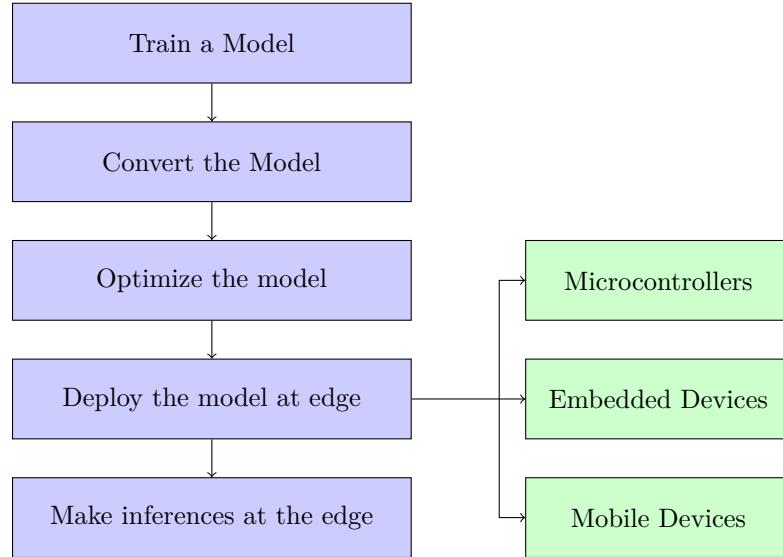


Figure 25.1.: Tensorflow Lite workflow [Kha20]

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 25.2 illustrates the basic process for creating a model that can be used on an edge computer. Most of the process uses standard TensorFlow tools.

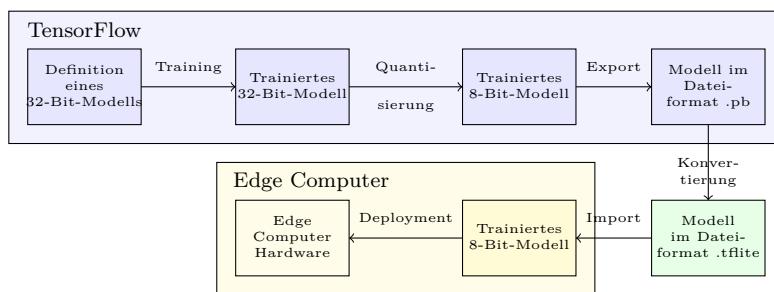


Figure 25.2.: The basic process for creating a model for an edge computer[Goo19a]

25.4. 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]

25.5. 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]

26. 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 26.1.

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.

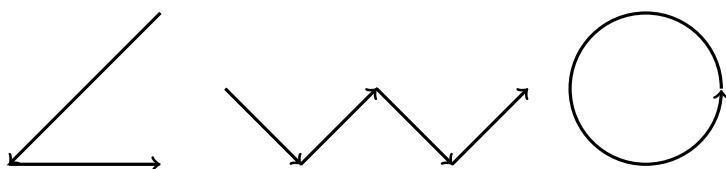


Figure 26.1.: Gestures used namely Slope, W and ring

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.

27. Project Magic Wand

27.1. 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

27.1.1. 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]

27.1.2. 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 27.1.: 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]

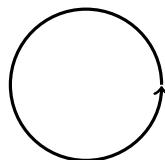


Figure 27.2.: Ring Gesture [WS20]

- 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]:

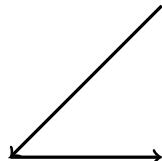


Figure 27.3.: Slope Gesture [WS20]

- 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]

27.1.3. 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 27.4.: 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.

27.1.4. 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^{27.6}.IMU is the electronic component which contains the accelerometer. The IMU object comes from the Arduino LSM9DS1 library.

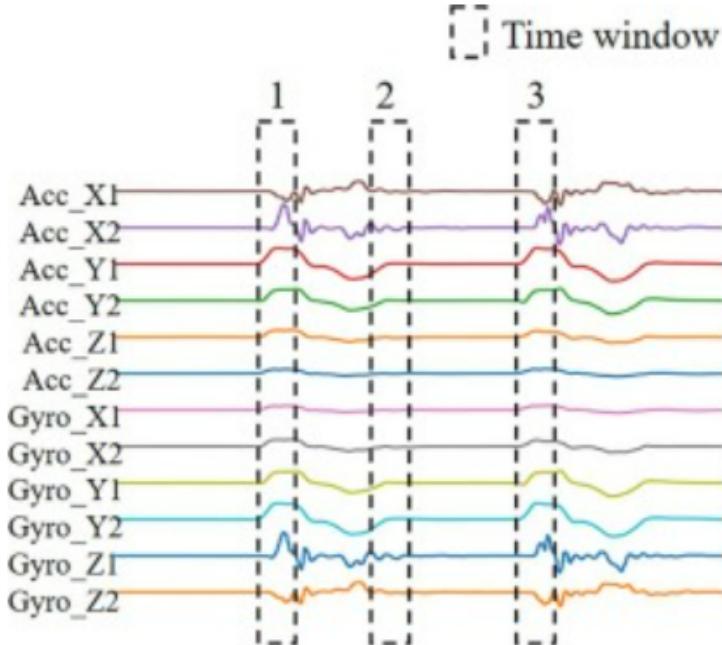


Figure 27.5.: IMU signals [Xu+22]

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 27.7 [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

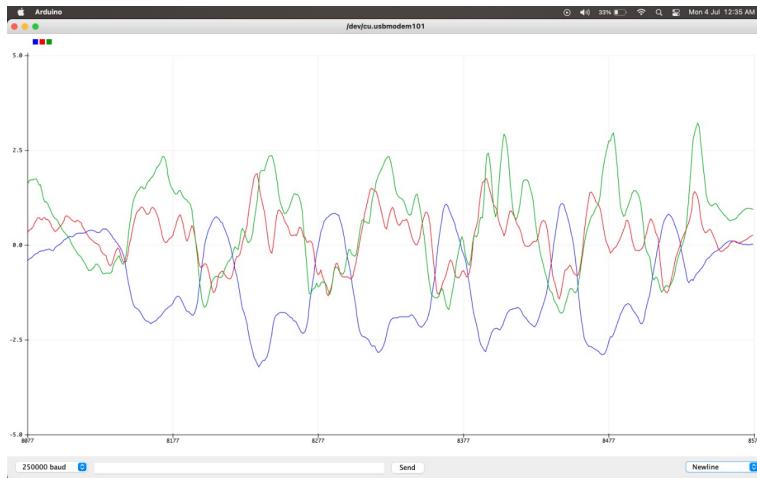


Figure 27.6.: IMU Accelerometer Graph [WS20]

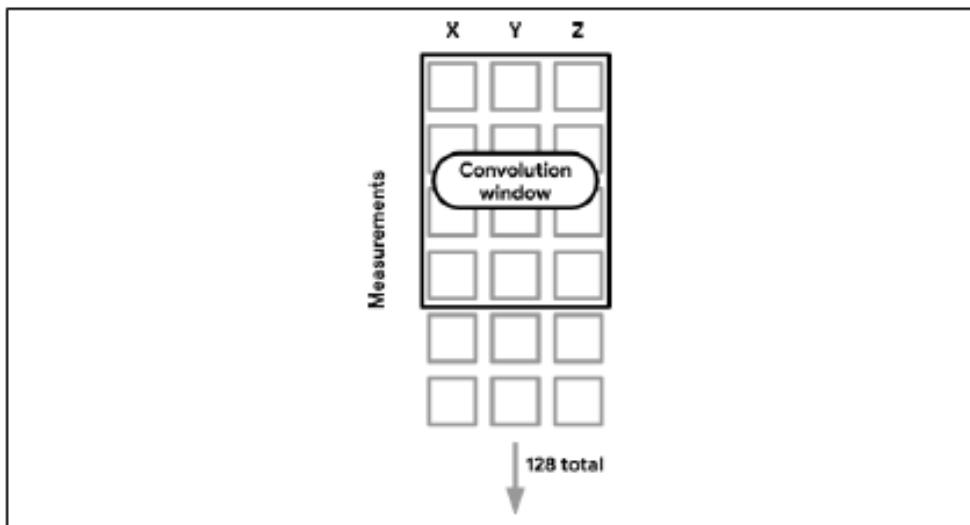


Figure 27.7.: A convolution window overlaid on the data. [WS20]

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. 27.8 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

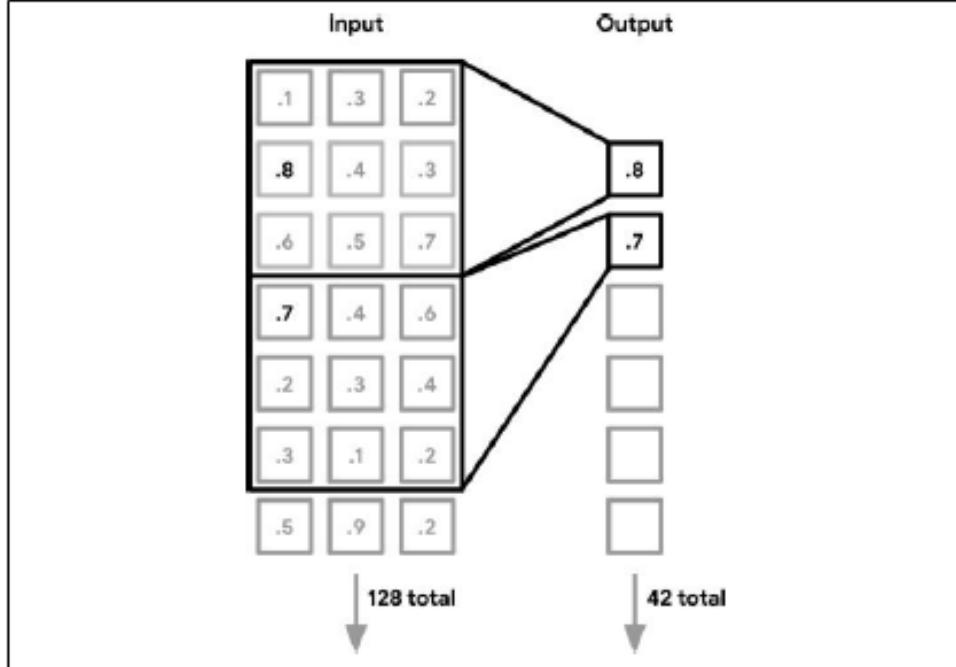


Figure 27.8.: Max Pooling. [WS20]

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.^{27.9} 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

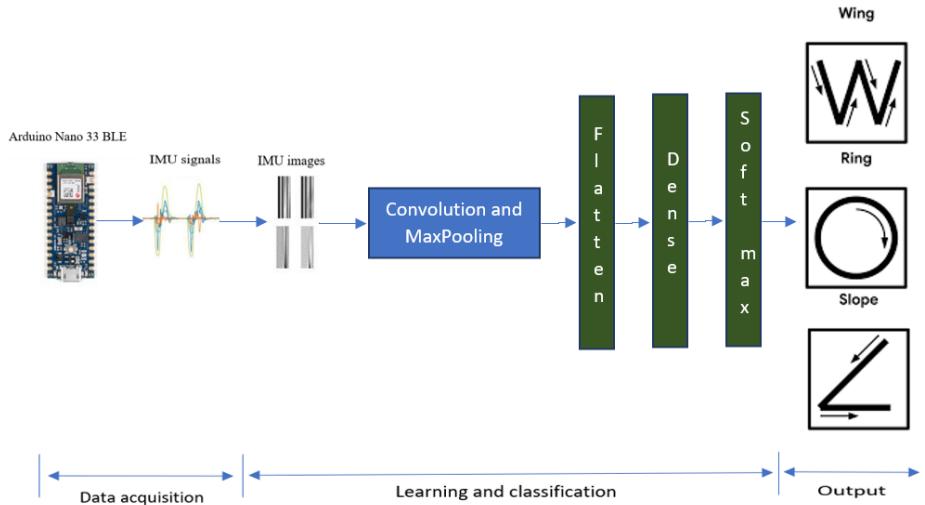


Figure 27.9.: CNN sequence to classify Wing,Ring and Slope.

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]

27.1.5. 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 27.10 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 27.10.: Sample Confusion Matrix for the dataset [WS20]

27.2. 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.

27.2.1. 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 27.11. Depending on the operating system environment, the necessary software is downloaded and installed.

Downloads

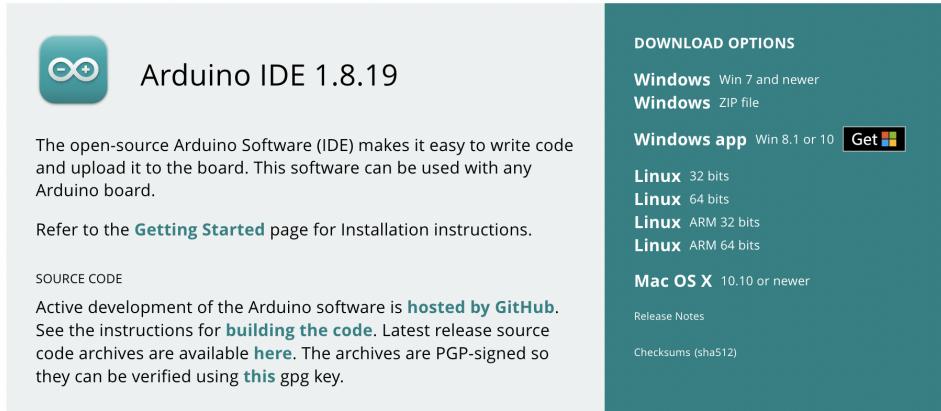


Figure 27.11.: Arduino Nano website downloads page

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 27.12.: Arduino Nano software version number

Configuration

After installation of Arduino IDE, a window as 27.13 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 27.14

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 27.15.

After installing the boards, the board has to be selected from the menu as shown in 27.16. 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 27.17. 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 27.18

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 27.19. Search for Tensor flow libraries and then download and install them

Recognizing Gestures

This code snippet in 27.20 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.

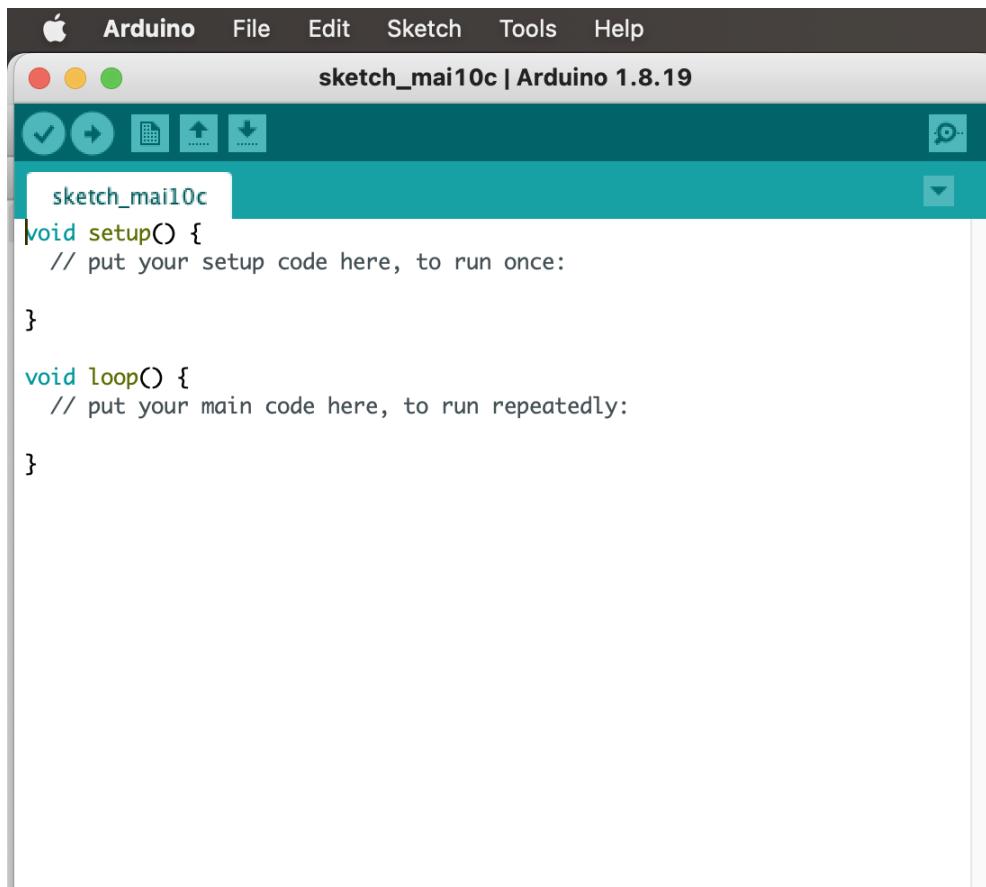


Figure 27.13.: Arduino IDE initial software window

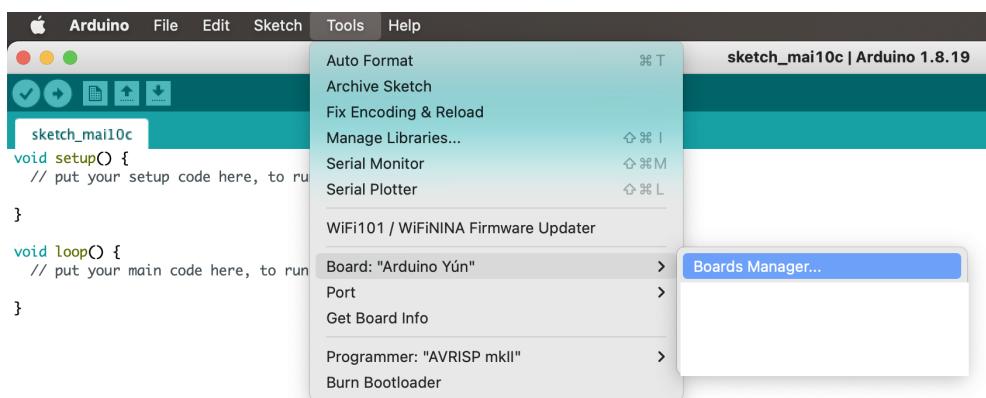


Figure 27.14.: Arduino IDE Boards Manager menu

Prediction of Gestures

The code snippet in 28.3 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.

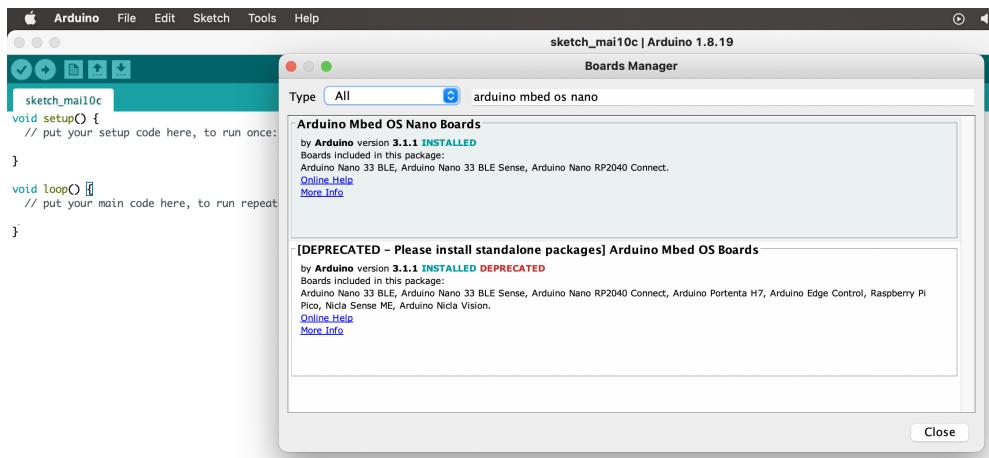


Figure 27.15.: Arduino IDE Boards Manager list

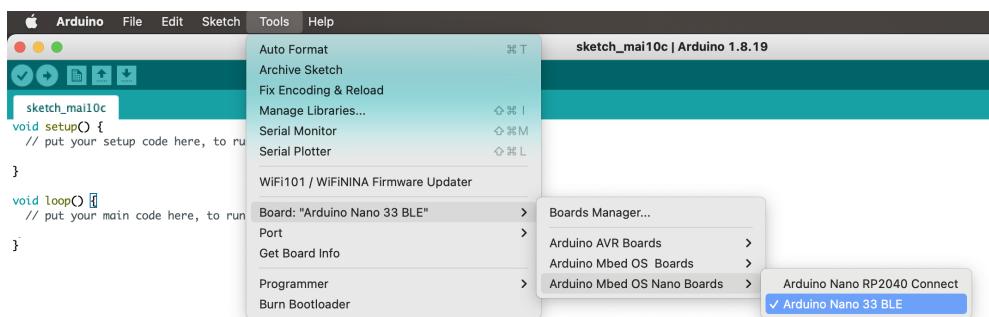


Figure 27.16.: Selection of correct board for the program

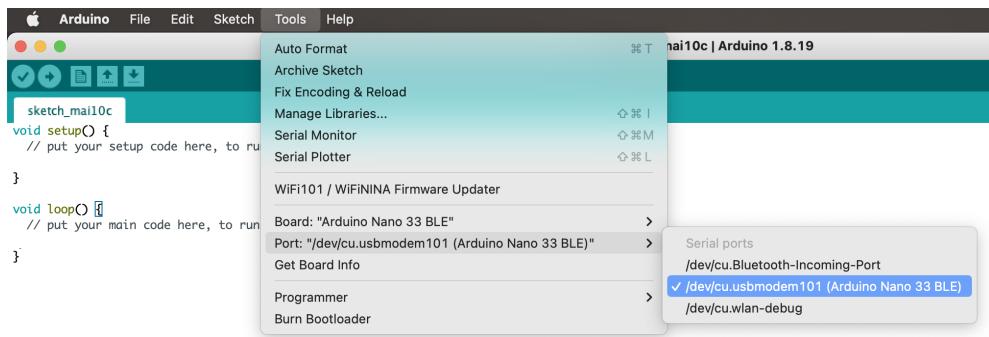


Figure 27.17.: Selection of correct port for uploading the program

Pending Movement

The code in 27.22 checks for any pending movement and waits for a new gesture to be recorded.

Training or Recognizing

The code in 27.23 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.

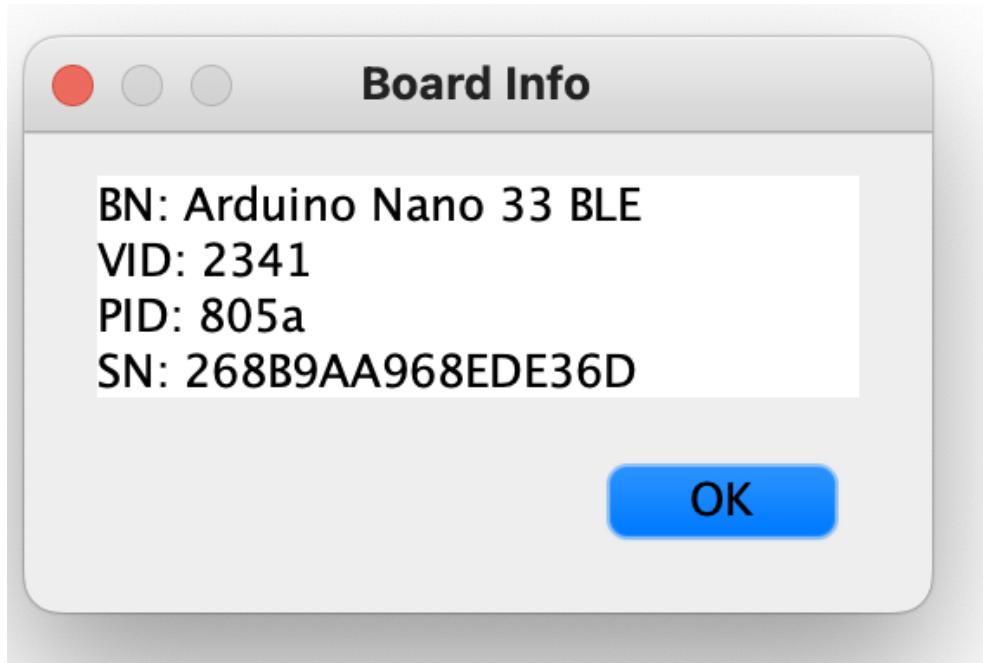


Figure 27.18.: Arduino Board Info

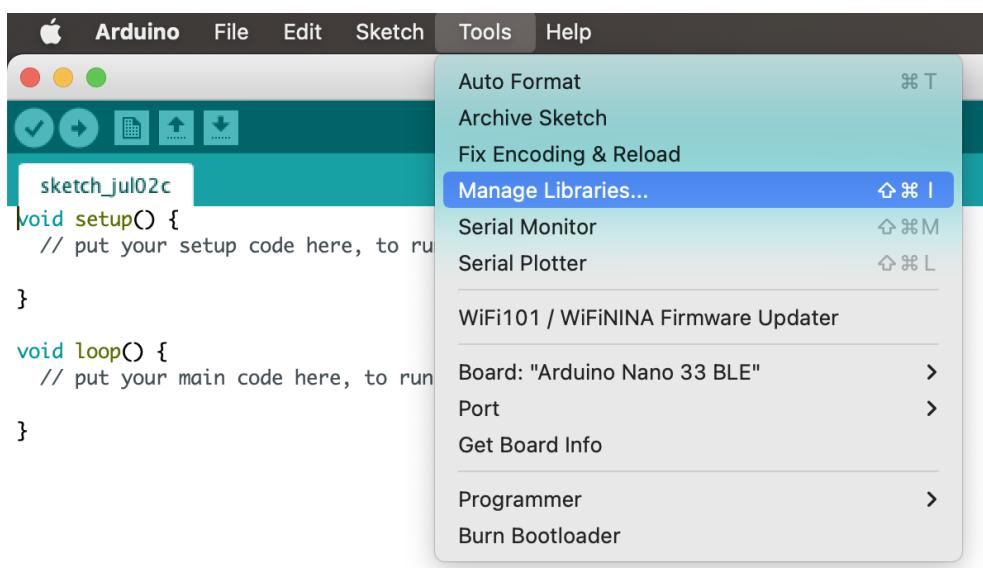


Figure 27.19.: Managing libraries in the Arduino IDE

Printing the output

The code snippet in 27.24 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 27.25 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 27.20.: 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 27.21.: 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 27.22.: Switch case statement, Pending Movement

```

const bool should_capture_data = false;
if (should_capture_data) {
    CaptureGestureData();
} else {
    RecognizeGestures();
}
}

```

Figure 27.23.: 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");
    }
}

```

Figure 27.24.: Code snippet displaying a Wing gesture

results achieved are not very good, more training has to be carried out.

```

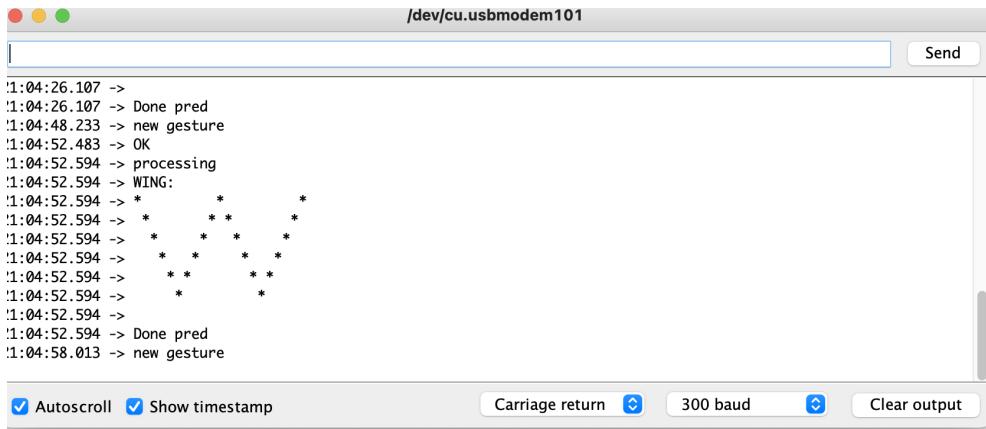
constexpr float kDetectionThreshold = 0.7f;
constexpr int kPredictionHistoryLength = 5;
constexpr int kPredictionSuppressionDuration = 25;

```

Figure 27.25.: 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 27.26 for Wing, 27.27 for Slope, 27.28 and for unknown 27.29



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' at timestamp 04:26.107, followed by 'OK' at 04:48.233, and then a series of 'processing' and 'WING' events. Each 'WING' event is preceded by a timestamp and followed by a sequence of asterisks (*). The log ends with 'Done pred' at 04:58.013 and 'new gesture' at 04:58.013. At the bottom of the window, there are checkboxes for 'Autoscroll' and 'Show timestamp', and buttons for 'Carriage return', '300 baud', and 'Clear output'.

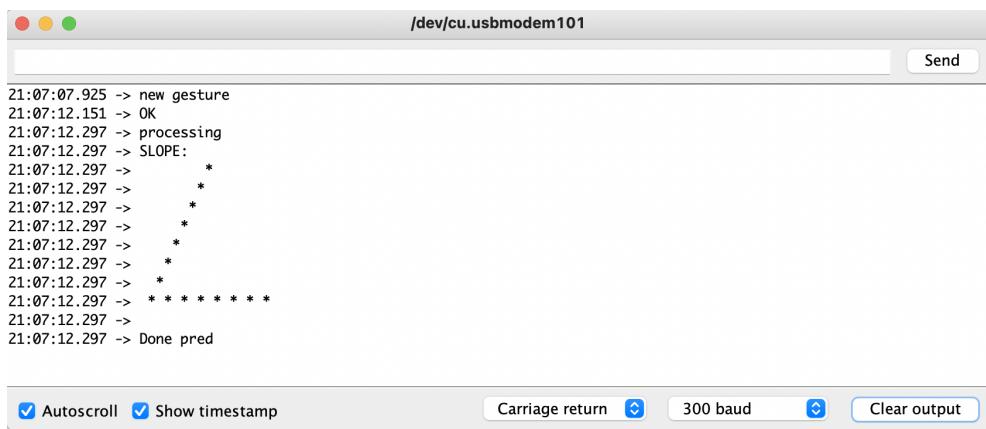
```

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

```

Autoscroll Show timestamp Carriage return 300 baud Clear output

Figure 27.26.: 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 starts with 'new gesture' at 07:07.925, followed by 'OK' at 07:12.151, and then a series of 'processing' and 'SLOPE:' events. Each 'SLOPE:' event is preceded by a timestamp and followed by a sequence of asterisks (*). The log ends with 'Done pred' at 07:12.297. At the bottom of the window, there are checkboxes for 'Autoscroll' and 'Show timestamp', and buttons for 'Carriage return', '300 baud', and 'Clear output'.

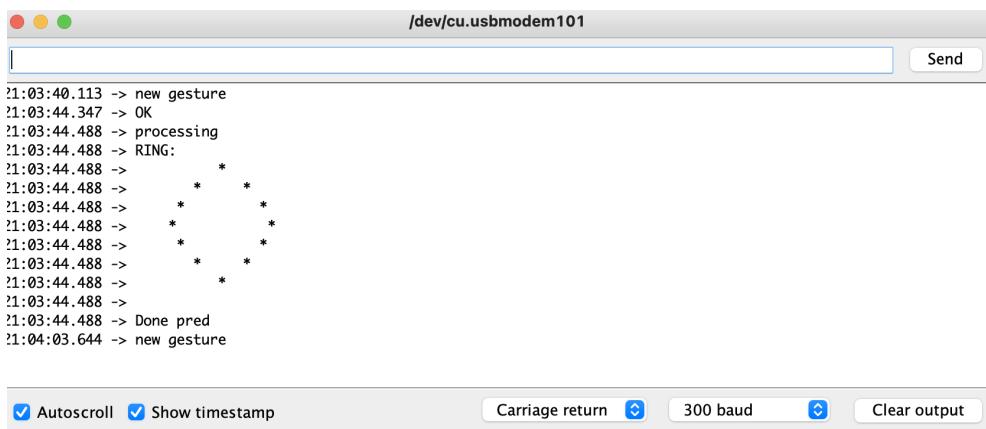
```

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

```

Autoscroll Show timestamp Carriage return 300 baud Clear output

Figure 27.27.: The displayed slope gesture



The screenshot shows a terminal window titled '/dev/cu.usbmodem101'. The window contains a text log of gesture recognition events. It starts with 'new gesture' at 03:40.113, followed by 'OK' at 03:44.347, and then a series of 'processing' and 'RING:' events. Each 'RING:' event is preceded by a timestamp and followed by a sequence of asterisks (*). The log ends with 'Done pred' at 04:03.644 and 'new gesture' at 04:03.644. At the bottom of the window, there are checkboxes for 'Autoscroll' and 'Show timestamp', and buttons for 'Carriage return', '300 baud', and 'Clear output'.

```

?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 -> Done pred
?1:04:03.644 -> new gesture

```

Autoscroll Show timestamp Carriage return 300 baud Clear output

Figure 27.28.: The displayed ring gesture

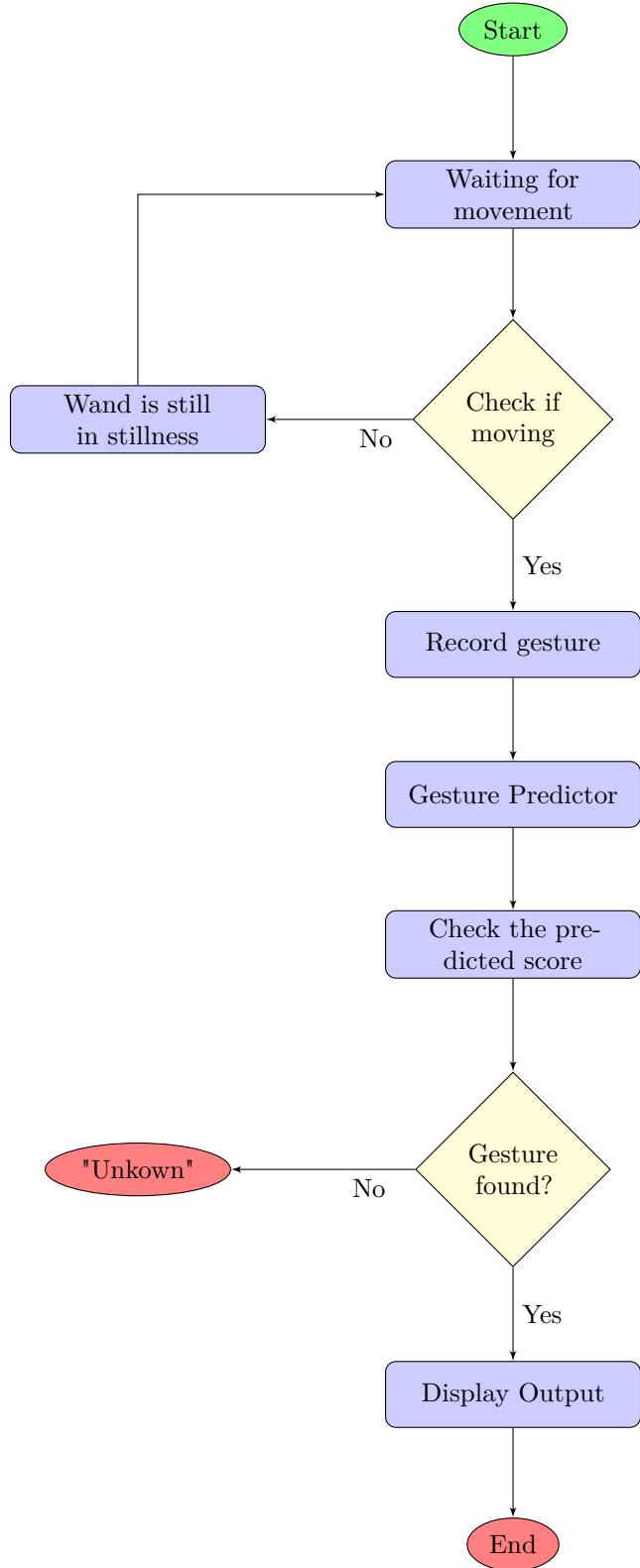
27.2.2. Program Flow Code

On clicking the option "Manage Libraries" a new window shows up where a person can search for the libraries and install them.



Figure 27.29.: The unknown gesture

28. 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.

28.1. Development

28.1.1. 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.

28.1.2. 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.

28.1.3. 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.

28.2. Code Structure

The code includes 12 major files. These files are coded to be compatible for the Arduino Platform.

Magic_Wand

This code snippet 28.1 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 28.1.: 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 28.2.: Code snippet for isMoving() function

isMoving()

Using the function isMoving() shown in 28.2 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 28.3.: Code snippet that shows processing of the waved gestures

The code snippet shown in 28.3 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 28.4 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 28.4 . 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 28.6. 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 28.4.: 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 28.5.: 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 28.6.: 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 28.7.: Code snippet to check if the wand is still

28.2.1. Cases - ePendingMovement, eRecordingGesture

These cases shown in 28.7 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 28.8.: Code snippet showing capturing gesture function for training

CaptureGestureData()

This function shown in 28.8 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 28.9 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.

28.2.2. arduino_accelerometer_handler

The function shown in 28.10 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 28.9.: 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 28.11 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.

28.2.3. Arduino_Output_Handler

The function shown in 28.12 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.

28.2.4. main_functions

The code snippet shown in 28.13 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 28.10.: The accelerometer handler function

28.2.5. constants.h

This code snippet shown in 28.14 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 28.11.: 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 28.12.: 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();
#ifndef __cplusplus
}
#endif
#endif // TENSORFLOW_LITE_MICRO_EXAMPLES_MAGIC_WAND_MAIN_FUNCTIONS_H_

```

Figure 28.13.: Code snippet showing the `loop()` function in the program

```

constexpr float kDetectionThreshold = 0.7f;
constexpr int kPredictionHistoryLength = 5;
constexpr int kPredictionSuppressionDuration = 25;

```

Figure 28.14.: Code displaying threshold values

28.2.6. Magic Wand Program Workflow

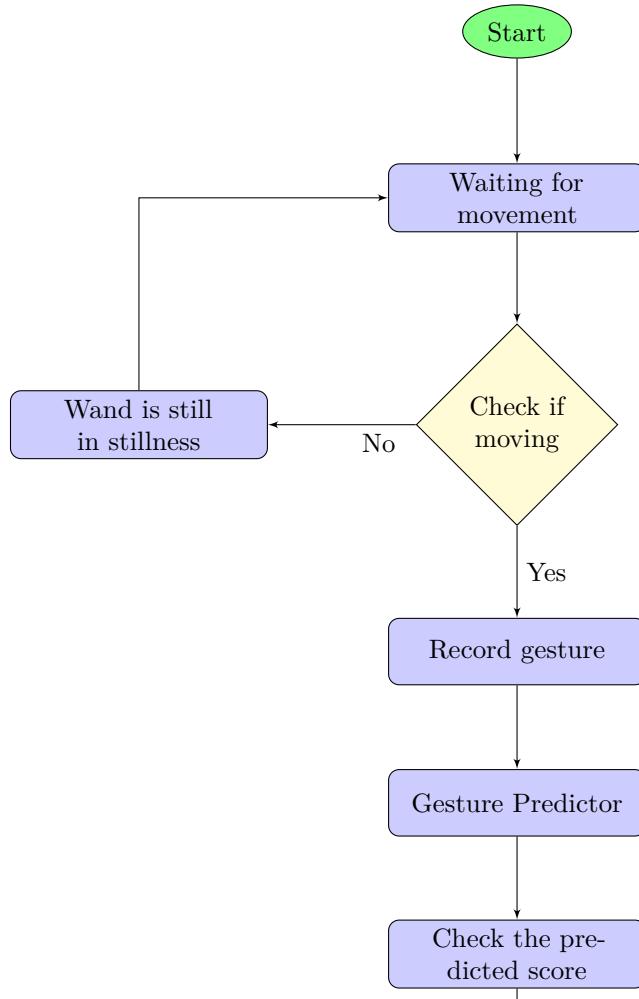


Figure 28.15.: Magic Wand Program Workflow

28.2.7. 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.

28.3. Getting to Know Arduino Nano Magic Wand

28.3.1. Key Features

- Can detect various gestures.
- Three gestures trained here are “wing” , “ring” and “slope”
- Compatible with Windows, linux and MacOS

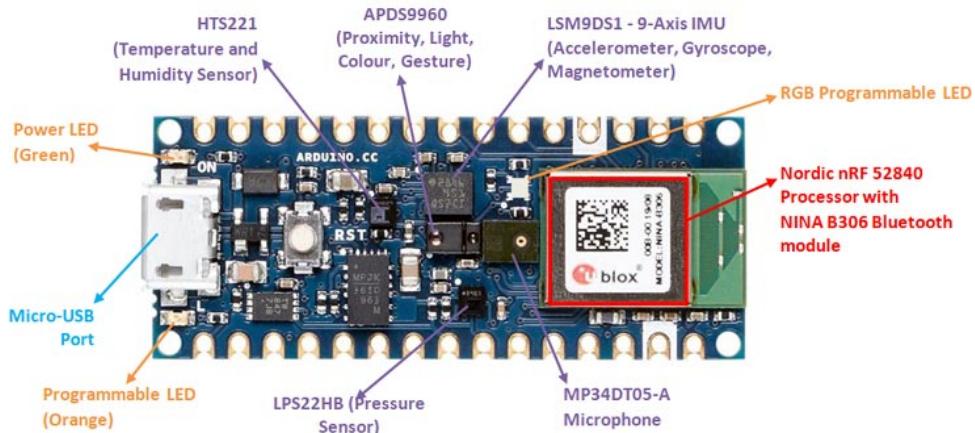


Figure 28.16.: Components in Arduino Nano 33 BLE Sense [Raj19]

28.3.2. 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.

28.4. 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 28.17.: 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 28.1.: Technical Specifications of Arduino Nano 33 BLE Sense [Ard21]

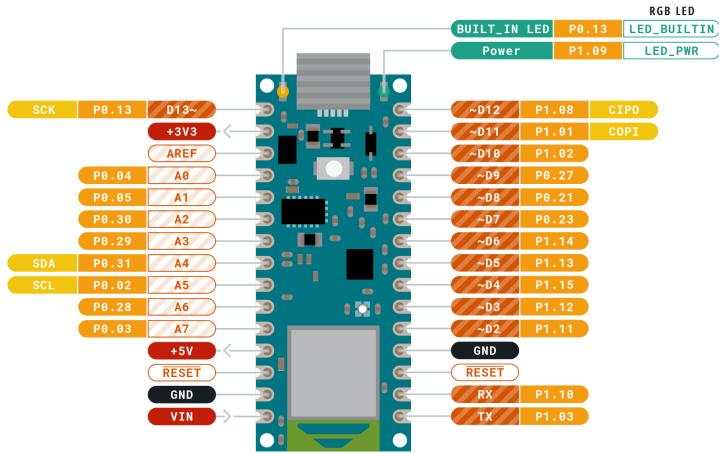


Figure 28.18.: Pinout
[Ard21]

28.5. LSM9DS1(IMU)

28.6. 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.

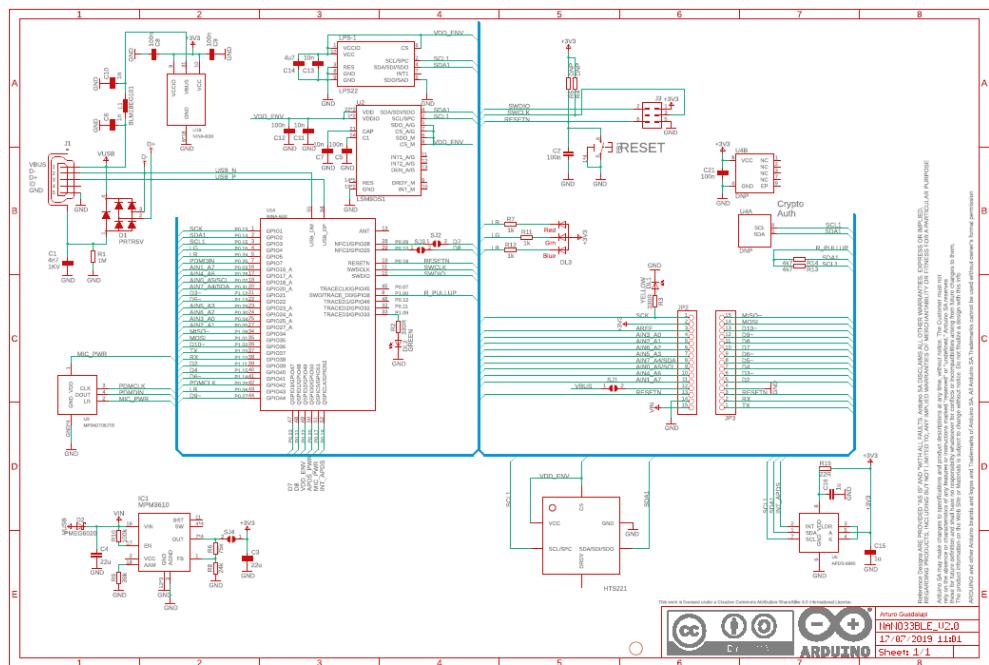


Figure 28.19.: Schematic connection diagram
[Ard21]

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 28.20.: 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.

28.7. Pin description LSM9DS1

28.8. Pin connections LSM9DS1

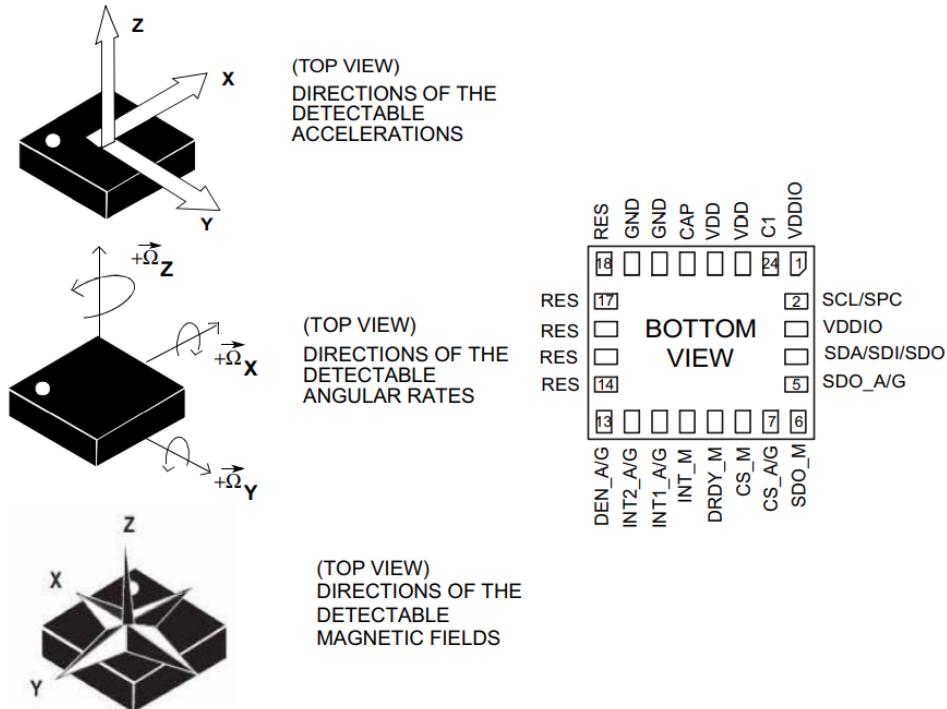


Figure 28.21.: Pin connections LSM9DS1
[Stmb]

Pin description LSM9DS1

28.8.1. 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.

28.8.2. 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 28.22.: 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 28.23.: 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 28.24.: Temperature Sensor Characteristics
[Stmb]

LSM9DS1 electrical connections

28.8.3. Using Magic Wand

28.9. Connect with Computer

Connect the device using Micro-B USB cable.

28.10. 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 28.25.: Absolute Maximum Temperature
[Stmb]

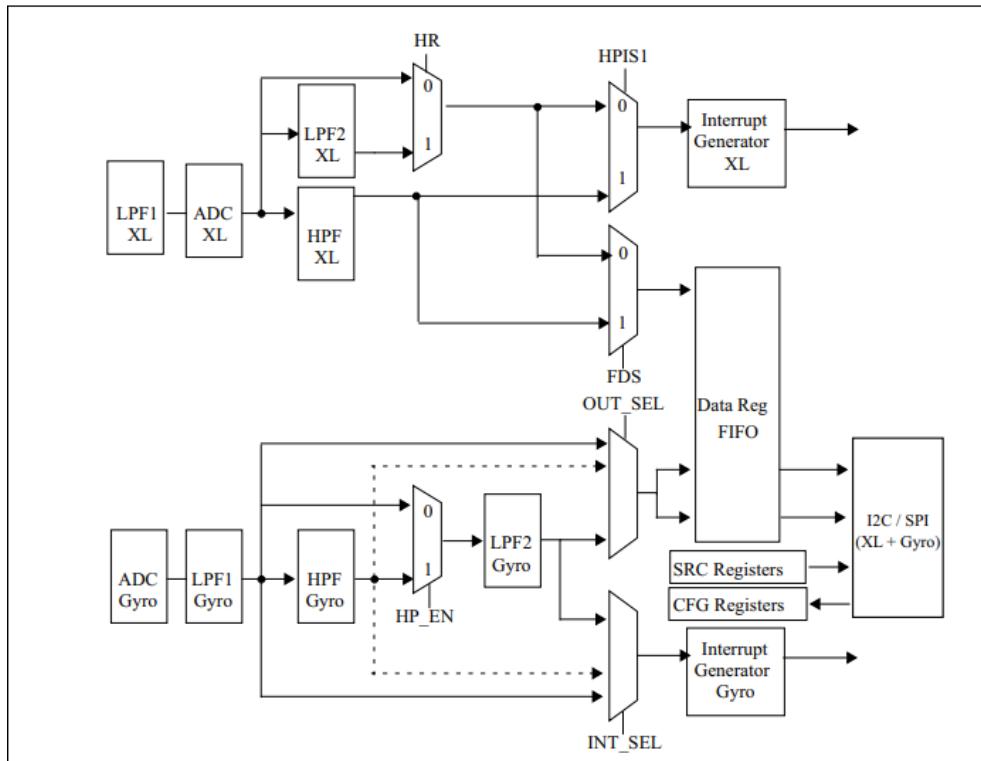


Figure 28.26.: Accelerometer and gyroscope block diagram
[Stmb]

28.11. Select Port

Tools → Board → Arduino Nano 33 BLE

28.12. Run the application

Hit the upload button to compile and upload the code to the Arduino device.

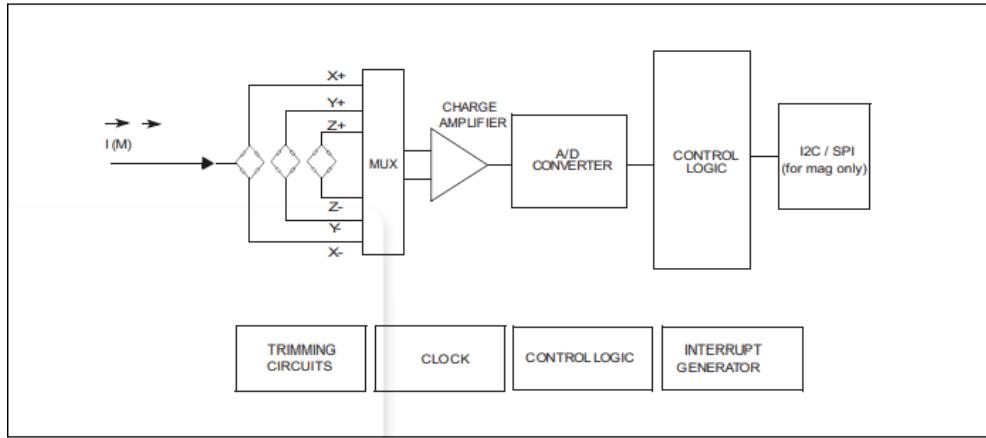


Figure 28.27.: Magnetometer block diagram
[Stmb]

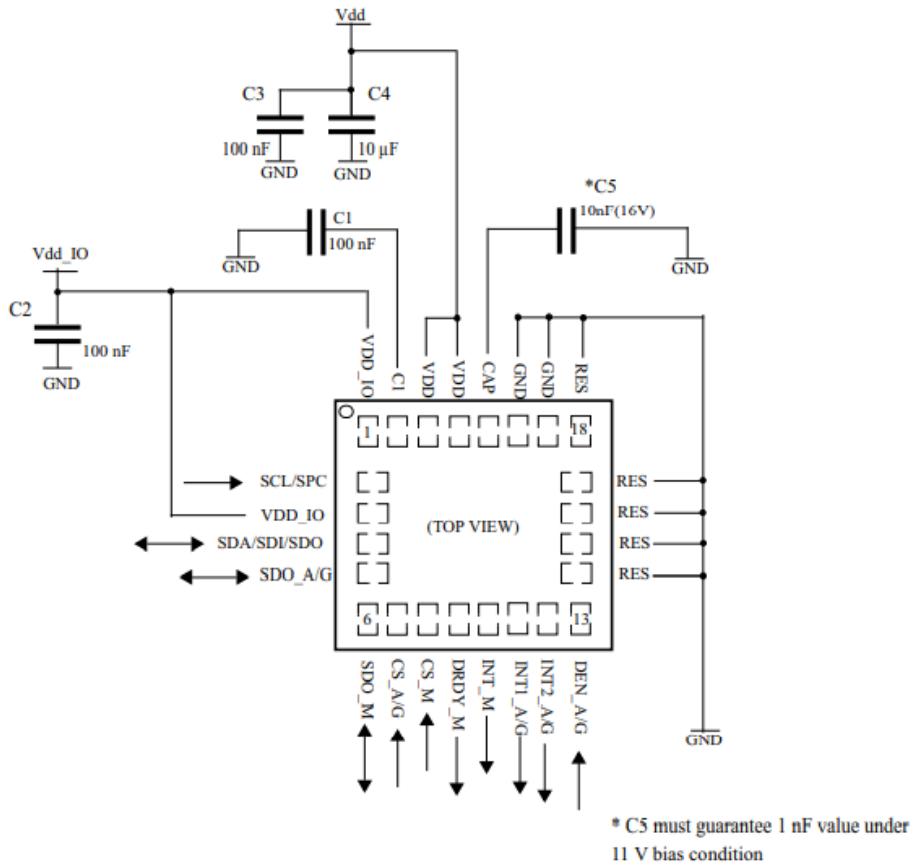


Figure 28.28.: LSM9DS1 electrical connections
[Stmb]

28.13. Open Serial Monitor

Tools → Serial Monitor

28.14. Start waving magic wand

Start waving for the desired output. Gestures will be predicted according to the movement of the wand.

28.14.1. 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

28.14.2. 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.

28.14.3. 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.

29. 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.

29.1. 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. Fig 29.1 shows the different steps involved for training the model.

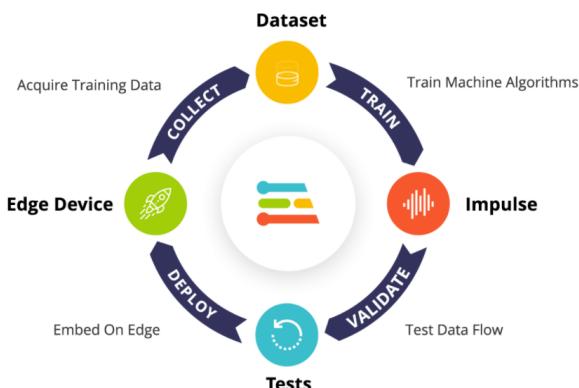


Figure 29.1.: Edge Impulse Flow Chart

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 hardware 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

29.2. 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.

29.2.1. 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.

29.3. 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). Fig29.2 shows the result of counting finger using the MediaPipe technique on the basis of hand landmarks.

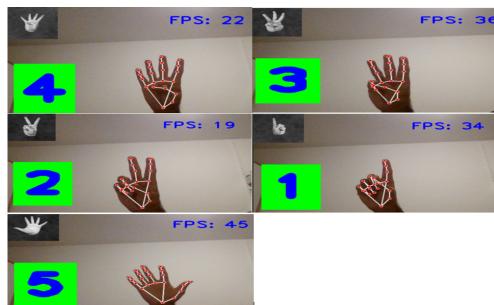


Figure 29.2.: Counting Finger Using MediaPipe and OpenCV

29.3.1. 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 figures or even the individual landmark on hand. Fig29.3 shows the 21 3D hand landmarks. MediaPipe module uses the convolutional neural network and took 30,000 3D images



Figure 29.3.: Hand Landmarks using MediaPipe

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. Fig29.4 shows for detecting the 21 hand landmarks of the hand are:

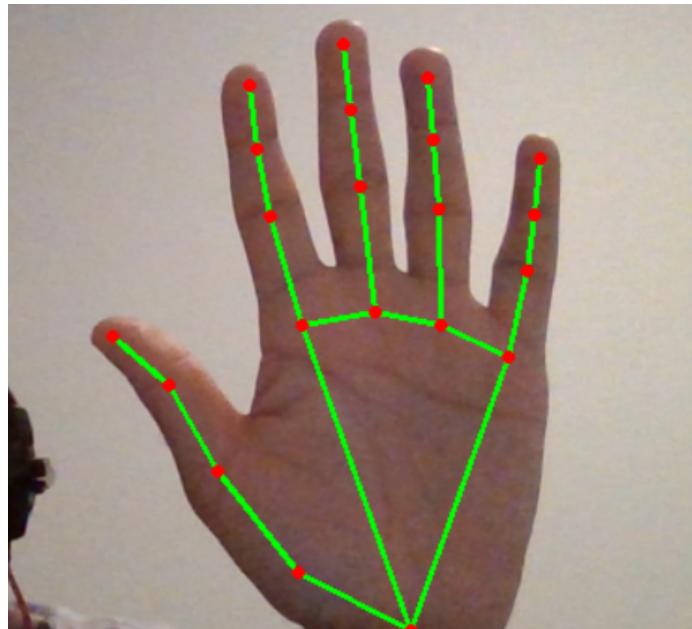


Figure 29.4.: Hand Landmarks using MediaPipe Function

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:

29.3.2. 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.

29.3.3. 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, Fast, 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. Fig29.5 shows the detected gestures and sign results are as follow.



Figure 29.5.: Gesture Detection on Arduino

29.4. 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.

29.4.1. 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. Fig29.6 shows the result of train model detecting the different gestures.

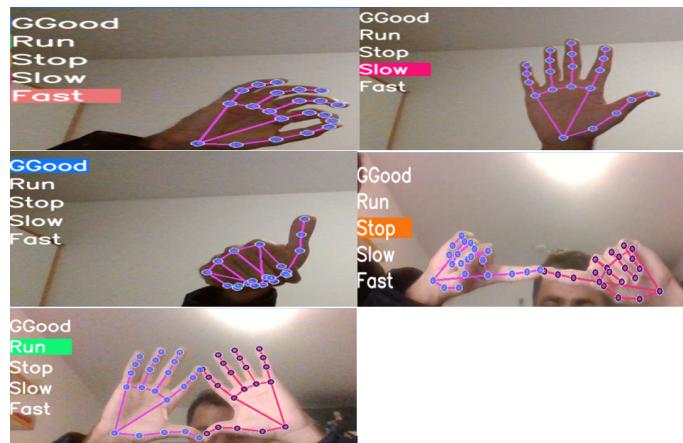


Figure 29.6.: Gesture Detection Using LSTM, TensorFlow, and MediaPipe

29.5. Color Detection

For this project, the task is to detect the Red, Green, Blue (RGB) color for different product. In the Arduino Nano 33 BLE Sense, there is on-board sensor APDS9960 whose one of the function is also to detect RGB color. I used this sensor and it give us the RGB percentage of each color in the product. The available program and library is also available in the Arduino IDE for executing this programm.

29.6. 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.

29.6.1. 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. Fig29.7 shows the comparison of YOLO version and other models for object detection is:

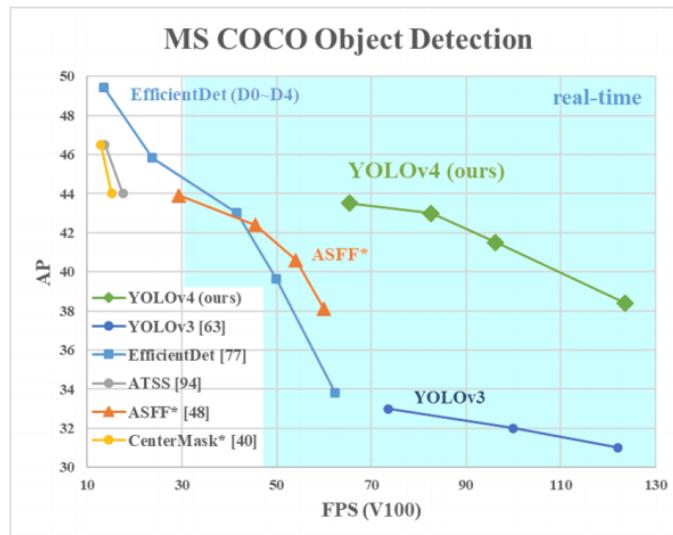


Figure 29.7.: Object detection Model Comparison

29.6.2. 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

29.6.3. Deep Sort Object Tracking Model

Deep sort is the model in machine learning used for tracking the object from the video having good frame per second on gpu. It also tracks 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.

30. 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.

30.1. 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.

30.1.1. 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 30.1 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

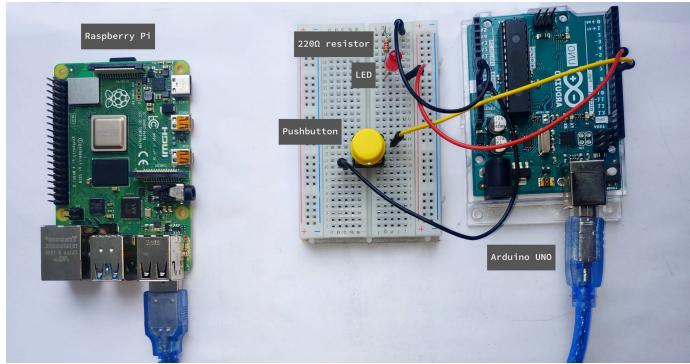


Figure 30.1.: Pyfirmata Communication for Python on Arduino

30.2. 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.

30.2.1. 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 30.2 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.

30.3. 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 30.2.: Logitech Camera Replacement

Sense due to C++ language environment issues.

30.3.1. 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.

30.4. 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.

30.5. 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

30.5.1. 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.

31. Source Codes

31.0.1. 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
```

31.0.2. 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

```

31.0.3. 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()

```

31.0.4. 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

```

31.0.5. 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)
```


32. Datenblätter

32.1. 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 IOs 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/

2/3

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.

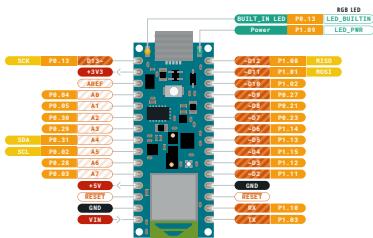
Pin	Function	Type	Description
1	+3V3	Power Out	Internally generated power output to be used as voltage reference
2	SWD	Digital	nRF52480 Single Wire Debug Data
3	SWCLK	Digital In	nRF52480 Single Wire Debug Clock
5	GND	Power	Power Ground
6	RST	Digital In	Active low reset input

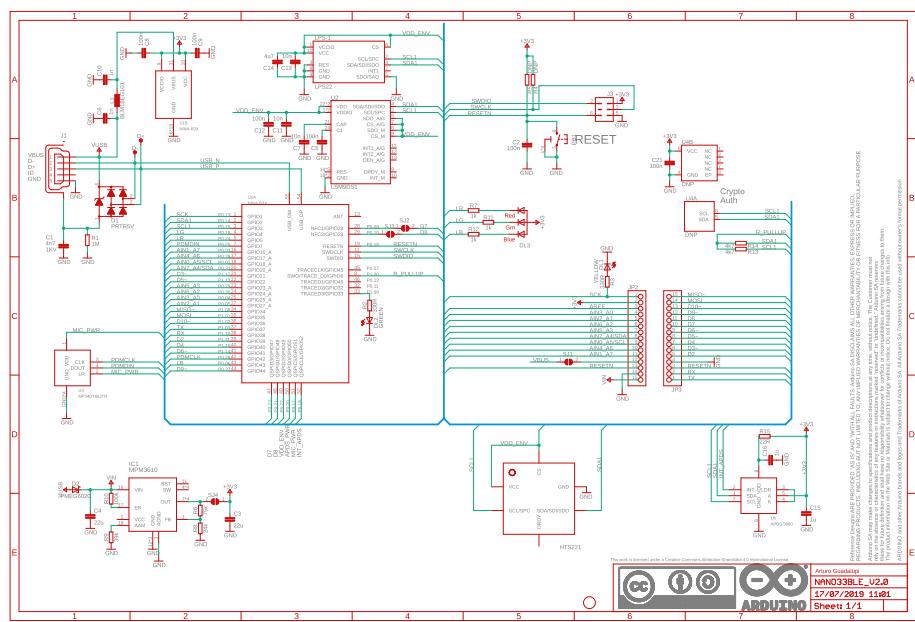
https://store.arduino.cc/usa/datasheet/index/index?url_key/nano-33-ble-sense-reseller/

3/3



ARDUINO
NANO 33 BLE SENSE
[STORE.ARDUINO.CC/NANO-33-BLUETOOTH-SENSE](https://store.arduino.cc/nano-33-bluetooth-sense)





33. 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



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.

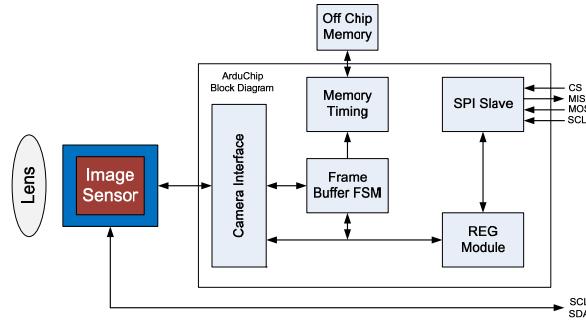


Figure 2 ArduCAM Mini Shield Block Diagram

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

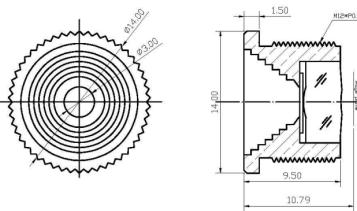


Figure 3 S Mount Lens Specification

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

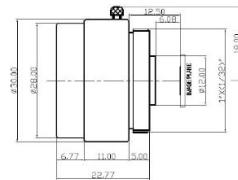
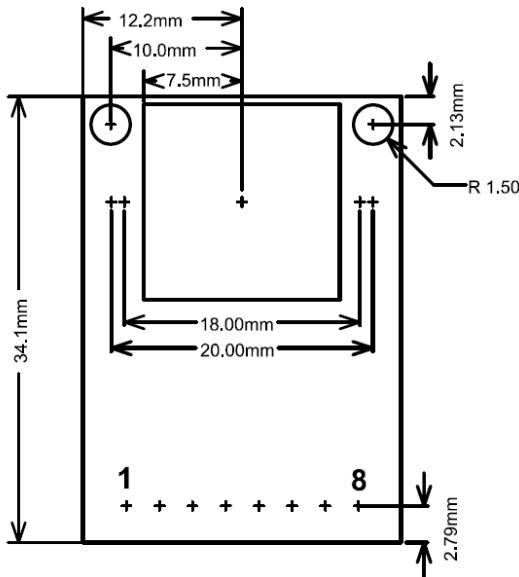


Figure 4 CS Mount Lens Specification

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.

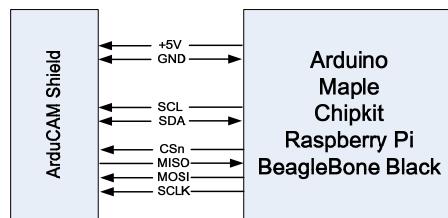


Figure 1 Typical Wiring

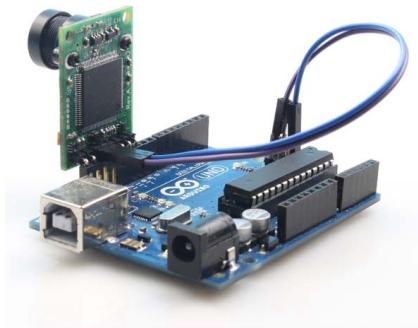


Figure 2 Wiring for Arduino UNO R3

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.

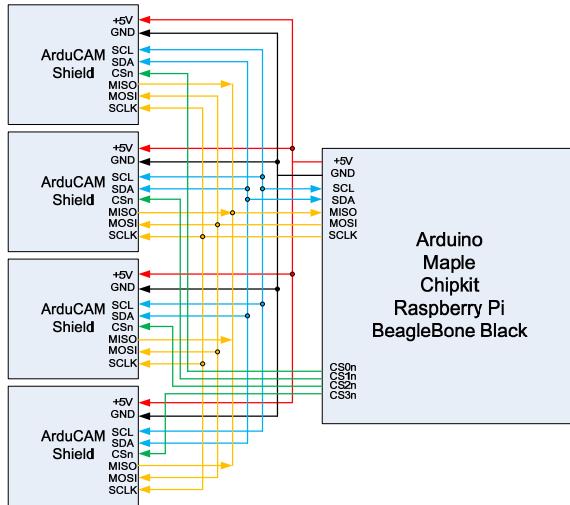


Figure 3 Multi-Cameras Wiring

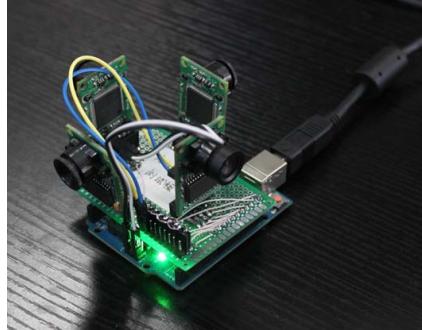


Figure 4 Mult-Cameras Wiring on Arduino UNO

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

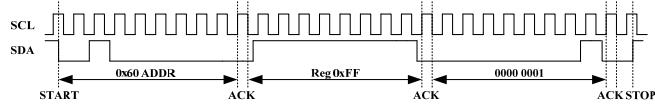


Figure 5 I2C Write Bus Timing

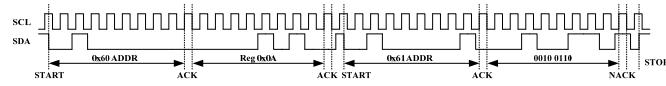


Figure 6 I2C Read Bus Timing

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.

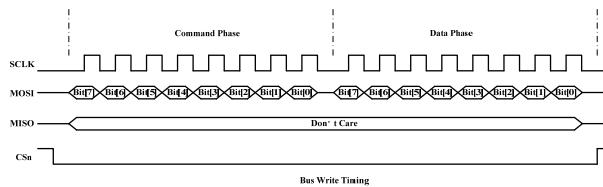


Figure 7 SPI Bus Write Timing

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.

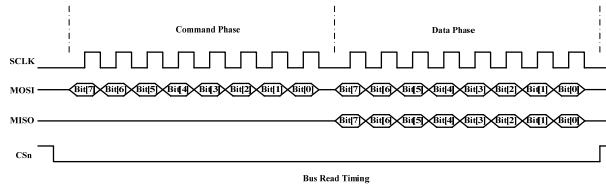


Figure 8 SPI Bus Single Read Timing

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.

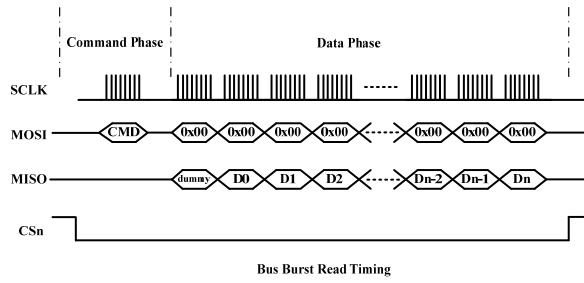


Figure 9 SPI Bus Burst Read Timing

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

- [Ada] *Adafruit GFX Library*. 2023. URL: <https://github.com/adafruit/Adafruit-GFX-Library> (visited on 06/14/2023).
- [Arda] *Arduino ABX00031 Nano 33 BLE Sense Module Benutzerhandbuch*. [pdf]. 2022. URL: <https://de.manuals.plus/arduino/abx00031-nano-33-ble-sense-module-manual> (visited on 06/26/2023).
- [Ardb] *Arduino Libraries – SSD1306Ascii*. 2023. URL: <https://reference.arduino.cc/reference/en/libraries/ssd1306ascii/> (visited on 06/26/2023).
- [Ardc] *Arduino Libraries - PDM*. 2023. URL: <https://docs.arduino.cc/learn/built-in-libraries/pdm> (visited on 06/14/2023).
- [Ardd] *Arduino Reference – Wire*. 2022. URL: <https://www.arduino.cc/reference/en/language/functions/communication/wire/> (visited on 06/26/2023).
- [Arde] *Arduino Reference – Wire.setClock()*. 2022. URL: <https://www.arduino.cc/reference/en/language/functions/communication/wire/setclock/> (visited on 06/26/2023).
- [Ardf] *Arduino Reference – loop()*. 2019. URL: <https://www.arduino.cc/reference/en/language/structure/sketch/loop/> (visited on 06/26/2023).
- [Ardg] *Arduino Reference – setup()*. 2019. URL: <https://www.arduino.cc/reference/en/language/structure/sketch/setup/> (visited on 06/26/2023).
- [Ardh] *Arduino Referenz – pinMode()*. 2019. URL: <https://reference.arduino.cc/reference/de/language/functions/digital-io/pinmode/> (visited on 06/26/2023).
- [Ardi] *Arduino Tiny Machine Learning Kit*. [pdf]. 2022. URL: <https://store.arduino.cc/products/arduino-tiny-machine-learning-kit> (visited on 06/23/2023).
- [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>.
- [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>.

- [Bab+23] N. R. Babu et al. “Case Study on Ni-MH Battery”. In: *2023 2nd International Conference on Applied Artificial Intelligence and Computing (ICAAIC)* (2023), pp. 1559–1564.
- [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>.
- [Cho+19] A. Chowdhery et al. *Visual Wake Words Dataset*. [pdf], [github]. 2019. DOI: 10.48550/ARXIV.1906.05721. arXiv: 1906.05721 [cs.CV]. URL: <https://github.com/Mxbonn/visualwakewords>.
- [DMM20] K. Dokic, M. Martinovic, and D. Mandusic. “Inference speed and quantisation of neural networks with TensorFlow Lite for Microcontrollers framework”. In: *2020 5th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM)*. [pdf]. 2020, pp. 1–6. DOI: 10.1109/SEEDA-CECNSM49515.2020.9221846.
- [Dej18] Dejan. *How to Control Servo Motors with Arduino – Complete Guide*. 2018.
- [Dew+14] A. Dewan et al. “Alternative power sources for remote sensors: A review”. In: *Journal of Power Sources* 245 (2014), pp. 129–143. ISSN: 0378-7753. DOI: <https://doi.org/10.1016/j.jpowsour.2013.06.081>. URL: <https://www.sciencedirect.com/science/article/pii/S0378775313010884>.
- [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>.
- [Fun1] *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/>.
- [HP20] C. Holz and A. Pusch. “Do powerbanks deliver what they advertise? Measuring voltage, current, power, energy and charge of powerbanks with an Arduino”. In: *Physics Education* 55.2 (2020), p. 025013. DOI: 10.1088/1361-6552/ab630c. URL: <https://dx.doi.org/10.1088/1361-6552/ab630c>.
- [HSH17] E. Hering and G. Schönfelder Hrsg. *Sensoren in Wissenschaft und Technik*. Wiesbaden: Springer Vieweg, 2017. ISBN: 978-3-658-12561-5.
- [Haz] *Event Driven Architecture*. accessed date 19.04.2022. 2022. URL: <https://hazelcast.com/glossary/event-driven-architecture/>.
- [Ibr18] D. Ibrahim. *Motorsteuerung mit Arduino & Raspberry Pi*. Aachen: Elektor, 2018. ISBN: 978-3-895-76336-6.
- [Iot] *BME280 Temperatursensor – Arduino Sketch*. 2020. URL: <https://iotspace.dev/bme280-temperatursensor-arduino-sketch/> (visited on 06/23/0023).
- [Jam] *High End Micro*. 033212. Jamara e.K., 2018.
- [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>.
- [Kon+18] L. Kong et al. “Li-Ion Battery Fire Hazards and Safety Strategies”. In: *Energies* 11.9 (2018). ISSN: 1996-1073. DOI: 10.3390/en11092191. URL: <https://www.mdpi.com/1996-1073/11/9/2191>.
- [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>.
- [Las] *Interface BME280 Temperature, Humidity and Pressure Sensor with Arduino*. 2023. URL: <https://lastminuteengineers.com/bme280-arduino-tutorial/> (visited on 06/23/0023).
- [Li+21] A. Li et al. “A Review on Lithium-Ion Battery Separators towards Enhanced Safety Performances and Modelling Approaches”. In: *Molecules* 26.2 (2021). DOI: 10.3390/molecules26020478. URL: <https://www.mdpi.com/1420-3049/26/2/478>.

- [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>.
- [MW23] H. Müller and F. Weichert. *Vorkurs Informatik – Der Einstieg ins Informatikstudium*. Wiesbaden: Springer Vieweg, 2023. ISBN: 978-3-658-36467-0.
- [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/>.
- [Ouy+19] D. Ouyang et al. “A Review on the Thermal Hazards of the Lithium-Ion Battery and the Corresponding Countermeasures”. In: *Applied Sciences* 9.12 (2019). ISSN: 2076-3417. DOI: 10.3390/app9122483. URL: <https://www.mdpi.com/2076-3417/9/12/2483>.
- [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>.
- [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.
- [SSC16] L. Samsung SDI Co., ed. *Technology – The Four Components of a Li-ion Battery*. 2016. URL: <https://www.samsungsdi.com/column/technology/detail/55272.html>.
- [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.
- [Sch22] T. Scherer. “Batteriemanagement”. In: *Elektor special: Stromversorgung und Batterien* (2022), pp. 6–8.
- [Sim] *SBC-OLED01 - Datenblatt*. [pdf]. SIMAC Electronics GmbH. URL: https://cdn-reichelt.de/documents/datenblatt/A300/DEBO_OLED_0-96_DB-DE.pdf.
- [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).

- [Sun+20a] P. Sun et al. “A Review of Battery Fires in Electric Vehicles”. In: *Fire Technology* 56.4 (2020), pp. 1361–1410. DOI: 10.1007/s10694-019-00944-3. URL: <https://doi.org/10.1007/s10694-019-00944-3>.
- [Sun+20b] N. Sunanda et al. “Smart Instant Charging of Power Banks”. In: *IOP Conference Series: Materials Science and Engineering* 981.2 (2020), p. 022066. DOI: 10.1088/1757-899X/981/2/022066. URL: <https://doi.org/10.1088/1757-899X/981/2/022066>.
- [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.
- [Vid+19] C. Vidal et al. “XEV Li-Ion Battery Low Temperature Effects - Review”. In: *IEEE Transactions on Vehicular Technology* PP (Mar. 2019), pp. 1–1. DOI: 10.1109/TVT.2019.2906487.
- [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>.
- [Waq+21] D. M. Waqar et al. “Design of a Speech Anger Recognition System on Arduino Nano 33 BLE Sense”. In: *2021 IEEE 7th International Conference on Smart Instrumentation, Measurement and Applications (ICSIMA)*. [pdf]. 2021, pp. 64–69. DOI: 10.1109/ICSIMA50015.2021.9526323.
- [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>.
- [Yan+17] C. Yang et al. “Flexible Aqueous Li-Ion Battery with High Energy and Power Densities”. In: *Advanced Materials* 29 (Oct. 2017), p. 1701972. DOI: 10.1002/adma.201701972.

Index

- File
- .txt, 199
 - „person_detection“, 172
 - Adafruit-GFX.h, 100
 - Adafruit-SSD1306.h, 100
 - ArduCAM, 173, 181
 - Arducam, 172
 - Arduino.h, 146
 - arduino/constants.cc, 148
 - arduino/detection_responder.cc, 169
 - arduino/image_provider.cc, 168
 - Arduino/libraries, 173
 - Arduino/libraries/ArduCAM/memoriesaver.h, 173, 177
 - Arduino/libraries/JPEGDecoder/src/User_PDM.h, 54
 - Config.h, 173, 177
 - arduino_constants.cpp, 148, 151
 - Arduino_TensorFlowLite, 171, 176
 - ArduinoBLE, 118
 - ArduinoBLE.h, 115
 - cactus_io_BME280, 90
 - cactus_io_BME280_I2C, 90
 - constants.cc, 145
 - constants.h, 146
 - detection_responder.cc, 163, 167
 - detection_responder.h, 163
 - detection_responder_test.cc, 158, 163
 - ESP8266, 182
 - Examples→_Arduino_TensorFlowLite_, 148
 - Examples->Arduino_TensorFlowLite, 172
 - Fade, 49
 - File/Examples/01.Basics, 49
 - Hello World, 103, 104
 - hello_world, 148
 - hello_world/arduino/constants.cc, 145
 - hello_world/arduino/output_handler.cc, 146
 - hello_world/constants.cc, 145
 - hello_world/output_handler.cc, 146
 - HelloWorldWire, 99
 - host_app, 182
 - image_provider.cc, 163, 167, 168
 - image_provider.h, 162
 - image_provider_test.cc, 158, 162
 - JPEGDecoder, 169, 173
 - LSM9DS1, 118
 - Mag_raw.txt, 35
 - main.cc, 148
 - main_functions.cc, 148, 172
 - memoriesaver.h, 182
 - micro/arduino/debug_log.cc, 148
 - micro/debug_log.cc, 148
 - Mini_5MP_Plus, 182
 - model_settings.cc, 160, 161
 - model_settings.h, 160, 162
 - Nano33BLE_Led_A0.ino, 114
 - output.txt, 199
 - output_handler.cc, 148, 151
 - output_handler.h, 146
 - Person_detection, 172
 - person_detection.zip, 176
 - person_detection/image_provider.cc, 162
 - Person_detection_test.cc, 158
 - person_detection_test.cc, 158
 - person_image_data.h, 160
 - RaspberryPi, 182
 - README.md, 155, 171
 - RevC, 182
 - Shield_V2, 182
 - SimpleAccelerometer, 70
 - Sketch -> Include Library -> Manage Library, 115
 - SSD1306Ascii.h, 71
 - tensorflow/lite/micro/tools/make/downloads/person_model_grayscale, 162
 - tf_lite_micro_person_data_grayscale.zip, 158, 160
 - UTFT4ArduCAM_SPI, 181
 - Wire.h, 71, 100
 - Inertial Measurement Unit
 - hello_world/arduino/output_handler.cc, see IMU, 13, 15, 33
 - Acoustic Overload Point
 - see AOP, 15, 37
 - AOP, 15, 37
 - Central Processing Unit
 - see CPU, 15, 193
 - CNN, 13, 15, 196, 200, 202–204
 - Convolutional Neural Network

see CNN, 13, 15, 196
CPU, 15, 193

 General Purpose Input Output
 see GPIO, 15, 84
 GPIO, 15, 84
 GPU, 15, 193
 Graphics Processing Unit
 see GPU, 15, 193

 I²C, 15, 79, 82
 IDE, 15, 49, 81, 82, 195
 IMU, 13, 15, 33–35, 201, 202, 228
 Integrated Development Environment
 see IDE, 15, 49
 Inter-Integrated Circuit
 see I²C, 15, 79

 KDD, 15, 196
 Knowledge Discovery in Databases
 see KDD, 15, 196

 LED, 15, 79, 81, 84, 195
 Light Emitting Diode
 see LED, 15, 79

 mcd, 15, 80
 Millicandela
 see mcd, 15, 80

 OLED, 15, 79, 81, 82
 Organic Light Emitting Diode
 see OLED, 15, 79

 PDM, 15, 37, 54
 Pulse Density Modulation
 see PDM, 15, 37

 SCL, 15, 59, 79
 SDA, 15, 59, 79
 Serial Clock
 see SCL, 15, 59
 Serial Data
 see SDA, 15, 59

 UART, 15, 83
 Universal Asynchronous Receiver Transmitter
 see UART, 15, 83