



Semester Project

Face Recognition Using Portenta H7

Author:

Vatsal Mahajan (7025694)

Author:

Manoj Selvaraju (7025649)

Author:

Vijay Singh (7025700)

Course of Studies:

Department of Electrical and Computer Science

First examiner: Prof. Dr. Elmar Wings

Submission date: January 31, 2025

Contents

Contents	i
List of Figures	xi
List of Tables	xvii
List of Listings	xix
I. Introduction	1
1. Face Recognition System for Access Control	3
1.1. Introduction	3
1.2. Applications of Face Recognition in Access Control Systems	3
1.2.1. Building Access and Security	3
1.2.2. Smart Home Systems	4
1.2.3. Time and Attendance Tracking	4
1.2.4. Healthcare and Critical Facilities	4
1.3. Topic Description	4
1.4. Challenges and Limitations	5
1.5. Report Structure	6
II. Arduino Development Tools	7
2. Arduino IDE 2.3.2	9
2.1. Overview of Arduino IDE	9
2.2. Installation on MacOS	10
2.3. Installation on Windows	12
2.3.1. Download	12
2.3.2. Installation	12
2.4. Features	14
2.4.1. Sketchbook	14
2.4.2. Boards Manager	14
2.4.3. Library Manager	15
2.4.4. Serial Monitor	15
2.4.5. Serial Plotter	16

2.4.6. Debugging	16
2.4.7. Autocompletion	17
2.5. Examples	17
2.6. Conclusion	17
3. Arduino Web Version	19
3.1. Introduction	19
3.1.1. Using the online IDE	19
3.2. Arduino Libraries	21
3.3. Data Security	21
3.3.1. Authentication and Authorization	22
3.3.2. Secure Code Execution	22
3.3.3. Protection Against Malware	22
3.3.4. Privacy Policies	22
3.3.5. Regular Updates and Maintenance	22
3.3.6. User Awareness	23
4. Arduino Command Line Interface (CLI)	25
4.1. Installing and using the CLI	25
4.1.1. Arduino CLI License	27
4.2. Configuration of the Arduino CLI	27
4.2.1. Configuration keys	27
4.2.2. Configuration methods	30
4.3. Overview of functions	32
4.3.1. Basic functions	34
4.3.2. Command reference	35
4.4. First steps with Arduino Portenta H7 using the CLI	42
4.4.1. Recognizing the connected boards	42
4.4.2. Configuration of Arduino CLI	43
4.4.3. Creating a new sketch	44
4.4.4. Compiling a sketch	44
4.4.5. Uploading a sketch	45
4.4.6. Installation of Libraries	45
4.4.7. Example program "Hello World"	45
5. Batch File	47
5.1. Batch File	47
5.2. List Batch File	47
5.3. Explanation of Batch commands	47
III. Portenta H7	53
6. PortentaH7	55
6.1. Introduction	55

6.2. Brief Description of Hardware Components in Arduino Portenta H7	56
7. First Step with the PortentaH7	61
7.1. Introduction	61
7.2. Configuration	61
7.2.1. Example Blink Sketch	61
7.2.2. Serial Blink Example for Dual Core Processing	64
8. Portenta Vision Shield	67
8.1. Key Features	67
8.2. Hardware Components	69
8.2.1. LoRa Transceiver Module	69
8.2.2. Microcontroller Unit (MCU)	69
8.2.3. Antenna	70
8.2.4. Power Supply	70
8.2.5. Sensors and Peripherals	70
8.2.6. Memory	70
8.3. Application Examples	71
9. First Step with Portenta Vision Shield	73
9.1. Introduction	73
9.2. Getting Started With the Portenta Vision Shield Camera	74
9.2.1. Required Hardware and Software:	74
9.2.2. Instructions:	74
9.2.3. Conclusion:	81
9.3. Saving Bitmap Camera Images to the SD Card and PC	85
9.3.1. Goals:	85
9.3.2. Required Hardware and Software:	85
9.3.3. Instructions:	86
9.3.4. Conclusion	95
9.4. Creating a Basic Face Filter With OpenMV	96
9.4.1. Goals:	96
9.4.2. Required Hardware and Software:	96
9.4.3. The Haar Cascade Algorithm	96
9.4.4. Instructions:	97
9.4.5. Conclusion	101
10. Portenta Cat. M1/NB IoT GNSS Shield)	103
10.1. Introduction	103
10.2. Portenta Cat. M1/NB IoT GNSS Shield	104
10.2.1. Board Topology	104
10.2.2. Features	107
10.2.3. Compabilities	108
10.2.4. GNSS Capabilities	109

10.2.5. GSM Capabilities	109
10.3. Portenta Cat. M1/NB IoT GNSS Shield Application Examples	110
11. First Step with Portenta Cat. M1/NB IoT GNSS Shield	111
11.1. Introduction	111
11.1.1. Configuration	111
11.1.2. Requirements	111
11.1.3. Procedures	111
11.1.4. Result of Sketch	114
IV. Arduino PortentaH7 - Onboard Sensors	117
12. Power LED	119
12.1. Description	119
12.2. Specific Sensor	120
12.3. Specification	120
12.4. Simple Code	120
12.5. Test	122
12.5.1. Simple Function Test	122
12.5.2. Test all Functions	122
12.6. Simple Application	122
13. Sensor Inertial Mesure Unit	125
13.1. General Description	125
13.1.1. Always On Mode	125
13.1.2. Tilt detection	126
13.1.3. Significant Motion Detection	126
13.2. Specific Sensor	126
13.2.1. LSM9DS1	126
13.2.2. Accelerometer	127
13.2.3. Gyroscope	128
13.3. Specification	128
13.3.1. Accelerometer Sensor	128
13.3.2. PIN Connction	129
13.4. Libraries	130
13.4.1. Library Description	130
13.4.2. Installation	131
13.5. Function	132
13.5.1. Library <code>Wire.h</code>	132
13.5.2. Function <code>IMU.begin()</code>	133
13.5.3. Function <code>IMU.end()</code>	133
13.5.4. Function <code>IMU.readAcceleration(x,y,z)</code>	135

13.5.5. Function <code>IMU.readGyroscope()</code>	135
13.5.6. Function <code>IMU.accelerationAvailable()</code>	136
13.5.7. Function <code>IMU.gyroscopeAvailable()</code>	137
13.5.8. Function <code>IMU.accelerationSampleRate()</code>	137
13.5.9. Function <code>IMU.gyroscopeSampleRate()</code>	138
13.6. Calibration	139
13.6.1. Introduction	139
13.6.2. Standard Operating Procedure	139
13.6.3. Reset part	139
13.7. Simple Application of code	140
13.7.1. Initialization example	140
13.7.2. Application example – gyroscope application	140
13.8. Tests	140
13.8.1. Simple Test Function	140
13.8.2. Acquisition Test	140
13.9. Error	144
13.9.1. Type of Error	144
13.9.2. Library <code>Kalman.h</code>	145
14. Bluetooth	149
14.1. Description	149
14.2. Specific Sensor	149
14.3. Specification	150
14.4. TESTS	150
14.4.1. control the built-in LED using Bluetooth	153
15. WIFI Module	163
15.1. Description	163
15.2. Specific Sensor	163
15.3. Specification	164
15.4. TESTS	164
15.4.1. WiFi Access Point	164
16. LORA	179
16.1. Description	179
16.2. Key Features of LoRa	179
16.3. Hardware Components of the LoRa Implementation	181
16.3.1. LoRa Transceiver Module	181
16.3.2. Antenna	181
16.3.3. Microcontroller Unit (MCU)	182
16.3.4. Power Supply	182
16.3.5. Interface	183
16.3.6. Memory	183
16.3.7. Peer-To-Peer (P2P) LoRa Communication	184

16.4. Low-Power Wide Area Networks(LPWAN)	186
16.4.1. Difference between LoRa and LoRaWAN?	187
16.4.2. LoRaWAN Network Architecture	187
16.4.3. Data Rates	188
16.4.4. Classes	190
16.4.5. Authentication and Security	191
16.4.6. Connecting to the LoRaWAN-TTN	192
16.5. Tests	198
16.5.1. Connecting the Portenta Vision Shield to TTN Using LoRa	198
16.5.2. Designing a Double LoRa Connectivity for the Ar- duino Portenta H7	201
16.5.3. LoRaWAN vision with Edge Impulse and Portenta H7	206
V. Tools and Packages	213
17. TensorFlow Lite	215
17.1. TensorFlow	215
17.2. pyTorch	215
17.3. JAX	216
17.4. Keras	216
17.5. TensorFlow Lite	216
17.6. Why TensorFlow Lite?	217
17.7. Working Procedure of TensorFlow Lite	217
17.7.1. Model Creation and Training	218
17.7.2. Model Conversion	218
17.7.3. Model Deployment	220
17.7.4. Running Inference	222
18. Comprehensive TensorFlow Lite Usage Methodologies	225
18.1. TensorFlow Lite as a Library for Arduino	225
18.1.1. Usage and Integration	225
18.1.2. TensorFlow Lite Micro Arduino Example	225
18.2. TensorFlow Lite as a Library Path	229
18.2.1. Usage and Integration:	229
18.3. TensorFlow Lite as a Tool	231
18.3.1. Relationship to Other TensorFlow Lite functionality	231
18.4. TensorFlow Lite for Microcontrollers	231
18.4.1. Integrating Edge TPU with Arduino	232

19. Model Optimization	235
19.1. Why models should be optimized?	235
19.1.1. Size reduction	235
19.1.2. Latency reduction	236
19.1.3. Accelerator compatibility	236
19.1.4. Trade-offs	236
19.2. Types of Optimization	236
19.2.1. Quantization	236
19.2.2. Pruning	238
19.2.3. Clustering	239
20. TensorFlow Installation on PC	243
20.1. System Requirements	243
20.2. Software Requirements	243
20.3. Installation procedure	243
20.4. TensorFlow Lite Library Installation on Arduino IDE	244
21. MicroPython & OpenMV	247
21.1. Introduction	247
21.2. MicroPython	247
21.3. OpenMV	248
21.4. Implementing MicroPython on Portenta H7 with OpenMV IDE	249
21.4.1. Example script for blinking builtin LED on the Arduino Portenta H7 with OpenMV	250
22. Edge Impulse	251
22.1. Edge Impulse	251
22.1.1. Features of the Edge Impulse Platform	251
22.1.2. Edge Impulse Framework Installation	252
22.1.3. Configuring Arduino Portenta H7 to Edge Impulse	253
22.1.4. Edge Impulse GUI Description	255
VI. Domain Machine Learning	259
23. Algorithms	261
23.1. Description	261
23.2. Application	261
23.3. Hyperparameter	261
23.4. Requirements	262
23.5. Input	262
23.6. Output	262
23.7. Example with a Program	262

24. Packages	267
24.1. edge-impulse-sdk, Version 1.3.0	267
24.1.1. Introduction	267
24.1.2. Installation	267
24.1.3. Example - Description	267
24.1.4. Example - Manual	267
24.1.5. Example - code	268
24.1.6. Future Reading	270
24.2. TensorFlow Lite, Version 2.12.0	271
24.2.1. Introduction	271
24.2.2. Installation	271
24.2.3. Example - Description	271
24.2.4. Example - Manual	271
24.2.5. Example - Code	271
24.2.6. Example - Files	272
24.2.7. Futher reading	272
VII. Methodology	273
25. Methodology	275
25.1. KDD Process	275
25.2. Justification	276
VIII. KDD	277
26. Database	279
26.1. Data Selection	279
26.1.1. Origin	279
26.1.2. Features	279
26.1.3. Quality	279
26.1.4. Quantity	280
26.1.5. Properties	280
26.1.6. Fairness/Bias	281
26.2. Data Processing	281
26.2.1. Outliers	281
26.2.2. Anomalies	281
26.2.3. Augmentation	281
27. Data Transformation and Data Mining	283
27.1. Data Transformation	283
27.2. Data Mining	283
27.2.1. Hyperparameters	283

27.2.2. Input	284
27.2.3. Training	284
27.2.4. Interpretation	284
27.2.5. Output	285
28. Evaluation and Verification	287
28.1. Evaluation	287
28.1.1. Concept	287
28.1.2. Application	288
28.1.3. Result	289
28.2. Validation	289
28.2.1. Concept	289
28.2.2. Ideas	289
28.2.3. Conclusion	291
28.2.4. Future Enhancements	291
29. Application Deployment	293
29.1. Description	293
29.2. Deployment Architecture	293
29.2.1. Edge Device (Arduino Portenta H7 + Vision Shield)	293
29.3. Flow Chart of Each Step in the Deployment	294
29.4. ML Pipeline	295
29.5. Process	295
29.5.1. Connect the Vision Shield to the Arduino Portenta H7	295
29.5.2. Connect the Arduino Portenta H7 to the System	296
29.5.3. Check Power Supply and Board Initialization	296
29.5.4. Load the Face Recognition Sketch	297
29.5.5. Compile and Upload the Sketch	297
29.5.6. Run and Test the Model	297
30. Monitoring and Maintenance	299
30.1. Idea	299
30.2. Plan/Description	299
30.3. Updating new Data	300
30.4. Data updating in the ML Pipeline	300
30.5. Checks	300
30.6. Privacy	301
30.7. Robustness	301
30.8. Process	302

IX. Appendix	303
31. Application Appendix	305
31.1. List of Materials	305
31.2. Software Bill of Materials	305
31.3. List of Packages	305
31.4. Requirement File	306
31.5. Programming Languages	306
32. doxygen - Example	307
Bibliography	309
Index	309

List of Figures

2.1.	Overview	10
2.2.	ArduinoIDE Create Agent Installation	10
2.4.	Copy to the Applications folder	11
2.3.	Open the Downloadf folder	11
2.6.	Arduino File	12
2.5.	Arduino Sketch	12
2.7.	Running the installation file	13
2.8.	Steps for Installation	13
2.9.	Sketchbook	14
2.10.	Board-Manager	15
2.11.	library-manager	15
2.12.	serial-monitor	16
2.13.	Debugging	16
2.14.	autocomplete	17
2.15.	Example	18
3.1.	web-editor	20
3.2.	finding-an-example	21
4.1.	Adding arduino-cli.exe file storage location to PATH system variable	26
4.2.	Installation of Arduino CLI	26
4.3.	Return values of the “board list” function	43
4.4.	Updating the board index	43
4.5.	Installing the core for Arduino Portenta H7	43
4.6.	List available boards	44
4.7.	Creating a new Sketch with CLI	44
4.8.	Compiling a sketch with CLI	45
4.9.	Uploading the compiled sketch	46
4.10.	The orange LED of the Arduino starts to flash	46
6.1.	PortentaH7 Hardware	56
7.1.	Arduino Portenta H7 Connected to a laptop	62
7.2.	Installation Arduino Mbed OS Portenta Board	62
7.3.	Upload Basic Sketch	62
7.4.	Compile Blink Sketch	63

7.5. Serial Blink Sketch	64
7.6. Serial Blink Sketch	64
7.7. Output LED Color	65
8.1. Portenta Vision Shield	67
8.2. Portenta Vision Shield PinDiagram	68
8.3. Board	69
8.4. Board Topology	71
9.1. VisionShield	73
9.2. Connection VS	75
9.3. Portentaport	75
9.4. Processingimage	77
9.5. CameraOutput	81
9.6. VisionShield	85
9.7. Connection Vision Shield	86
9.8. SavingImage	92
9.9. HaarCascade	97
9.10. BitmapImage	99
9.11. ImageOutput	101
10.1. Portenta Cat. M1/NB IoT GNSS-Shield	104
10.2. Portenta Cat. M1/NB IoT GNSS Shield Pinout	106
10.3. Portenta Cat. M1/NB IoT GNSS Shield Top View	106
10.4. Portenta Cat. M1/NB IoT GNSS Shield Bottom View	106
11.1. Arduino Mbed OS Portenta core installation in Arduino IDE	112
11.2. Arduino Mbed OS Portenta core installation in Arduino IDE	113
11.3. GSM Client Code for Portenta IoT GNSS Shield	115
11.4. Arduino Mbed OS Portenta core installation in Arduino IDE	116
13.1. LSM9DS1 axis on this card	127
13.2. LSM9DS1 axis on this card	128
13.3. Explication of pin mode	130
13.4. Wire include	131
13.5. Board card installation	131
13.6. Library choice	132
14.1. Bluetooth-portentaH7	154
14.2. PortentaH7-connection	155
14.3. Bluetooth-Library	155
14.4. select-board-h7	158
14.5. Select the Port to which the Portenta is connected to	159
14.6. NRF-Connect	160
14.7. Bluetooth-scan	161

15.1. Full Interface	166
15.2. SetUpServer	166
15.3. BasicSetup	167
15.4. LEDState	167
15.5. NewTab	172
15.6. NamingNewTab	173
15.7. HeaderFile	174
15.8. Uplaoding	174
15.9. SerialMonitor	174
15.10PortentaAccessPoint	175
15.11URLAccessPoint	175
15.12HTML Page	176
15.13ClientDetails	176
15.14GETRequest	177
16.1. LoRa	180
16.2. Bandwidth vs. range of short distance, cellular and LPWA networks. Image credits: The Things Network.	180
16.3. LoRa with Antenna	182
16.4. Source: The Things Network	184
16.5. LoRaWAN Topology	185
16.6. P2P LoRa	185
16.7. Typical LoRaWAN network architecture example.	188
16.8. LoRaWAN network layers. Image credits: Semtech.	189
16.9. Class A default configuration profile. Image credits: The Things Network.	190
16.10Class B default configuration profile. Image credits: The Things Network.	191
16.11Class C default configuration profile. Image credits: The Things Network	191
16.12Over-The-Air activation process. Image credits: Heath Raftery.	192
16.13Activation by Personalization process. Image credits: Heath Raftery.	192
16.14Things Network	193
16.15Select Application	193
16.16AddingApplication	194
16.17AppParameter	194
16.18MCore	195
16.19UploadCode	196
16.20RegisteringDevice	197
16.21RegisteringDevice	197
16.22Secondstep	198
16.23TTN	198

16.24Install MKRWAN	200
16.25MKRWAN Standalone	200
16.26SerialMonitor	200
16.27Monitor Output	202
16.28Connection for SX1276 chip to the Arduino Portenta H7 .	204
16.29Communication test of two Arduino Portenta H7 by LoRa	205
17.1. TensorLite Workflow	218
18.1. TFlite Micro github Cloning to local repository	226
18.2. TensorFlow Lite as a Library for Arduino	226
18.3. Board and Port selection	227
18.4. Directory structure with TensorFlow Lite as Library Path	233
19.1. Quantization Techniques	237
19.2. Quantization Aware Training Technique by introducing simulated quantization during model training	239
19.3. Synapses and neurons in Deep Learning Model before and after pruning technique	240
19.4. Weight Clustering Optimization technique	240
20.1. Python Installation	244
20.2. Python and pip version	244
20.3. TensorFlow Installation on PC	245
20.4. TensorFlow Version	245
20.5. Arduino TensorFlowLite Library Installation	245
21.1. RGB LED blinking blue, red and green in Portenta H7 .	250
22.1. Workflow of Edge Impulse Platform	251
22.2. Node.js Installation	253
22.3. CLI commands for Version	254
22.4. Edge Impulse CLI version	254
22.5. Installing Portenta H7 board in Arduino IDE	255
22.6. Vision Shield connected to Portenta H7	256
22.7. Edge Impulse Firmware	256
22.8. Edge Impulse User Interface	257
25.1. KDD Process	276
26.1. Dataset dashboard	280
27.1. Feature Explorer	284
27.2. Model Interpretation and Output	286
28.1. ML Pipeline for Face Detection Evaluation Workflow .	288
28.2. Model Evaluaiton Metrics	290

29.1. Deployment Process Flowchart	294
29.2. Machine Learning Pipeline	295
29.3. Arduino PortentaH7 Connected to a Laptop	296
29.4. Running the inference	298
29.5. Debug Mode	298
30.1. Model Retraining	299
30.2. Process Workflow for Face Access System Monitoring	301
32.1. Doxygen	307

List of Tables

4.1. List of Basic Functions	35
6.1. Specifications Table	55
9.1. Description of the bitmap file structure.	87
10.1. Technical Specifications of Portenta Cat. M1/NB IoT GNSS-Shield	105
10.2. Portenta Cat. M1/NB IoT GNSS Shield Top View Component Description	107
10.3. Portenta Cat. M1/NB IoT GNSS Shield Bottom View Component Description	107
16.1. Specifications of LoRa Device on Portenta Vision Shield	183
16.2. LoRa Spreading Factor Specifications	189
28.1. Performance Metrics	289
31.1. Material List	305
31.2. Software Bill of Materials	305
31.3. List of Packages	305
31.4. Programming Languages	306

List of Listings

9.1. Simple sketch to capture the image from vision shield	82
9.2. Simple sketch to capture the image from vision shield	82
9.3. Simple sketch to save the image from vision shield	92
9.4. Simple sketch for bitmap image	100
12.1. Simple sketch to check the battery state using the power LED	121
12.2. Simple sketch to check the battery state using the power LED	123
12.3. Simple sketch to check the battery state using the power LED	124
13.1. Simple sketch using the sensor LSM9DS1 detection.	134
13.2. Simple test for sensor LSM9DS1 motion detection	141
13.3. Simple test for gyroscope application	142
13.4. Simple test for motion acquisition sensor LSM9DS1 acquisition	143
13.5. Simple example with Kalman Filter.	147
14.1. Simple sketch to control the LED using bluetooth	150
14.2. Simple sketch to control the LED using bluetooth	156
15.1. Simple sketch to control the LED using bluetooth	167
23.1. Algorithm exmaple code	262
24.1. Package exmaple code	268
24.2. Example Code for running Tensorflow Lite model.	271

Part I.

Introduction

1. Face Recognition System for Access Control

1.1. Introduction

Advancements in technology have transformed smart devices, enabling them to learn and adapt through machine learning (ML) [LZW18] and edge computing [JKL+20]. Edge devices allow AI models to operate locally, reducing latency, preserving privacy, and cutting costs—ideal for real-time applications like access control. By handling data directly on devices, edge AI mitigates issues like bandwidth constraints and security concerns, supporting fast and secure decision-making even in low-connectivity environments [Cha24] [ZCL+19] [CL+23].

Face recognition is a technology that uses face image of someone to verify his identity by finding this person in a given photos database. It becomes very practical in access control systems as it does not require any physical interaction for gaining access as traditional ways with keys. Moreover, these systems only require a camera for recognition and are easy to install and use. This is why they are already in use by companies as access control to their offices, in home automation systems. [YXW+20] Face recognition in access control offers a contactless, secure, and efficient solution by authenticating individuals through unique biometric features, delivering convenience and enhanced security [SMS21] .

1.2. Applications of Face Recognition in Access Control Systems

Face recognition for access control is gaining popularity across a wide range of industries due to its security, efficiency, and convenience. The technology is applied in various settings, including:

1.2.1. Building Access and Security

Face recognition systems in commercial and residential buildings grant access only to authorized personnel, enhancing security with a contactless approach suitable for offices, healthcare, and residential complexes [Nor24]

1.2.2. Smart Home Systems

In smart homes, face recognition enables personalized, secure access, managing locks and automating entry for family members and approved visitors [SMS21]

1.2.3. Time and Attendance Tracking

Face recognition automates attendance in workplaces and schools, replacing badges with accurate, contactless tracking. [Sur+22]

1.2.4. Healthcare and Critical Facilities

In healthcare, face recognition restricts access to sensitive areas like drug storage, ensuring only authorized staff can access, providing a higher level of security in sensitive environments. [Cyb24]

The use of face recognition in access control systems simplifies user interactions by offering a contactless and efficient way to authenticate and authorize individuals.

1.3. Topic Description

In this project, we aim to develop a face recognition-based access control system using the Arduino Portenta H7 6.1 and the Vision Shield with the help of Edge Impulse 22.1. The system is designed to offer a secure, contactless, and efficient method of controlling access to restricted areas, such as offices, homes, or high-security zones.

The core functionality revolves around capturing an individual's face using the Vision Shield's camera and processing the image using a machine learning model deployed on the Arduino Portenta H7 [Ard24n]. The model is trained to recognize authorized personnel and differentiate them from unauthorized individuals. Upon successful recognition, the system grants access, such as unlocking a door or enabling entry, while denying it in case of unrecognized faces.

By deploying the solution on an embedded device like Arduino Portenta H7, we enable local image processing, which reduces the need for internet connectivity. Although the Portenta H7 [Ard24m] is not a strict real-time device, it performs sufficiently fast for practical access control applications. The system is also scalable, allowing for easy updates to include new faces as needed, making it a flexible and secure solution for access control.

1.4. Challenges and Limitations

- **Processing Power:** The Arduino Portenta H7 provides up to 1 MB of internal RAM and 8 MB of external SDRAM, suitable for constrained models but it's typically not enough for running big networks without pruning, quantization, or other optimizations. [AAO21a] [WHW+23]
- **Storage Constraints:** With 2 MB of internal Flash and 16 MB of external NOR Flash, the Portenta H7 provides sufficient storage for basic applications. However, tasks like managing large face datasets or multiple user profiles can quickly exceed its capacity, requiring careful memory management.
- **Camera Quality and Vision Shield Limitations:** The Vision Shield camera we utilize QQVGA resolution to support upto 120FPS, which may affect face recognition in low or dynamic lighting. [Ard24n].
- **Latency and Performance:** Real-time face recognition on the Portenta H7 can experience delays, particularly with multiple faces in view or in crowded settings, as it processes each image in roughly 20–50 ms. [WHW+23] [AAO21b]
- **Model customization:** While Edge Impulse offers many advantages, it has limitations compared to platforms like TensorFlow and PyTorch, which provide greater flexibility in model customization. Edge Impulse is focused on simplicity and rapid deployment, leaving less options for advanced tuning. [LLL+22]
- **Environmental Constraints:** Variability in ambient lighting, such as bright sunlight or shadows, can impact model accuracy. Additionally, camera placement is crucial; poor angles can reduce recognition effectiveness.
- **Face Dataset and Profile Capacity:** The limited dataset (3 individuals, 50–70 images each) restricts the model's ability to generalize well across diverse faces. Profile management is required due to storage constraints, and processing may need to be offloaded periodically.

1.5. Report Structure

The first chapter introduces the project, covering its background, applications, topic description and challenges. Chapter two discusses the

development tools used, including the Arduino IDE and its web version. Chapter three outlines the hardware specifications of the Arduino Portenta H7, Vision Shield and Cat. M1/NB IoT GNSS Shield.

Chapter four focuses on the onboard sensors of the Portenta H7 and outlines their features and capabilities. The fifth chapter covers the description of major tools and libraries relevant to Machine Learning along with TensorFlow Lite, detailing its integration for efficient model inference and finally with Edge impulse studio which is used in our project. Chapter six provides an overview of machine learning algorithms and packages relevant to the system. Chapter seven introduces the Knowledge Discovery in Databases (KDD) methodology and explains its significance for the project. Finally, chapter eight describes the practical implementation of the KDD process in developing the face recognition system.

Part II.

Arduino Development Tools

2. Arduino IDE 2.3.2

2.1. Overview of Arduino IDE

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. 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 Arduino IDE features a new sidebar, making the most commonly used toolsas.

2.1 [Doc24d] [Ard25a]

- **Verify / Upload-** compile and upload your code to your Arduino Board.
- **Select Board and Port-** detected Arduino boards automatically show up here, along with the port number.
- **Sketchbook-** here you will find all of your sketches locally stored on your computer. Additionally, you can sync with the Arduino Cloud, and also obtain your sketches from the online environment.
- **Boards Manager-** browse through Arduino and third party packages that can be installed. For example, using a MKR WiFi 1010 board requires the Arduino SAMD Boards package installed.
- **Library Manager-** browse through thousands of Arduino libraries, made by Arduino and its community.
- **Debugger-** test and debug programs in real time.

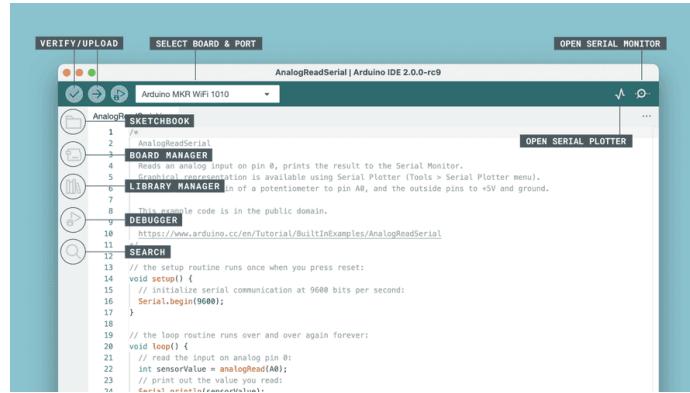


Figure 2.1.: Overview

- **Search-** search for keywords in your code.
- **Open Serial Monitor-** opens the Serial Monitor tool, as a new tab in the console.

2.2. Installation on MacOS

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 2.3.2 for a MacOS(Sonoma 14.4.1) operating system. The set up file name is [arduino-ide_2.3.2_macOS_64bit.dmg](#) and the size of it is 1,93,600 KB. The file is in Zip format. If you use Safari it will be automatically extracted. If you use a different browser you may need to extract it manually. The most recent offline arduino IDE 2.3.2 can be seen in Figure 2.2 it is also supportive for all operating systems [Doc25d].



Figure 2.2.: ArduinoIDE Create Agent Installation

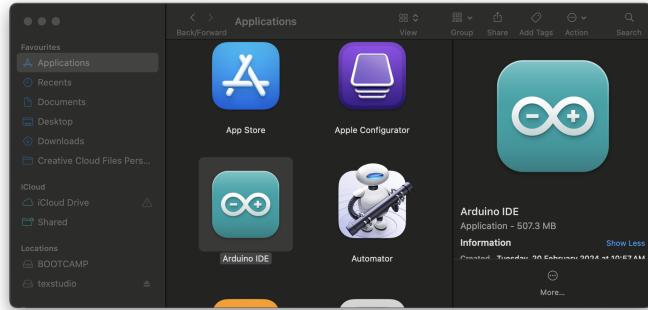


Figure 2.4.: Copy to the Applications folder

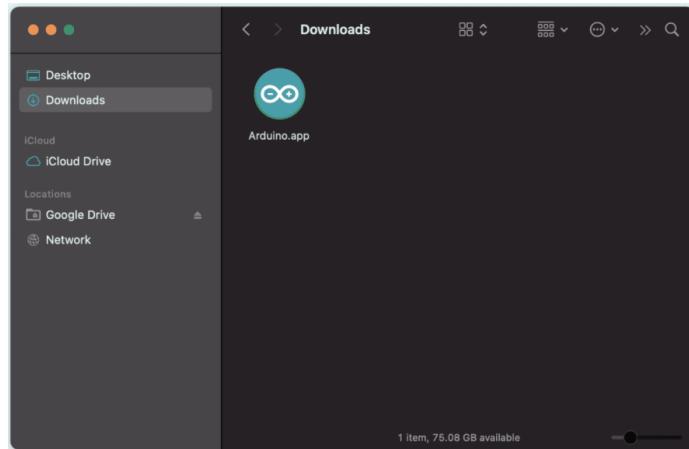


Figure 2.3.: Open the Downloadf folder

Copy the Arduino application bundle into the Applications folder (or elsewhere on your computer) then it look like Figure 2.4

It can be seen from the Figure 2.2 that the basic Arduino sketch has two parts.

- **void setup():** This function returns void and we do the intiliazation such as the output LED color, specifying the core etc
- **void loop():** In this function we define functions which are to be performed throughout the loop. These codes are placed between paranthesis and each function has a return type, here it has void return type.

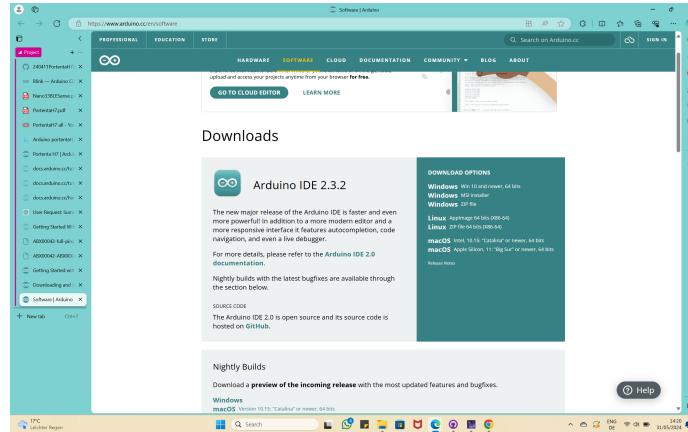


Figure 2.6.: Arduino File

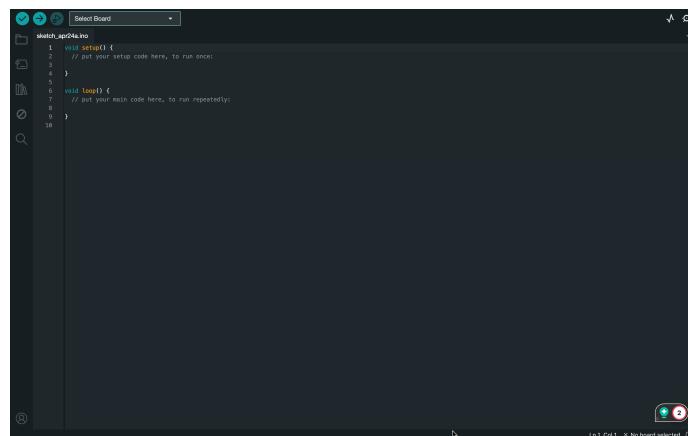


Figure 2.5.: Arduino Sketch

2.3. Installation on Windows

2.3.1. Download

You can easily download the editor from the Arduino Software page. 2.6 [Doc25c]

2.3.2. Installation

To install the Arduino IDE 2 on a Windows computer, simply run the file downloaded from the software page. 2.7 2.8 [Doc25c]

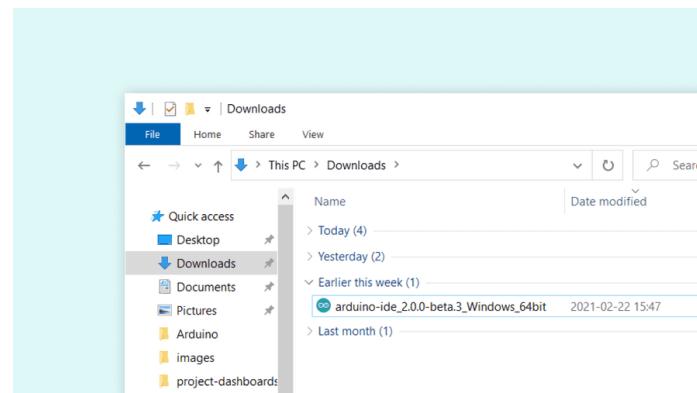


Figure 2.7.: Running the installation file

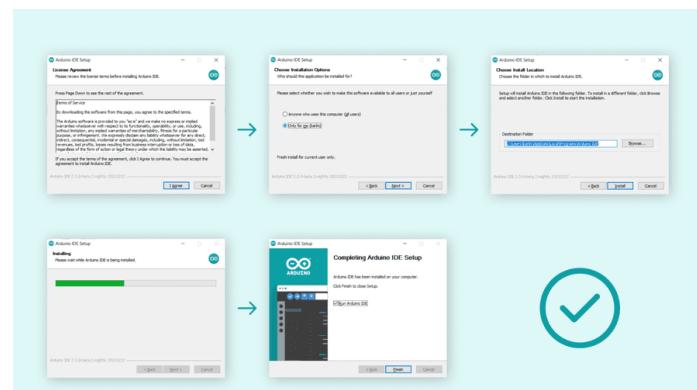


Figure 2.8.: Steps for Installation

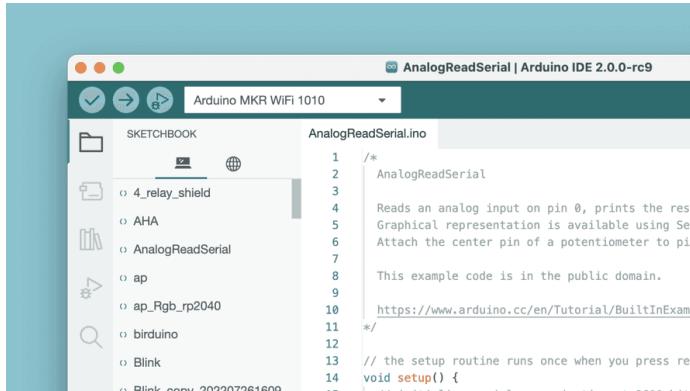


Figure 2.9.: Sketchbook

2.4. Features

The Arduino IDE 2 is a versatile editor with many features. You can install libraries directly, sync your sketches with Arduino Cloud, debug your sketches and much more. In this section, some of the core features are listed, along with a link to a more detailed article. [Doc24d] [Ard25a]

2.4.1. Sketchbook

Your sketchbook is where your code files are stored. Arduino sketches are saved as .ino files, and must be stored in a folder of the exact name. For example, a sketch named my sketch.ino must be stored in a folder named my sketch. 2.9

Typically, your sketches are saved in a folder named Arduino in your Documents folder.

To access your sketchbook, click on the folder icon located in the sidebar.

2.4.2. Boards Manager

With the Boards Manager, you can browse and install board packages. A board package contains the "instructions" for compiling your code to the boards that are included in the board package. 2.10

There are several Arduino board packages available, such as avr, samd, megaavr and more.

2.4.3. Library Manager

With the library manager you can browse and install thousands of libraries. Libraries are extensions of the Arduino API, and makes it easier to for

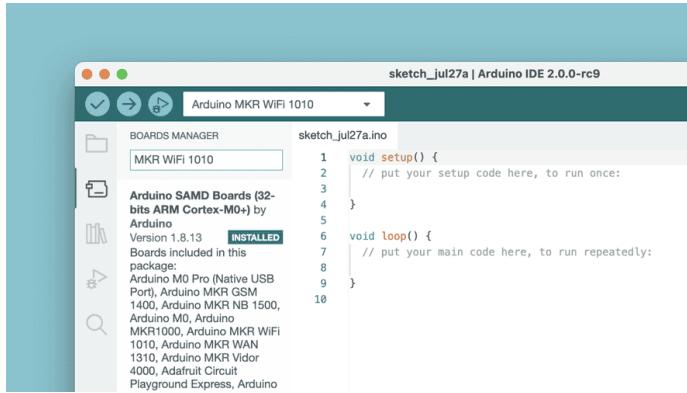


Figure 2.10.: Board-Manager

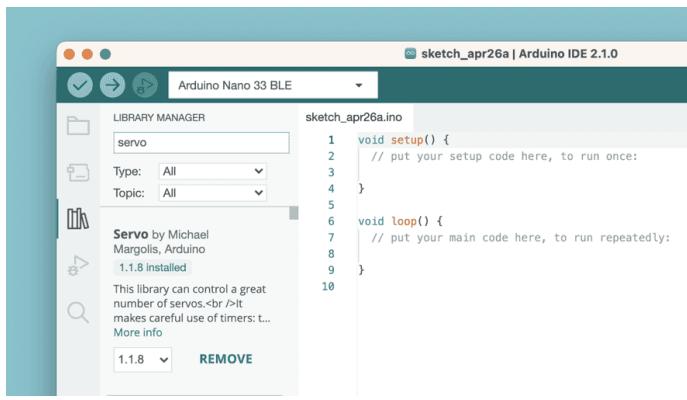


Figure 2.11.: library-manager

example control a servo motor, read specific sensors, or use a Wi-Fi module. 2.11

2.4.4. Serial Monitor

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

Historically, this tool has been located in a separate window, but is now integrated with the editor. This makes it easy to have multiple instances running at the same time on your computer. 2.12

2.4.5. Serial Plotter

The Serial Plotter tool is great for visualizing data using graphs, and to monitor for example peaks in voltage.

You can monitor several variables simultaneously, with options to enable only certain types.

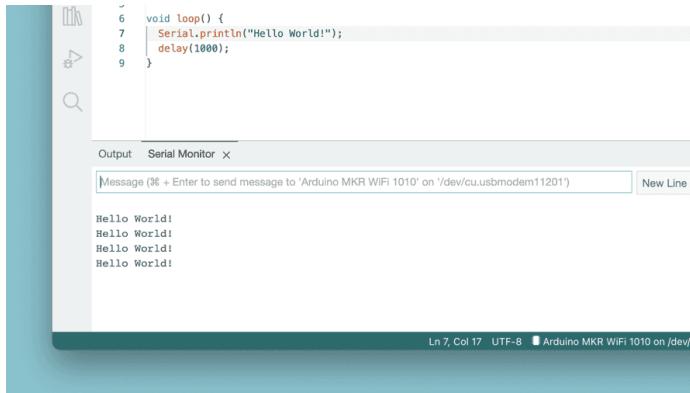


Figure 2.12.: serial-monitor

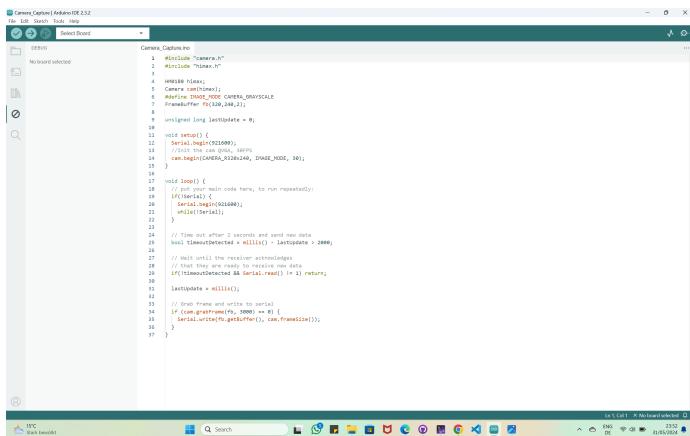


Figure 2.13.: Debugging

2.4.6. Debugging

The debugger tool is used to test and debug programs, hence the name. It can be used to navigate through a program's execution in a controlled manner. 2.13

2.4.7. Autocompletion

Autocompletion is a must-have for code editors, and the 2 version comes well equipped. When writing code, this is useful to understand more about the elements of the Arduino API.

Note that you always need to select your board for autocompletion to work. 2.14

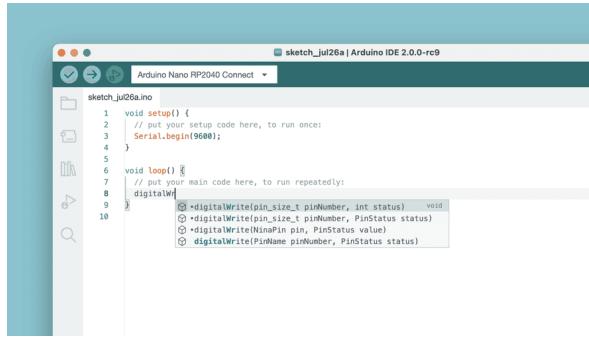


Figure 2.14.: autocomplete

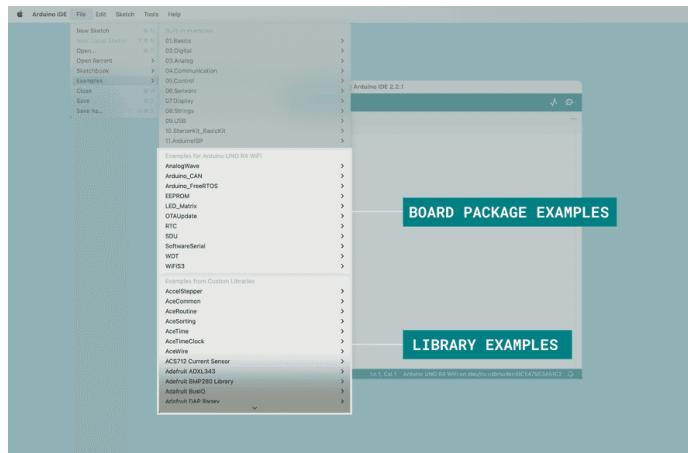


Figure 2.15.: Example

2.5. Examples

An important part of the Arduino Documentation are the example sketches that come bundled with libraries. They will show examples of the functions used in practice, illustrating the intended use and features of a library.

Libraries that come bundled as a part of a boards package may also include libraries, and those libraries often include example sketches.

To open the example sketches bundled in either the libraries you have installed manually or that come bundled in board packages, navigate to **File > Examples** and find the library you're searching for in the list that appears. 2.15

In the image above, you can see what the examples list looks like when Arduino board is connected to your computer.

From here, you can for example navigate to **File > Examples > LED Matrix > MatrixIntro** and upload the sketch to your board to show the Tetris animation that came pre-loaded on your UNO R4 WiFi when

you first took it out of its box.

2.6. Conclusion

In this guide, we have presented a series of features and more detailed articles to follow, so that you can enjoy each and every one of the features included in the IDE 2.

3. Arduino Web Version

3.1. Introduction

The Arduino Web Editor allows you to write code and upload sketches to any official Arduino board directly from your web browser (Chrome, Firefox, Safari and Edge). [Doc24i]

This IDE (Integrated Development Environment) is part of Arduino Create, an online platform that enables developers to write code, access tutorials, configure boards, and share projects. Designed to provide users with a continuous workflow, Arduino Create connects the dots between each part of a developer's journey from inspiration to implementation. Meaning, you now have the ability to manage every aspect of your project right from a single dashboard.

The Arduino Web Editor is hosted online, therefore it is always be up-to-date with the latest features and support for new boards. This IDE lets you write code and save it to the Cloud, always backing it up and making it accessible from any device. It automatically recognizes any Arduino board connected to your PC, and configures itself accordingly. All you need to get started is an Arduino account. The following steps can guide you to start using the Arduino Web Editor: [Doc24i]

3.1.1. Using the online IDE

After logging in, you are ready to start using the Arduino Web Editor.

The web app is divided into three main columns. 3.1

The Arduino Web Editor's interface is as follows:

1. The first column lets you navigate between:
 - **Your Sketchbook:** a collection of all your sketches (a sketch is a program you upload on your board).
 - **Examples:** read-only sketches that demonstrate all the basic Arduino commands (built-in tab), and the behavior of your libraries (from the libraries tab).
 - **Libraries:** packages that can be included to your sketch to provide extra functionalities.
 - **Serial monitor:** a feature that enables you to monitor, receive and send data to and from your board via the USB cable.

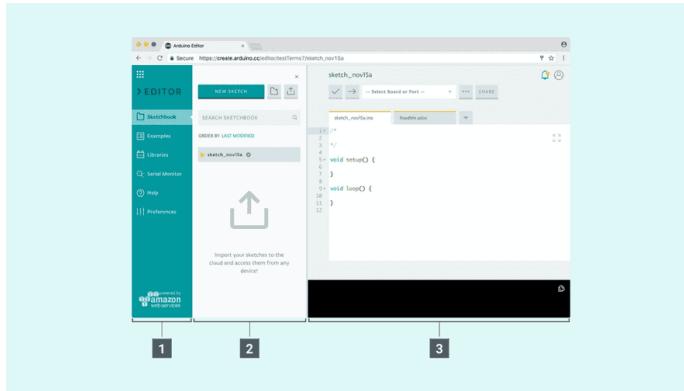


Figure 3.1.: web-editor

- **Help:** helpful links and a glossary about Arduino terms.
- **Preferences:** options to customize the look and behavior of your editor, such as text size and color theme.

2. The second column views the content of the chosen option.
3. The third column, the code area, is the one you will use the most. Here, you can write code, verify it and upload it to your boards, save your sketches on the Cloud, and share them with anyone you want.

Now that we are all set up, let's try to make the board blink!

1. Connect your Arduino or Genuino board to your computer. Boards and serial ports are auto-discovered and selectable in a single dropdown. Pick the Arduino/Genuino board you want to upload to from the list at the top of the third column.[Doc24i]
2. Let's try with an example: Choose Examples on the menu on the left (first column), then Basic and Blink. The Blink sketch is now displayed in the code area.
3. To upload it to your board, press the "Upload" button near the dropdown menu. While the code is verifying and uploading, a "BUSY" label replaces the upload button. If the upload is successful, the message "Success: done uploading" will appear in the bottom output area.
4. Once the upload is complete, you should see on your board the yellow LED with an L next to it start blinking. You can adjust the speed of blinking by changing the delay number in the parenthesis to 100, and upload the Blink sketch again. Now the LED should blink much faster 3.2.



Figure 3.2.: finding-an-example

3.2. Arduino Libraries

The process of setting up libraries on the online IDE (Arduino Cloud Editor) is quite similar to the offline one: [Doc25b] [Doc25a]

1. Login to the Arduino Cloud.
2. Create or open a sketch.
3. Open the "Libraries" tab from the left menu, and search for libraries. The list displays read-only libraries, authored and maintained by the Arduino team and its partners.
4. When you find the library, you can add it to your sketch by selecting the "Include" button. You can also see the related examples, and select a specific version, if available.
5. If you can't find a specific library on the list, you can search every existing library through the search bar. You also have the option to add them to your favorites list by clicking on the star next to the library you want. Once you star a library, you can view it under the "favorites" tab and use its examples (if available).

3.3. Data Security

- **Encryption:** The web version of Arduino IDE should use strong encryption protocols to secure data transmitted between the user's browser and the server, preventing unauthorized access.
- **Secure Storage:** User data, including login credentials and project files, should be encrypted at rest to protect against unauthorized access or tampering.

3.3.1. Authentication and Authorization

- **Strong Password Policies:** Enforce password complexity and offer multi-factor authentication to enhance account security.
- **Role-Based Access Control:** Implement role-based access to ensure users only access necessary resources, preventing unauthorized access to sensitive data.

3.3.2. Secure Code Execution

- **Sandboxing:** Execute user-uploaded code in a sandboxed environment to restrict access to system resources and prevent malicious actions.
- **Code Validation:** Validate user code to detect and prevent vulnerabilities like code injection or buffer overflows before execution.

3.3.3. Protection Against Malware

- **Code Scanning:** Use code scanning to detect and block malicious code by analyzing for malware signatures or suspicious patterns.
- **Antivirus Integration:** Integrate antivirus solutions to scan uploaded files for viruses or malicious content before execution.

3.3.4. Privacy Policies

- **Transparency:** Maintain clear privacy policies outlining data collection, storage, usage, retention periods, and third-party sharing practices.
- **User Consent:** Inform users about privacy practices and obtain explicit consent for data processing, allowing them to review and modify privacy settings.

3.3.5. Regular Updates and Maintenance

- **Patch Management:** Regularly update software components, libraries, and dependencies to address security vulnerabilities promptly.
- **Security Audits:** Conduct regular security audits and penetration testing to identify and remediate potential weaknesses.

3.3.6. User Awareness

- **Security Education:** Provide resources on secure coding practices to help users avoid vulnerabilities like XSS, SQL injection, and insecure object references.
- **Security Alerts:** Notify users about security-related issues, such as suspicious login attempts or unauthorized access.

4. Arduino Command Line Interface (CLI)

Arduino Command Line Interface (CLI) is an all-in-one solution that provides Boards/Library Managers, sketch builder, board detection, uploader, and many other tools needed to use any Arduino compatible board and platform from command line or machine interfaces. In addition to being a standalone tool, Arduino CLI is the heart of all official Arduino development software (Arduino IDE, Arduino Web Editor). This means that all functionalities provided by the development environment are also available here. This can be used under Windows, MacOS and Linux [Ard24p]. Development using the command line interpreter makes it possible to carry out quality assurance measures in a targeted manner if batch files are used consistently. This means that the configuration of the development environment can be saved and repeated in a traceable manner.

The source code can be used freely, but a license must be requested from the company for commercial use [Ard24b]. Detailed documentation is provided. [Ard24d]

The development of the Command Line Interface (CLI) focuses on three aspects.

1. On the one hand, the interface can be embedded within familiar development tools, making it easier to work with platforms like Edge Impulse for machine learning at the edge.
2. Secondly, it enables the integration of development into the Continuous Development and Continuous Integration processes. It thus enables the automation of typical software development activities.
3. Furthermore, it now enables simplified communication with edge computers, for example with a Raspberry PI.

4.1. Installing and using the CLI

As Command Line Interface (CLI) is constantly being further developed, the current version must first be downloaded from the GitHub project Arduino-cli-0.36-installation, see [Ard24b]. The version used here is 0.36.0 [Ard24a]

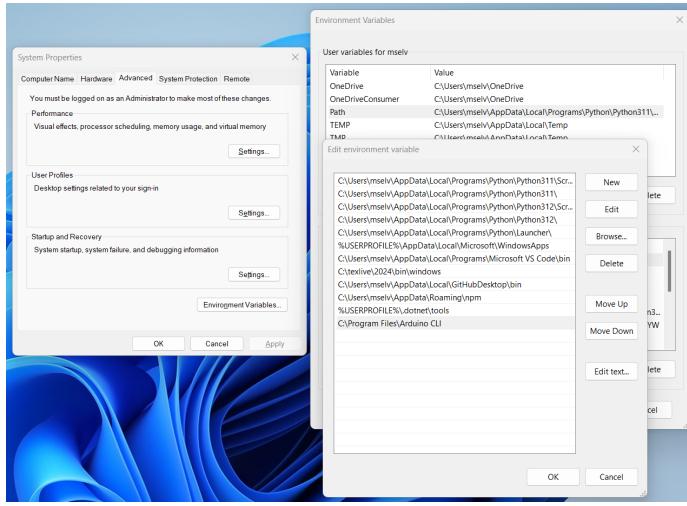


Figure 4.1.: Adding arduino-cli.exe file storage location to PATH system variable

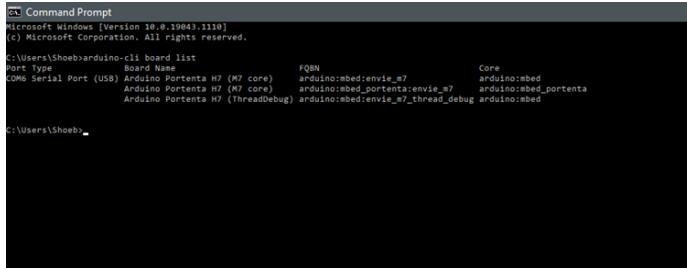


Figure 4.2.: Installation of Arduino CLI

After successfully downloading the file [arduino-cli_1.0.4_Windows_-64bit.zip](#), the folder contents can be extracted to a self-defined storage path [C:/Program Files/Arduino CLI](#). The packed file contains the license terms and the executable file [arduino-cli.exe](#).

In order to use the function of the command line interpreter in different paths, the path [C:/Program Files/Arduino CLI](#) to the storage location of the file [arduino-cli.exe](#) should be added to the system variable [PATH](#) as shown in the below Figure 4.1. The interface can then be used by entering [arduino-cli](#) in the command line.

After the installation is complete, [arduino-cli board list](#) can be entered in the command prompt. The connected device is displayed as Arduino Portenta H7 with port number and type, as shown in Figure 4.2.

4.1.1. Arduino CLI License

Arduino CLI is licensed under the GPL 3.0 license. The software is open-source and licensed under the GNU General Public License (GPL) version 3, which means it is free to use, modify, and distribute it as long as you comply with the GPL terms. While there is typically a license file included in the download to explain the legal terms, but its presence is not required for the software to function. Buying a commercial license is mandatory if you want to modify or otherwise use the software for commercial activities involving the Arduino software without disclosing the source code of your own applications. [Ard24b]

4.2. Configuration of the Arduino CLI

So that CLI can find the Arduino installation, it helps to create a configuration file for CLI. This configuration file is defined in a format **YAML**. To create a basic configuration file, CLI can be used in which a command line **arduino-cli config init** is entered. This command creates a new file **arduino-cli.yaml** in the following path **C:/Users/mselv/AppData/Local/Arduino15/arduino-cli.yaml**. All necessary parameters are declared there. The default settings of the configuration file are shown in the listing 4.1.

The following parameters usually need to be changed:

- **sketchbook_path**: Directory of the Arduino sketchbook. All libraries and hardware definitions are installed here.
- **Arduino_data**: Installation location of the Arduino board and the library manager. In most cases, this should not need to be changed.

The other options can generally remain at their default values.

4.2.1. Configuration keys

Configuration keys refer to the parameters or settings stored in the configuration file **arduino-cli.yaml**. These keys determine various aspects of how the Arduino CLI operates. [Ard24c]

- **board_manager**
 - **additional_urls** - the URLs to any additional Boards Manager package index files needed for your boards platforms.
- **daemon** - options related to running Arduino CLI as a gRPC server.
 - **port** - TCP port used for gRPC client connections.

```
1 board_manager:
2 additional_urls: []
3 build_cache:
4 compilations_before_purge: 10
5 ttl: 720h0m0s
6 daemon:
7 port: "50051"
8 directories:
9 data: C:\Users\mselv\AppData\Local\Arduino15
10 downloads: C:\Users\mselv\AppData\Local\Arduino15\
11     staging
12 user: C:\Users\mselv\OneDrive\Documents\Arduino
13 library:
14 enable_unsafe_install:false
15 logging:
16 file: ""
17 format: text
18 level: info
19 metrics:
20 addr: :9090
21 enabled: true
22 output:
23 no_color:false
24 sketch:
25 always_export_binaries:false
26 updater:
enable_notification: true
```

Listing 4.1: Default settings of the configuration file

- **directories** - directories used by Arduino CLI.
 - **data** - directory used to store Boards/Library Manager index files and Boards Manager platform installations.
 - **downloads** - directory used to stage downloaded archives during Boards/Library Manager installations.
 - **user** - the equivalent of the Arduino IDE's "sketchbook" directory. Library Manager installations are made to the libraries subdirectory of the user directory.
 - **builtin.libraries** - the libraries in this directory will be available to all platforms without the need for the user to install them, but with the lowest priority over other installed libraries with the same name, it's the equivalent of the Arduino IDE's bundled libraries directory.
 - **builtin.tools** - it's a list of directories of tools that will be available to all platforms without the need for the user to install them, it's the equivalent of the Arduino IDE 1.x bundled tools directory.
- **library** - configuration options relating to Arduino libraries.
 - **enable_unsafe_install** - set to true to enable the use of the --git-url and --zip-file flags with arduino-cli lib install. These are considered "unsafe" installation methods because they allow installing files that have not passed through the Library Manager submission process.
- **locale** - the language used by Arduino CLI to communicate to the user, the parameter is the language identifier in the standard POSIX format <language>[_<TERRITORY>[.<encoding>]] (for example it or it_IT, or it_IT.UTF-8).
- **logging** - configuration options for Arduino CLI's logs.
 - **file** - path to the file where logs will be written.
 - **format** - output format for the logs. Allowed values are text or json.
 - **level** - messages with this level and above will be logged. Valid levels are: trace, debug, info, warn, error, fatal, panic.
- **metrics** - settings related to the collection of data used for continued improvement of Arduino CLI.
 - **addr** - TCP port used for metrics communication.
 - **enabled** - controls the use of metrics.

- **output** - settings related to text output.
 - **no_color** - ANSI color escape codes are added by default to the output. Set to true to disable colored text output.
- **sketch** - configuration options relating to Arduino sketches.
 - **always_export_binaries** - set to true to make arduino-cli compile always save binaries to the sketch folder. This is the equivalent of using the –export-binaries flag.
- **updater** - configuration options related to Arduino CLI updates
 - **enable_notification** - set to false to disable notifications of new Arduino CLI releases, defaults to true
- **build_cache** - configuration options related to the compilation cache
 - **compilations_before_purge** - interval, in number of compilations, at which the cache is purged, defaults to 10. When 0 the cache is never purged.
 - **ttl** - cache expiration time of build folders. If the cache is hit by a compilation the corresponding build files lifetime is renewed. The value format must be a valid input for time.ParseDuration(), defaults to 720h (30 days).
- **network** - configuration options related to the network connection.
 - **proxy** - URL of the proxy server.

4.2.2. Configuration methods

Arduino CLI may be configured in three ways:

1. Command line flags
2. Environment variables
3. Configuration file

If a configuration option is configured by multiple methods, the value set by the method highest on the above list overwrites the ones below it. If a configuration option is not set, Arduino CLI uses a default value. **arduino-cli config dump** displays the current configuration values. Detailed documentation is provided [Ard24c]

Command line flags

Configuration file of Arduino CLI can be modified using Command line flags. [Ard24c]

- Example: Setting an additional Boards Manager URL using the `-additional-urls` command line flag:

```
arduino-cli core update-index -additional-urls  
https://downloads.arduino.cc/packages/package_staging_-  
index.json
```

Environment variables

All configuration options can be set via environment variables. The variable names start with `ARDUINO`, followed by the configuration key names, with each component separated by `_`. For example, the `ARDUINO_DIRECTORIES_USER` environment variable sets the `directories.user` configuration option.

On Linux or macOS, you can use the `export` command to set environment variables. On Windows command, you can use the `set` command. [Ard24c]

- Example: Setting an additional Boards Manager URL using the `ARDUINO_BOARD_MANAGER_ADDITIONAL_URLS` environment variable:

```
set ARDUINO_BOARD_MANAGER_ADDITIONAL_URLS=  
https://downloads.arduino.cc/packages/package_staging_-  
index.json
```

- The export or set command should be run in terminal session. It sets the environment variable for the duration of that terminal session.
- If the terminal is closed, the environment variable will be unset. To make the variable persistent across terminal sessions, we should add the export or set command to shell's configuration file (e.g., `.bashrc`, `.zshrc`, or `.profile`).

Configuration file

`arduino-cli config init` creates or updates a configuration file with the current configuration settings. This allows saving the options set by command line flags or environment variables. [Ard24c]

- Example: `arduino-cli config init -additional-urls https://downloads.arduino.cc/packages/package_staging_-index.json`

The configuration file must be named `arduino-cli`, with the appropriate file extension for the file's format.

Refer the listing 4.2, for the updation of the configuration file using **arduino-cli config init**

4.3. Overview of functions

The file [README.md](#) in the GitHub repository "Arduino CLI" contains an excellent overview of the functions and possibilities [Ard24b] The following functions are available [Ard24a]:

- **arduino-cli**
- **board**
 - **board attach**
 - **board details**
 - **board list**
 - **board listall**
 - **board search**
- **burn-bootloader**
- **cache**
- **cache clean**
- **compile**
- **completion**
- **config**
 - **config dump**
 - **config init**
 - **config add**
 - **config delete**
 - **config remove**
 - **config set**
- **core**
 - **core download**
 - **core install**
 - **core list**
 - **core search**

```
1  board_manager:
2    additional_urls:
3      : https://downloads.arduino.cc/packages/
4        package_staging_index.json
5  build_cache:
6  compilations_before_purge: 10
7  ttl: 720h0m0s
8  daemon:
9    port: "50051"
10   directories:
11     data: C:\Users\mselv\AppData\Local\Arduino15
12     downloads: C:\Users\mselv\AppData\Local\Arduino15\
13       staging
14     user: C:\Users\mselv\OneDrive\Documents\Arduino
15   library:
16     enable_unsafe_install:false
17   logging:
18     file: ""
19     format: text
20     level: info
21   metrics:
22     addr: :9090
23     enabled: true
24   output:
25     no_color:false
26   sketch:
27     always_export_binaries:false
28   updater:
29     enable_notification: true
```

Listing 4.2: Updated settings of the configuration file

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

4.3.1. Basic functions

The functions required to load a finished sketch onto an Arduino are defined as basic functions. Communication with a connected Arduino is implemented later using the serial monitor.

The first basic function is the board list command. This can be used to display all Arduino boards connected to the PC with additional information - such as the port used or the core of the board. This information

is necessary for using the correct board. The core list and core install commands are also required. The already installed cores can be listed with the core list command. If the core specified under board list is not installed, it can be added with the core install command. The third basic function is the lib command with the lib list and lib install extensions. The lib list command can be used to check which libraries are already installed on the system. If an additional library is required for a sketch, it can be installed with the lib install command.

The preparations for compiling and uploading a sketch can be made with these basic functions. By executing the compile command, a sketch is compiled into code that can be read by a selected board. A compiled sketch is then uploaded to a board using the upload command. The port of the board known from the board list command is specified as the destination for the upload. The basic functions are summarized in Table 4.1

Table 4.1.: List of Basic Functions

No.	BasicFunction	Description
1	<code>board list</code>	List the connected Arduinos with additional information
2	<code>core list/install</code>	List and install cores
3	<code>lib list/install</code>	Listing and installing libraries
4	<code>compile</code>	Compiling a sketch for a specific board
5	<code>upload</code>	Uploading a sketch to an Arduino

4.3.2. Command reference

Arduino Command Line Interface includes the following syntax format.
[Ard24a] `arduino-cli <command> [flags]`

- **<command>** - represents the CLI commands to be executed.
- **flags** - represents optional command-line arguments that can modify the behavior of the command. These flags provide additional options or configurations that can be passed to the command to customize its execution.

The Command Line Interface (CLI) functions are explained in details along with the command format below.

- **board**

The command `arduino-cli board` is used to manage Arduino boards. This command allows you to perform various operations

related to boards, such as attaching a sketch, listing available boards, searching for boards, and more.

`arduino-cli board <subcommand> [flags]`

- **board attach**

The command `arduino-cli board attach` is used to attach a sketch to a board. This command allows you to specify the port and the Fully Qualified Board Name (FQBN) of the board you want to attach the sketch to. SketchPath is the optional path to the directory containing the sketch you want to attach to the board.

`arduino-cli board attach [-p <port>] [-b <FQBN>] [sketch-Path] [flags]`

- **board details**

The command `arduino-cli board details` is used to display detailed information about a specific board. This command requires you to specify the Fully Qualified Board Name (FQBN) of the board you want to get details for.

`arduino-cli board details -b <FQBN> [flags]`

- **board list**

The command `arduino-cli board list` detects and displays a list of boards connected to the current computer. This command returns the Fully Qualified Board Name (FQBN) of each board along with its vendor and architecture.

`arduino-cli board list [flags]`

- **board listall**

The command `arduino-cli board listall` is used to list all available boards that have the support platform installed. You can search for a specific board if you specify the board name.

`arduino-cli board listall [boardname] [flags]`

- **board search**

The command `arduino-cli board search` is used to search a board in the Boards Manager using the specified keywords.. This command returns a list of boards that match the specified criteria.

`arduino-cli board search [boardname] [flags]`

- **burn-bootloader**

The command `arduino-cli burn-bootloader` is used to burn the bootloader onto an Arduino board using an external programmer. This command requires you to specify the port and the Fully Qualified Board Name (FQBN) of the target board.

```
arduino-cli burn-bootloader -p <port> -b <FQBN> [flags]
```

- **cache**

The command `arduino-cli cache` is used to manage the cache used by Arduino CLI. This command allows you to clean the cache or list the contents of the cache directory.

```
arduino-cli cache <subcommand> [flags]
```

- **cache clean**

The command `arduino-cli cache clean` is used to clean the cache used by Arduino CLI. This command deletes contents of the `directories.downloads` folder, where archive files are staged during installation of libraries and boards platforms.

```
arduino-cli cache clean [flags]
```

- **compile**

The command `arduino-cli compile` is used to compile Arduino sketches. This command requires you to specify the path to the sketch directory and optionally the board and output directory.

```
arduino-cli compile -b <FQBN> <sketchPath> [flags]
```

- **completion**

The command `arduino-cli completion` is used to generate completion scripts for the specified shell (bash, zsh, fish, powershell).

```
arduino-cli completion [bash|zsh|fish|powershell] [-no-descriptions] [flags]
```

- **config**

The command `arduino-cli config` is used to manage Arduino CLI configuration. This command allows you to interact with the configuration file, including initializing a new configuration, adding, deleting, setting, and listing configuration values.

```
arduino-cli config <subcommand> [flags]
```

- **config dump**

The command `arduino-cli config dump` is used to print the current configuration to the console.

```
arduino-cli config dump [flags]
```

- **config init**

The command `arduino-cli config init` creates or updates the configuration file in the data directory or custom directory with the current configuration settings.

`arduino-cli config init [flags]`

- **config add**

The command `arduino-cli config add` is used to add a new configuration value.

`arduino-cli config add [flags]`

- **config delete**

The command `arduino-cli config delete` is used to delete a settings key and all its sub keys in configuration value.

`arduino-cli config delete [flags]`

- **config remove**

The command `arduino-cli config remove` is an alias for config delete, removes one or more values from a setting

`arduino-cli config remove [flags]`

- **config set**

The command `arduino-cli config set` is used to set a setting value in configuration value.

`arduino-cli config set [flags]`

- **core download**

The command `arduino-cli core download` is used to download Arduino cores. This command requires you to specify the Fully Qualified Core Name (FQCN) of the core you want to download.

`arduino-cli core download <FQCN> [flags]`

- **core install**

The command `arduino-cli core install` is used to install Arduino cores.

`arduino-cli core install <FQCN> [flags]`

- **core list**

The command `arduino-cli core list` is used to list installed Arduino cores.

`arduino-cli core list [flags]`

- **core search**

The command `arduino-cli core search` is used to search for Arduino cores by specified keywords.

```
arduino-cli core search <keywords> [flags]
```

- **core uninstall**

The command `arduino-cli core uninstall` is used to uninstall Arduino cores. This command requires you to specify the Fully Qualified Core Name (FQCN) of the core you want to uninstall.

```
arduino-cli core uninstall <FQCN> [flags]
```

- **core update-index**

The command `arduino-cli core update-index` is used to update the index of available Arduino cores.

```
arduino-cli core update-index [flags]
```

- **core upgrade**

The command `arduino-cli core upgrade` is used to upgrade installed Arduino cores.

```
arduino-cli core upgrade [flags]
```

- **daemon**

The command `arduino-cli daemon` is used to start the Arduino CLI daemon, allowing it to listen for commands from other processes.

```
arduino-cli daemon [flags]
```

- **debug**

The command `arduino-cli debug` is used to debug Arduino sketches. This command requires you to specify the port, the Fully Qualified Board Name (FQBN), and the path to the sketch directory.

```
arduino-cli debug -p <port> -b <FQBN> <sketchPath> [flags]
```

- **lib**

The command `arduino-cli lib` is used to manage Arduino libraries. This command allows you to perform various operations related to libraries, such as installing, uninstalling, searching, listing, and updating libraries.

```
arduino-cli lib <subcommand> [flags]
```

- **lib deps**

The command `arduino-cli lib deps` is used to check dependencies status for the specified library.

```
arduino-cli lib deps <libraryName> [flags]
```

- **lib download**

The command `arduino-cli lib download` is used to download a library without installing them.

```
arduino-cli lib download <libraryName> [flags]
```

- **lib examples**

The command `arduino-cli lib examples` is used to list the examples available in a library. A name may be given as argument to search a specific library.

```
arduino-cli lib examples <libraryName> [flags]
```

- **lib install**

The command `arduino-cli lib install` is used to install one or more specified libraries into the system.

```
arduino-cli lib install <libraryName> [flags]
```

- **lib list**

The command `arduino-cli lib list` is used to list installed libraries. If the LIBNAME parameter is specified the listing is limited to that specific library. By default the libraries provided as built-in by platforms/core are not listed, they can be listed by adding the -all flag.

```
arduino-cli lib list <libraryName> [flags]
```

- **lib search**

The command `arduino-cli lib search` is used to search for libraries by name.

```
arduino-cli lib search <libraryName> [flags]
```

- **lib uninstall**

The command `arduino-cli lib uninstall` is used to uninstall a library.

```
arduino-cli lib uninstall <libraryName> [flags]
```

- **lib update-index**

The command `arduino-cli lib update-index` is used to update the libraries index to the latest version.

```
arduino-cli lib update-index [flags]
```

- **lib upgrade**

The command `arduino-cli lib upgrade` upgrades an installed library to the latest available version. Multiple libraries can be passed separated by a space. If no arguments are provided, the command will upgrade all the installed libraries where an update is available.

```
arduino-cli lib upgrade [flags]
```

- **monitor**

The command `arduino-cli monitor` is used to monitor the serial output of an Arduino board. This command requires you to specify the port and optionally the baud rate.

```
arduino-cli monitor -p <port> [flags]
```

- **outdated**

The command `arduino-cli outdated` shows a list of installed cores and/or libraries that can be upgraded. If nothing needs to be updated the output is empty.

```
arduino-cli outdated [flags]
```

- **sketch**

The command `arduino-cli sketch` is used to manage Arduino sketches. This command allows you to create, archive, and list sketches.

```
arduino-cli sketch <subcommand> [flags]
```

- **sketch archive**

The command `arduino-cli sketch archive` creates a zip file containing all sketch files. This command requires you to specify the path to the sketch directory.

```
arduino-cli sketch archive <sketchPath> <archivePath> [flags]
```

- **sketch new**

The command `arduino-cli sketch new` is used to create a new Arduino sketch. This command requires you to specify the name of the sketch.

`arduino-cli sketch new <sketchName> [flags]`

- `update`

The command `arduino-cli update` is used to update the index of cores and libraries to the latest versions.

`arduino-cli update [flags]`

- `upgrade`

The command `arduino-cli upgrade` is used to upgrade the index of cores and libraries to the latest versions.

`arduino-cli upgrade [flags]`

- `upload`

The command `arduino-cli upload` is used to upload Arduino sketches. This does NOT compile the sketch prior to upload. This command requires you to specify the port, the Fully Qualified Board Name (FQBN), and the path to the sketch directory.

`arduino-cli upload -p <port> -b <FQBN> <sketchPath> [flags]`

- `version`

The command `arduino-cli version` is used to display the current version of Arduino CLI installed on your system.

`arduino-cli version [flags]`

NOTE: The flags used in Arduino CLI commands can vary based on both the version of Arduino CLI being used and the specific command being executed.

4.4. First steps with Arduino Portenta H7 using the CLI

4.4.1. Recognizing the connected boards

CLI can be used to query which boards are installed using `board listall`. If the connected board is missing, it must be installed using `core install`. CLI can be used to query which boards are connected using `board list`. After the installation of the Arduino CLI and connecting the board to the PC, the command `arduino-cli board list` can be used to check whether the board is recognized by the computer. The function `board list` has several return values. The port via which the board is connected to the PC, the port, the type, the board name and the FQBN and core are returned, as shown in Figure 4.3.

Port	Type	Board Name	FQBN	Core
/dev/ttyACM0	Serial Port	Arduino Portenta H7 (M7)	arduino:mbed_portenta	arduino:mbed

Figure 4.3.: Return values of the “board list” function

```
C:\Users\mselv>arduino-cli core update-index
Downloading index: package_index.tar.bz2 downloaded
```

Figure 4.4.: Updating the board index

4.4.2. Configuration of Arduino CLI

To begin with, the correct core for the Arduino Portenta H7 should be installed. Before that, add the staging index URL to the Arduino CLI configuration file `arduino-cli.yaml` with the following command `arduino-cli config add board_manager.additional_urls https://downloads.arduino.cc/packages/package_staging_index.json`. Then update the board index to fetch the latest package information from the staging index using the command `arduino-cli core update-index` as shown in the Figure 4.4.

Once the index is updated, install the Portenta H7 boards core that are available in the staging index using the command `arduino-cli core install arduino:mbed_portenta` as shown in the Figure 4.5 and the correct core name "arduino:mbed" can be fetched from the Figure 4.3. Now run the command `arduino-cli board listall` to view all boards supported by the installed cores as shown in the Figure 4.6

```
C:\Users\mselv>arduino-cli core install arduino:mbed_portenta
Downloading packages...
arduino:adb@32.0.0 arduino:adb@32.0.0 already downloaded
arduino:arm-none-eabi-gcc@7-2017q4 arduino:arm-none-eabi-gcc@7-2017q4 already downloaded
arduino:dfu-util@0.10.0-arduino1 arduino:dfu-util@0.10.0-arduino1 already downloaded
arduino:imgtool@1.8.0-arduino2 arduino:imgtool@1.8.0-arduino2 already downloaded
arduino:openocd@0.11.0-arduino2 arduino:openocd@0.11.0-arduino2 already downloaded
arduino:mbed_portenta@4.1.5 arduino:mbed_portenta@4.1.5 already downloaded
Installing arduino:adb@32.0.0...
Configuring tool...
arduino:adb@32.0.0 installed
Installing arduino:arm-none-eabi-gcc@7-2017q4...
Configuring tool...
arduino:arm-none-eabi-gcc@7-2017q4 installed
Installing arduino:dfu-util@0.10.0-arduino1...
Configuring tool...
arduino:dfu-util@0.10.0-arduino1 installed
Installing arduino:imgtool@1.8.0-arduino2...
Configuring tool...
arduino:imgtool@1.8.0-arduino2 installed
Installing arduino:openocd@0.11.0-arduino2...
Configuring tool...
arduino:openocd@0.11.0-arduino2 installed
Installing platform arduino:mbed_portenta@4.1.5...
Configuring platform...
Platform arduino:mbed_portenta@4.1.5 installed
```

Figure 4.5.: Installing the core for Arduino Portenta H7

```
C:\Users\mselv>arduino-cli board listall
Board Name          FQBN
Arduino Portenta H7 arduino:mbed_portenta:envie_m7
Arduino Portenta X8 arduino:mbed_portenta:portenta_x8
```

Figure 4.6.: List available boards

```
C:\Users\mselv>arduino-cli sketch new cli_test
Sketch created in: C:\Users\mselv\cli_test
```

Figure 4.7.: Creating a new Sketch with CLI

4.4.3. Creating a new sketch

The first step is to create a new sketch:

arduino-cli sketch new cli_test

This command creates a directory named `cli_test`, which contains a file with the same name as shown in the Figure 4.7

4.4.4. Compiling a sketch

The function `compile` of CLI can be used to compile a sketch for any supported board. The crucial option required by this function is the board type, which can be specified with the option `-fqbn`. The abbreviation `fqbn` means "fully-qualified board name". For example, the following boards are available:

- Arduino Uno: `arduino:avr:uno`
- Arduino Mega: `arduino:avr:mega`
- Arduino Micro: `arduino:avr:micro`
- Arduino Mini: `arduino:avr:mini`
- Arduino Portenta H7: `arduino:mbed_portenta`
- Arduino Portenta X8 : `arduino:mbed_portenta:portenta_x8`
- ESP32 Dev Module: `esp32:esp32`

The possible boards are structured as follows:

`manufacturer:architecture:board`.

With the following command, the example sketch, which is located in the folder `C:/Users/user.name` can be compiled for an Arduino Portenta H7: `arduino-cli compile -fqbn arduino:mbed_portenta:envie_m7 C:/Users/mselv/cli_test`

Figure 4.8 shows the CLI messages. The following flags can be added:

```
C:\Users\mse1v>arduino-cli compile --fqbn arduino:mbed_portenta:envie_m7 C:\Users\mse1v\cli_test
Used platform      Version Path
arduino:mbed_portenta 4.1.5  C:\Users\mse1v\AppData\Local\Arduino15\packages\arduino\hardware\mbed_portenta\4.1.5
```

Figure 4.8.: Compiling a sketch with CLI

- Verbose **-v**: Useful if you want to display all options and files that are compiled in your sketch.
- Build path **-build-path [string]**: Useful if the compiled object and hex files are to be saved in a custom path. On my Windows system, the value of this parameter must be a complete path.

4.4.5. Uploading a sketch

A compiled sketch can be uploaded. Similar to the compile command, the upload command also requires the use of the option **-fqbn**. Additionally, a serial interface is necessary for uploading, which is specified with the option **-p**. The following command uploads the example sketch to a Windows COM port on COM18 as shown in Figure 4.9:

```
arduino-cli upload -p COM18 -fqbn arduino:mbed_portenta:envie_m7 C:/Users/mse1v/cli_test
```

If executed correctly, the RX/TX LEDs of the Arduino should start flashing, and shortly afterward, an empty sketch will be executed.

4.4.6. Installation of Libraries

If a library is required, you can first use the command

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

to check if the library (in this case, Ethernet) is installed. Then, with the command

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

you can install it.

4.4.7. Example program "Hello World"

The "Hello World!" counterpart for microcontrollers will be used to demonstrate how to compile a sketch and then upload it using the Arduino CLI. A simple sketch that makes the Arduino's LED flash is used for this.

In the function **setup()**, which is executed once at the beginning, the built-in LED is defined as an output **pinMode(LED_BUILTIN, OUTPUT)**. This allows the LED to be controlled later. Within the **loop()** function, the LED is first switched on **digitalWrite(LED_BUILTIN, HIGH)** and then switched off again with a delay of one second **delay(1000)**. Before

```

1 // This is a simple Arduino program to blink an LED
2 void setup() {
3     pinMode(LED_BUILTIN, OUTPUT); // Initialize the LED
4         pin as an output
5 }
6 void loop() {
7     digitalWrite(LED_BUILTIN, HIGH); // Turn the LED on
8     delay(1000); // Wait for a second
9     digitalWrite(LED_BUILTIN, LOW); // Turn the LED off
10    delay(1000); // Wait for a second
11 }
```

..//Code/Arduino/ArduinoCLI/BLINK.ino

```
C:\Users\mselv>arduino-cli upload -p COM18 --fqbn arduino:mbed_portenta:envie_m7 C:/Users/user_name/Documents/Arduino/cli_test
```

Figure 4.9.: Uploading the compiled sketch

the LED is switched on again at the beginning of the loop, another second is waited at the end of the loop to achieve the flashing effect.

To compile the sketch, the command previously defined as the basic function **compile**, as shown in Figure 4.8, can be used.

The compiled sketch can then be uploaded to the Arduino Portenta H7 using the function **upload** as shown in Figure 4.9

After the successful upload, the LED of the Arduino Portenta H7 shown in Figure 4.10 starts to flash.

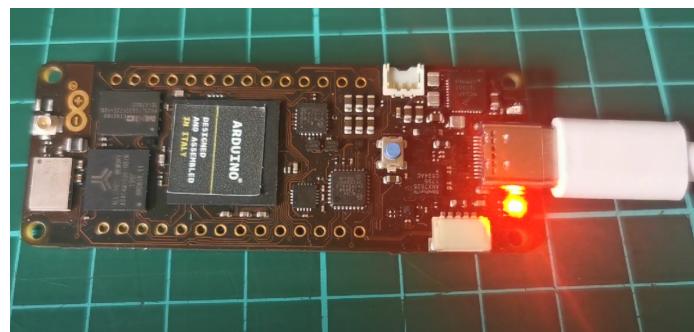


Figure 4.10.: The orange LED of the Arduino starts to flash

5. Batch File

5.1. Batch File

A batch file is a text file that contains a sequence of commands intended to be executed by the command-line interpreter. On Windows operating systems, these files typically have the extension `.bat` or `.cmd`. Batch files are used to automate repetitive tasks, perform complex sequences of commands, and streamline system administration processes. The results are documented in a log file. [Doc25g]

5.2. List Batch File

A "list batch file" refers to a batch file designed to list specific items, such as files in a directory, the contents of a directory, or any other listable resources. In the context of Arduino CLI, a list batch file might include commands to list available Arduino boards, libraries, or sketches.

```
1 @echo off
2 :: List available Arduino boards
3 arduino-cli board list
```

`@echo off` command is used at the beginning of the batch file to suppress the display of each command as it executes. By default, batch files echo each command to the console as they execute, but `@echo off` prevents this behavior, making the output cleaner. `:: List available Arduino boards` is a comment in the batch file. In batch scripting, comments start with `::` or `rem`, and they are ignored by the command interpreter. `arduino-cli board list` command executes the arduino-cli command-line tool with the board list subcommand. This subcommand lists all available Arduino boards connected to the system

5.3. Explanation of Batch commands

This section describes how the batch script works.

```
1 @echo off
2 echo log file Arduino setup, %time% clock, %date% > setup-log.
3 echo. >> setup-log.txt
4 echo. >> setup-log.txt
```

The command **echo off** prevents the script content from being output during execution. This enables more appealing front-end programming. The log file is created in the second line. When it is created, the first line of the log is given the heading **Logfile Arduino setup** and the system time **%time%** and date **%date%** are read out. For later clarity of the log file, two empty lines **echo. » setup-log.txt**. If the greater than operator **> setup-log.txt** is used once, a new file is created or an existing file with the same name is overwritten. If the greater than operator **» setup-log.txt** is used twice, the left-hand content is appended to an existing file in a new line. In this way, the log file is written line by line and not once completely at the end of the batch script.

```
1 :: If necessary, the core is installed
2 echo core status: >> setup-log.txt
3 arduino-cli core install arduino:mbed_portenta >> setup
    -log.txt
```

Comments in the batch script are identified by two consecutive colons **::**. **If necessary, the core is installed** and are intended to increase traceability within the script. The command **echo core status: » setup-log.txt** is used in the log file to indicate that the next line contains information about the core to be installed for the Arduino Portenta H7. The Arduino CLI already has the intelligence to check whether the specified core is already installed. If this is the case, the corresponding documentation is included in the log file. The installation of the core is initialized with the line **arduino-cli core install arduino:mbed_portenta » setup-log.txt**. The core required for the Arduino Portenta H7 is specified with the name **arduino:mbed_portenta**. The return value of the function for core installation is saved in the log file.

```
1 echo A search is made for connected boards...
2 :: List the connected Arduinos
3 echo The following boards are connected to the PC: >>
    setup-log.txt
4 echo.
```

```

5 echo. >> setup-log.txt
6 arduino-cli board list >> setup-log.txt
7 arduino-cli board list
8 echo.

```

At the beginning of this script section, the user of the software is first informed about the search for the connected Arduino boards **echo A search for connected boards is performed....**. The following command **echo The following boards are connected to the PC: » setup-log.txt** informs the user about the content of the next line in the log file. Information about which Arduinos are connected to the PC can be obtained with the command **arduino-cli board list**, which is executed twice here. In the first execution, the return of the Arduino-CLI is saved in the log file **» setup-log.txt**, the second execution is used for display in the currently executed script. The user needs the information about the connected Arduinos for an input in the next step.

```

1 :: Query the port name for later upload
2 :: of the sensor test file
3 set /p port="Please enter the port name of the Portenta
   H7 and confirm: "
4 echo The port %port% was selected. >> setup-log.txt

```

In the line **set /p port="Please enter the port name of the Portenta H7 and confirm:"** a user query is displayed. The command **set** allows the variable **port** to be set to a specific value. With the addition **/p**, this specific value is set to the following user input. Execution of the script is stopped until the user input is successful. The user input contains the port of the connected Arduino Nano 33 BLE Sense Lite. For later traceability, the selected port is entered in the log file with the line **echo The port %port% was selected. » setup-log.txt** is documented.

```

1 :: Create the folder for the compiled sensor test
2 set folder=SensorTestCompiledData
3 mkdir %folder%

```

The variable **folder** is assigned the value **SensorTestCompiledData**. In the next line **mkdir %folder%** the folder with the name **SensorTestCompiledData** is created for saving the compiled sketch later.

```

1 :: If necessary, the required libraries for
2 :: the sensor test are installed
3

```

```

4 :: Library for the onboard LPS22HB barometer
5 echo Lib for LPS22HB barometer >> setup-log.txt
6 arduino-cli lib install Arduino_LPS22HB >> setup-log.txt
7 echo. >> setup-log.txt
8
9 :: Library for the onboard HTS221 temperature and
   humidity sensor
10 echo Lib for HTS221 temperature and humidity sensor >>
    setup-log.txt
11 arduino-cli lib install Arduino_HTS221 >> setup-log.txt
12 echo. >> setup-log.txt

```

The log file documents which library is involved in the following line **SensorentestCompiledData**. Then use the command **arduino-cli lib install Arduino_LPS22HB » setup-log.txt** to install the library and save the return value of the installation in the log file. A library that has already been installed is recognized by the Arduino CLI and a corresponding return value is written to the log file. The procedure is identical for all three libraries to be installed.

```

1 :: Compiling the sensor test sketch
2 echo Compiling the sensor test sketch: >> setup-log.txt
3 echo. >> setup-log.txt
4 arduino-cli compile -b arduino:mbed_portenta:envie_m7
5 %cd%\SensortestPortentaH7
6 --build-path %cd%\%folder% >> setup-log.txt

```

Once the necessary libraries have been installed, the previously created sketch for testing the sensors can be compiled. The sketch for the Arduino Nano 33 BLE Sense Lite is compiled with the command **arduino-cli compile -b arduino:mbed_portenta:envie_m7 %cd%/SensortestPortentaH7**. The last part of the command specifies the memory path of the sketch to be compiled. Furthermore, the target folder for the compiled sketch can be defined with the addition **-build-path %cd%/%folder% » setup-log.txt**. The return value of compiling with the Arduino CLI is saved in the log file.

```

1 :: Upload the compiled sketch to the Arduino
2 echo Upload the compiled sketch to the Arduino >> setup-
   log.txt
3 arduino-cli upload -p %port% --input-dir %cd%\%folder%
4 >> setup-log.txt

```

The compiled sketch is stored in the line `arduino-cli upload -p %port% -input-dir %cd%/%folder% » setup-log.txt` to the Arduino Portenta H7 connected via `port`. The storage path of the compiled data is specified with the addition `-input-dir %cd%/%folder%`.

```
1 :: Opening the serial monitor
2 start monitor_log
3 :: Automatic closing of the serial monitor after 4
   seconds
4 :: (n-1 seconds, with n=5)
5 ping 127.0.0.1 -n 5 > nul
6 taskkill /im serial-monitor.exe /F
```

The command `start monitor_log` calls another batch script to read out the serial monitor. The line `ping 127.0.0.1 -n 5 > nul` stops the execution of the software for four seconds. Five requests are sent to the local computer and one second is waited between each request. With `> nul`, the output of the responses is directed to the void and not displayed. Once the four-second period has elapsed, the serial monitor for reading the sensor data is automatically closed with the command `taskkill /im serial-monitor.exe /F`. The window to be closed can be named `serial-monitor.exe` using the suffix `/im`. The forced closing of the serial monitor takes place with the appendix `/F`. After the serial monitor has been read out and closed, the software can be closed. The results of the sensor test can then be found in the log file.

The serial monitor is opened in the script `monitor_log.bat`.

```
1 echo Sensor data: >> setup-log.txt
2 echo. >> setup-log.txt
3
4 echo The serial monitor is opened, the sensor data is
5 read out and saved in the log file...
6
7 arduino-cli monitor -p %port% >> setup-log.txt
```

The serial monitor can be activated using the Arduino CLI command `arduino-cli monitor -p %port% » setup-log.txt` to open it. The connection for the specified port is established with `-p %port%`. With the addition `» setup-log.txt` the received data is written to the log file.

Part III.

Portenta H7

6. PortentaH7

6.1. Introduction

The Arduino Portenta H7 is a versatile computing platform with dual cores for efficient multitasking. One core gathers data from sensors, while the other processes it with machine learning. Its dual 80-pin connectors allow scalability and compatibility with additional boards like the Vision shield, making it ideal for complex applications. Below table shows the specifications for the PortentaH7 6.1. [Doc24c]

Table 6.1.: Specifications Table

Specification	Details
Main Processor	STM32H747XI-dual Cortex-M7+M4 32bit low power Arm MCU
SDRAM	8-64 MByte option
QSPI Flash	2-128 MByte option
Ethernet	10/100 Phy option
Wireless	BT5.0 + WiFi 802.11 b/g/n 65Mbps
Crypto chip	ECC608 or SE050C2 (Common Criteria EAL 6+) option
Display Connector	MIPI DSI host and MIPI D-PHY to interface with low-pin count large displays
GPU	Chrom-ART graphical hardware Accelerator
Timers	22x timers and watchdogs
UART	4x ports (2 with flow control)
SD Card	Interface for SD Card connector (through expansion port only)
Temperature	-40°C to +85°C
Power	Through USB-C connector or LiPo battery (integrated charger)
Current Consumption	2.95 micro-Ampere in Standby mode (Backup SRAM OFF, RTC/LSE ON)
USB-C	Host / Device, DisplayPort out, High / Full Speed, Power delivery
MKR Headers	Use any of the existing industrial MKR shields
High Density Connectors	Two 80 pin connectors will expose all of the board's peripherals to other devices
ESLOV	Arduino's open connector standard for self-identifiable hardware
Camera Interface	8-bit, up to 80 MHz
ADCs	3 × ADCs with 16-bit max. resolution (up to 36 channels, up to 3.6 MSPS)
DACs	2 × 12-bit DAC (1 MHz)

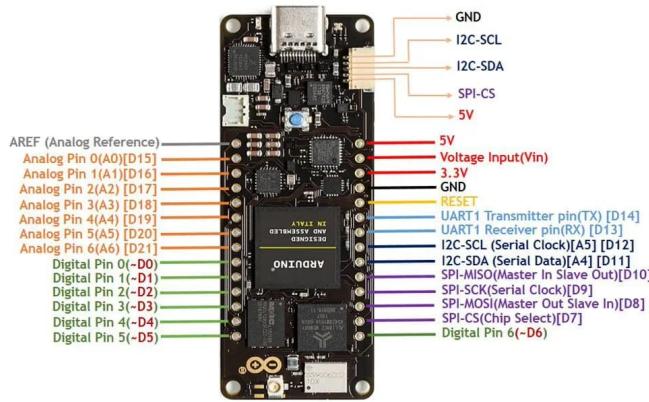


Figure 6.1.: PortentaH7 Hardware
[Ard24m]

6.2. Brief Description of Hardware Components in Arduino Portenta H7

Here's a comprehensive list of hardware parts commonly found in the Arduino Portenta H7 see Figure 6.1.

- Microcontroller Unit (MCU):** The MCU is the main processing unit of the Arduino Portenta H7. It consists of a powerful dual-core STM32H747 processor, with an Arm Cortex-M7 core running at up to 480 MHz and an Arm Cortex-M4 core running at up to 240 MHz. This dual-core architecture provides the device with significant processing power and flexibility for handling complex tasks and applications. [Doc24c]
- Memory:** The Arduino Portenta H7 supports various types of memory to accommodate different requirements for program storage and data manipulation. Here are the main varieties of memory available in the Portenta H7:
 - Flash Memory:** Ideal for storing program code and non-volatile data, ensuring data retention even during power loss.
 - RAM (Random Access Memory):** Offers speedy access for runtime data and variable storage during program execution.
 - External Storage:** Supports expansion via external storage devices like SD cards, facilitating additional space for large datasets and multimedia content.
 - Cache Memory:** Enhances performance by temporarily storing frequently accessed data and instructions close to the CPU

cores, reducing memory access latency.

3. **Connectivity Interfaces:** The Portenta H7 offers a variety of connectivity options, including Ethernet, Wi-Fi, Bluetooth, USB, and serial interfaces such as SPI, I2C, UART, and CAN bus. These interfaces enable the device to communicate with other devices, sensors, actuators, and networks, facilitating data exchange, control, and networking capabilities.
4. **I/O Pins:** The Arduino Portenta H7 provides a variety of I/O (Input/Output) pins, both digital and analog, offering flexibility for interfacing with external components and peripherals [Doc24c]. Here are the different types of I/O pins available in the Portenta H7:
 - **Digital I/O Pins:** These pins (D0 to D6 green color in above figure 6.1) can be configured as either input or output and are used for digital communication or controlling digital devices. They typically support binary states, either HIGH (5V) or LOW (0V).
 - **Analog Input Pins:** The Portenta H7 features analog input pins (A0 to A6 orange color in above figure 6.1) that can measure continuous voltage levels. These pins are often used to interface with analog sensors or read analog signals from external devices. They provide a range of values rather than discrete digital states.
 - **PWM (Pulse Width Modulation) Pins:** Some of the digital I/O pins on the Portenta H7 support PWM output. PWM pins can generate analog-like signals with varying duty cycles, allowing for control of devices such as motors or LED brightness.
 - **I2C (Inter-Integrated Circuit) Pins:** I2C pins (dark-blue color pins in 6.1) allow for serial communication with devices using the I2C protocol. This communication protocol enables multiple devices to communicate with each other using only two wires, making it suitable for connecting sensors, displays, and other peripherals.
 - **SPI (Serial Peripheral Interface) Pins:** SPI pins (purple color pins in 6.1) facilitate high-speed serial communication between the Portenta H7 and other devices using the SPI protocol. SPI is commonly used to interface with devices such as SD cards, displays, and sensors that require fast data transfer rates.

- **UART (Universal Asynchronous Receiver-Transmitter) Pins:** UART pins (blue color pins in 6.1) provide serial communication capabilities for transmitting and receiving data asynchronously between the Portenta H7 and other devices. UART is widely used for communication between microcontrollers, sensors, GPS modules, and other peripherals.
 - **CAN (Controller Area Network) Pins:** The Portenta H7 may include CAN pins for communication over the Controller Area Network protocol. CAN is commonly used in automotive and industrial applications for robust, high-speed communication between devices in a network.
5. **Analog-to-Digital Converter (ADC):** The ADC allows the Portenta H7 to convert analog signals from sensors or other external devices into digital values that can be processed by the system. This enables the device to interface with analog sensors and acquire analog data for processing and analysis. Here are some common types of ADCs that may be available in the Portenta H7 [Doc24c] :
- **Single-Channel ADCs:** These ADCs have a single input channel, making them suitable for applications that only require the conversion of a single analog signal.
 - **Multi-Channel ADCs:** These ADCs have multiple input channels, allowing them to simultaneously convert multiple analog signals. This is useful for applications that involve monitoring multiple sensors or acquiring data from multiple sources.
 - **Differential ADCs:** These ADCs measure the voltage difference between two input channels, providing higher accuracy and noise immunity compared to single-ended ADCs. They are suitable for applications that require precise measurements or operate in noisy environments.
 - **High-Resolution ADCs:** These ADCs offer higher resolution, typically with more bits of resolution, allowing them to capture finer details in the analog signal. This is beneficial for applications that require high accuracy or deal with low-level signals.
 - **High-Speed ADCs:** These ADCs can sample analog signals at high speeds, enabling them to capture rapidly changing signals or high-frequency signals. They are suitable for applications that require real-time data acquisition or signal processing.

- **Low-Power ADCs:** These ADCs consume minimal power during operation, making them suitable for battery-powered or energy-efficient applications. They optimize power consumption while maintaining adequate performance.
 - **Integrated ADCs with Signal Conditioning:** Some variants of the Portenta H7 may include integrated ADCs with built-in signal conditioning features such as programmable gain amplifiers (PGAs), filters, or input multiplexers. These features enhance the ADC's functionality and performance, simplifying the design of analog front-end circuits.
6. **Digital-to-Analog Converter (DAC):** The DAC enables the Portenta H7 to convert digital signals into analog voltages or currents, facilitating control over analog devices like motors, actuators, or audio amplifiers. While the Portenta H7 doesn't have built-in DACs, external DAC modules or shields can be added for this functionality. Examples include modules like MCP4922, ADS1115, DAC8831, and DAC7564, which offer varying resolutions and channels for interfacing with the Portenta H7 [Doc24c].
 7. **Real-Time Clock (RTC):** The RTC is a dedicated hardware component that keeps track of the current date and time, even when the device is powered off. This enables the Portenta H7 to perform timekeeping functions, schedule tasks, and timestamp data with accurate time information [Doc24c].
 8. **Secure Element/TrustZone:** Some variants of the Portenta H7 may include hardware-based security features such as a secure element or TrustZone technology. These security features provide hardware-based protection for sensitive data, cryptographic operations, and secure boot processes, enhancing the device's security against unauthorized access and attacks.
 9. **Power Management Unit (PMU):** The PMU is responsible for managing the power distribution and consumption within the device. It typically includes voltage regulators, power switches, and monitoring circuits to ensure efficient power usage, regulate voltage levels, and protect the system from power-related issues such as overvoltage, undervoltage, or overcurrent conditions [Doc24c].
 10. **Peripherals and Expansion Interfaces:** The Portenta H7 may feature various peripherals and expansion interfaces to support additional functionality and connectivity options. These peripherals may include SD card slots, audio codecs, cryptographic accelerators, motor control interfaces, and more. Additionally, the device may

offer expansion interfaces such as GPIO headers, expansion slots, or mezzanine connectors for attaching external modules, shields, or custom expansion boards to extend its capabilities.

7. First Step with the PortentaH7

7.1. Introduction

In this chapter we will be looking at how to connect the Arduino Portenta H7 with a PC/laptop in order to use the board. Then, we will be going through the first steps of installing the relevant packages to use the Arduino Portenta H7 board and then we will be implementing a basic example sketch from the Arduino IDE. [Doc24h]

7.2. Configuration

- Connect the Arduino Portenta H7 board to the computer via USB Type-C cable.
- Press the reset button twice on the board and the LED on the board starts blinking green as shown in the figure 7.1 indicating that it is ready.
- Next, open the Arduino IDE and navigate to the Boards Manager by selecting **Tools > Board > Boards Manager**. In the Boards Manager, search for "Portenta," select **Arduino Mbed OS Portenta Boards**, and click "Install". Fig 7.2

7.2.1. Example Blink Sketch

To upload the example sketch, go to **File > Examples > Basics > Blink** as shown in Figure 7.3.

In the program, we initialize **LED_BUILTIN** as the output in the setup function. In the loop function we turn on the LED by using **digitalWrite()** function. Click on upload and wait for it to complete. After the upload is done, the green LED on the board starts blinking with a delay of 1000ms. The blink sketch appears on the screen as shown in figure 7.4.



Figure 7.1.: Arduino Portenta H7 Connected to a laptop

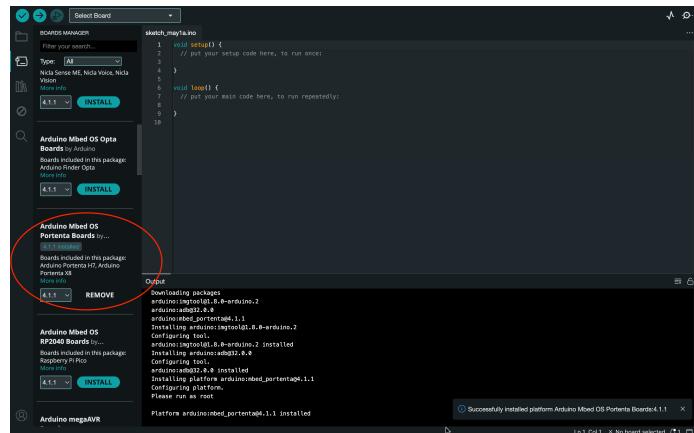


Figure 7.2.: Installation Arduino Mbed OS Portenta Board

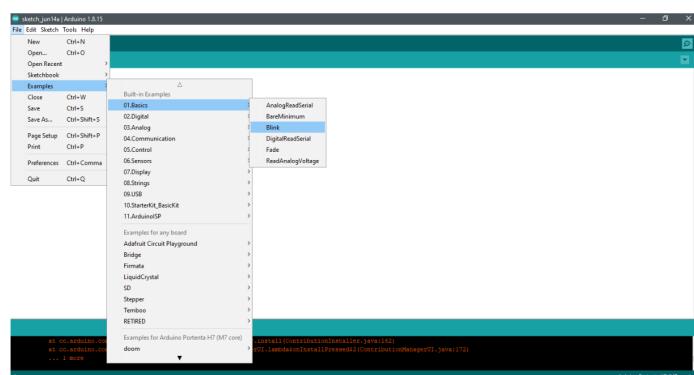


Figure 7.3.: Upload Basic Sketch

```
1 // This is a simple Arduino program to blink an LED
2 void setup() {
3     pinMode(LED_BUILTIN, OUTPUT); // Initialize the LED
4         pin as an output
5 }
6
7 void loop() {
8     digitalWrite(LED_BUILTIN, HIGH); // Turn the LED on
9     delay(1000); // Wait for a second
10    digitalWrite(LED_BUILTIN, LOW); // Turn the LED off
11    delay(1000); // Wait for a second
12 }
```

..//Code/Arduino/ArduinoCLI/BLINK.ino

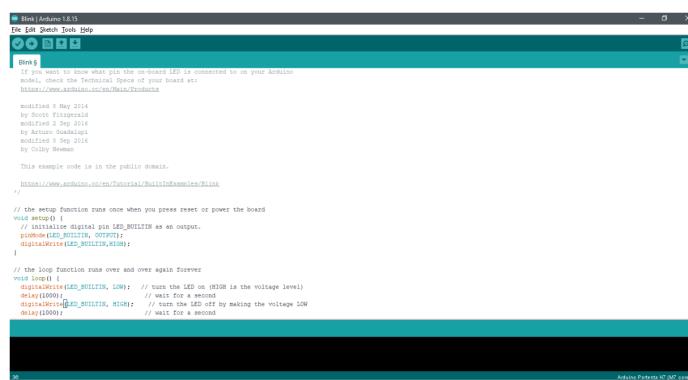


Figure 7.4.: Compile Blink Sketch

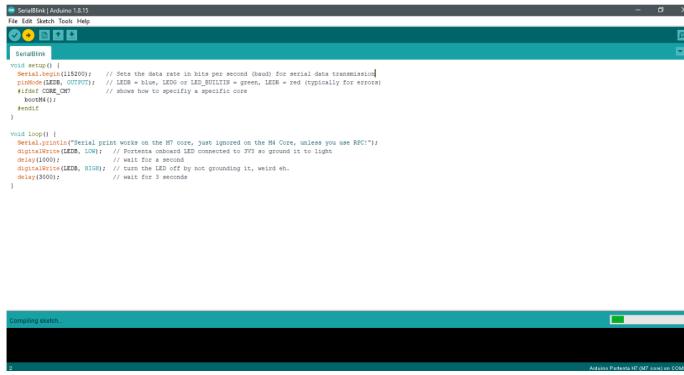


Figure 7.5.: Serial Blink Sketch

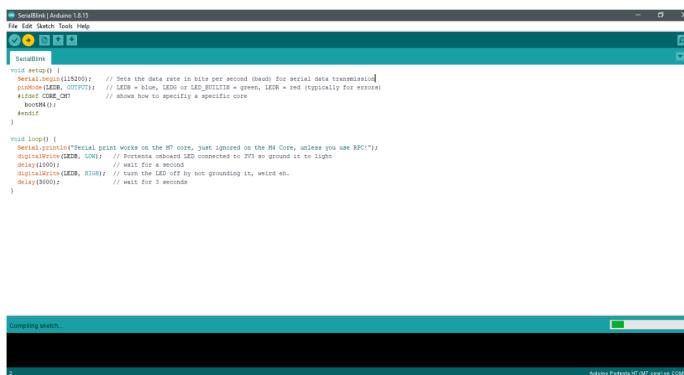


Figure 7.6.: Serial Blink Sketch

7.2.2. Serial Blink Example for Dual Core Processing

The following figure 7.6 represents the serial blink sketch. This sketch helps in demonstrating the serial data transmission by showing the LED blinking after a certain time period.

Here, **Serial.begin(115200)** command sets the data transmission rate to 115200 bits per second. The LEDB is the output which blinks in blue color and we have specified core M7. If we would like to use core M4 then we need to boot it first using **bootM4()** command. The ouput LED blinks blue in color after a delay of 3 seconds as defined in the sketch.

In order to use core M4, we need to de-select the core M7 from the board and select core M4. After selecting, we can see at the bottom the selected core for confirmation. Also, We need to write another sketch and now change the LED color to red in the sketch and add a delay of 4 seconds.

Notice that in the output both LED blue and green blinks and at a point

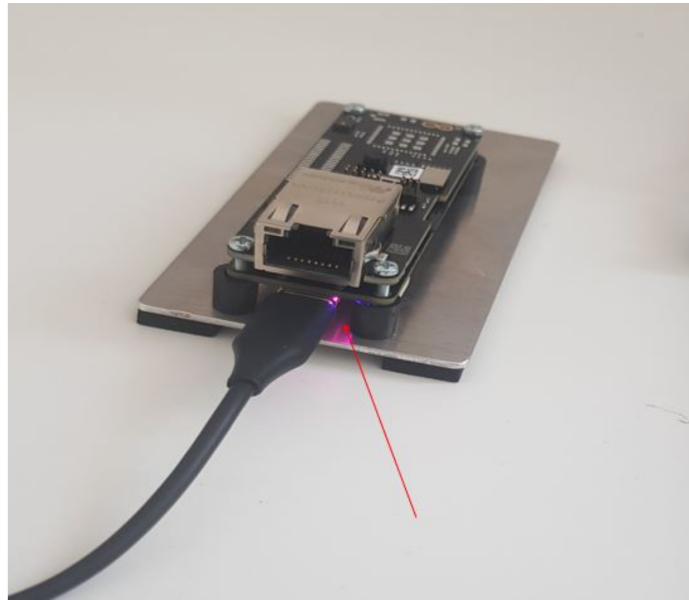


Figure 7.7.: Output LED Color

in time they almost blink together resulting in a Lila color. This shows the dual core processing concept by using an LED.

It can be seen from the figure 7.7 that the output LED blinks in Lila color due to the overlapping of the green and blue LED blinking at the same time.

8. Portenta Vision Shield

The Arduino Portenta Vision Shield is an add-on board providing machine vision capabilities and additional connectivity to the Portenta family of Arduino boards, designed to meet the needs of industrial automation. The Portenta Vision Shield connects via a high-density connector to the Portenta boards with minimal hardware and software setup, refer figure 8.1. [Ard24n]

8.1. Key Features

- **Onboard Camera:**

Himax HM-01B0 Camera Module

- Ultra-Low-Power Image Sensor designed for always-on vision devices and applications
- Window, vertical flip, and horizontal mirror readout
- Programmable black level calibration target, frame size, frame rate, exposure, analog gain (up to 8x), and digital gain (up to 4x)
- Automatic exposure and gain control loop with support for 50 Hz / 60 Hz flicker avoidance

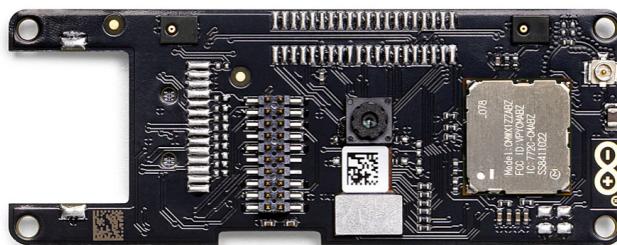


Figure 8.1.: Portenta Vision Shield

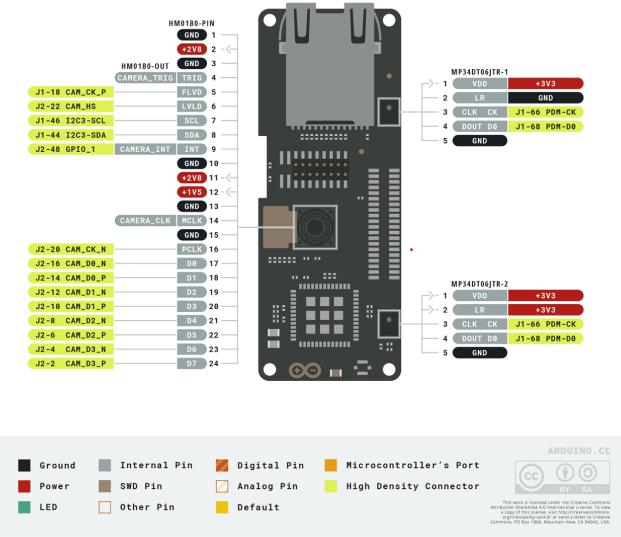


Figure 8.2.: Portenta Vision Shield PinDiagram
[Ardnd]

- Motion Detection circuit with programmable ROI and detection threshold with digital output to serve as an interrupt

Supported Resolutions:

- QQVGA (160x120) at 15, 30, 60, and 120 FPS
- QVGA (320x240) at 15, 30, and 60 FPS [Ardnd]

- **Onboard Microphone:** Dual ultra-compact, low-power, omnidirectional, digital MEMS microphones (MP34DT06J)
- **Connectivity:**
 - LoRa Module (CMWX1ZZABZ-078) for the LoRa variant (SKU: ASX00026)
 - RJ45 Connector for Ethernet capabilities in the Ethernet variant (SKU: ASX00021)
- **External Memory:** Onboard microSD card slot
- **Dimensions:** 66.04 x 25.40 mm
- **Weight:** 8 g [Ardnd]

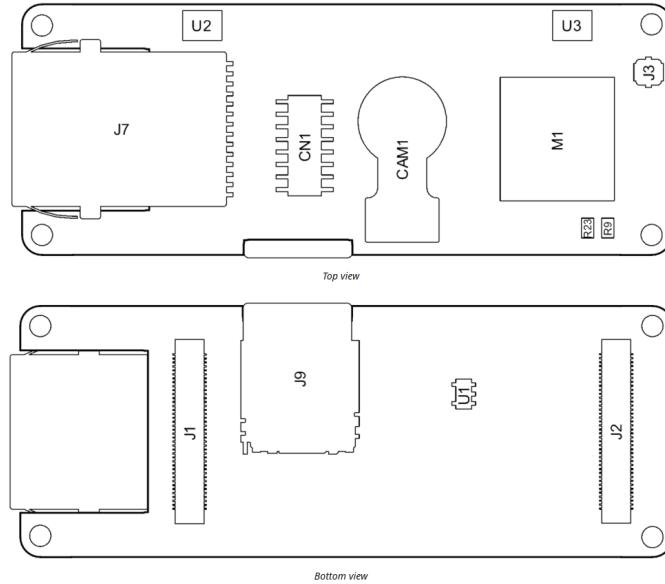


Figure 8.3.: Board

8.2. Hardware Components

The LoRa implementation in the Portenta Vision Shield comprises several essential hardware components(Refer figure 16.1) that work together to enable seamless communication, data handling, and energy management. Below are the key components: [Ard25b].

8.2.1. LoRa Transceiver Module

- **Function:** Provides radio communication using LoRa modulation for sending and receiving data [Cor25a].
- **Integrated Module:** Murata CMWX1ZZABZ module based on Semtech SX1276 transceiver [Man25].
- **Features:** Supports configurable frequency bands (868 MHz and 915 MHz), spreading factors, and adaptive data rates [Cor25a].

8.2.2. Microcontroller Unit (MCU)

- **Function:** Serves as the core processor for managing data, interfacing with sensors, and handling communication protocols .
- **MCU in Portenta H7:** Dual-core architecture with an ARM Cortex-M7 (480 MHz) and Cortex-M4 (240 MHz) .

- **Features:** High processing power, low energy consumption, and extensive I/O support for peripheral devices [Ard25b].

8.2.3. Antenna

- **Function:** Facilitates the transmission and reception of LoRa signals [Cor25b].
- **Antenna Type:** External antenna connected via a U.FL connector for enhanced signal quality [Man25].
- **Considerations:** Designed for optimal frequency compatibility and communication range [Cor25b].

8.2.4. Power Supply

- **Function:** Provides power to the Vision Shield and its components, ensuring energy efficiency .
- **Sources:** Compatible with the Portenta H7's onboard power supply and external batteries .
- **Components:** Includes voltage regulators to maintain stable operation and extend battery life [Ard25b].

8.2.5. Sensors and Peripherals

- **Function:** Collects environmental data and enhances functionality for specific applications .
- **Examples:** Integrated camera for vision applications, microphones for audio input. .
- **Interface:** Communicates with the MCU using I2C, SPI, or UART protocols [Ard25b].

8.2.6. Memory

- **Function:** Stores firmware, application data, and communication logs [Ard25b].
- **Memory in Portenta H7:** Flash memory for non-volatile storage and SRAM for temporary processing tasks [Ard25b].
- **Capacity:** 16 MB of NOR Flash and 8 MB of SDRAM, providing ample space for advanced applications [Ard25b].

««< Updated upstream

Ref.	Description	Ref.	Description
U1	Voltage Regulator	J3	LoRa® Radio Antenna U.FL Connector (ASX00026 Only)
U2,U3	ST MP34DT06jTR Digital Microphones	J7	Ethernet Connector (ASX00021 Only)
M1	Murata CMWX1ZZABZ LoRa® Module (ASX00026 Only)	J9	Micro SD Card Connector
J1,J2	High-Density Connectors	CN1	JTAG Connector
CAM1	Camera Module Himax HM-01B0		

Figure 8.4.: Board Topology

8.3. Application Examples

- **Industrial Automation:**
 - Quality control: Automatically detect product defects in production lines.
 - Predictive maintenance: Monitor equipment for early signs of wear or failure.
 - Automated sorting: Sort items based on color, shape, or size in conveyor systems. [Ardnd]
- **Surveillance:**
 - Real-time threat detection: Identify and alert authorities of potential threats.
 - Perimeter surveillance: Detect intrusions or breaches in perimeter security.
- **Machine Vision and Edge Computing:**
 - Smart agriculture: Monitor crops and soil conditions for issues like pest infestations or nutrient deficiencies.
 - Autonomous vehicles: Enhance navigation and obstacle detection.
 - Robotics: Enable robots to see and interpret their surroundings for complex tasks.
- **Audio Analysis and Sound-Based Applications:**
 - Acoustic monitoring: Detect anomalies or predict maintenance needs based on machinery noise patterns.
 - Voice-activated systems: Enable voice commands for hands-free control.
 - Anomaly detection: Detect unusual sounds like breaking glass or alarms in security systems. [Ardnd]

9. First Step with Portenta Vision Shield

The Arduino Portenta Vision Shield is an addon board providing machine vision capabilities and additional connectivity to the Portenta family of Arduino boards, designed to meet the needs of industrial automations. The Portenta Vision Shield connects via a high density connector to the Portenta H7 with minimal hardware and software setup.

9.1. Introduction

In this chapter we will be going through the steps required for connecting the Arduino Portenta H7 with the Vision Shield using the OpenMV IDE and processing sketch. Further, we will be looking at the overview of the OpenMV IDE and what are the tools provided in the IDE. Finally, we will be implementing some of the example programs provided in the OpenMV IDE. [Wh24]

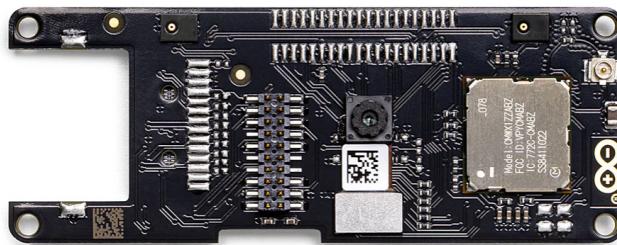


Figure 9.1.: VisionShield

9.2. Getting Started With the Portenta Vision Shield Camera

This example shows you how to capture frames from the Arduino Portenta Vision Shield Camera module and visualize the video output through a Processing sketch, refer figure 9.6 [Doc24e]

9.2.1. Required Hardware and Software:

- Portenta H7
- Portenta Vision Shield (LoRa or Ethernet)
- USB-C cable
- Arduino IDE 2.3.2
- Processing software [Doc24e]

9.2.2. Instructions:

Accessing the Portenta Vision Shield's camera data is done with the help of both Arduino and the Processing IDE. The Arduino sketch handles the capture of image data by the on-board camera, while the java applet created with Processing helps to visualize this data with the help of a serial connection. The following steps will run you through how to capture, package the data through the serial port and visualize the output in Processing. [Doc24e]

The Basic Setup:

Connect the Portenta Vision Shield to your Portenta H7 as shown in the figure 9.7. The top and bottom high density connectors are connected to the corresponding ones on the underside of the H7 board. Plug in the H7 to your computer using the USB-C cable. [Doc24e]

Open the board manager in the Arduino IDE and install the latest version of the Portenta Core which is v4.1.5 as shown in the figure 9.3.

Capturing the Frames:

Create a new Arduino sketch called `CameraCaptureRawBytes.ino` To capture the frames you will need to use the functions contained in `camera.h` which comes with the Portenta core. This library contains all APIs related to frame capturing, motion detection and pattern recognition. Include the header file in your sketch.

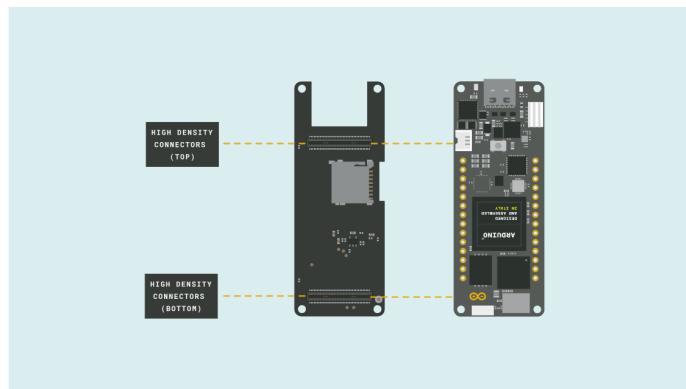


Figure 9.2.: Connection VS

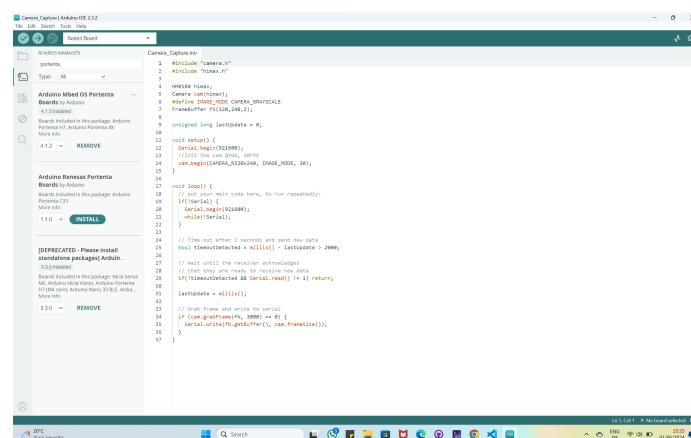


Figure 9.3.: Portentaport

```
1 #include "camera.h"
2 #include "himax.h"
```

Next, let's initialize a camera object and a frame buffer of the size **320*240 (76'800 bytes)**.

```
1 HM01B0 himax;
2 Camera cam(himax);
3 #define IMAGE_MODE CAMERA_GRAYSCALE
4 FrameBuffer fb(320,240,2);
5 unsigned long lastUpdate = 0;
```

In the **setup()** function, let's start the Serial communication at **921600 baud rate** and initialize the camera using **cam.begin()**.

```
1 void setup() {
2     Serial.begin(921600);
3     //Init the cam QVGA, 30FPS
4     cam.begin(CAMERA_R320x240, IMAGE_MODE, 30);
5 }
```

In the loop you need to capture each Frame and send it over a serial connection to the Processing sketch that will display the frames. You will use the function to fetch the frame from the frame buffer and save it into your custom data buffer.

```
1 void loop() {
2     // put your main code here, to run repeatedly:
3     if(!Serial) {
4         Serial.begin(921600);
5         while(!Serial);
6     }
7
8     // Time out after 2 seconds and send new data
9     bool timeoutDetected = millis() - lastUpdate > 2000;
10
11    // Wait until the receiver acknowledges
12    // that they are ready to receive new data
13    if(!timeoutDetected && Serial.read() != 1) return;
14
15    lastUpdate = millis();
16
17    // Grab frame and write to serial
18    if (cam.grabFrame(fb, 3000) == 0) {
```

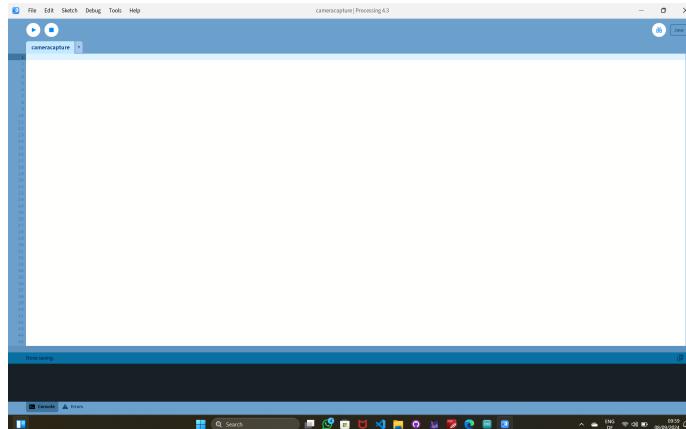


Figure 9.4.: Processingimage

```

19         Serial.write(fb.getBuffer(), cam.frameSize());
20     }
21 }
```

Create the Processing Sketch:

Open a new processing sketch file and name it CameraCapture.pde. 9.4 [Doc24e]

Let's start by importing the libraries and initializing the variables you will need to process. To process the data sent by the Portenta Vision Shield, you will need to import the following libraries:

1. **processing.serial.***: a Serial Library that is used to read and write data to external devices over the serial line.
2. **java.nio.ByteBuffer**: a java class that provides access to operations on byte buffers

```

1 import processing.serial.*;
2 import java.nio.ByteBuffer;
```

Next, you can initialize the following variables to process the received pixels from the serial port. You can set the dimensions, pixel count and bytes required per frame.

```

1 // must match resolution used in the sketch
2 final int cameraWidth = 320;
```

```

3   final int cameraHeight = 240;
4   final int cameraBytesPerPixel = 1;
5   final int cameraPixelCount = cameraWidth * cameraHeight;
6   final int bytesPerFrame = cameraWidth * cameraHeight *
      cameraBytesPerPixel;

```

To receive the frames, you will need a Serial port, a PImage object and an array to store the pixel values of the frame. Add the following variables to the code.

```

1   Serial myPort;
2   PImage myImage;
3   byte[] frameBuffer = new byte[bytesPerFrame];
4   int pixelPosition = 0;
5   int lastUpdate = 0;
6   boolean shouldRedraw = false;

```

Here, you will establish a connection to the serial port and prepare the buffer to store the frame pixels. Additionally, you can send a byte to the Arduino sketch from Processing to let it know that it is ready to receive data.

```

1 void setup() {
2   size(640, 480);
3
4   // if you know the serial port name
5   //myPort = new Serial(this, "COM5", 921600);
6   //           // Windows
7   //myPort = new Serial(this, "/dev/ttyACM0", 921600);
8   //           // Linux
9   myPort = new Serial(this, "/dev/cu.usbmodem14101",
10                      921600); // Mac
11
12   // Set the number of bytes to buffer
13   myPort.buffer(bytesPerFrame)
14
15   // Create an image based on the camera's dimensions and
16   // format
17   myImage = createImage(cameraWidth, cameraHeight, ALPHA);
18
19   // Let the Arduino sketch know we're ready to receive
20   // data
21   myPort.write(1);
22 }

```

The draw function checks if the connection is still alive and if there is any new data that can be drawn as an image. In that case, the original image gets copied into a new image object so that it can be scaled up.

```
1 void draw() {
2     // Time out after 1.5 seconds and ask for new data
3     if(millis() - lastUpdate > 1500) {
4         println("Connection timed out.");
5         myPort.clear();
6         myPort.write(1);
7     }
8
9     if(shouldRedraw){
10        PImage img = myImage.copy();
11        img.resize(640, 480);
12        image(img, 0, 0);
13        shouldRedraw = false;
14    }
15 }
```

Visualizing the Frames:

For this step, you will use the `serialEvent()` callback function to update the `myImage` when a new data is received on the serial port. [Doc24e]

```
1 void serialEvent(Serial myPort) {
2     lastUpdate = millis();
3
4     // read the received bytes
5     myPort.readBytes(frameBuffer);
6
7     // Access raw bytes via byte buffer
8     ByteBuffer bb = ByteBuffer.wrap(frameBuffer);
9
10    int i = 0;
11
12    while (bb.hasRemaining()) {
13        // read 8-bit pixel
14        byte pixelValue = bb.get();
15
16        // set pixel color
17        myImage.pixels[i++] = color(Byte.toUnsignedInt(
18            pixelValue));
19    }
20
21    myImage.updatePixels();
```

```

21     // Ensures that the new image data is drawn in the next
22     // draw loop
23     shouldRedraw = true;
24
25     // Let the Arduino sketch know we received all pixels
26     // and are ready for the next frame
27     myPort.write(1);
28 }
```

The first thing you can do inside this method is to update the timestamp when the last data was read. This is to detect and recover from a connection timeout. Then read the bytes from the frameBuffer array which you can do with the help of the `readBytes()` method that returns the number of bytes read.

```

1     lastUpdate = millis();
2
3     // read the received bytes
4     myPort.readBytes(frameBuffer);
```

Then the frame buffer is translated into a ByteBuffer that allows for easy and safe access to the underlying bytes without having to worry about the array indices.

```

1     // Access raw bytes via byte buffer
2     ByteBuffer bb = ByteBuffer.wrap(frameBuffer);
```

Next we read the frame buffer and convert the bytes into pixel color values. The image gets constructed by sequentially filling the pixels array of the image. The conversion of the raw data is done with `color()` and `Byte.toUnsignedInt()`.

```

1     int i = 0;
2
3     while (bb.hasRemaining()) {
4         // read 8-bit pixel
5         byte pixelValue = bb.get();
6
7         // set pixel color
8         myImage.pixels[i++] = color(Byte.toUnsignedInt(
9             pixelValue));
}
```

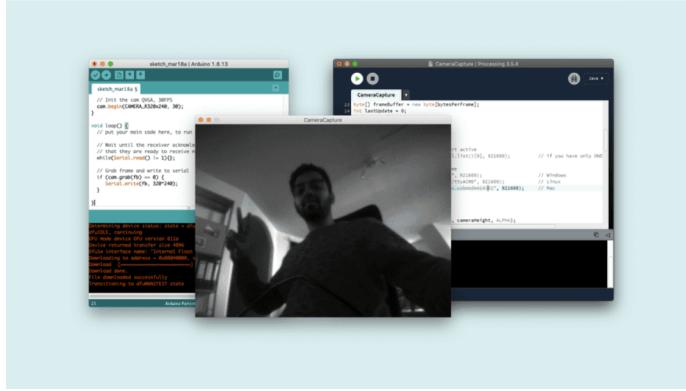


Figure 9.5.: CameraOutput

Once all the pixels have been updated, you need to tell the sketch to redraw the image. Additionally, you can send an acknowledgement back to the arduino sketch to ask it to send the pixels for the next frame. You can update the image with `updatePixels()` and write `1` to the serial port for the acknowledgement.

```

1 myImage.updatePixels();
2
3 // Ensures that the new image data is drawn in the next
4 // draw loop
5 shouldRedraw = true;
6
7 // Let the Arduino sketch know we received all pixels
8 // and are ready for the next frame
myPort.write(1);

```

Upload the Sketch:

Select the right serial port on your IDE and upload the Arduino sketch to your Portenta H7. After a successful upload, run the `CameraViewer.pde` sketch in Processing. You should be able to see the rendered camera output on the Processing canvas. 9.5 [Doc24e]

9.2.3. Conclusion:

In this example you learnt how to capture the frames from your Portenta Vision Shield's Camera and to visualize the frames through Processing. This knowledge can be useful for you to build and experiment simple computer vision applications for both outdoor and indoor environments.

Complete Sketch:

The **CaptureRawBytes.ino** Sketch. 9.1

Listing 9.1.: Simple sketch to capture the image from vision shield

```

1000 #include "camera.h"
1001 #include "himax.h"
1002
1003 HM01B0 himax;
1004 Camera cam(himax);
1005 #define IMAGE_MODE CAMERA_GRAYSCALE
1006 FrameBuffer fb(320,240,2);
1007
1008 unsigned long lastUpdate = 0;
1009
1010 void setup() {
1011     Serial.begin(921600);
1012     //Init the cam QVGA, 30FPS
1013     cam.begin(CAMERA_R320x240, IMAGE_MODE, 30);
1014 }
1015
1016 void loop() {
1017     // put your main code here, to run repeatedly:
1018     if (!Serial) {
1019         Serial.begin(921600);
1020         while (!Serial);
1021     }
1022
1023     // Time out after 2 seconds and send new data
1024     bool timeoutDetected = millis() - lastUpdate > 2000;
1025
1026     // Wait until the receiver acknowledges
1027     // that they are ready to receive new data
1028     if (!timeoutDetected && Serial.read() != 1) return;
1029
1030     lastUpdate = millis();
1031
1032     // Grab frame and write to serial
1033     if (cam.grabFrame(fb, 3000) == 0) {
1034         Serial.write(fb.getBuffer(), cam.frameSize());
1035     }
1036 }
```

..//Code/Vision shield/Camera Capture/CaptureRawBytes.ino

The **CameraViewer.pde** Sketch. 9.2

Listing 9.2.: Simple sketch to capture the image from vision shield

```

1000 /*
1001 This sketch reads a raw Stream of RGB565 pixels
1002 from the Serial port and displays the frame on
1003 the window.
```

```
1004 Use with the Examples -> CameraCaptureRawBytes Arduino
1005 sketch.
1006 This example code is in the public domain.
1007 */
1008
1009 import processing.serial.*;
1010 import java.nio.ByteBuffer;
1011 import java.nio.ByteOrder;
1012
1013 Serial myPort;
1014
1015 // must match resolution used in the sketch
1016 final int cameraWidth = 320;
1017 final int cameraHeight = 240;
1018 final int cameraBytesPerPixel = 1;
1019 final int cameraPixelCount = cameraWidth * cameraHeight;
1020 final int bytesPerFrame = cameraPixelCount *
1021     cameraBytesPerPixel;
1022
1023 PImage myImage;
1024 byte[] frameBuffer = new byte[bytesPerFrame];
1025 int lastUpdate = 0;
1026 boolean shouldRedraw = false;
1027
1028 void setup() {
1029     size(640, 480);
1030
1031     // if you have only ONE serial port active
1032     //myPort = new Serial(this, Serial.list()[0], 921600);
1033         // if you have only ONE serial port active
1034
1035         // if you know the serial port name
1036         //myPort = new Serial(this, "COM5", 921600);
1037             // Windows
1038         //myPort = new Serial(this, "/dev/ttyACM0", 921600);
1039             // Linux
1040         myPort = new Serial(this, "/dev/cu.usbmodem14401", 921600);
1041             // Mac
1042
1043     // wait for full frame of bytes
1044     myPort.buffer(bytesPerFrame);
1045
1046     myImage = createImage(cameraWidth, cameraHeight, ALPHA);
1047
1048     // Let the Arduino sketch know we're ready to receive data
1049     myPort.write(1);
1050 }
```

```

1052     myPort.write(1);
1053 }
1054 if (shouldRedraw){
1055     PImage img = myImage.copy();
1056     img.resize(640, 480);
1057     image(img, 0, 0);
1058     shouldRedraw = false;
1059 }
1060 }
1061 void serialEvent(Serial myPort) {
1062     lastUpdate = millis();
1063
1064     // read the received bytes
1065     myPort.readBytes(frameBuffer);
1066
1067     // Access raw bytes via byte buffer
1068     ByteBuffer bb = ByteBuffer.wrap(frameBuffer);
1069
1070     /*
1071      Ensure proper endianness of the data for > 8 bit values.
1072      When using > 8bit values uncomment the following line and
1073      adjust the translation to the pixel color.
1074     */
1075     //bb.order(ByteOrder.BIG_ENDIAN);
1076
1077     int i = 0;
1078
1079     while (bb.hasRemaining()) {
1080         // read 8-bit pixel
1081         byte pixelValue = bb.get();
1082
1083         // set pixel color
1084         myImage.pixels[i++] = color(Byte.toUnsignedInt(pixelValue));
1085     }
1086
1087     myImage.updatePixels();
1088
1089     // Ensures that the new image data is drawn in the next
1090     // draw loop
1091     shouldRedraw = true;
1092
1093     // Let the Arduino sketch know we received all pixels
1094     // and are ready for the next frame
1095     myPort.write(1);
1096 }

```

..../Code/Vision shield/Camera Capture/cameraviewer.pde

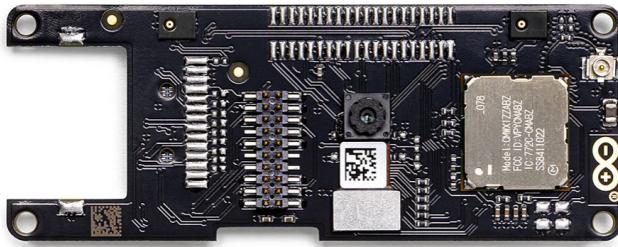


Figure 9.6.: VisionShield

9.3. Saving Bitmap Camera Images to the SD Card and PC

This tutorial shows you how to capture a frame from the Portenta Vision Shield Camera module and save the output as a bitmap image. It will allow you to see the output directly on your computer without using any third party tool. 9.6 [Doc24g]

9.3.1. Goals:

- Capturing a frame from the camera.
- Make the bitmap binary file with the correct settings.
- Save the bitmap on an SD Card.
- Save the bitmap on an PC:
- Visualize the captured image on your computer. [Doc24g]

9.3.2. Required Hardware and Software:

- Portenta H7
- Portenta Vision Shield (LoRa or Ethernet)
- USB-C cable
- Micro SD card
- Arduino IDE [Doc24g]

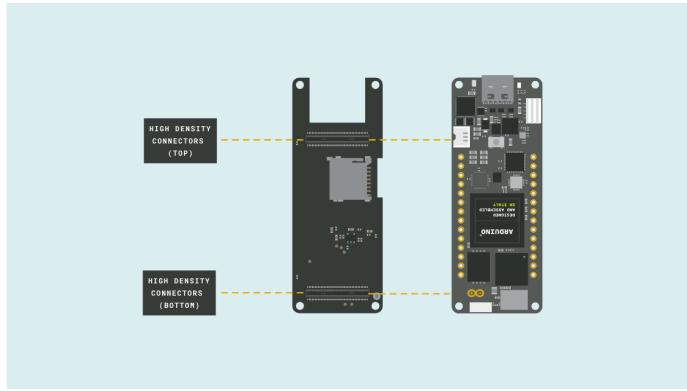


Figure 9.7.: Connection Vision Shield

9.3.3. Instructions:

The Setup:

Connect the Portenta Vision Shield to your Portenta H7 as shown in the figure. The top and bottom high density connectors are connected to the corresponding ones on the underside of the H7 board. Plug in the H7 to your computer using the USB-C cable. [Doc24g]

The Camera

You will be using the **Himax HM-01B0** camera module which has a resolution of 320 by 240 and the output data its in grayscale with 8 bits per pixel (bpp). It is important to have this in mind as the **.bmp** (bitmap) format has some needed configuration depending on the data being used. [Doc24g]

```

1 #include "camera.h" // Multi Media Card APIs
2 #include "himax.h" // API to read from the Himax camera
    found on the Portenta Vision Shield

```

Bitmap File Format

The bitmap binary file needs to contain some information in order to tell the computer for example the resolution of the picture and the bit-depth (bpp). Bit depth refers to the color information stored in the image. The higher the bit depth of an image, the more colors it can store. As the bit depth increases, the file size of the image also increases, because more color information has to be stored for each pixel in the image.

The following table 9.1 shows all the headers, the size of its buffer, offsets, the settings that are being used with their details: [Doc24g]

The final size of the file is 77.879 KB.

Name	Size	Details
DIB	14 Bytes	Bitmap information, setting the size of the file.
File Header	40 Bytes	This header requires the resolution and the bpp.
Palette (Color Map)	1025 Bytes	This header is mandatory on bitmaps with a bpp ≤ 8 , setting the grayscale.
Image data	76800 Bytes	The raw image data, in this case, each pixel has 8 bits (1 Byte) and the resolution is 320 x 240, with no compression.

Table 9.1.: Description of the bitmap file structure.

The Sketch:

You can find the sketch on the latest version of the Arduino Pro Tutorials at [examples > Vision Shield to SD Card bmp > visionShield-Bitmap.ino](#)

First you need to include the needed libraries [Doc24g]

```

1 #include "SDMMCBlockDevice.h" // Multi Media Card APIs
2 #include "FATFileSystem.h"    // Mbed API for portable and
3                                // embedded systems
4
5 #include "camera.h" // Arduino Mbed Core Camera APIs
6 #include "himax.h"  // API to read from the Himax camera
7                                // found on the Portenta Vision Shield

```

Then define the following objects with their respective constructor blockDevice and fileSystem objects, needed for getting access to the SD Card and the file system. [Doc24g]

```

1 #include "SDMMCBlockDevice.h" // Multi Media Card APIs
2 #include "FATFileSystem.h"    // API to run operations on a
3                                // FAT file system
4 SDMMCBlockDevice blockDevice;
5 mbed FATFileSystem fileSystem("fs");
6
7 #include "camera.h" // Arduino Mbed Core Camera APIs
8 #include "himax.h"  // API to read from the Himax camera
9                                // found on the Portenta Vision Shield
10 HM01B0 himax;
11 Camera cam(himax);
12
13 FrameBuffer frameBuffer; // Buffer to save the camera stream

```

For the bitmap headers binary file you will need some information like the resolution of the image, the bits per pixel and more; so you can define your settings as shown: [Doc24g]

```
1 // Settings for our setup
2 #define IMAGE_HEIGHT (unsigned int)240
3 #define IMAGE_WIDTH (unsigned int)320
4 #define IMAGE_MODE CAMERA_GRAYSCALE
5 #define BITS_PER_PIXEL (unsigned int)8
6 #define PALETTE_COLORS_AMOUNT (unsigned int)(pow(2,
    BITS_PER_PIXEL))
7 #define PALETTE_SIZE (unsigned int)(PALETTE_COLORS_AMOUNT *
        4) // 4 bytes = 32bit per color (3 bytes RGB and 1 byte
        0x00)
8 #define IMAGE_PATH "/fs/image.bmp"
9
10 // Headers info
11 #define BITMAP_FILE_HEADER_SIZE (unsigned int)14 // For
    storing general information about the bitmap image file
12 #define DIB_HEADER_SIZE (unsigned int)40 // For storing
    information about the image and define the pixel format
13 #define HEADER_SIZE (BITMAP_FILE_HEADER_SIZE +
    DIB_HEADER_SIZE)
```

The program has the following functions declared: [Doc24g]

1. `void mountSDCard()`
2. `unsigned char * captureImage()`
3. `void setFileHeaders(bitmapFileHeaders, bitmapDIBHeader, fileSize)`
4. `void setColorMap(colorMap)`
5. `saveImage(imageData, imagePath)`

To mount the SD Card you will use the following function in the sketch: [Doc24g]

```
1 // Mount File system block
2 void mountSDCard(){
3     int error = fileSystem.mount(blockDevice);
4     if (error){
5         Serial.println("Trying to reformat...");
```

```

6     int formattingError = fileSystem.reformat(blockDevice);
7     if (formattingError) {
8         Serial.println("No SD Card found");
9         while (1);
10    }
11 }
12 }
```

The function to capture the frame from the camera: [Doc24g]

```

1 // Get the raw image data (8bpp grayscale)
2 unsigned char * captureImage(){
3     if (cam.grabFrame(frameBuffer, 3000) == 0){
4         return frameBuffer.getBuffer();
5     } else {
6         Serial.println("could not grab the frame");
7         while (1);
8     }
9 }
```

To manipulate the file you will need to set the headers on the binary information as follows: [Doc24g]

```

1 // Set the headers data
2 void setFileHeaders(unsigned char *bitmapFileHeader,
3                     unsigned char *bitmapDIBHeader, int fileSize){
4     // Set the headers to 0
5     memset(bitmapFileHeader, (unsigned char)(0),
6            BITMAP_FILE_HEADER_SIZE);
6     memset(bitmapDIBHeader, (unsigned char)(0),
7            DIB_HEADER_SIZE);
8
9     // File header
10    bitmapFileHeader[0] = 'B';
11    bitmapFileHeader[1] = 'M';
12    bitmapFileHeader[2] = (unsigned char)(fileSize);
13    bitmapFileHeader[3] = (unsigned char)(fileSize >> 8);
14    bitmapFileHeader[4] = (unsigned char)(fileSize >> 16);
15    bitmapFileHeader[5] = (unsigned char)(fileSize >> 24);
16    bitmapFileHeader[10] = (unsigned char)HEADER_SIZE +
17                           PALETTE_SIZE;
18
19     // Info header
20    bitmapDIBHeader[0] = (unsigned char)(DIB_HEADER_SIZE);
21    bitmapDIBHeader[4] = (unsigned char)(IMAGE_WIDTH);
22    bitmapDIBHeader[5] = (unsigned char)(IMAGE_WIDTH >> 8);
```

```

20     bitmapDIBHeader[8] = (unsigned char)(IMAGE_HEIGHT);
21     bitmapDIBHeader[9] = (unsigned char)(IMAGE_HEIGHT >> 8);
22     bitmapDIBHeader[14] = (unsigned char)(BITS_PER_PIXEL); }
```

In this case that our image (320x240) is 8 bits per pixel and grayscale on the bitmap rules you need to define the color table (color map) to assign an specific RGB color for our 8 bit color. On the following function it sets the color map as a grayscale of RGB colors from [R:0x00 G:0x00 B:0x00] to [R:0xFF G:0xFF B:0xFF] [Doc24g]

The function in charge to save the image will use the previous functions to set the headers and write the buffers into the file **image.bmp** [Doc24g]

```

1 // Save the headers and the image data into the .bmp file
2 void saveImage(unsigned char *imageData, const char*
   imagePath){
3     int fileSize = BITMAP_FILE_HEADER_SIZE + DIB_HEADER_SIZE +
   IMAGE_WIDTH * IMAGE_HEIGHT;
4     FILE *file = fopen(imagePath, "w");
5
6     // Bitmap structure (Head + DIB Head + ColorMap + binary
   image)
7     unsigned char bitmapFileHeader[BITMAP_FILE_HEADER_SIZE];
8     unsigned char bitmapDIBHeader[DIB_HEADER_SIZE];
9     unsigned char colorMap[PALETTE_SIZE]; // Needed for < 8bpp
   grayscale bitmaps
10
11    setFileHeaders(bitmapFileHeader, bitmapDIBHeader, fileSize
   );
12    setColorMap(colorMap);
13
14    // Write the bitmap file
15    fwrite(bitmapFileHeader, 1, BITMAP_FILE_HEADER_SIZE, file)
   ;
16    fwrite(bitmapDIBHeader, 1, DIB_HEADER_SIZE, file);
17    fwrite(colorMap, 1, PALETTE_SIZE, file);
18    fwrite(imageData, 1, IMAGE_HEIGHT * IMAGE_WIDTH, file);
19
20    // Close the file stream
21    fclose(file);
22 }
```

Then to have visual feedback lets add a blink function to make 3 blinks after the photo is taken, once the blue LED is ON it means the picture was taken. [Doc24g]

```
1 void countDownBlink(){
2     for (int i = 0; i < 6; i++){
3         digitalWrite(LEDG, i % 2);
4         delay(500);
5     }
6     digitalWrite(LEDG, HIGH);
7     digitalWrite(LEDB, LOW);
8 }
```

Now that you have all the functions to be used, inside the `setup()` its call them only once after the board restarts. [Doc24g]

```
1 void setup(){
2     Serial.begin(115200);
3     while (!Serial & millis() < 5000);
4
5     Serial.println("Mounting SD Card...");
6     mountSDCard();
7     Serial.println("SD Card mounted.");
8
9     // Init the cam QVGA, 30FPS, Grayscale
10    if (!cam.begin(CAMERA_R320x240, IMAGE_MODE, 30)){
11        Serial.println("Unable to find the camera");
12    }
13
14    countDownBlink();
15    Serial.println("Fetching camera image...");
16    unsigned char *imageData = captureImage();
17
18    Serial.println("Saving image to SD card...");
19    saveImage(imageData, IMAGE_PATH);
20
21    fileSystem.unmount();
22    Serial.println("Done. You can now remove the SD card.");
23 }
```

The `loop()` is empty, as it only does one shot when the Serial Monitor is open or after 5 seconds passed after boot. [Doc24g]

Upload the Sketch

Select the right serial port on your IDE and upload the Arduino sketch to your Portenta H7. [Doc24g]

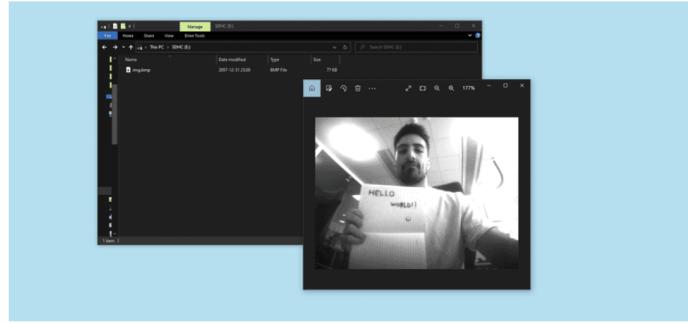


Figure 9.8.: SavingImage

Try It Out

Insert a micro SD Card into the Portenta Vision Shield.

Connect the Portenta Vision Shield to the Portenta H7.

Once the sketch is uploaded, open the Serial Monitor or wait 5 seconds: you should see that everything is fine and the capture has been taken.

9.8

Once the capture is saved, remove the SD Card and plug it into a computer/phone with an SD Card reader, open the storage unit, look for a bitmap called `image.bmp` and open it to check the result. You will be able to see a grayscale image on your device's image viewer. [Doc24g]

Full Sketch

Listing 9.3.: Simple sketch to save the image from vision shield

```

1000 #include "SDMMCBlockDevice.h" // Multi Media Card APIs
1001 #include "FATFileSystem.h"    // API to run operations on a
1002   FAT file system
1003 SDMMCBlockDevice blockDevice;
1004 mbed::FATFileSystem fileSystem("fs");
1005
1006 #include "camera.h" // Arduino Mbed Core Camera APIs
1007 #include "himax.h" // API to read from the Himax camera
1008   found on the Portenta Vision Shield
1009 HM01B0 himax;
1010 Camera cam(himax);
1011
1012 FrameBuffer frameBuffer; // Buffer to save the camera stream
1013
1014 // Settings for our setup
1015 #define IMAGE_HEIGHT (unsigned int)240
1016 #define IMAGE_WIDTH (unsigned int)320
1017 #define IMAGE_MODE CAMERA_GRAYSCALE
1018 #define BITS_PER_PIXEL (unsigned int)8
1019 #define PALETTE_COLORS_AMOUNT (unsigned int)(pow(2,
1020   BITS_PER_PIXEL))
1021 #define PALETTE_SIZE (unsigned int)(PALETTE_COLORS_AMOUNT *
1022

```

```
    4) // 4 bytes = 32bit per color (3 bytes RGB and 1 byte 0
       x00)
#define IMAGE_PATH "/fs/image.bmp"
1020
// Headers info
1022#define BITMAP_FILE_HEADER_SIZE (unsigned int)14 // For
      storing general information about the bitmap image file
#define DIB_HEADER_SIZE (unsigned int)40 // For storing
      information about the image and define the pixel format
1024#define HEADER_SIZE (BITMAP_FILE_HEADER_SIZE +
      DIB_HEADER_SIZE)

1026 void setup(){
1028     Serial.begin(115200);
1029     while (!Serial && millis() < 5000);
1030
1031     Serial.println("Mounting SD Card... ");
1032     mountSDCard();
1033     Serial.println("SD Card mounted.");
1034
1035     // Init the cam QVGA, 30FPS, Grayscale
1036     if (!cam.begin(CAMERA_R320x240, IMAGE_MODE, 30)){
1037         Serial.println("Unable to find the camera");
1038     }
1039     countDownBlink();
1040     Serial.println("Fetching camera image... ");
1041     unsigned char *imageData = captureImage();
1042     digitalWrite(LEDB, HIGH);

1043     Serial.println("Saving image to SD card... ");
1044     saveImage(imageData, IMAGE_PATH);
1045
1046     fileSystem.unmount();
1047     Serial.println("Done. You can now remove the SD card.");
1048 }
1049
1050 void loop(){
1051 }

1052 // Mount File system block
1053 void mountSDCard(){
1054     int error = fileSystem.mount(&blockDevice);
1055     if (error){
1056         Serial.println("Trying to reformat... ");
1057         int formattingError = fileSystem.reformat(&
1058             blockDevice);
1059         if (formattingError) {
1060             Serial.println("No SD Card found");
1061             while (1);
1062         }
1063     }
1064 }
```

```

1066 // Get the raw image data (8bpp grayscale)
1068 unsigned char * captureImage(){
1069     if (cam.grabFrame(frameBuffer, 3000) == 0){
1070         return frameBuffer.getBuffer();
1071     } else {
1072         Serial.println("could not grab the frame");
1073         while (1);
1074     }
1076 // Set the headers data
1078 void setFileHeaders(unsigned char *bitmapFileHeader, unsigned
1079     char *bitmapDIBHeader, int fileSize){
1080     // Set the headers to 0
1081     memset(bitmapFileHeader, (unsigned char)(0),
1082             BITMAP_FILE_HEADER_SIZE);
1083     memset(bitmapDIBHeader, (unsigned char)(0),
1084             DIB_HEADER_SIZE);
1085
1086     // File header
1087     bitmapFileHeader[0] = 'B';
1088     bitmapFileHeader[1] = 'M';
1089     bitmapFileHeader[2] = (unsigned char)(fileSize);
1090     bitmapFileHeader[3] = (unsigned char)(fileSize >> 8);
1091     bitmapFileHeader[4] = (unsigned char)(fileSize >> 16);
1092     bitmapFileHeader[5] = (unsigned char)(fileSize >> 24);
1093     bitmapFileHeader[10] = (unsigned char)HEADER_SIZE +
1094             PALETTE_SIZE;
1095
1096     // Info header
1097     bitmapDIBHeader[0] = (unsigned char)(DIB_HEADER_SIZE);
1098     bitmapDIBHeader[4] = (unsigned char)(IMAGE_WIDTH);
1099     bitmapDIBHeader[5] = (unsigned char)(IMAGE_WIDTH >> 8);
1100     bitmapDIBHeader[8] = (unsigned char)(IMAGE_HEIGHT);
1101     bitmapDIBHeader[9] = (unsigned char)(IMAGE_HEIGHT >> 8);
1102     bitmapDIBHeader[14] = (unsigned char)(BITS_PER_PIXEL);
1103 }
1104
1105 void setColorMap(unsigned char *colorMap){
1106     //Init the palette with zeroes
1107     memset(colorMap, (unsigned char)(0), PALETTE_SIZE);
1108
1109     // Gray scale color palette, 4 bytes per color (R, G, B,
1110     // 0x00)
1111     for (int i = 0; i < PALETTE_COLORS_AMOUNT; i++) {
1112         colorMap[i * 4] = i;
1113         colorMap[i * 4 + 1] = i;
1114         colorMap[i * 4 + 2] = i;
1115     }
1116 }
1117
1118 // Save the headers and the image data into the .bmp file

```

```

1114 void saveImage( unsigned char *imageData , const char *
1115   imagePath){
1116   int fileSize = BITMAP_FILE_HEADER_SIZE + DIB_HEADER_SIZE
1117   + IMAGE_WIDTH * IMAGE_HEIGHT;
1118   FILE *file = fopen(imagePath , "w");
1119
1120   // Bitmap structure (Head + DIB Head + ColorMap + binary
1121   // image)
1122   unsigned char bitmapFileHeader[BITMAP_FILE_HEADER_SIZE];
1123   unsigned char bitmapDIBHeader[DIB_HEADER_SIZE];
1124   unsigned char colorMap[PALETTE_SIZE]; // Needed for <= 8
1125   bpp grayscale bitmaps
1126
1127   setFileHeaders(bitmapFileHeader , bitmapDIBHeader ,
1128     fileSize);
1129   setColorMap(colorMap);
1130
1131   // Write the bitmap file
1132   fwrite(bitmapFileHeader , 1 , BITMAP_FILE_HEADER_SIZE , file
1133 );
1134   fwrite(bitmapDIBHeader , 1 , DIB_HEADER_SIZE , file );
1135   fwrite(colorMap , 1 , PALETTE_SIZE , file );
1136   fwrite(imageData , 1 , IMAGE_HEIGHT * IMAGE_WIDTH , file );
1137
1138   // Close the file stream
1139   fclose(file );
1140 }
1141
1142 void countDownBlink(){
1143   for (int i = 0; i < 6; i++){
1144     digitalWrite(LEDG, i % 2);
1145     delay(500);
1146   }
1147   digitalWrite(LEDG, HIGH);
1148   digitalWrite(LEDB, LOW);
1149 }
```

..../Code/Vision shield/SaveImage/SaveImage.ino

9.3.4. Conclusion

In this example you learned how to capture frames with your Portenta Vision Shield's Camera in the Arduino IDE, encode it with the bitmap standards and save it to an SD Card and PC.

9.4. Creating a Basic Face Filter With OpenMV

In this example you will build a MicroPython application with OpenMV, to use the Portenta Vision Shield to detect faces and overlay them with a custom bitmap image. Think of it as building your own camera filter that puts a smile on every face it detects. This tutorial is based on the face detection example that comes with the OpenMV IDE. 9.6 [Doc24a]

9.4.1. Goals:

- How to use the OpenMV IDE to run MicroPython on Portenta
- How to use the built-in face detection algorithm of OpenMV
- Copying files to the internal Flash of the Portenta
- Using MicroPython to read files from the internal Flash [Doc24a]

9.4.2. Required Hardware and Software:

- Portenta H7
- Portenta Vision Shield (LoRa or Ethernet)
- USB-C cable
- Arduino IDE 2.3+
- Portenta Bootloader Version 20+
- OpenMV IDE 2.6.4+ [Doc24a]

9.4.3. The Haar Cascade Algorithm

By harnessing the power of machine vision algorithms, objects can be detected in a camera stream. Those algorithms can be trained to detect the desired type of object. In this tutorial, you will use a machine learning based approach called Haar Cascade to detect faces.

This approach uses a cascade algorithm that has multiple stages, where the output from one stage acts as additional information for the next stage in the cascade. The different stages are responsible for detecting edges, lines, contrast checks and calculating pixel values in a given image. Larger areas of the image are checked first in the earlier stages, followed by more numerous and smaller area checks in later stages. The Haar Cascade function provided by OpenMV allows to specify the amount of

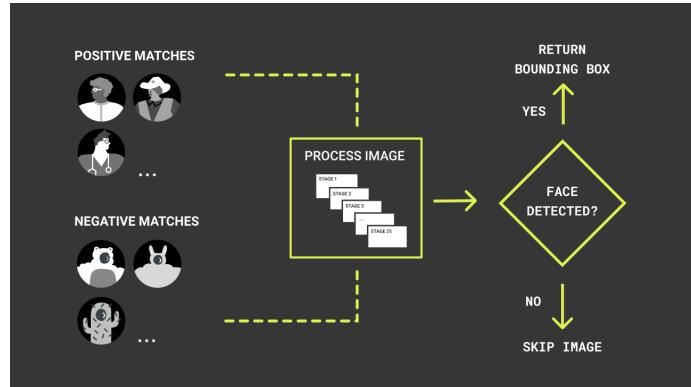


Figure 9.9.: HaarCascade

stages. Fewer stages make the detection faster, while leading to more false positives.

The built-in Haar Cascade model for faces was trained with hundreds of images containing faces that are labeled as such and images that do not contain faces labeled differently. That allows the algorithm to distinguish such images after it is being trained.

9.4.4. Instructions:

Creating the Face Detection Script For this tutorial, you will be using the OpenMV IDE along with the OpenMV firmware on your Portenta H7 to build the face detection script. If this is your first time using the Portenta Vision Shield and OpenMV, we recommend you to take a look at the "Configuring the Development Environment" section inside the Blob Detection tutorial to configure the development environment.

The Basic Setup

Attach your Portenta Vision Shield to your Portenta H7 and open the OpenMV Editor. For this tutorial, you will create a new script that is based on the face detection example provided by OpenMV. Create a new script by clicking the "New File" button in the toolbar on the left side and save it as **face detection.py**. [Doc24a]

Importing the Modules

The script starts by importing the sensor, image and time modules for handling the camera sensor, using machine vision algorithms and time tracking functions. **examples > Vision Shield to SD Card bmp > visionShieldBitmap.ino**

First you need to include the needed libraries [Doc24a]

```
1 import sensor # Import the module for sensor related
   functions
2 import image # Import module containing machine vision
   algorithms
3 import time # Import module for tracking elapsed time
```

Preparing the Sensor

The next step is to calibrate the camera sensor to achieve the best results using the sensor module. You can use the `set contrast()` function to set the contrast of the sensor to its highest value (3). This can help the algorithm identifying lines and edges more easily. `set gainceiling()` controls the amplification of the signal from the camera sensor including any associated background noise. For maximizing the detection success rate, it is recommended to set the camera frame size to HQVGA. [Doc24a]

```
1 # Sensor settings
2 sensor.set_contrast(3)
3 sensor.set_gainceiling(16)
4 sensor.set_framesize(sensor.HQVGA)
5 sensor.set_pixformat(sensor.GRAYSCALE)
```

Displaying a Bitmap Image

Once you know the location of the faces in the camera image, you can overlay them with an image of your choice. OpenMV currently supports bmp, pgm or ppm image formats. Image formats with an alpha layer such as PNG are not supported yet.

In this tutorial you will use a preloaded smiley image in the monochrome Portable Bitmap Image (.pbm) format. This format consists of a matrix of zeroes and ones denoting black and white pixels. 1 stands for a black pixel, 0 for a white one. If you want to create your custom image, make sure you save it in one of the supported bitmap formats (bmp, pgm or ppm). You may use an image editor of your choice which supports exporting images in one of these formats. For this tutorial Adobe Photoshop was used.

Connect your Portenta board to your computer if you have not done so yet. Make sure you are running the OpenMV firmware on the Portenta. If you have not installed the OpenMV firmware yet, take a look at the "Configuring the Development Environment" section which explains how to proceed in that case.

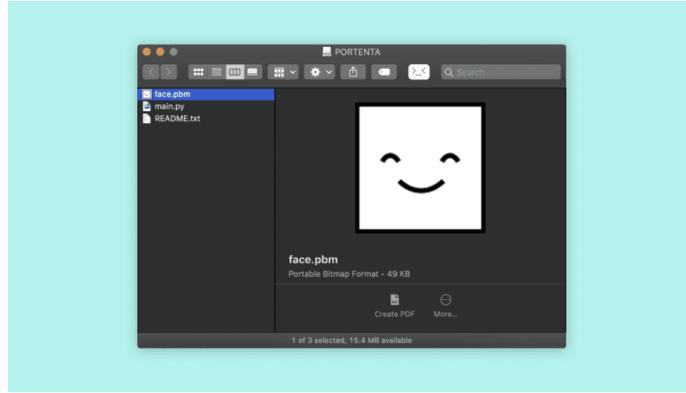


Figure 9.10.: BitmapImage

Download this file containing the smiley bitmap and copy it to the Flash drive that was mounted when you connected the Portenta running the OpenMV firmware. [Doc24a]

Load the image into a variable called `faceImage` using the `Image()` function from the `image` module. The initial slash refers to the root directory of the Flash drive. In order to use the image as an overlay to the camera stream, instead of directly displaying it, set the `copy_to_fb` to `False` such that it does not get copied into the frame buffer automatically.

```
1  faceImage = image.Image("/face.pbm", copy_to_fb=False)
```

Before drawing the image on top of the camera stream you need to figure out the scale ratio to match the detected face size in the camera stream. The provided bitmap image comes in a 128x128 px resolution. You can calculate the correct scale ratio with the following formula:

```
1  faceX = boundingBox[0]
2  faceY = boundingBox[1]
3  faceWidth = boundingBox[2]
4
5  # Calculates the scale ratio to scale the bitmap image to
   # match the bounding box
6  scale_ratio = faceWidth / faceImage.width()
```

You can then draw the scaled bitmap image on top of the camera image using the `draw_image` function:

```
1  # Draws the bitmap on top of the camera stream
2  cameraImage.draw_image(faceImage, faceX, faceY, x_scale=
   scale_ratio, y_scale=scale_ratio)
```

Uploading the Script

Let's program the Portenta with the complete script and test if the algorithm works. Copy the following script and paste it into the new script file that you created. [Doc24a]

Listing 9.4.: Simple sketch for bitmap image

```
1000 import sensor # Import the module for sensor related
# functions
1001 import image # Import module containing machine vision
# algorithms
1002 import time # Import module for tracking elapsed time
1003
1004 sensor.reset() # Resets the sensor
1005 sensor.set_contrast(3) # Sets the contrast to the highest
# level (min -3, max 3)
1006 sensor.set_gainceiling(16) # Sets the amplification of camera
# sensor signal
1007
1008 # HQVGA and GRayscale are the best for face tracking.
1009 sensor.set_framesize(sensor.HQVGA)
1010 sensor.set_pixformat(sensor.GRAYSCALE)
1011
1012 # Load the built-in frontal face Haar Cascade
# By default this will use all stages, lower stages is faster
# but less accurate.
1013 face_cascade = image.HaarCascade("frontalface", stages=25)
1014 print(face_cascade) # Prints the Haar Cascade configuration
1015
1016 faceImage = image.Image("/face.pbm", copy_to_fb=False) #
# Loads a bitmap file from the flash storage
1017 clock = time.clock() # Instantiates a clock object to
# calculate the FPS
1018
1019 while (True):
1020     clock.tick() # Advances the clock
1021     cameraImage = sensor.snapshot() # Takes a snapshot and
# saves it in memory
1022
1023     # Find objects.
# Note: Lower scale factor scales-down the image more and
# detects smaller objects.
1024     # Higher threshold results in a higher detection rate,
# with more false positives.
1025     boundingBoxes = cameraImage.find_features(face_cascade,
threshold=1, scale_factor=1.5)
1026
1027     # Draw objects
```

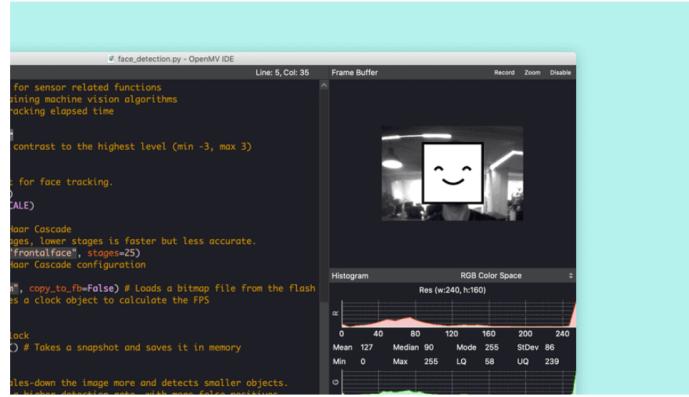


Figure 9.11.: ImageOutput

```

1030    for boundingBox in boundingBoxes:
1031        faceX = boundingBox[0]
1032        faceY = boundingBox[1]
1033        faceWidth = boundingBox[2]
1034
1035        # Calculates the scale ratio to scale the bitmap
1036        # image to match the bounding box
1037        scale_ratio = faceWidth / faceImage.width()
1038        # Draws the bitmap on top of the camera stream
1039        cameraImage.draw_image(faceImage, faceX, faceY,
1040                               x_scale=scale_ratio, y_scale=scale_ratio)
1041
1042    # Print FPS.
1043    # Note: The actual FPS is higher when not displaying a
1044    # frame buffer preview in the IDE
1045    print(clock.fps())

```

..//Code/Vision shield/Face Detection/face_detection.py

Click on the "Play" button at the bottom of the left toolbar. Point the camera on the Portenta Vision Shield towards your face and check if the Portenta can detect it. Once it detects your face, it should be covered with a smiley.

9.4.5. Conclusion

In this example you learned how to use OpenMV's built-in face detection algorithm which is based on Haar Cascade. Furthermore, you learned how to copy a file to the internal flash and how to load an image from the Flash into the memory. You have also learned how to draw an image on top of a snapshot from the camera stream.

10. Portenta Cat. M1/NB IoT GNSS Shield)

Arduino is a prestigious open-source electronics platform that seamlessly integrates user-friendly hardware and software, making it highly appealing to inventors, hobbyists, engineers, students, and professionals alike. At the heart of this platform is a versatile microcontroller, a mini-computer, that can be programmed and manipulated to interact with external signals from a wide variety of modules, including sensors and actuators, resulting in a wide array of control systems and interactive setups. [Ard24i]

Part of the appeal of Arduino is its commitment to making technology accessible to beginners. This is exemplified in the Arduino integrated development environment (IDE), which provides a streamlined, beginner-friendly programming environment. The IDE assists users in developing and uploading code to the Arduino microcontroller. In summary, Arduino stands as a beacon for the open-source movement in electronics and has played a pivotal role tech-educational contexts, providing an easy-to-understand yet powerful platform for exploring the world of electronics and computer programming. [Ard24i]

10.1. Introduction

The Portenta Cat. M1/NB IoT GNSS-Shield allows you to improve the connectivity capabilities of your Portenta H7 applications. The Shield uses a Thales Cinterion TX62 wireless module designed for high-efficiency, low-power IoT applications to ensure optimized bandwidth and performance.

The Portenta Cat. M1/NB IoT GNSS-Shield combines with the strong edge computing performance of the Portenta H7 and enables the development of asset tracking and remote monitoring applications in industrial settings as well as in agriculture, public facilities and smart cities. The Shield offers a cellular connection for both Cat. M1 and NB-IoT networks with the option to use eSIM technology. Easily track your valuable assets across the city worldwide with your choice of GPS, GLONASS, Galileo or BeiDou.

The following below figures shows a Portenta Cat. M1/NB IoT GNSS Shield [Ard24g]

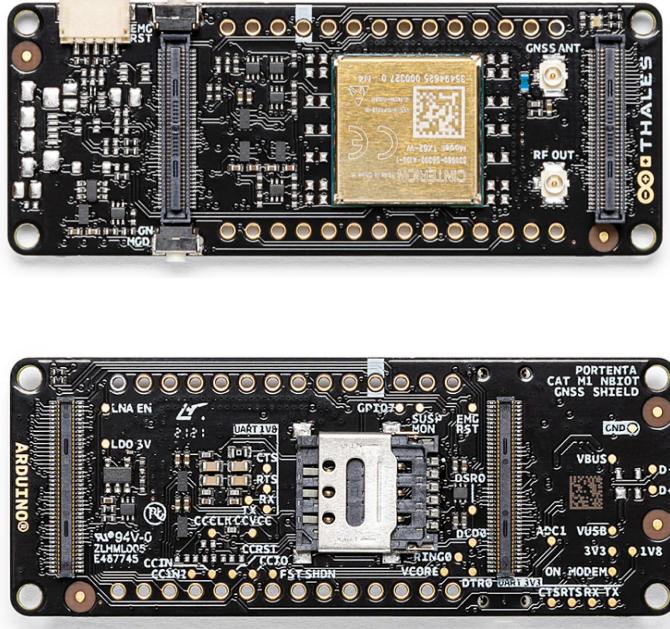


Figure 10.1.: Portenta Cat. M1/NB IoT GNSS-Shield
[Ard24g]

10.2. Portenta Cat. M1/NB IoT GNSS Shield

The Arduino Portenta Cat. M1/NB IoT GNSS Shield is designed to enhance the connectivity and localization capabilities of your Portenta or MKR boards. Portenta Cat. M1/NB IoT GNSS Shield is very small 66mm × 25.4mm in size and 9g in weight. It is a low power consumption board and operate normally on 3.3 V. Arduino Portenta Cat. M1/NB IoT GNSS Shield can still utilize UART, I2C, SPI communication protocols by connecting external devices to the appropriate pins on the Portenta board communication. It is designed to provide reliable connectivity to the Portenta family of boards. It incorporates Cinterion TX62 wireless module for Cellular connectivity and provides connectivity to both Cat.M1 and NB-IoT networks. Also it supports GNSS capabilities for real-time positioning. The table below 10.1 shows the technical specifications of Portenta Cat. M1/NB IoT GNSS Shield. [Ard24f]

10.2.1. Board Topology

The figure 10.2 represents the Pinout details of Portenta Cat. M1/NB IoT GNSS Shield Board. The Arduino Portenta Cat. M1/NB IoT GNSS Shield requires a Portenta/MKR host board to operate. Power to the Arduino Portenta Cat. M1/NB IoT GNSS Shield is provided by the host Portenta board via the highdensity connector. As can be seen from the

Connectivity	Cinterion TX62 wireless module; NB-IoT - LTE CAT.M1; 3GPP Rel.14 Compliant Protocol LTE Cat. M1/NB1/NB2; UMTS BANDS: 1 / 2 / 3 / 4 / 5 / 8 / 12(17) / 13 / 18 / 19 / 20 / 25 / 26 / 27 / 28 / 66 / 71 / 85; LTE Cat.M1 DL: max. 300 kbps, UL: max. 1.1 Mbps; LTE Cat.NB1 DL: max. 27 kbps, UL: max. 63 kbps; LTE Cat.NB2 DL: max. 124 kbps, UL: max. 158 kbps
Short Messaging Service (SMS)	Point-to-point mobile terminated (MT) and mobile originated (MO) Text Mode; Protocol Data Unit (PDU) Mode
Localization Support	GNSS capability (GPS/BeiDou/Galileo/GLONASS)
Dimensions	66 mm x 25.4 mm
Other	Embedded IPv4 and IPv6 TCP/IP stack access; Internet Services: TCP server/client, UDP client, DNS, Ping, HTTP client, FTP client, MQTT client Secure Connection with TLS/DTLS Secure boot
Operating Temperatures	-40° C to +85° C (-104° F to 185°F)

Table 10.1.: Technical Specifications of Portenta Cat. M1/NB IoT GNSS-Shield

[Ard24f]

figure, the pins A0 to A6 represents analog pins and D0 to D14 represents the digital pins. The ground pins (GND) are denoted in black color and power pins in red color. Also there is a RESET pin in the board. We can trigger the reset in the board by using Push-button switch either manually else programmable reset. VIN is the input voltage pin. This pin is used to power the Portenta Cat. M1/NB IoT GNSS Shield along with Portenta H7 board when no power source in portenta boards. The 3v3 represents 3.3 volts output voltage pin and +5V pin is also a output voltage pin which outputs 5V from the board when powered using portenta host board or using the VIN pin. [Ard24f]

The figure 10.3 shows the Top View of Portenta Cat. M1/NB IoT GNSS Shield Board with the components listed in the table 10.2 The figure 10.4 shows the Top View of Portenta Cat. M1/NB IoT GNSS Shield Board with the components listed in the table 10.3 [Ard24f]

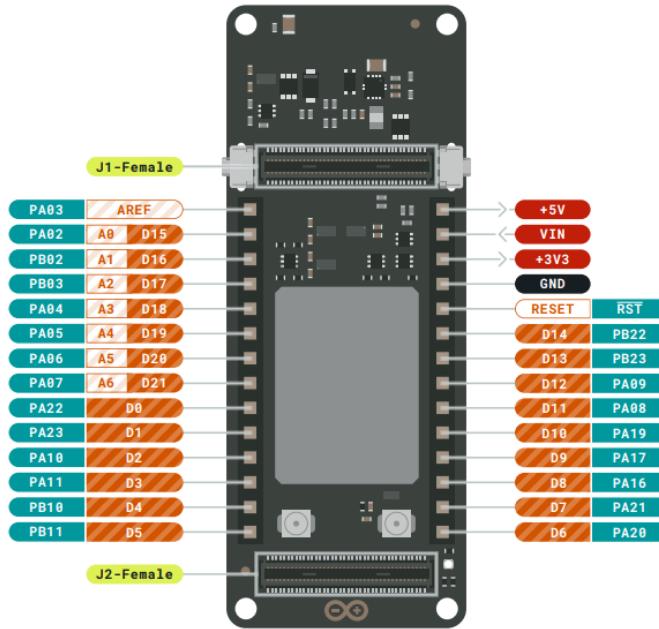


Figure 10.2.: Portenta Cat. M1/NB IoT GNSS Shield Pinout
[Ard24f]

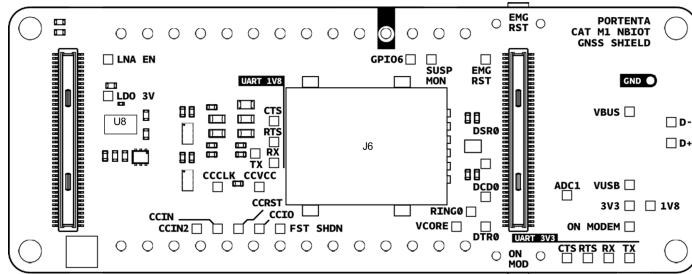


Figure 10.3.: Portenta Cat. M1/NB IoT GNSS Shield Top View

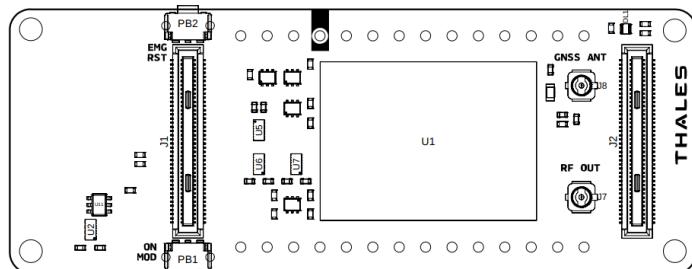


Figure 10.4.: Portenta Cat. M1/NB IoT GNSS Shield Bottom View

Ref	Description
DL1	SMLP34RGB2W3 RGB LED
PB1	TL3340AF160QG Mode Select button
U1	TX62-W Cellular-GNSS Module
J1,12	Female High Density Connector
PB2	TL3340AF160QG Reset button
U2,U5,U7	74LVCH2T45GT Level Translator

Table 10.2.: Portenta Cat. M1/NB IoT GNSS Shield Top View Component Description

Ref.	Description
J3, J4	Male High Density Connectors
J6	SIM8060-6-0-14-00-A Hinged Nano SIM card module
J7	U.FL-R-SMT-1(60) micro UFL Cellular Antenna Connector
J8	U.FL-R-SMT-1(60) micro UFL GNSS Antenna Connector
U3,U4,U6	74LVCH2T45GT Level Translator
U8	TC1185-3.0VCT713 3V 150mA LDO IC

Table 10.3.: Portenta Cat. M1/NB IoT GNSS Shield Bottom View Component Description

10.2.2. Features

[Ard24f]

- **Cinterion TX62 wireless module**
 - Cellular connectivity and positioning support
 - Embedded IPv4 and IPv6 TCP/IP stack access
 - Internet Services
 - TCP server/client
 - UDP client
 - DNS
 - Ping
 - HTTP client
 - FTP client
 - MQTT client Secure Connection with TLS/DTLS Secure boot
- **Cellular Connectivity**
 - LTE Cat. M1/NB1/NB2
 - 3GPP Rel.14 Compliant Protocol LTE Cat. M1/NB1/NB2

- UMTS BANDS: 1 / 2 / 3 / 4 / 5 / 8 / 12 / 13 / 18 / 19 / 20 / 25 / 26 / 27 / 28 / 66 / 71 / 85
 - LTE Bands
 - LTE Cat.M1 DL: max. 300 kbps, UL: max. 1.1 Mbps
 - LTE Cat.NB1 DL: max. 27 kbps, UL: max. 63 kbps
 - LTE Cat.NB2 DL: max. 124 kbps, UL: max. 158 kbps
 - Short Messaging Service (SMS)
 - Point-to-point mobile terminated (MT) and mobile originated (MO)
 - Text Mode
 - Protocol Data Unit (PDU) Mode
- **Positioning Support**
 - Multiple GNSS Support
 - GPS
 - GLONASS
 - Galileo
 - BeiDou
 - MKR and Portenta Compatible
 - MKR requires headers

10.2.3. Compabilities

[Ard24f] The following **software tools** allow you to program your board both online and offline.

- Arduino IDE
- Arduino CLI
- Web Editor

The **hardware** listed below is compatible with this product

- Portenta Breakout
- Portenta H7
- Portenta H7 Lite
- Portenta H7 Lite Connected

10.2.4. GNSS Capabilities

The Arduino Portenta Cat M1/NB IoT GNSS Shield supports precise location tracking and navigation through multiple GNSS systems. [Ard24f]

- **Supported GNSS Systems:** The shield supports four major GNSS constellations: GPS (Global Positioning System), GLONASS (Global Navigation Satellite System), Galileo, and BeiDou.
- **NMEA Protocol:** GNSS information is transmitted using the NMEA (National Marine Electronics Association) protocol, a standard for interfacing marine electronics.
- **Antenna Requirements:** An active GNSS antenna can be connected via the micro UFL connector (J8). The antenna should have a bias voltage of 3.0V and an input impedance of 50Ω . To ensure compatibility with all GNSS systems, the antenna should support frequency bands over the 1559 - 1606 MHz range.

GNSS and cellular services cannot be used simultaneously. [Ard24f]

10.2.5. GSM Capabilities

The Arduino Portenta Cat M1/NB IoT GNSS Shield provides robust cellular connectivity through its GSM features, making it suitable for a wide range of IoT applications. [Ard24f]

- **TX62-W Module:** The shield uses the TX62-W Module (U1) to access various cellular networks, supporting LTE Cat M1 and NB-IoT technologies.
- **External Antenna:** It is possible to connect an external cellular antenna via the micro UFL connector (J7). The input impedance for the cellular antenna is 50Ω .
- **Data Transmission:** Supports both SMS and data transfer functionalities. SMS messages can be stored in the SIM card module.
- **AT Commands:** The modem can be controlled and configured using AT commands, allowing for precise management of the module's functions.
- **Status Indicator:** Modem status is indicated by the RGB LED (DL1), providing visual feedback on the module's operation.
- **SIM Card Integration:** A standard SIM card slot allows for easy integration with mobile network services. Users need to configure the APN and PIN settings in their code.

- **IPv4 and IPv6 Support:** Supports both IPv4 and IPv6, ensuring future-proof connectivity.

The Portenta Cat. M1/NB IoT GNSS Shield requires a physical nano-SIM for cellular connectivity. [Ard24f]

10.3. Portenta Cat. M1/NB IoT GNSS Shield Application Examples

Arduino Portenta Cat. M1/NB IoT GNSS Shield shield can connect to your Portenta/MKR board enabling a wide range of applications. [Ard24f]

- **Global Asset Tracking:** The Arduino Portenta Cat. M1/NB IoT GNSS Shield provides connectivity to four major satellite positioning networks, allowing you to track your inventory reliably. The multi-band cellular connectivity ensures you get live updates on your inventory nearly anywhere in the world.
- **Remote Node Monitoring:** The Arduino Portenta Cat. M1/NB IoT GNSS Shield can relay geo-tagged data from local sensors located worldwide to provide real-time insight for increasing your business revenues.
- **Fleet Management:** Manage your Mobility as a Service (MaaS) solution across the city or between borders. Track, analyze, and dynamically manage your fleet to optimize fuel usage, increase customer satisfaction, and reduce transport times. Enable predictive maintenance and remote diagnostics to ensure your business runs smoothly with minimal downtime.

11. First Step with Portenta Cat. M1/NB IoT GNSS Shield

11.1. Introduction

In this chapter, we will be looking how to use GSM networks to connect to a server and display it's content in the serial monitor. [Ard24q]

11.1.1. Configuration

First, we need to install Arduino Mbed OS Portenta core in the IDE platform and connect the Arduino Portenta Cat. M1/NB IoT GNSS shield to the Arduino Portenta H7 using HD connectors. Then we need to connect with the GSM network using NB IoT or Cat. M1 technology and print the website's HTML content in the Serial Monitor.

11.1.2. Requirements

- Arduino IDE (online or offline)
- Portenta H7
- Portenta Cat. M1/NB IoT GNSS Shield
- Dipole Antenna (connected to the RF OUT antenna connector on the top side of the shield)
- SIM card (standard pre-paid or post-paid SIM card that supports Cat M1 or NB-IoT connectivity, along with details of APN settings and PIN code usually 0000 or 1234. These are usually provided by your SIM card provider)

11.1.3. Procedures

First insert the SIM card into the SIM card slot at the bottom of the Portenta Cat. M1/NB IoT GNSS Shield [Ard24q]. The figure 11.1 shows where to connect the dipole antenna to the shield. It should connect to the antenna connector marked RF OUT.

Now connect the shield to the Portenta H7. Do this by attaching it to

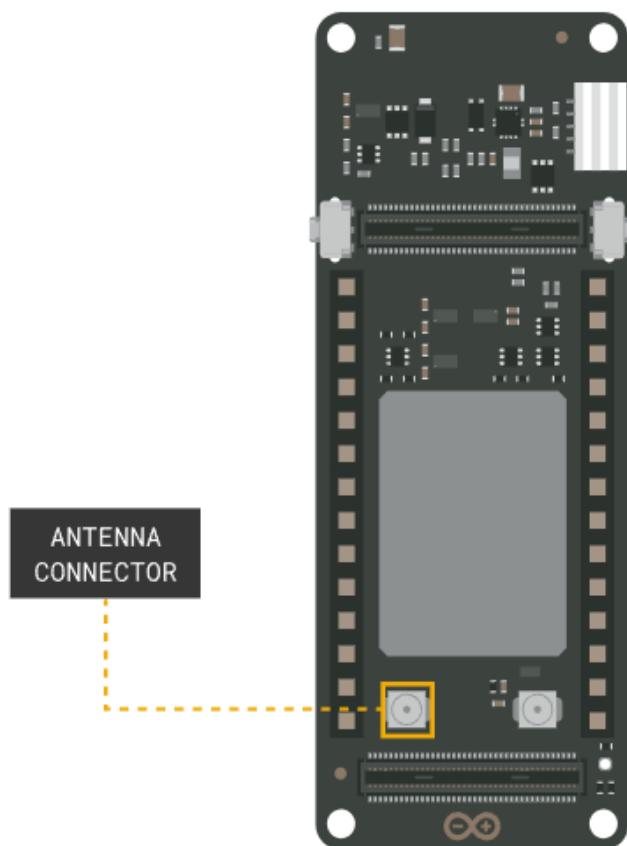


Figure 11.1.: Arduino Mbed OS Portenta core installation in Arduino IDE

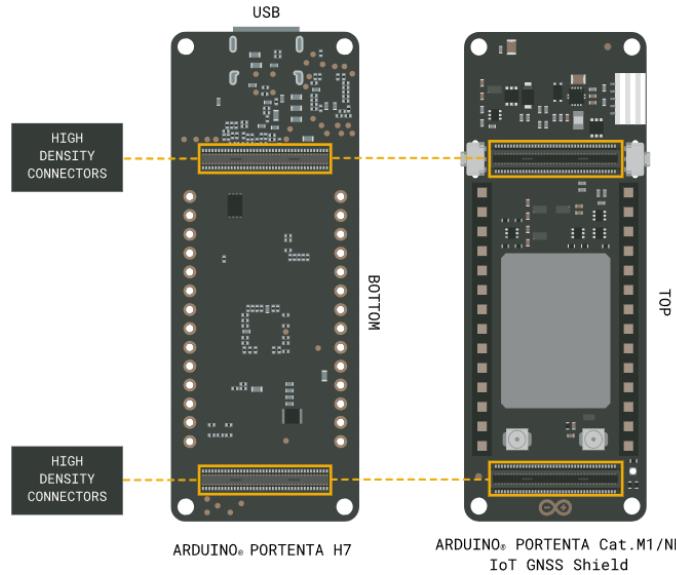


Figure 11.2.: Arduino Mbed OS Portenta core installation in Arduino IDE

the HD connectors at the bottom of the Portenta H7 board. The figure 11.2 shows the top and bottom high density connectors on the shield are connected to the corresponding ones on the lower side of the H7 board. Press firmly to let it snap in. Once attached, plug the Portenta H7 into your computer using a USB-C cable.

Arduino IDE

Make sure that the latest Arduino Mbed OS Portenta core installed. We can install it with the board manager under **Tools > Board > Board Manager** as shown in figure 11.4 With the core installed and the board selected, navigate to **File > Examples > GSM > GSMClient**. The listing reference 11.3 shows the example sketch we will use to test out the Cat. M1 or NB IoT connection. [Ard24q]

Programming the Board

Go to the `arduino_secrets.h` tab that opens with the example and enter the PIN of the SIM card you are using into the `SECRET_PIN` variable. Check your SIM card provider's mobile APN (e.g., "online.provider.com") and save it inside `SECRET_APN`.

NOTE: An Access Point Name (APN) is a gateway between a GSM, GPRS, 3G or 4G mobile network and another computer network, frequently the public Internet. The username and password depend on your provider; these are required to authenticate with the APN gateway. These

values can usually be found by searching online for APN credentials and provider names. Sometimes they can be left blank.

Using `GSM.begin()` will make the GSM module start with the PIN and APN entered in `arduino_secrets.h`. To decide whether to use NB IoT or Cat. M1 technology, you need to change the last argument. Use either `CATNB` for NB IoT or `CATM1` for Cat. M1. If you are unsure what technology to use, check with your SIM card provider. Using `GSM.begin()` will make the GSM module start with the PIN and APN entered in `arduino_secrets.h`. To decide whether to use NB IoT or Cat. M1 technology, you need to change the last argument. Use either `CATNB` for NB IoT or `CATM1` for Cat. M1. If you are unsure what technology to use, check with your SIM card provider.

```
GSM.begin(pin, apn, username, pass, CATNB);
```

When the GSM module starts, you can connect to a remote server using the server URL. The port to connect to is also defined; it is set to `80` by default.

```
client.connect(server, port);
```

11.1.4. Result of Sketch

After finishing this setup, compile and upload the program. You should see the HTML content of the server printed in the Serial Monitor. The server is set as `doc.arduino.cc` as default. [Ard24q]

```

1  /*
2   * GSMClient
3   *
4   * This sketch connects to a website (https://docs.arduino.cc/)
5   * using the Portenta CAT.M1/NB IoT GNSS Shield.
6   *
7   */
8
9 #include <GSM.h>
10 #include <Arduino_DebugUtils.h>
11 #include <GSMDDebug.h>
12 #include "arduino_secrets.h"
13
14 char pin[]      = SECRET_PIN;
15 char apn[]       = SECRET_APN;
16 char username[] = SECRET_USERNAME;
17 char pass[]      = SECRET_PASSWORD;
18
19 const char server[] = "doc.arduino.cc";
20 const char* ip_address;
21 int port = 80;
22 GSMClient client;
23
24 void setup() {
25   Serial.begin(9600);
26   while(!Serial) {}
27
28 #if defined(ARDUINO_EDGE_CONTROL)
29   // Power ON MKR2
30   pinMode(ON_MKR2, OUTPUT);
31   digitalWrite(ON_MKR2, HIGH);
32 #endif
33   // To enable AT Trace debug uncomment the following
34   // lines
35   //GSM.trace(Serial);
36   //GSM.setTraceLevel(4);
37
38 Debug.setDebugOutputStream(&Serial);
39 Debug.setDebugLevel(4);
40
41 Serial.println("Starting Carrier Network registration");
42 if(!GSM.begin(pin, apn, username, pass, CATNB, BAND_20 |
43   BAND_19)){
44   Serial.println("The board was not able to register to
45     the network...");  

46   // do nothing forevermore:  

47   while(1);
48 }
49 Serial.println("\nStarting connection to server...");  

50 // if you get a connection, report back via serial:  


```

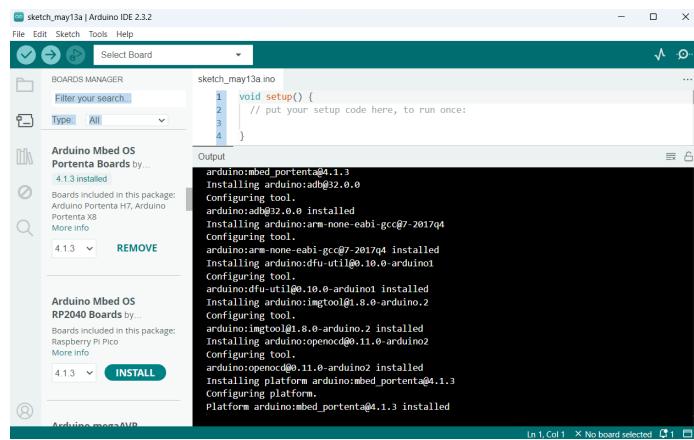


Figure 11.4.: Arduino Mbed OS Portenta core installation in Arduino IDE

Part IV.

Arduino PortentaH7 - Onboard Sensors

12. Power LED

12.1. Description

The Arduino Portenta H7 features a Power LED, an essential component designed to provide a visual indication of the board's power status. This LED is located on the board and serves the following purposes: [Doc25e]

- **Power Status Indicator:** The primary function of the Power LED is to indicate whether the board is receiving power. When the Arduino Portenta H7 is connected to a power source, such as via a USB Type-C cable or a connected battery, the Power LED lights up. This immediate visual feedback confirms that the board is powered on and operational.
- **Troubleshooting Aid:** The Power LED is a valuable tool for troubleshooting. If the board does not appear to be functioning correctly and the Power LED is off, this indicates that the board is not receiving power, prompting the user to check the power connections and source.
- **Continuous Operation:** The Power LED remains illuminated as long as the board is powered. This continuous indication helps users quickly verify that the board is in a powered state during development, testing, and deployment.

Key Features of the Power LED:

- **Visibility:** The Power LED is positioned for easy visibility, ensuring that users can quickly ascertain the power status without needing to connect additional peripherals or interfaces. [Doc25e]
- **Simplicity:** The Power LED provides a straightforward, unambiguous indication of the board's power state, simplifying the user experience and aiding in efficient board management.

The inclusion of the Power LED on the Arduino Portenta H7 enhances its usability, providing clear and immediate feedback on the power status, which is crucial for effective development and troubleshooting.

12.2. Specific Sensor

While labelled as a power LED, the single green LED on the PortentaH7 isn't solely for power indication. It has multi-functionality depending on board state and user code interaction. It lights steadily green when powered during typical operation, similar to other Arduino models. However, it flashes rapidly during serial communication and bootloader mode. Notably, when the user code enters deep sleep mode, the LED turns off entirely for power saving. This includes power status, communication activity, and sleep mode activation. Understanding these LED behaviours can aid in troubleshooting and code debugging. The green power LED's primary function indicates power and basic board states. [Doc25e]

12.3. Specification

- The power LED on the Arduino Portenta H7 is a green LED.
- It is connected to pin 25 on the board.
- The brightness of the power LED can be adjusted using **Pulse Width Modulation (PWM)** techniques.
- The power LED operates in an active-high configuration, meaning it turns on when the pin is set to a high voltage level.
- Users can control the power LED programmatically by setting the pin connected to it (pin 25) to either **HIGH** or **LOW**.
- In order to use the power LED, the pin (pin 25) must be defined as an output in the function **setup** using **pinMode(LED_PWR, OUTPUT)**. Otherwise, the LED will not function properly.
- Additionally, pin 25 can be utilized for other purposes. In such cases, the power LED will only turn on when the board is connected to a power source.

12.4. Simple Code

In the below sketch 12.1, a variable is connected to pin 25. The pin 25 is defined as an output in the function **setup**. In the function **loop**, the LED is switched on for 1 second and switched off for 1 second so that the LED flashes accordingly.

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

```
1 // How to control the power LED of the Arduino Nano 33
2 // BLE Sense //
3 // file : TestLEDPower.ino //
4 // The LED is switched on for 1 second and switched off
5 // for 1 second so that the LED flashes accordingly .
6
7 #define LED_PWR 25 // Define the pin for the power LED
8
9 void setup () {
10     // Initialize the pin as an output
11     pinMode(LED_PWR, OUTPUT) ;
12 }
13
14 void loop () {
15     // Turn the LED on
16     digitalWrite (LED_PWR, HIGH) ;
17     // Wait for one second
18     delay (1000) ;
19     // Turn the LED off
20     digitalWrite (LED_PWR, LOW) ;
21     // Wait for one second
22     delay (1000) ;
23 }
```

..../Code/Arduino/LED/TestLEDPower/TestLEDPower.ino

Listing 12.1.: Simple sketch to check the battery state using the power LED.

12.5. Test

12.5.1. Simple Function Test

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

12.5.2. Test all Functions

The brightness of the power LED can be controlled. This is demonstrated in the example sketch 10.3. Using the pulse width modulation, the brightness is gradually increased to the maximum value and then gradually reduced to 0 again.

12.6. Simple Application

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

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

A simple application is to check the condition of the battery. The sketch in Listing demonstrates that if the voltage drops too low, the power LED flashes.

```
1 // How to control the power LED of the Arduino PortentaH7
2 // File: TestLEDPowerBrightness.ino
3 // Using PWM, the LED's brightness is increasing until
4 // full and reverse.
5
6 // Define the initial brightness values (0-255)
7 int Brightness = 0;
8 // Define the increment/decrement value
9 int Step = 5;
10
11
12 void setup() {
13     // Initialize the pin as an output
14     pinMode(LED_PWR, OUTPUT);
15 }
16
17 void loop() {
18     // Write the PWM values to the LED pin
19     analogWrite(LED_PWR, Brightness);
20     // Update the brightness values
21     Brightness += Step;
22
23     // Check if the brightness values are out of range and
24     // reverse the direction
25     if (Brightness <= 0 || Brightness >= 255) {
26         Step = -Step;
27     }
28
29     // Wait for 10 milliseconds
30     delay(10);
31 }
```

..../Code/Arduino/LED/TestLEDPowerBrightness/TestLEDPower-Brightness.ino

Listing 12.2.: Simple sketch to check the battery state using the power LED

```
1 // Test of the Power LED
2 //
3 // File: TestLEDPowerBattery.ino
4 //
5 // Check the battery state.
6
7 #include <Arduino.h>
8
9 #define LED_PWR 25 // Define the pin for the power LED
10
11 // Battery measurement pin
12 const int BATTERY_PIN = A0;
13 // Reference voltage for 3.3V (adjust if using external
14 // power)
14 const float REFERENCE_VOLTAGE = 3.3;
15
16 const int LED_PWR = 25;
17
18 void setup() {
19     // Set LED pin as output
20     pinMode(LED_PWR, OUTPUT);
21     // Initialize serial communication for debugging
22     Serial.begin(9600);
23 }
24
25 // Input:
26 // SendMessages false - no messages
27 // true - with messages
28 // Return: 0 - okay // 1 - < 20%
29 int BatteryState(bool SendMessages) {
30     int ret;
31
32     // Read battery voltage from analog pin
33     float rawVoltage = analogRead(BATTERY_PIN) * (
34         REFERENCE_VOLTAGE / 1023.0);
35     // Calculate percentage based on reference voltage (
36     // adjust based on battery specs)
37     float batteryPercentage = (rawVoltage / 4.2) * 100.0;
38
39     if (SendMessages) {
40         // Print battery voltage and percentage (for
41         // debugging)
42         Serial.print("Battery voltage: ");
43         Serial.print(rawVoltage);
44         Serial.println(" V");
45         Serial.print("Battery percentage: ");
46         Serial.print(batteryPercentage);
47         Serial.println("%");
48     }
49
50     // Optional: blink LED based on battery level (adjust
51     // thresholds)
52     if (batteryPercentage < 20) {
```

13. Sensor Inertial Mesure Unit

Inertial Measurement Units (IMU) are an essential component of many navigation and control systems. These devices integrate several sensors, including accelerometers and gyroscopes, to measure and track an object's movements in three-dimensional space. IMUs are widely used in a variety of applications, such as autonomous vehicle navigation, virtual reality, image stabilization, and even in wearable devices for tracking physical activity. Their ability to provide precise data on linear and angular acceleration makes them indispensable tools for systems requiring precise control and orientation, and ongoing technological advances have reduced their size, power consumption and cost, widening their scope of application in many fields. [STM25a] [Dev24]

13.1. General Description

The Inertial Measurement Unit is used to mesured acceleration or rotation, this is an electronic device which used three types of sensor magnetometer, accelerometer and gyroscope. These three sensors is used in LSM9DS1 and they give velocity, orientation and gravitational force.

The magnetometer measures the strength of magnetic fields, providing various types of information. For instance, when you pass a current through an electromagnet or generate a magnetic field in another way, the magnetometer can detect and quantify the strength of that field. [STM24c] [STM24b]

13.1.1. Always On Mode

This feature implies that the LSM9DS1 can remain operational continuously, even when a device is in sleep mode, without excessively draining power. Such functionality proves invaluable for applications that require uninterrupted motion tracking, like fitness monitoring or navigation systems. [STM24c]

The power consumption is an important factor when we have battery-powered device like smartphone. These devices need to be lightweight and portable, which requires minimizing battery weight and maximizing battery life.

The LSM9DS1 consumes only 0.55 mA when operating in combo high-performance mode. This low power consumption allows the sensor to deliver exceptional data quality while efficiently using battery power. In this mode, the sensor can simultaneously measure acceleration and angular rate, providing precise and reliable data even at high sampling rates.

13.1.2. Tilt detection

These features imply that tilt detection relies on the accelerometer to detect movement. When movement occurs, the tilt detection feature identifies it. Additionally, the tilt function operates with ultra-low power consumption. [STM24c]

For example, if you have your smartphone in your front pant pocket, the tilt detection feature can determine whether you are moving from a standing to a sitting position or from sitting to standing.

13.1.3. Significant Motion Detection

This feature indicates that the Significant Motion Detection in the LSM9DS1 uses only the accelerometer to detect large movements.[STM24c]

13.2. Specific Sensor

13.2.1. LSM9DS1

The LSM9DS1 sensor, engineered by STMicroelectronics, is a highly advanced 3-axis Inertial Measurement Unit (IMU) that incorporates a high-performance 3-axis digital accelerometer and 3-axis digital gyroscope. This meticulously crafted device is designed for precise motion tracking and finds extensive use in wearable devices and smartphones. With its capability to concurrently measure both linear and rotational motion, it stands as a benchmark in motion-sensing technology. [STM24b]

Developed to detect various types of motion, the LSM9DS1 sensor offers features such as stationary/motion detection, tilt sensing, pedometer functions, and timestamping. Additionally, it supports the integration of an external magnetometer, expanding its functionality further. The sensor provides hardware flexibility, allowing for different pin connections to external sensors. This flexibility enables the expansion of functionalities, such as adding a sensor hub or auxiliary SPI connections, to meet diverse application requirements.

13.2.2. Accelerometer

The STMicroelectronics LSM9DS1 accelerometer is an advanced sensor crafted to measure linear acceleration along the x, y, and z axes. Its wide measurement range, spanning from $\pm 2\text{g}$ to $\pm 16\text{g}$, ensures versatility across various applications. Boasting exceptional precision and reliability, it stands as a go-to choice for engineers and designers seeking accurate motion tracking solutions. [STM24b]

Moreover, its low power consumption, typically ranging from 0.55 mA to 1.25 mA, makes it ideal for battery-powered devices. This feature ensures prolonged operation without draining excessive power, enhancing the efficiency and longevity of battery-powered systems.

With an operating voltage range of 1.71 V to 3.6 V, the LSM9DS1 accelerometer easily adapts to different system configurations, offering flexibility in integration. Whether employed in wearable devices for motion tracking, drones for stabilization, or structural monitoring for vibration analysis, the LSM9DS1 accelerometer consistently delivers outstanding performance.

Compact yet robust, the LSM9DS1 accelerometer is engineered to withstand demanding environments while providing precise and reliable measurements. Its reputation as a preferred choice among engineers and designers underscores its ability to meet the stringent requirements of diverse projects.

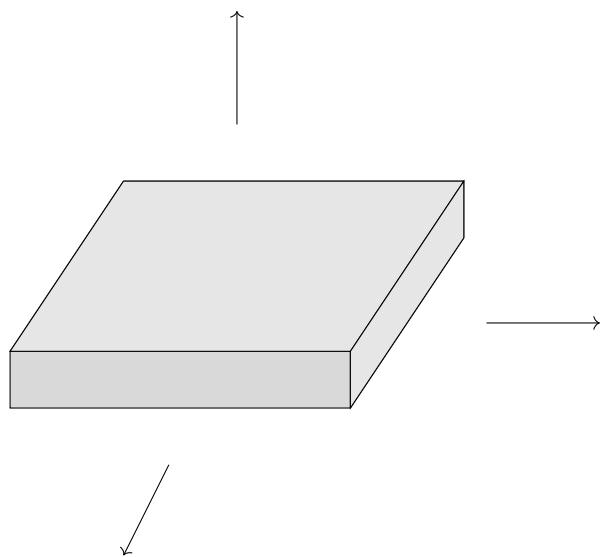


Figure 13.1.: LSM9DS1 axis on this card

13.2.3. Gyroscope

The STMicroelectronics LSM9DS1 gyroscope is a sophisticated measuring instrument designed to assess rotation rates around the x, y, and z axes. With a typical measurement range of $\pm 245/500/2000$ dps, it offers great flexibility to adapt to the specific needs of various applications. [STM24b]

Operating within a voltage range of 1.71 V to 3.6 V, this gyroscope can seamlessly integrate into different system configurations. Furthermore, its typical operating current ranges from 1 mA to 2 mA, ensuring optimal energy efficiency for battery-powered devices.

Whether it's for inertial navigation, drone stabilization, or other applications requiring precise motion measurement, the LSM9DS1 gyroscope delivers reliable and accurate performance in a compact and robust form factor, meeting the highest requirements of engineers and designers.

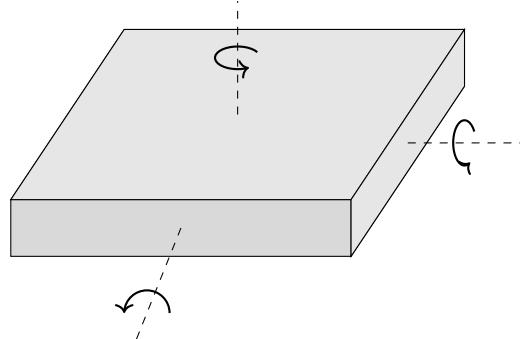


Figure 13.2.: LSM9DS1 axis on this card

13.3. Specification

13.3.1. Accelerometer Sensor

The accelerometer is composed by sensor type MEMS (Micro-Electro-Mechanical Systems) and they used microscopic structures like suspended beams or springs, which deform in response to acceleration. The deformation of these structures is measured using displacement sensors or changes in electrical resistance. MEMS accelerometers are commonly used due to their small size, low power consumption, and comparatively lower cost compared to other types of accelerometers. [Pie24]

The Accelerometer sensor function by [Ele24] :

- Acceleration Integration for Velocity Estimation: To estimate velocity, the acceleration measured by the accelerometer is integrated over time. Integrating acceleration over time yields velocity.
- Error Compensation: However, it's crucial to note that acceleration integration is susceptible to cumulative errors. Errors stemming from sensor noise, drift, and inaccuracies can accumulate over time, leading to inaccurate or biased velocity estimates.
- Utilization of Filters and Sensor Fusion Techniques: To compensate for these errors, advanced techniques such as Kalman filters, Complementary filters, or sensor fusion techniques can be employed. These techniques combine accelerometer data with other sensors such as gyroscopes to enhance velocity estimation accuracy.
- Calibration and Adjustment: It's also vital to calibrate and adjust the IMU to correct systematic errors and ensure precise measurements. This may involve steps such as compensating for Earth's gravity, correcting gyroscope drift, and reducing sensor noise.

13.3.2. PIN Connection

They are 4 ways to connect pin, ways depend of what they have after [STM24a].

- Mode 1: You can utilize either the I²C / MIPI I3CSM slave interface or the SPI (3- and 4-wire) serial interface.
- Mode 2: This mode supports both the I²C / MIPI I3CSM slave interface or the SPI (3- and 4-wire) serial interface, along with an I²C interface master for external sensor connections.
- Mode 3: Here, you have the choice of employing either the I²C / MIPI I3CSM slave interface or the SPI (3- and 4-wire) serial interface for the application processor interface. Additionally, an auxiliary SPI (3- and 4-wire) serial interface is designated for external sensor connections, exclusively for the gyroscope.
- Mode 4: Similar to Mode 3, this mode offers the I²C / MIPI I3CSM slave interface or the SPI (3- and 4-wire) serial interface for the application processor interface. Additionally, it provides an auxiliary SPI (3- and 4-wire) serial interface for external sensor connections, catering to both the accelerometer and gyroscope.

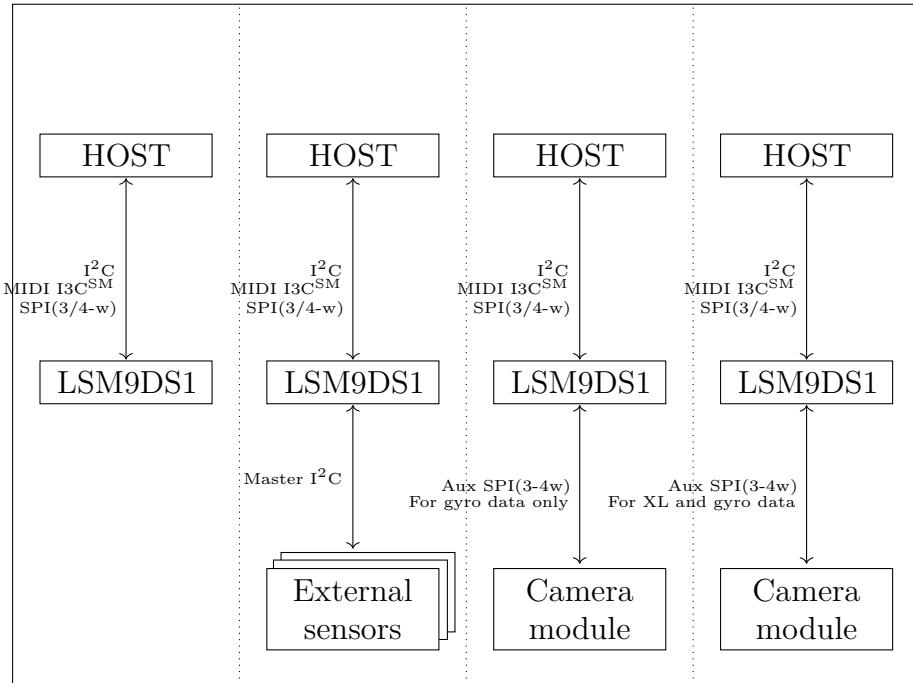


Figure 13.3.: Explication of pin mode

13.4. Libraries

13.4.1. Library Description

1. [Wire.h](#) : This library is a standard Arduino library that facilitates communication between I²C (Inter-Integrated Circuit) devices. I²C is a synchronous serial communication protocol used to connect microcontrollers to external devices. [Wire.h](#) provides functions for initializing the I²C bus, sending and receiving data on the bus, and managing multiple devices on the same bus. With this library, developers can easily connect multiple I²C devices, such as sensors or displays, to an Arduino board. [Ard24o]
2. [Kalman.h](#) : It's a library that implements the Kalman filter algorithm. The Kalman filter is a mathematical technique used to estimate the state of a system from incomplete measurements. It is commonly used in control systems, robotics and navigation applications to improve the accuracy of sensor measurements and reduce errors. [Kalman.h](#) provides a simple interface for developers to implement the Kalman filter in their Arduino projects. [Ard24k]
3. [Arduino LSM9DS1.h](#) : The LSM9DS1 library is a software library designed to facilitate interaction with the LSM9DS1 inertial sen-

sor, developed by STMicroelectronics. This sensor combines an accelerometer, a gyroscope and a magnetometer in a single package. The LSM9DS1 library provides an interface for accessing the sensor's functionalities, such as reading acceleration, angular velocity and magnetic field data. It can be used to configure various sensor parameters, such as measurement scales, sampling rates and operating modes. [Ard24e]

13.4.2. Installation

The bibliographie give coding example for used it, we need to download Arduino Library. The library [Arduino LSM9DS1.h](#) and [Kalman.h](#) can be download directly on Arduino. The library [wire.h](#) is used if you imput : include <wire.h>

```
1 #include <Wire.h>
2
```

Figure 13.4.: Wire include

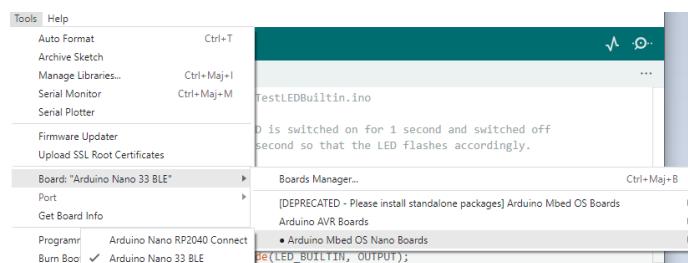


Figure 13.5.: Board card installation

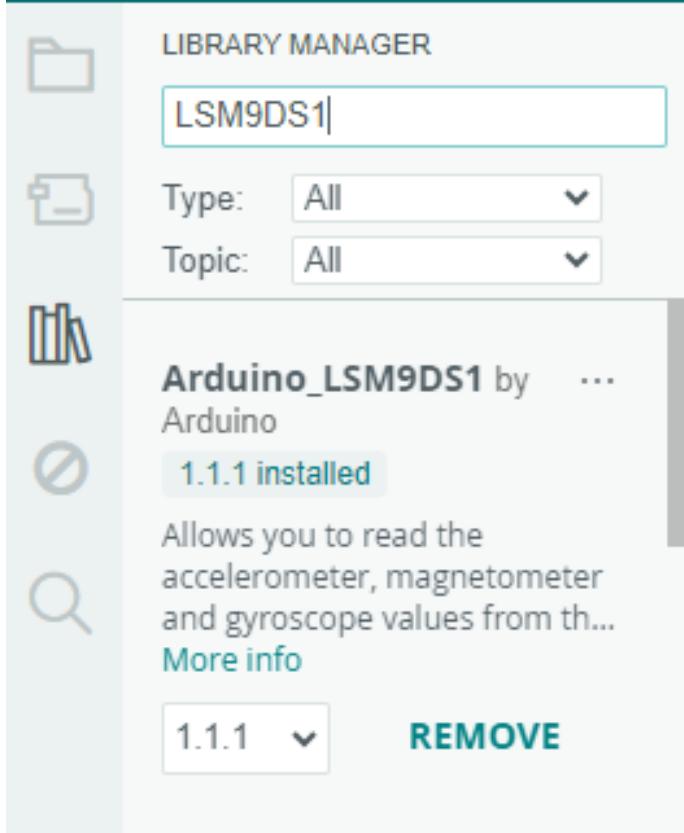


Figure 13.6.: Library choice

13.5. Function

The LSM9DS1 library on Arduino PortentaH7 is composed of different function. The function is different if you was on I²C protocol or SPI protocol.

13.5.1. Library `Wire.h`

This library is used to develop communication between different components of arduino, it's used for I²C. `Wire.h` is composed by [Ard24o] :

- `Wire.begin()`: Initializes the Wire library and prepares the Arduino to act as master or slave on the bus I²C.
- `Wire.beginTransmission(address)`: Starts a transmission to a slave device with the specified address.
- `Wire.write(data)` : Sends data on the I²C bus during transmission.

- `Wire.endTransmission()` : Ends the current transmission and releases the I2C bus.
- `Wire.requestFrom(address, quantity)` : Requests data from a slave device with the specified address.
- `Wire.available()`: Checks whether data is available to be read after a request.
- `Wire.read()`: Reads data received from a slave device.

13.5.2. Function `IMU.begin()`

Calling the function triggers `IMU.begin()`, the IMU initialization process. This step is important to configures the IMU's basic parameters, establishes communications with other devices, and prepares the unit for proper operation. Accurate initialization is essential to ensure the accuracy and reliability of the data provided by the IMU. [Ard24e] Incorrect initialization can lead to inaccurate measurements or unpredictable behavior. For example, incorrectly configured measurement parameters or sampling rates can compromise data quality, affecting overall system performance.

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

13.5.3. Function `IMU.end()`

No parameters is required. A return value of 1 indicates that all went well, while a return value of 0 indicates a problem. [Ard24e]

This function is used to stop or deactivate the IMU once it is no longer required. It frees up the resources used by the IMU and ends its operation cleanly. When called, it ensures that all tasks associated with the IMU are properly finalized, avoiding memory leaks or other problems associated with incomplete de-initialization.

This step helps to ensure the stability and reliability of the system as a whole, by avoiding problems associated with incorrect IMU de-initialization.

The data was return at the end.

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

```

1 #include <Wire.h>
2
3 #define LSM9DS1_M_ADDR 0x1E // I2C Address of the LSM9DS1
4   magnetometer module
5 #define LSM9DS1_AG_ADDR 0x6B // I2C Address of the
6   LSM9DS1 accelerometer-gyroscope module
7
8 void setup() {
9   Serial.begin(9600); // Initialize serial
10  communication
11  Wire.begin(); // Initialize Wire library
12  // Initialize LSM9DS1
13  initLSM9DS1();
14 }
15
16 void loop() {
17  // Read LSM9DS1 data
18  readLSM9DS1();
19  delay(1000); // Pause for one second between readings
20 }
21
22 void initLSM9DS1() {
23  // Initialize magnetometer module
24  Wire.beginTransmission(LSM9DS1_M_ADDR);
25  Wire.write(0x20); // Control register address for
26    mode
27  Wire.write(0x1C); // Activate continuous mode at 10
28    Hz
29  Wire.endTransmission();
30
31  // Initialize accelerometer-gyroscope module
32  Wire.beginTransmission(LSM9DS1_AG_ADDR);
33  Wire.write(0x10); // Control register address for
34    gyro mode
35  Wire.write(0x38); // Activate continuous m

```

..../Code/Arduino/IMU/WireEx.ino

Listing 13.1.: Simple sketch using the sensor LSM9DS1 detection.

13.5.4. Function **IMU.readAcceleration(x,y,z)**

IMU accelerometer to retrieve acceleration data, expressed in gravity (g). This function provides information on the movements detected by the IMU's accelerometer. The resulting acceleration data can be used for a variety of applications, such as shock or motion detection, or inertial navigation. By regularly interrogating the accelerometer and analyzing variations in acceleration, it is possible to understand and react to changes in motion with precision, which is essential in many modern embedded systems and electronic devices. [Ada24]

The parameters x, y and z are floats giving information on the x, y or z axis

```
1 float x, y, z;
2
3 if (IMU.accelerationAvailable()) {
4     IMU.readAcceleration(x, y, z);
5
6     Serial.print(x);
7     Serial.print('\t');
8     Serial.print(y);
9     Serial.print('\t');
10    Serial.println(z); }
```

13.5.5. Function **IMU.readGyroscope()**

Retrieves data from the IMU's gyroscope and returns angular velocity in degrees per second (dps). This function provides information on the rotational movement detected by the IMU's gyroscope. The angular velocity data retrieved can be used in various applications such as robotics, motion tracking or navigation systems. By regularly interrogating the gyroscope and analyzing angular velocity variations, it becomes possible to understand and respond precisely to rotational movements, which is crucial in applications requiring precise orientation control or stabilization. [Ada24]

The parameters x, y and z are floats giving information on the x, y or z axis

```
1 float x, y, z;
```

```

2
3  if (IMU.gyroscopeAvailable()) {
4      IMU.readGyroscope(x, y, z);
5
6      Serial.print(x);
7      Serial.print('\t');
8      Serial.print(y);
9      Serial.print('\t');
10     Serial.println(z); }
```

13.5.6. Function **IMU.accelerationAvailable()**

This function allows you to check the status of IMU acceleration data, indicating whether or not new data is ready to be retrieved.

In scenarios such as motion tracking, robotics or navigation systems, where the IMU continuously captures acceleration data, the availability of new data determines the system answers. Test the IMU at regular intervals is used for application and they can check for acceleration currently or non, ensuring that the system remains synchronized with object movements in real time. [Ada24]

When the function returns a value of 1, this means that new acceleration data is available, enabling the system to extract the data and process it further. A value of 0 indicates that there have been no recent acceleration updates.

```

1  float x, y, z;
2
3  if (IMU.accelerationAvailable()) {
4      IMU.readAcceleration(x, y, z);
5
6      Serial.print(x);
7      Serial.print('\t');
8      Serial.print(y);
9      Serial.print('\t');
10     Serial.println(z); }
```

13.5.7. Function **IMU.gyroscopeAvailable()**

This function ask if new gyro data are available for the IMU and they returns a value 0 if no new gyro data is available, and 1 if new gyro data is ready to be retrieved.

In applications requiring real-time monitoring of rotational motion, such as drone stabilization, virtual reality systems or inertial navigation, the system can determine whether to wait for updated gyroscope readings or proceed to process available data. [Ard24j]

A value of 1 indicates that new gyro data is available, enabling the system to retrieve and use this information for adapted the orientation tracking or motion control. Conversely, a feedback value of 0 indicates that there have been no recent updates to the gyroscope measurements, prompting the system to wait for new data to become available before proceeding.

```
1 float x, y, z;
2
3 if (IMU.gyroscopeAvailable()) {
4     IMU.readGyroscope(x, y, z);
5
6     Serial.print(x);
7     Serial.print('t');
8     Serial.print(y);
9     Serial.print(' t');
10    Serial.println(z); }
```

13.5.8. Function **IMU.accelerationSampleRate()**

This function is designed to provide information on the sampling rate of the IMU's accelerometer. The function **IMU.accelerationSampleRate()** returns the accelerometer sampling rate, expressed in Hertz (Hz). This value represents the frequency at which the accelerometer takes measurements and provides acceleration data. It indicates how many times per second the accelerometer registers changes in acceleration along its axes. [STM24b]

In activities such as motion tracking, gesture recognition or vibration analysis, knowledge of the accelerometer's sampling rate ensures that the system can accurately capture and respond to changes in motion dynamics.

```
1 Serial.print("Accelerometer sample rate = ");
2 Serial.print(IMU.accelerationSampleRate());
3 Serial.println(" Hz");
4 Serial.println();
5 Serial.println("Acceleration in g's");
6 Serial.println("X \t Y \t Z");
```

13.5.9. Function **IMU.gyroscopeSampleRate()**

To recover and communicate the specific rate at which the gyroscope, integrated into the inertial measurement unit (IMU), collects data samples. This frequency is usually expressed in hertz (Hz), corresponding to the number of samples acquired per second. This is the frequency at which the gyroscope takes measurements, giving an idea of its operational efficiency and performance. [STM24b]

```
1 Serial.print("Gyroscope sample rate = ");
2 Serial.print(IMU.gyroscopeSampleRate());
3 Serial.println(" Hz");
4 Serial.println();
5 Serial.println("Angular speed in degrees/second");
6 Serial.println("X tY tZ");
```

13.6. Calibration

13.6.1. Introduction

Calibrating an IMU (Inertial Measurement Unit) involves determining the bias, drift, and noise values of the sensors within the IMU. This is achieved by measuring the IMU's output in various positions and orientations, then employing algorithms to calculate the bias, drift, and noise values. This process is crucial to ensure the accuracy of the IMU's measurements. [Seg24]

13.6.2. Standard Operating Procedure

The standard operation procedure for calibrating LSM9DS1 IMU involves the following steps [Seg24] :

- Scope and Measurand(s) of Calibrations
- Description of the Item to be Calibrated
- Measurement Parameters, Quantities, and Ranges to be Determined
- Environmental Conditions and Stabilization Periods
- Procedure Include:
 - Handling, transporting, storing and preparation of items
 - Checks to be made before the work is started
 - Step by step process
- Handling, transporting, storing and preparation of items
- Checks to be made before the work is started
- Step by step process

The signal can be cleaned by different filters using different libraries such as Kalman.

13.6.3. Reset part

Resetting is particularly important for programming purposes or in the event of an external bug. There are 4 main types of programming:

- Soft reset: This method involves using code to trigger a microcontroller reset. For example, in Arduino, this can be done using the `reset()` function or by executing `asm volatile ("jmp 0")`. [Ins24]

- Hard reset : This method generally involves using an external component, such as a pushbutton or switch, to trigger a hard reset of the microcontroller. This can be achieved by connecting the pushbutton between the RESET pin and ground (GND). [For24]
- Auto Reset: On some Arduino boards, a reset is automatically triggered when a new USB connection is detected. This usually occurs when a new program is downloaded from the Arduino IDE. [Com25]
- Watchdog reset: Some microcontrollers have a built-in “watchdog timer”, which can be used to trigger a reset if the microcontroller remains stuck or unwanted behavior is detected. [You25]

13.7. Simple Application of code

13.7.1. Initialization example

In this section, we can see simple code test for sensor LSM9DS1 motion detection as shown in the listing 13.2.

13.7.2. Application example – gyroscope application

In this section, we can see simple code test for gyroscope application as shown in the listing 13.3.

13.8. Tests

13.8.1. Simple Test Function

Tests are essential to see the first bases of a code, and to have an answer when you do a part of the code.

If there's a part of the code where you're stuck in a loop, or something else, testing provides a clear answer to this kind of eventuality.

When we apply the test, we'll do it with 3 LEDs and test the accelerometer's data acquisition.

13.8.2. Acquisition Test

In this section, we can see simple test for motion acquisition sensor LSM9DS1 acquisition as shown in the listing 13.4.

```
1 #include <Wire.h>
2 #include <SPI.h>
3 #include <Adafruit_LSM9DS1.h>
4 #include <Adafruit_Sensor.h> // not used in this demo
5 // but required!
6
7 // i2c
8 Adafruit_LSM9DS1 lsm = Adafruit_LSM9DS1();
9
10 //define LSM9DS1_SCK A5
11 //define LSM9DS1_MISO 12
12 //define LSM9DS1_MOSI A4
13 //define LSM9DS1_XGCS 6
14 //define LSM9DS1_MCS 5
15 // You can also use software SPI
16 //Adafruit_LSM9DS1 lsm = Adafruit_LSM9DS1(LSM9DS1_SCK,
17 //                                         LSM9DS1_MISO, LSM9DS1_MOSI, LSM9DS1_XGCS, LSM9DS1_MCS
18 //                                         );
19 // Or hardware SPI! In this case, only CS pins are passed
20 // in
21 //Adafruit_LSM9DS1 lsm = Adafruit_LSM9DS1(LSM9DS1_XGCS,
22 //                                         LSM9DS1_MCS);
23
24
25 void setupSensor()
26 {
27     // 1.) Set the accelerometer range
28     lsm.setupAccel(lsm.LSM9DS1_ACCEL RANGE_2G, lsm.
29                     LSM9DS1_ACCELDATARATE_10HZ);
30     //lsm.setupAccel(lsm.LSM9DS1_ACCEL RANGE_4G, lsm.
31                     LSM9DS1_ACCELDATARATE_119HZ);
32     //lsm.setupAccel(lsm.LSM9DS1_ACCEL RANGE_8G, lsm.
33                     LSM9DS1_ACCELDATARATE_476HZ);
34     //lsm.setupAccel(lsm.LSM9DS1_ACCEL RANGE_16G, lsm.
35                     LSM9DS1_ACCELDATARATE_952HZ);
36
37     // 2.) Set the magnetometer sensitivity
38     lsm.setupMag(lsm.LSM9DS1_MAGGAIN_4GAUSS);
39     //lsm.setupMag(lsm.LSM9DS1_MAGGAIN_8GAUSS);
40     //lsm.setupMag(lsm.LSM9DS1_MAGGAIN_12GAUSS);
41     //lsm.setupMag(lsm.LSM9DS1_MAGGAIN_16GAUSS);
42
43     // 3.) Setup the gyroscope
44     lsm.setupGyro(lsm.LSM9DS1_GYROS SCALE_245DPS);
45     //lsm.setupGyro(lsm.LSM9DS1_GYROS SCALE_500DPS);
46     //lsm.setupGyro(lsm.LSM9DS1_GYROS SCALE_2000DPS);
47 }
48
49
50 void setup()
51 {
52     Serial.begin(115200);
53 }
```

```

1 #include <Wire.h>
2 #include <LSM9DS1.h>
3
4 #define LSM9DS1_SCK 13
5 #define LSM9DS1_MISO 12
6 #define LSM9DS1_MOSI 11
7 #define LSM9DS1_CS_AG 10
8 #define LSM9DS1_CS_M 9
9
10 LSM9DS1 imu;
11
12 void setup() {
13   Serial.begin(9600);
14   while (!Serial);
15
16   if (!imu.begin()) {
17     Serial.println("Failed to communicate with LSM9DS1.");
18     ;
19   }
20
21   imu.calibrate();
22 }
23
24 void loop() {
25   imu.readGyro();
26   float gyroZ = imu.calcGyro(imu.g.z);
27
28   float angle = gyroZ * 0.01; // ajustez ce facteur selon
29   vos besoins
30
31   if (angle > 2) {
32     Serial.println("Rotation vers la droite");
33     Serial.print("Angle: ");
34     Serial.println(angle);
35   } else if (angle < -2) {
36     Serial.println("Rotation vers la gauche");
37     Serial.print("Angle: ");
38     Serial.println(angle);
39   }
}

```

..../Code/Arduino/IMU/gyrorotation.ino

Listing 13.3.: Simple test for gyroscope application

```
1 #include <ArduinoBLE.h>
2 #include <Wire.h>
3 #include <Arduino_LSM9DS1.h>
4
5 // LED pins
6 const int ledPin1 = 13; // Up/Down
7 const int ledPin2 = 14; // Left/Right
8 const int ledPin3 = 15; // Connected and calibrated
9
10 void setup() {
11     // Initialize LED pins
12     pinMode(ledPin1, OUTPUT);
13     pinMode(ledPin2, OUTPUT);
14     pinMode(ledPin3, OUTPUT);
15
16     // Initialize I2C communication
17     Wire.begin();
18
19     // Initialize LSM9DS1 sensor
20     if (!imu.begin()) {
21         digitalWrite(ledPin3, HIGH); // Light up LED3 in red
22             if sensor initialization fails
23             while (1);
24     }
25
26     // LED indication: BLE initialized successfully
27     digitalWrite(ledPin3, HIGH); // Light up LED3 in red
28     delay(500);
29     digitalWrite(ledPin3, LOW); // Turn off LED3
30 }
31
32 void loop() {
33     // Read accelerometer data
34     imu.readAccel();
35
36     // Check the accelerometer data for movement direction
37     if (imu.ax > 1 || imu.ax < -1) {
38         // Up/Down
39         digitalWrite(ledPin1, HIGH); // Turn on LED1 for Up/
40             Down
41         digitalWrite(ledPin2, LOW); // Turn off LED2
42     } else if (imu.ay > 1 || imu.ay < -1) {
43         // Left/Right
44         digitalWrite(ledPin1, LOW); // Turn off LED1
45         digitalWrite(ledPin2, HIGH); // Turn on LED2 for Left/
46             Right
47     } else {
48         // No significant movement
49         digitalWrite(ledPin1, LOW); // Turn off LED1
50         digitalWrite(ledPin2, LOW); // Turn off LED2
51     }
52 }
```

13.9. Error

13.9.1. Type of Error

1. Bias error : Biases are constant values added to IMU sensor measurements. They can result from various factors, such as inaccuracies in electronic components or variations in the characteristics of the materials used in IMU construction. Biases can be different for each measurement axis (linear acceleration on the x, y and z axes, as well as angular velocity around these axes), and their presence can lead to systematic errors in IMU data. [Nov25]
2. Random walk: Random walk is a noise-type error that manifests itself as random fluctuations in IMU measurements. These fluctuations can be caused by external factors such as vibration or temperature variations. Random deviation can make it difficult to separate the useful signal from the noise in IMU data, particularly when used over long periods of time. [Vec25]
3. Gyroscopic drift: Gyroscopic drift occurs when IMU gyroscopes gradually accumulate orientation errors over time. These errors can be caused by temperature variations, mechanical vibrations or imperfections in the manufacture of gyroscopic components. Gyroscopic drift can lead to significant errors in IMU orientation estimation, particularly when used over long periods of time without recalibration. [Exc25]
4. Gravity compensation error: When the IMU measures linear acceleration, it must compensate for the gravitational component to obtain accurate measurements. Incorrect gravity compensation can result in orientation errors, particularly when the IMU is used in environments where the direction of gravity varies, such as on board moving vehicles. These errors can lead to inaccuracies in IMU orientation estimates. [Nav25]
5. Sensor synchronization error: If the IMU's sensors are not correctly synchronized, measurement errors may occur. For example, delays or time differences between acceleration and rotation measurements may occur if the sensors are not properly synchronized. These errors can affect the accuracy of IMU orientation and position estimates. [Nav25]
6. Calibration error: Incorrect calibration of IMU sensors can lead to measurement errors. Calibration is the critical process of adjusting sensor parameters to correct systematic errors such as bias and incorrect measurement scales. Inadequate calibration can compromise

the accuracy of IMU measurements and lead to incorrect estimates of position, orientation and velocity. [Res25]

These errors can be mitigated using filtering techniques such as Kalman filters, sensor fusion filters or advanced calibration methods. These methods make it possible to optimally combine information from the IMU's various sensors to obtain more accurate estimates of the position, orientation and velocity of the object to which the IMU is attached. [(PM25)]

13.9.2. Library `Kalman.h`

The `Kalman.h` library brings together functions essential to the implementation of the Kalman filter, guaranteeing precise filtering and accurate predictions. Each function within this library is designed to handle specific aspects of the filtering process. The library `Kalman.h` is an essential tool for signal processing and estimation tasks, enabling reliable and efficient filtering in a variety of applications. [Ard24k]

Function

- `KalmanFilter.init()`: This function is used to initialize the Kalman filter with necessary parameters, such as the initial state covariance matrix and the process noise covariance matrix, as well as the measurement noise covariance matrix.
- `KalmanFilter.predict()`: This function is used to predict the future state of the system using the Kalman filter prediction equations. It is typically called at each iteration of the filter, even when new measurements are not available.
- `KalmanFilter.iupdate()`: This function is used to update the state estimate based on the new available measurement. It utilizes the Kalman filter update equations to combine the predicted estimate with the new measurement.
- `KalmanFilter.setMeasurementNoise()`: This function allows setting the measurement noise covariance, which represents the uncertainty associated with the measurements provided by sensors.
- `KalmanFilter.setProcessNoise()`: This function allows setting the process noise covariance, which represents the uncertainty associated with the dynamics of the system.
- `KalmanFilter.setEstimateError()`: This function allows setting the initial state covariance, which represents the initial uncertainty of the state estimate.

- `KalmanFilter.getEstimation()`: This function allows retrieving the current estimate of the system state after the update.

Kalman filter example

```
1 #include <Kalman.h>
2
3 // Kalman filter example
4
5
6 void setup() {
7     // Initialize the Kalman filter with appropriate
8     // parameters
9     kalmanFilter.init(/* initialization parameters */);
10 }
11
12 void loop() {
13     // Get the current sensor measurement
14     double measurement = obtain_measurement();
15
16     // Predict the future state
17     kalmanFilter.predict();
18
19     // Update the state estimate based on the new
20     // measurement
21     kalmanFilter.update(measurement);
22
23     // Get the current state estimate
24     double estimation = kalmanFilter.getEstimation();
25
26     // Use the estimate to control other parts of the
27     // program
28     // or perform actions based on the estimated position
29     // of the object
30 }
```

..../Code/Arduino/IMU/KalmanFilter.ino

Listing 13.5.: Simple example with Kalman Filter.

14. Bluetooth

14.1. Description

Bluetooth sensors are devices that utilize Bluetooth technology to wirelessly transmit data over short distances. They are commonly used in various applications including health monitoring, fitness tracking, smart home automation, industrial monitoring, and asset tracking. [Doc24f]

Key Features:

- **Integrated BLE Module:** The Portenta H7 includes a built-in Bluetooth module that supports BLE, allowing for low-power, short-range wireless communication. It enables the board to interface with smartphones, tablets, and other BLE devices.
- **High Performance:** Equipped with a dual-core ARM Cortex-M7 and Cortex-M4 processor, the Portenta H7 provides robust processing power to handle complex tasks, including Bluetooth communications and data processing.
- **Easy Integration:** The Portenta H7 is compatible with the Arduino ecosystem, allowing for easy programming and integration with other Arduino shields and modules, including those utilizing Bluetooth communication. [Doc24f]

14.2. Specific Sensor

The BLE functionality in the Portenta H7 is provided by an integrated module that adheres to the Bluetooth 5.0 standard. This module supports features such as enhanced range, increased speed, and improved data throughput compared to previous Bluetooth versions.

The onboard Bluetooth module of the Portenta H7 offers low energy Bluetooth functionality, in order to provide the board with the flexibility to be easily connected to devices which also support Bluetooth Low Energy, such as the Arduino Nano 33 IoT or most modern smartphones. Compared to classic Bluetooth, Low Energy Bluetooth is intended to provide considerably reduced power consumption and cost while maintaining a similar communication range. [Doc24f]

14.3. Specification

- The BLE sensor on the Arduino Portenta H7 operates on the Bluetooth 5.0 standard.
- It is integrated into the board, eliminating the need for additional Bluetooth hardware.
- The BLE sensor supports various data transfer modes including Advertising, Scanning, and Connection.
- The module operates on a low power mode to ensure efficient energy usage.
- It provides a communication range of up to 100 meters in open space.
- The BLE sensor can be programmed using the Arduino IDE with appropriate libraries for BLE communication. [Doc24f]

14.4. TESTS

In the listing 14.1, the Bluetooth sensor is initialized and used to advertise a simple message. This basic setup demonstrates how to get started with BLE communication on the Portenta H7.

Listing 14.1.: Simple sketch to control the LED using bluetooth

```

1000  /*
1001   LED Control
1002
1003   This example scans for Bluetooth Low Energy peripherals
1004   until one with the advertised service
1005   "19b10000-e8f2-537e-4f6c-d104768a1214" UUID is found. Once
1006   discovered and connected,
1007   it will remotely control the Bluetooth Low Energy
1008   peripheral's LED, when the button is pressed or released.
1009
1010  The circuit:
1011  - Arduino MKR WiFi 1010, Arduino Uno WiFi Rev2 board,
1012    Arduino Nano 33 IoT,
1013    Arduino Nano 33 BLE, or Arduino Nano 33 BLE Sense board.
1014  - Button with pull-up resistor connected to pin 2.
1015
1016  You can use it with another board that is compatible with
1017  this library and the
1018  Peripherals -> LED example.
1019
1020  This example code is in the public domain.
1021 */

```

```
1018 #include <ArduinoBLE.h>
1020 // variables for button
1021 const int buttonPin = 2;
1022 int oldButtonState = LOW;
1023
1024 void setup() {
1025     Serial.begin(9600);
1026     while (!Serial);
1027
1028     // configure the button pin as input
1029     pinMode(buttonPin, INPUT);
1030
1031     // initialize the Bluetooth Low Energy hardware
1032     BLE.begin();
1033
1034     Serial.println("Bluetooth Low Energy Central - LED control");
1035 }
1036
1037 // start scanning for peripherals
1038 BLE.scanForUuid("19b10000-e8f2-537e-4f6c-d104768a1214");
1039
1040 void loop() {
1041     // check if a peripheral has been discovered
1042     BLEDevice peripheral = BLE.available();
1043
1044     if (peripheral) {
1045         // discovered a peripheral, print out address, local name
1046         //, and advertised service
1047         Serial.print("Found ");
1048         Serial.print(peripheral.address());
1049         Serial.print(" ");
1050         Serial.print(peripheral.localName());
1051         Serial.print(" ");
1052         Serial.print(peripheral.advertisedServiceUuid());
1053         Serial.println();
1054
1055         if (peripheral.localName() != "LED") {
1056             return;
1057         }
1058
1059         // stop scanning
1060         BLE.stopScan();
1061
1062         controlLed(peripheral);
1063
1064         // peripheral disconnected, start scanning again
1065         BLE.scanForUuid("19b10000-e8f2-537e-4f6c-d104768a1214");
1066     }
1067 }
```

```

1068 void controlLed(BLEDevice peripheral) {
1069     // connect to the peripheral
1070     Serial.println("Connecting ...");
1071
1072     if (peripheral.connect()) {
1073         Serial.println("Connected");
1074     } else {
1075         Serial.println("Failed to connect!");
1076         return;
1077     }
1078
1079     // discover peripheral attributes
1080     Serial.println("Discovering attributes ...");
1081     if (peripheral.discoverAttributes()) {
1082         Serial.println("Attributes discovered");
1083     } else {
1084         Serial.println("Attribute discovery failed!");
1085         peripheral.disconnect();
1086         return;
1087     }
1088
1089     // retrieve the LED characteristic
1090     BLECharacteristic ledCharacteristic = peripheral.
1091         characteristic("19b10001-e8f2-537e-4f6c-d104768a1214");
1092
1093     if (!ledCharacteristic) {
1094         Serial.println("Peripheral does not have LED
1095             characteristic!");
1096         peripheral.disconnect();
1097         return;
1098     } else if (!ledCharacteristic.canWrite()) {
1099         Serial.println("Peripheral does not have a writable LED
1100             characteristic!");
1101         peripheral.disconnect();
1102         return;
1103     }
1104
1105     while (peripheral.connected()) {
1106         // while the peripheral is connected
1107
1108         // read the button pin
1109         int buttonState = digitalRead(buttonPin);
1110
1111         if (oldButtonState != buttonState) {
1112             // button changed
1113             oldButtonState = buttonState;
1114
1115             if (buttonState) {
1116                 Serial.println("button pressed");
1117
1118                 // button is pressed, write 0x01 to turn the LED on
1119                 ledCharacteristic.writeValue((byte)0x01);
1120             } else {

```

```
1118     Serial.println("button released");

1120     // button is released, write 0x00 to turn the LED off
1121     ledCharacteristic.writeValue((byte)0x00);
1122 }
1123 }
1124 }

1125 Serial.println("Peripheral disconnected");
1126 }
```

..//Code/Bluetooth/LedControl/LedControl.ino

This is just a simple example. The necessary libraries such as <Arduino-BLE.h> should be included, and the BLE device name and services can be defined as per the application requirements. [Doc24f]

Example

14.4.1. control the built-in LED using Bluetooth

The onboard Bluetooth module of the Portenta H7 offers low energy Bluetooth functionality, in order to provide the board with the flexibility to be easily connected to devices which also support Bluetooth Low Energy, such as the Arduino Nano 33 IoT or most modern smartphones. Compared to classic Bluetooth, Low Energy Bluetooth is intended to provide considerably reduced power consumption and cost while maintaining a similar communication range. [Doc24f]

Overview

In this Example we will enable low energy Bluetooth on the Portenta H7 to allow an external Bluetooth device to control the built-in LED either by turning it on or off. [Doc24f]

Goals

- Enabling Bluetooth Low Energy connectivity on the Portenta H7.
- Connecting the Portenta to an external Bluetooth Low Energy Mobile Application (In this case nRF Connect by Nordic Semiconductor). [Doc24f]

Required Hardware and Software

1. Portenta H7 or Portenta H7 Lite Connected
2. USB-C cable (either USB-A to USB-C or USB-C to USB-C)
3. Arduino IDE 1.8.13+ or Arduino Pro IDE 0.0.4+

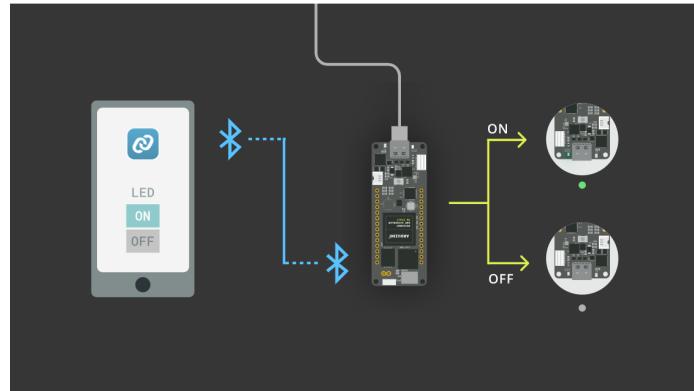


Figure 14.1.: Bluetooth-portentaH7
[Doc24f]

4. Mobile device, phone or tablet
5. nRFconnect or equivalent tool downloaded on your mobile device:
nRF Connect for iOS or nRF Connect for android [Doc24f]

Instructions

- **Configuring the Development Environment:** To communicate with the Portenta H7 via Bluetooth, you need to upload a pre-built sketch that starts a Bluetooth network and allows your mobile device, which will be used to control the LEDs, to connect to it refer figure 14.1. The sketch uses the ArduinoBLE Library that enables the Bluetooth Low Energy module and handles important functions, such as scanning, connecting and interacting with services provided by other devices. You will also be using a third party application (e.g. nRF Connect), running on your mobile device in order to connect your device to the board and help you control the built-in LED. [Doc24f]
- **1. The Basic Setup:** Begin by plugging in your Portenta board to the computer using a USB-C cable and open the Arduino IDE. If this is your first time running Arduino sketch files on the board, we suggest you check out how to set up the Portenta H7 for Arduino before you proceed. 14.2 [Doc24f]
- **2. Install the ArduinoBLE Library:** You will need to install the ArduinoBLE library in the Arduino IDE you are using. To install the library go to : **Tools > Manage Libraries...** type ArduinoBLE and click Install. Make sure you install ArduinoBLE version 1.1.3 or higher. 14.3 [Doc24f]



Figure 14.2.: PortentaH7-connection
[Doc24f]

```
James Gaultier 10/20/2023 2:32
File Edit Search Tools Help
Select Board
ARDUINO IDE
Arduino SLE
Type: All
Analog Mbed OS Name Boards
by Ardunio
Analog Mbed OS Name Boards is a package for Arduino Nano 33 BLE Sense, Nano 33 BLE, Arduino Nano 33 BLE Sense, and Arduino Particle M0 (Thread-based... More info
4.1.3 + INSTALL
[DEPRECATED] - Please install ... [DEPRECATED]
Smartphone package for Arduino
Boards included in this package: Arduino Nano 33 BLE Sense, Arduino Nano 33 BLE, Arduino Particle M0 (Thread-based... More info
3.3.0 REMOVE
Camera, Capture.hc
1 #include <camera.h>
2 #include <time.h>
3
4 #include <math.h>
5
6 #define THRESH_HUE 0.05 // grayscale
7 #define THRESH_SAT 0.05 // saturation
8 #define THRESH_VAL 0.05 // value
9
10 unsigned long lastPhotoTime = 0;
11
12 void setup() {
13   Serial.begin(500000);
14   //init the ram photo, serial
15   if (!begin(CAMERA_CS_PIN, D7, D8, D9, 24)) {
16     }
17 }
18
19 void loop() {
20   // put your main code here, to run repeatedly:
21   if (millis() - lastPhotoTime > 1000) {
22     Serial.begin(500000);
23     if (takePhoto()) {
24       lastPhotoTime = millis();
25     }
26   }
27   // Read until we receive a character
28   if (Serial.available() > 0) {
29     // Read the characters until the end of the line
30     if (Serial.read() == 'r') {
31       //latitudine = +millilitre;
32
33       // Grab Frame and write to serial
34       camera.setFrameSize(1, 1);
35       serial.write((char*)getFrame(1, cam_frameSize));
36     }
37   }
}
```

Figure 14.3.: Bluetooth-Library

- **3. Create the Bluetooth Low Energy Sketch:** Let's program the Portenta with the following example sketch. If the Bluetooth Low Energy module has been initialized correctly, you will see the blue LED lighting up for one second after uploading the sketch. If it fails, you will see the red LED lighting up instead. Copy and paste the following code into a new sketch in your IDE or download it from: [Examples > Arduino-Pro-Tutorials > BLE Connectivity on Portenta H7 > PortentaBLE](#) 15.1 [Doc24f]

Listing 14.2.: Simple sketch to control the LED using bluetooth

```

1000 #include <ArduinoBLE.h>

1002 BLEService ledService("19b10000-e8f2-537e-4f6c-
1003   d104768a1214");

1004 // Bluetooth Low Energy LED Switch Characteristic -
1005 // custom 128-bit UUID, read and writable by central
1006 BLEByteCharacteristic switchCharacteristic("19b10000-
1007   e8f2-537e-4f6c-d104768a1214", BLERead | BLEWrite);

1008 const int ledPin = LED_BUILTIN; // Pin to use for the
1009   LED

1010 void setup() {
1011   Serial.begin(9600);
1012   //while (!Serial); // Uncomment to wait for serial
1013     port to connect.

1014   // Set LED pin to output mode
1015   pinMode(ledPin, OUTPUT);
1016   digitalWrite(ledPin, HIGH);

1017   // Begin initialization
1018   if (!BLE.begin()) {
1019     Serial.println("Starting Bluetooth Low Energy failed
1020       !");
1021     digitalWrite(LED_R, LOW);
1022     delay(1000);
1023     digitalWrite(LED_R, HIGH);

1024     // Stop if Bluetooth Low Energy couldn't be
1025       initialized.
1026     while (1);
1027   }

1028   // Set advertised local name and service UUID:
1029   BLE.setLocalName("LED-Portenta-01");
1030   BLE.setAdvertisedService(ledService);

1031   // Add the characteristic to the service

```

```
1034     ledService.addCharacteristic(switchCharacteristic);  
1035  
1036 // Add service  
1037 BLE.addService(ledService);  
1038  
1039 // Set the initial value for the characteristic:  
1040 switchCharacteristic.writeValue(0);  
1041  
1042 // start advertising  
1043 BLE.advertise();  
1044 digitalWrite(LED_BUILTIN, LOW);  
1045 delay(1000);  
1046 digitalWrite(LED_BUILTIN, HIGH);  
1047 Serial.println("BLE LED Control ready");  
1048 }  
1049  
1050 void loop() {  
1051 // Listen for Bluetooth Low Energy peripherals to  
1052 // connect:  
1053 BLEDevice central = BLE.central();  
1054  
1055 // If a central is connected to peripheral:  
1056 if (central) {  
1057     Serial.print("Connected to central: ");  
1058     // Print the central's MAC address:  
1059     Serial.println(central.address());  
1060     digitalWrite(LED_BUILTIN, HIGH);  
1061     delay(100);  
1062     digitalWrite(LED_BUILTIN, LOW);  
1063     delay(100);  
1064     digitalWrite(LED_BUILTIN, HIGH);  
1065  
1066     // While the central is still connected to  
1067     // peripheral:  
1068     while (central.connected()) {  
1069         // If the remote device wrote to the  
1070         // characteristic,  
1071         // Use the value to control the LED:  
1072         if (switchCharacteristic.written()) {  
1073             if (switchCharacteristic.value()) { // Any  
1074                 value other than 0  
1075                 Serial.println("LED on");  
1076                 digitalWrite(ledPin, LOW); // Will  
1077                 turn the Portenta LED on  
1078             } else {  
1079                 Serial.println("LED off");  
1080                 digitalWrite(ledPin, HIGH); // Will  
1081                 turn the Portenta LED off  
1082             }  
1083         }  
1084     }  
1085  
1086     // When the central disconnects, print it out:  
1087 }
```

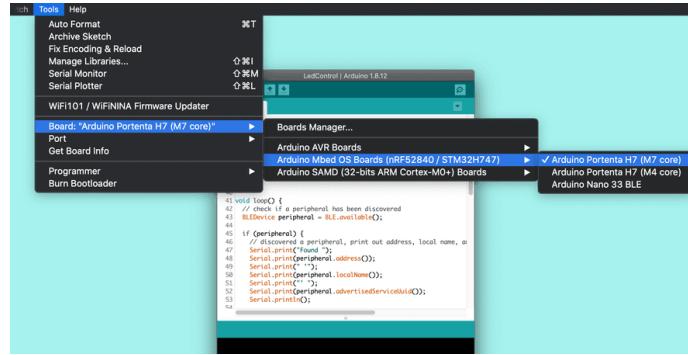


Figure 14.4.: select-board-h7
[Doc24f]

```

1080    Serial.print("Disconnected from central: ");
1082    Serial.println(central.address());
1083    digitalWrite(LEDB, HIGH);
1084    delay(100);
1085    digitalWrite(LEDB, LOW);
1086    delay(100);
1087    digitalWrite(LEDB, HIGH);
1088 }
```

..//Code/Bluetooth/LedControl/PortentaBLE.ino

In this example, you use a pre-defined Bluetooth number code pre-setup for controlling a device's LED. This code can also be referred to as GATT codes, which define how two Bluetooth low energy devices transfer data. Once a connection is established with a device, its respective GATT code, which is a 16 bit identifier, is stored in a lookup table for future reference.

These GATT codes are very long, but, in this example, it is always the same code: [Doc24f]

```

1 BLEService ledService("19b10000-e8f2-537e-4f6c-
d104768a1214"); // BLE LED Service
```

- **4. Upload the Sketch:** Double press the reset button so the built-in LED is slowly pulsing green. Then, select your board in the menu: **Tools > Board > Arduino Portenta H7 (M7 core)** 14.4

Choose the Port where your Portenta is connected to and Upload the sketch. Open the Serial Monitor once you have uploaded the code to the board to see debugging messages. If the Bluetooth

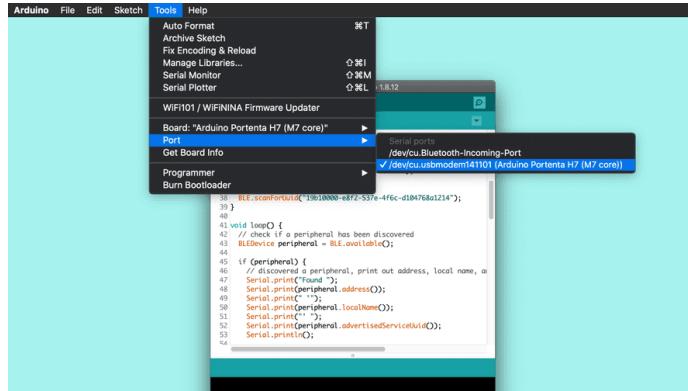


Figure 14.5.: Select the Port to which the Portenta is connected to
[Doc24f]

Low Energy setup was successful, you should see the message BLE LED Control ready. If something went wrong, you will see the message Starting Bluetooth Low Energy failed!. In that case update the Arduino BLE library (in the Library Manager) and the board (in the Board Manager) to the latest version and try again. 14.5 [Doc24f]

- **5. Connect an External Device :** On your mobile device install nRF Connect or an equivalent app that allows for Bluetooth Low Energy connections. We will refer to nRF Connect for the rest of this tutorial. 14.6

Once you have downloaded the nRF application on your mobile device, look for your Portenta in the device list. You may filter the list by "Portenta" to easierly find your board in case you are using nRF Connect. [Doc24f]

1. When you find your board in the list tap "Connect". refer figure 14.7
2. Navigate to the "Services" screen and tap the arrow up button.
3. Switch to "Bool" type and move the toggle to "True". Confirm the dialog with a tap on "Write" and you should see the built-in LED turned on. If you do the same procedure again but setting the toggle switch to "False", it will turn off the LED. [Doc24f]

- **6. Conclusion:** This example shows how to connect and control the built-in LED using a Bluetooth Low Energy connection. You have learned how a simple Bluetooth Low Energy connection between your Portenta and your cell phone, which has basic communication abilities between the two devices, works. 14.6 [Doc24f]

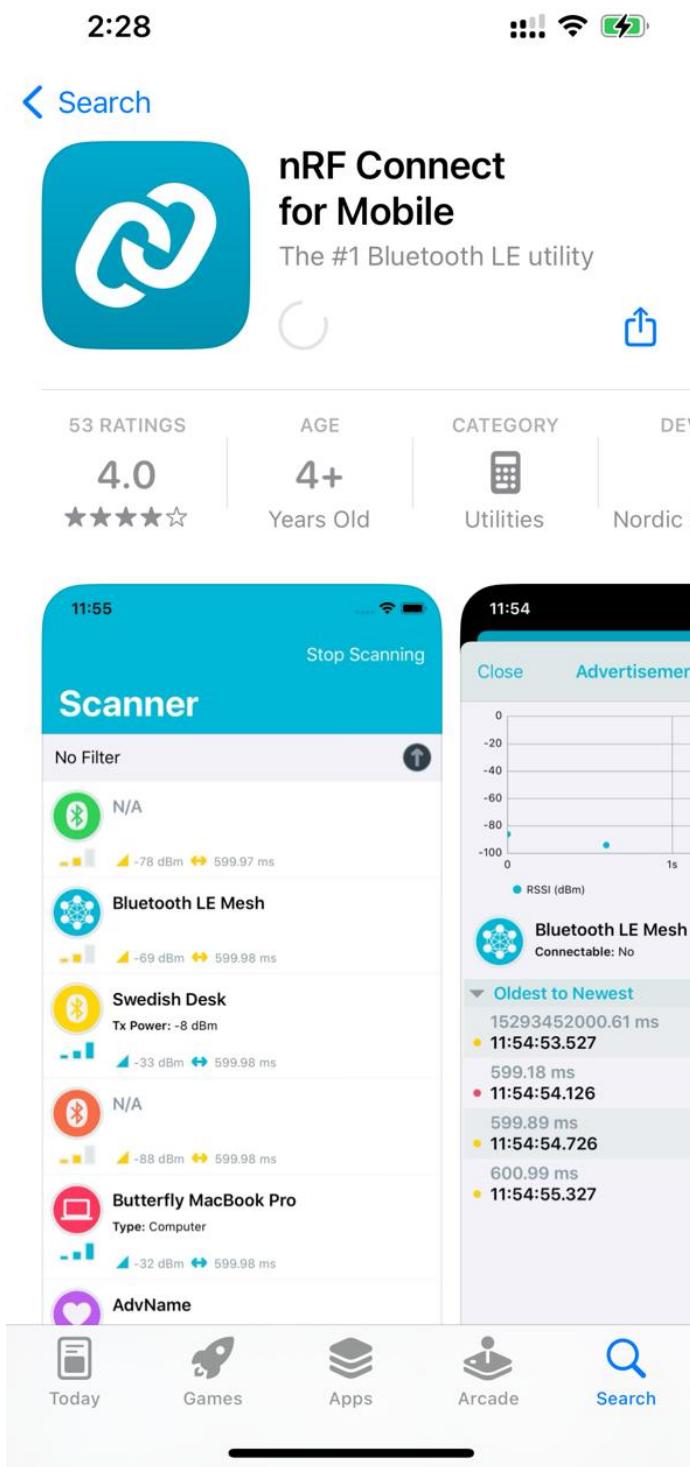


Figure 14.6.: NRF-Connect



Figure 14.7.: Bluetooth-scan
[Doc24f]

15. WIFI Module

15.1. Description

Portenta H7 comes with an on-board Wi-Fi Module that allows to develop IoT applications that require wireless connectivity and Internet access. Turning the board into an access point allows to create a Wi-Fi network on its own and allows other devices to connect to it. [Doc24b]

Key Features:

- **Dual-Band WiFi Connectivity:** The Portenta H7 supports both 2.4 GHz and 5 GHz WiFi bands, ensuring a reliable and fast connection in various network environments, whether in congested areas or high-speed applications.
- **Robust Security:** Equipped with WPA3 support, the WiFi module on the Portenta H7 provides cutting-edge security features, ensuring that data transmission is secure and protected from unauthorized access.
- **Seamless IoT Integration:** The WiFi module allows for easy integration into cloud platforms and IoT applications, enabling seamless data collection, remote monitoring, and real-time communication in connected systems.
- **High Throughput:** Capable of high data transfer rates, the WiFi module can handle large volumes of data, making it ideal for applications such as real-time video streaming, telemetry, or large-scale sensor networks.
- **Energy Efficient:** Designed to support low-power modes, the WiFi module can maintain wireless connectivity while conserving energy, ideal for battery-operated devices and power-conscious applications. [Doc24b]

15.2. Specific Sensor

The WiFi functionality in the Portenta H7 is powered by an integrated module that adheres to the 802.11 standards, supporting both 2.4 GHz

and 5 GHz frequency bands. This module enables seamless wireless communication, offering high data throughput and robust performance in a variety of network conditions.

The onboard WiFi/Bluetooth module provides the Portenta H7 with the capability to connect wirelessly to local networks, cloud services, and IoT platforms. The dual-band WiFi support allows the board to connect to a wide range of network environments, optimizing both speed and stability. [Doc24b]

15.3. Specification

- The WiFi module on the Arduino Portenta H7 supports the IEEE 802.11 a/b/g/n standards.
- It offers dual-band connectivity (2.4 GHz and 5 GHz) for flexible network compatibility.
- The WiFi module can achieve data transfer rates up to 150 Mbps, suitable for demanding applications such as real-time data streaming and cloud communication.
- It supports WPA3 encryption, ensuring secure communication across networks.
- The WiFi module operates with low power consumption, making it ideal for IoT applications and battery-powered projects.
- The WiFi module can be programmed using the Arduino IDE with the WiFi libraries available for the Portenta H7. [Doc24b]

15.4. TESTS

15.4.1. WiFi Access Point

The onboard Bluetooth module of the Portenta H7 offers low energy Bluetooth functionality, in order to provide the board with the flexibility to be easily connected to devices which also support Bluetooth Low Energy, such as the Arduino Nano 33 IoT or most modern smartphones. Compared to classic Bluetooth, Low Energy Bluetooth is intended to provide considerably reduced power consumption and cost while maintaining a similar communication range. [Doc24b]

Overview

Portenta H7 comes with an on-board Wi-Fi and a Bluetooth Module that allows to develop IoT applications that require wireless connectivity and Internet access. Turning the board into an access point allows to create a Wi-Fi network on its own and allows other devices to connect to it. In this tutorial you will learn to set up your board as an access point web server and remotely control the red, green and blue LEDs on the built-in RGB LED by accessing an HTML page on your mobile device's browser. [Doc24b]

Goals

- About the built-in Wi-Fi + Bluetooth module.
- How a client-server model works. [Doc24b]
- How to create an HTTP communication channel between the board and an external device. [Doc24b]

Required Hardware and Software

1. Portenta H7 or Portenta H7 Lite Connected
2. USB-C cable (either USB-A to USB-C)
3. Arduino IDE 1.8.13+ or Arduino Pro IDE 0.0.4+
4. A smart phone [Doc24b]

Access Point Configuration

The Portenta H7 features a Murata 1DX, which is a high performance chipset which supports Wi-Fi 802.11b/g/n + Bluetooth 5.1 BR/EDR/LE, up to 65 Mbps PHY data rate on Wi-Fi and 3 Mbps PHY data rate on Bluetooth. This module helps to configure the Portenta into three different modes of operation - an Access Point, a Station, or both. In this tutorial we will only focus on the access point configuration.

When the board is configured to operate as an access point, it can create its own wireless LAN (WLAN) network. In this mode, the board transmits and receives signals at 2.4 GHz allowing other electronic devices with Wi-Fi capabilities using the same bandwidth to connect to the board.

With the access point set up, you create a client server architecture where the board provides a web server communicating with the client devices over HTTP. The connected devices can then make HTTP GET requests to the server to retrieve web pages served by the web server on the board. This makes the Portenta H7 an ideal board for developing IoT solutions

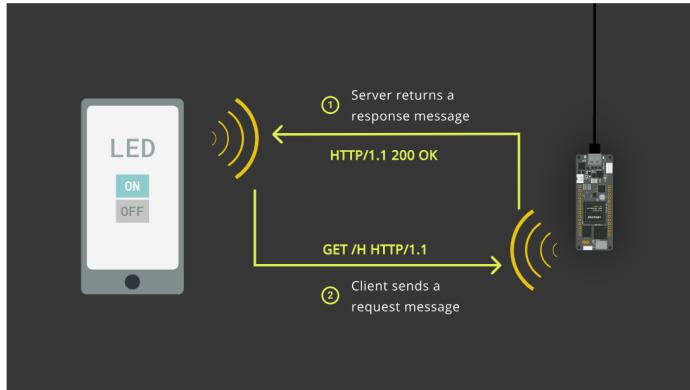


Figure 15.1.: Full Interface
[Doc24b]

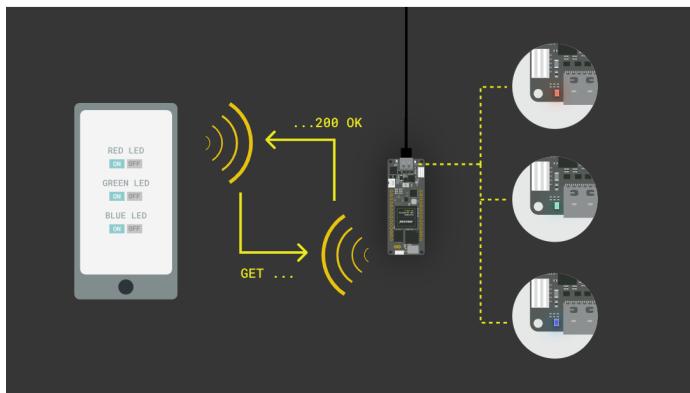


Figure 15.2.: SetUpServer
[Doc24b]

where external client devices can send and receive information while more complex processing tasks take place on the server. [Doc24b]

Instructions

- **Setting Up the Web Server:** In this tutorial you are going to convert the board into an access point and use it to set up a web server which provides a HTML webpage. This page contains buttons to toggle the red, green and blue color of the built-in LED. You will then connect your mobile device to this access point and access this web page through the browser on your mobile phone. Once retrieved, you will be able to control the state of the red, green and blue LED on the built-in RGB LED from your mobile device. 15.2 [Doc24b]



Figure 15.3.: BasicSetup
[Doc24b]

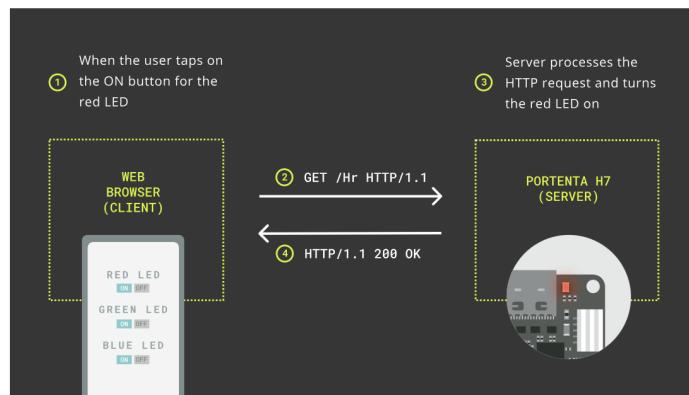


Figure 15.4.: LEDState
[Doc24b]

- **1. The Basic Setup:** Begin by plugging in your Portenta board to your computer using a USB-C cable and open the Arduino IDE. If this is your first time running Arduino sketch files on the board, we suggest you check out how to set up the Portenta H7 for Arduino before you proceed. 15.3 [Doc24b]
- **2. Create the Web Server Sketch:** Next you need to create a web server sketch that will handle the HTTP GET requests and provide the client devices with the HTML web page. The Wi-Fi library provides all necessary methods that allows Arduino boards to use their Wi-Fi features provided by the on-board Wi-Fi module. To set up the web server copy the following code, paste it into a new sketch file and name it **SimpleWebServer.ino**. 15.4 [Doc24b]

Listing 15.1.: Simple sketch to control the LED using bluetooth

```
1000 #include <WiFi.h>
```

```

1002 #include "arduino_secrets.h"
1003
1004 //+++++please enter your sensitive data in the Secret
1005 //tab/arduino_secrets.h
1006 char ssid [] = SECRET_SSID;      // your network SSID (name
1007 )
1008 char pass [] = SECRET_PASS;     // your network password (
1009   use for WPA, or use as key for WEP)
1010 int keyIndex = 0;               // your network key Index
1011   number (needed only for WEP)
1012
1013 int status = WL_IDLE_STATUS;
1014
1015 WiFiServer server(80);
1016
1017 void setup() {
1018   // put your setup code here, to run once:
1019   Serial.begin(9600);
1020   while (!Serial) {
1021     ; // wait for serial port to connect. Needed for
1022       native USB port only
1023   }
1024
1025   Serial.println("Access Point Web Server");
1026
1027   pinMode(LED_R,OUTPUT);
1028   pinMode(LED_G,OUTPUT);
1029   pinMode(LED_B,OUTPUT);
1030
1031   // by default the local IP address of will be
1032   // 192.168.3.1
1033   // you can override it with the following:
1034   // WiFi.config(IPAddress(10, 0, 0, 1));
1035
1036   if(strlen(pass) < 8){
1037     Serial.println("Creating access point failed");
1038     Serial.println("The Wi-Fi password must be at least
1039       8 characters long");
1040     // don't continue
1041     while(true);
1042   }
1043
1044   // print the network name (SSID);
1045   Serial.print("Creating access point named: ");
1046   Serial.println(ssid);
1047
1048   //Create the Access point
1049   status = WiFi.beginAP(ssid,pass);
1050   if(status != WL_AP_LISTENING){
1051     Serial.println("Creating access point failed");
1052     // don't continue
1053     while (true);
1054   }

```

```
1048 // wait 10 seconds for connection:  
1049 delay(10000);  
1050  
1051 // start the web server on port 80  
1052 server.begin();  
1053  
1054 // you're connected now, so print out the status  
1055 printWiFiStatus();  
1056 }  
1057  
1058 void loop() {  
1059  
1060 // compare the previous status to the current status  
1061 if (status != WiFi.status()) {  
1062 // it has changed update the variable  
1063 status = WiFi.status();  
1064  
1065 if (status == WL_AP_CONNECTED) {  
1066 // a device has connected to the AP  
1067 Serial.println("Device connected to AP");  
1068 } else {  
1069 // a device has disconnected from the AP, and we  
1070 // are back in listening mode  
1071 Serial.println("Device disconnected from AP");  
1072 }  
1073 }  
1074  
1075 WiFiClient client = server.available(); // listen  
1076 // for incoming clients  
1077  
1078 if (client) { // if you  
1079 // get a client,  
1080 // Serial.println("new client"); // print a  
1081 // message out the serial port  
1082 // String currentLine = ""; // make a  
1083 // String to hold incoming data from the client  
1084  
1085 while (client.connected()) { // loop  
1086 // while the client's connected  
1087  
1088 if (client.available()) { // if there's  
1089 // bytes to read from the client,  
1090 // char c = client.read(); // read a  
1091 // byte, then  
1092 // Serial.write(c); // print it  
1093 // out the serial monitor  
1094 // if (c == '\n') { // if the  
1095 // byte is a newline character  
1096  
1097 // if the current line is blank, you got two  
1098 // newline characters in a row.
```

```

        // that's the end of the client HTTP request ,
        so send a response:
1090      if (currentLine.length() == 0) {
          // HTTP headers always start with a response
          // code (e.g. HTTP/1.1 200 OK)
1092      // and a content-type so the client knows
      what's coming, then a blank line:
1094      client.println("HTTP/1.1 200 OK");
      client.println("Content-type:text/html");
      client.println();

1096      // the content of the HTTP response follows
      the header:
1098      client.print("<html><head>");
      client.print("<style>");
1100      client.print("* { font-family: sans-serif; }"
);
      client.print("body { padding: 2em; font-size
: 2em; text-align: center; }");
1102      client.print("a { -webkit-appearance: button
;-moz-appearance: button; appearance: button; text-
decoration: none; color: initial; padding: 25px; } #red
{color:red;} #green{color:green;} #blue{color:blue;}"
);
      client.print("</style></head>");
1104      client.print("<body><h1> LED CONTROLS </h1>"
);
      client.print("<h2><span id=\"red\">RED </
span> LED </h2>");
      client.print("<a href=\"/Hr\">ON</a> <a href
=\":/Lr\">OFF</a>");
      client.print("<h2> <span id=\"green\">GREEN
</span> LED </h2>");
1108      client.print("<a href=\"/Hg\">ON</a> <a href
=\":/Lg\">OFF</a>");
      client.print("<h2> <span id=\"blue\">BLUE</
span> LED </h2>");
      client.print("<a href=\"/Hb\">ON</a> <a href
=\":/Lb\">OFF</a>");
      client.print("</body></html>");

1112      // The HTTP response ends with another blank
      line:
1114      client.println();
      // break out of the while loop:
      break;
1116    } else { // if you got a newline , then
      clear currentLine:
      currentLine = " ";
    }
1118  } else if (c != '\r') { // if you got
      anything else but a carriage return character ,
      currentLine += c; // add it to the end of

```

```
1122     the currentLine
1123     }
1124
1125     // Check to see if the client request was "GET /H" or "GET /L":
1126     if (currentLine.endsWith("GET /Hr")) {
1127         digitalWrite(LED_R, LOW); // GET
1128         // Hr turns the Red LED on
1129         }
1130         if (currentLine.endsWith("GET /Lr")) {
1131             digitalWrite(LED_R, HIGH); // GET
1132             // Lr turns the Red LED off
1133             }
1134             if (currentLine.endsWith("GET /Hg")) {
1135                 digitalWrite(LED_G, LOW); // GET
1136                 // Hg turns the Green LED on
1137                 }
1138                 if (currentLine.endsWith("GET /Lg")) {
1139                     digitalWrite(LED_G, HIGH); // GET
1140                     // Lg turns the Green LED on
1141                     }
1142
1143         }
1144     }
1145     // close the connection:
1146     client.stop();
1147     Serial.println("client disconnected");
1148 }
1149
1150 }
1151
1152 void printWiFiStatus() {
1153     // print the SSID of the network you're attached to:
1154     Serial.print("SSID: ");
1155     Serial.println(WiFi.SSID());
1156
1157     // print your Wi-Fi shield's IP address:
1158     IPAddress ip = WiFi.localIP();
1159     Serial.print("IP Address: ");
1160     Serial.println(ip);
1161
1162     // print where to go in a browser:
1163     Serial.print("To see this page in action, open a
1164         browser to http://");
1165     Serial.println(ip);
```

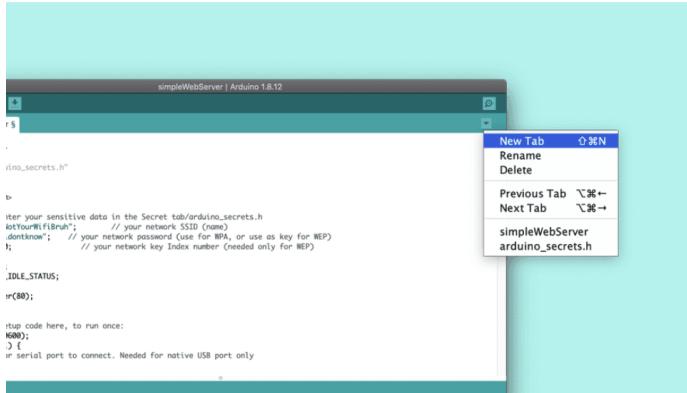


Figure 15.5.: NewTab
[Doc24b]

1166 }

.../Code/Wifi/SimpleWebServer/SimpleWebServer.ino

This sketch describes how the server will handle an incoming HTTP GET request from a client, both to request the HTML web page from the server and the requests to change the LED states using dedicated URLs.

Here the web page is just a simple HTML page with buttons to toggle the LED states. The way in which the web page works is: whenever a button on the web page is pressed, the client device (in this case your phone) sends a HTTP GET request to a URL denoted by a letter, in this case H or L (H stands for HIGH, L stands for LOW) followed by the LED color that should be turned on or off r, g or b. For example, to turn on the red LED the URL is /Hr . Once the server receives this request, it changes the corresponding LED state, closes the connection and continues to listen to next requests. [Doc24b]

- **3. Create the arduino secrets.h Tab:** A good practice is to have sensitive data like the SSID and the password required to identify and connect to a certain network within a separate file. Click on the arrow icon below the Serial Monitor button and open a new tab in the Arduino IDE. This will create a new file.. 15.1 [Doc24b]

Name the file **arduino secrets.h** and click OK.

Once you have created the new tab, you will see an empty page in the IDE. Define two constants **SECRET SSID** and **SECRET PASS** that will hold the name of the Wi-Fi network and the corresponding password. Add the following lines to your **arduino secrets.h** file:



Figure 15.6.: NamingNewTab
[Doc24b]



Figure 15.7.: HeaderFile
[Doc24b]

```

1 # define SECRET_SSID "PortentaAccessPoint"
2 # define SECRET_PASS "123Qwerty"

```

In order to access the **SECRET_SSID** and **SECRET_PASS** constants in the **simpleWebServer.ino** sketch file, the header file that you have just created needs to be included. In your sketch file this has already been taken care of by the following line at the beginning of the sketch:

```

1 #include "arduino_secrets.h"

```

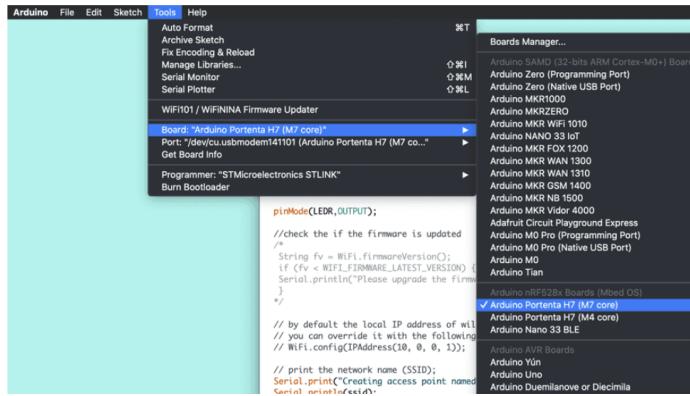


Figure 15.8.: Uploading
[Doc24b]

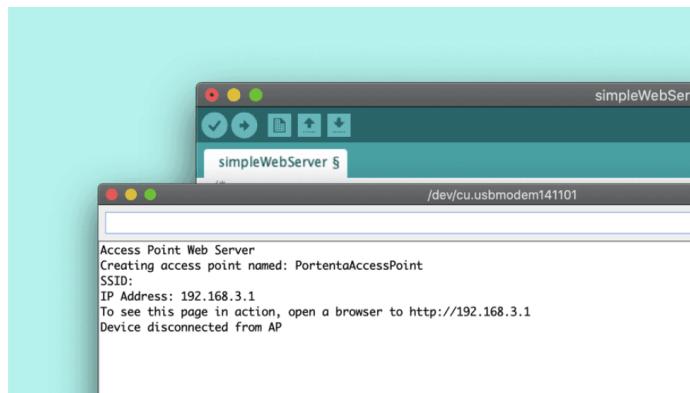


Figure 15.9.: SerialMonitor
[Doc24b]

- **4. Upload the Code:** Select the Arduino Portenta H7 (M7 core) from the Board menu and the port the Portenta is connected to. Upload the simpleWebServer.ino sketch. Doing so will automatically compile the sketch beforehand. 15.8

Once you have uploaded the code, open the Serial Monitor. You will be able to see the IP address of the access point. You will also see the message, Device disconnected from AP which means there are no devices connected to the Access point yet.

- **5. Connecting to the Portenta Access Point:** Once the access point is active and ready to be connected with external devices, you will be able to find the PortentaAccessPoint on the list of networks on your mobile device. Once you have entered the password you have defined earlier, your smart phone will connect to access point.. 15.10 [Doc24b]

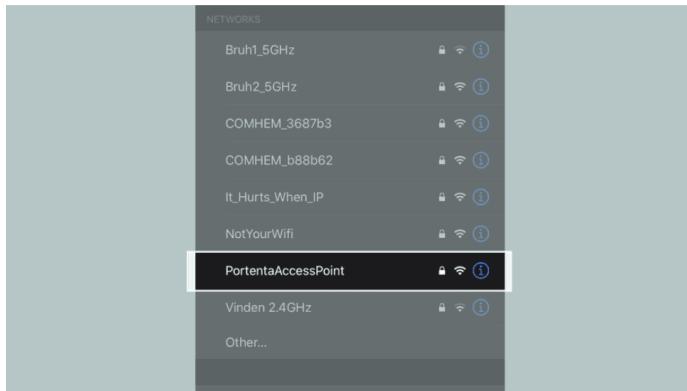


Figure 15.10.: PortentaAccessPoint
[Doc24b]

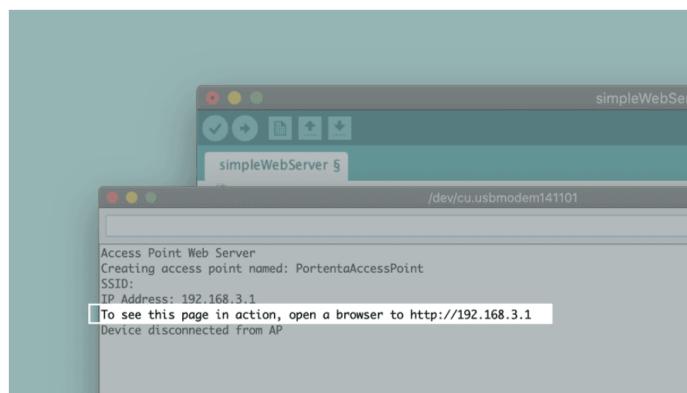


Figure 15.11.: URLAccessPoint
[Doc24b]

Now open a browser window on your mobile device and copy and paste the URL containing Portenta's IP address that is displayed on the serial monitor.

Once you have entered the URL, the client sends a GET request to the web server to fetch the HTML web page specified in the code. Once loaded, you will see the web page in your mobile browser.

- **5. Access the Board From Your Mobile Device:** If you take a look at the Serial Monitor, you can see the details of the HTTP GET request and other details of the device connected to the access point. The GET request is always in the following format:.. [Doc24b]

```
1   GET URL HTTP/1.1
```

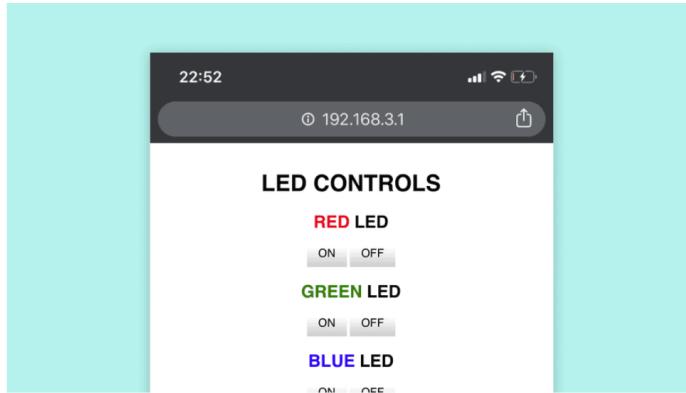


Figure 15.12.: HTML Page
[Doc24b]

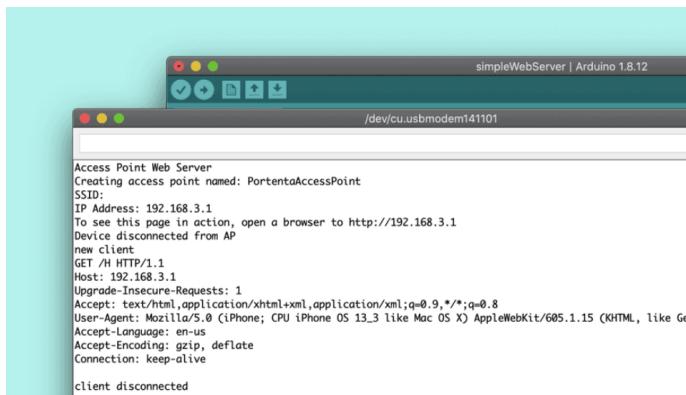


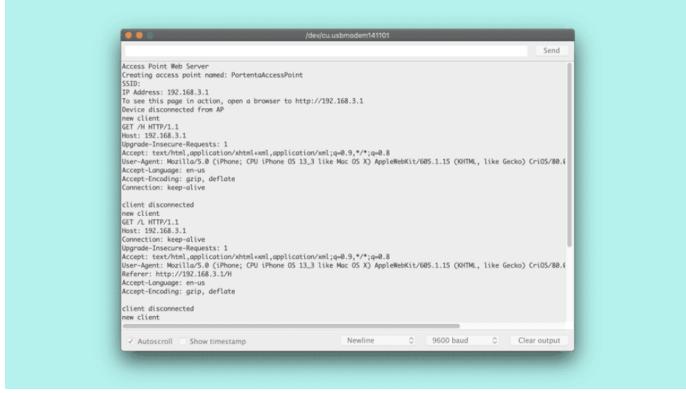
Figure 15.13.: ClientDetails
[Doc24b]

The URL is a string of characters sent to the server, in this case /Hx (where x stands for the color of the LED). This request containing the URL is received on the server and it replies with the following HTTP/1.1 response indicating that the connection was successful:

1 **HTTP/1.1 200 OK**

Once the server has responded to this request, it closes the connection and continues listening to next GET requests.

You are now be able to toggle the states of the red, green and blue LED through the buttons displayed on your mobile browser. Every time you press a button, the client sends a GET request to a URL in the format /Hx or /Lx ,where x can be 'r', 'g' or 'b', depending



The screenshot shows a terminal window titled "Access Point Web Server" with the identifier "/dev/cu.usbmodem141101". The window displays a log of HTTP requests and responses. A client connects at IP address 192.168.3.1 and sends a GET request for the root URL. The server responds with an HTML file containing instructions to open a browser at http://192.168.3.1. The client then disconnects.

```
Access Point Web Server
Creating access point named: PortentaAccessPoint
IP Address: 192.168.3.1
To see this page in action, open a browser to http://192.168.3.1
client disconnected from AP
new client
GET / HTTP/1.1
Host: 192.168.3.1
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 13_3 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) CriOS/86.4
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: keep-alive
client disconnected
new client
GET / HTTP/1.1
Host: 192.168.3.1
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Referer: http://192.168.3.1/
Accept-Language: en-us
Accept-Encoding: gzip, deflate
client disconnected
new client
```

Figure 15.14.: GETRequest
[Doc24b]

on the button pressed on the HTML page. The web server then reads the URL requested by the client, changes the state of the LED corresponding to the URL and closes the connection. [Doc24b]

Conclusion:

This tutorial shows one of the several capabilities of the on-board WiFi+Bluetooth module by configuring the board as an access point and setting up a web server. You have also learnt how a simple client-server model and the underlying HTTP requests and responses work. [Doc24b]

16. LORA

The Internet of Things (IoT) is often referred to as a collection of objects connected to the Internet using wireless networks; these connected objects aim to collect and exchange information from their surroundings. IoT enables a connection between the physical and the digital worlds; that connection produces a massive amount of data that can be used for the optimization of resources and to improve the efficiency of existing systems.

Many of the existing IoT devices will be connected to the Internet using short-range wireless networks such as Wi-Fi, Bluetooth, ZigBee, Z-Wave, etc. Cellular connections using networks such as 2G, 3G, and 4G will also connect IoT devices to the Internet. See Figure 16.3 for reference. Still, these short and medium-range wireless networks are not always suitable for IoT devices since they were developed for applications where power consumption and battery life are not significant issues. IoT devices usually have low-power consumption and send and receive low amounts of data. [Ard24l]

16.1. Description

LoRa (Long Range) is a wireless communication technology developed to facilitate long-distance, low-power data transmission, making it ideal for Internet of Things (IoT) applications. Utilizing the LoRa modulation technique, this technology enables devices to communicate over several kilometers with minimal energy consumption, which is essential for battery-powered devices deployed in remote or widespread locations. [Cor25b]

16.2. Key Features of LoRa

- **Long Range:** Capable of transmitting data over distances up to 15 kilometers in rural areas and 2-5 kilometers in urban settings.
- **Low Power Consumption:** LoRa devices consume as little as 1-2 μ A in sleep mode, around 20-50 mA during transmission, and can operate for 5-10 years on a standard battery. [STM25b]

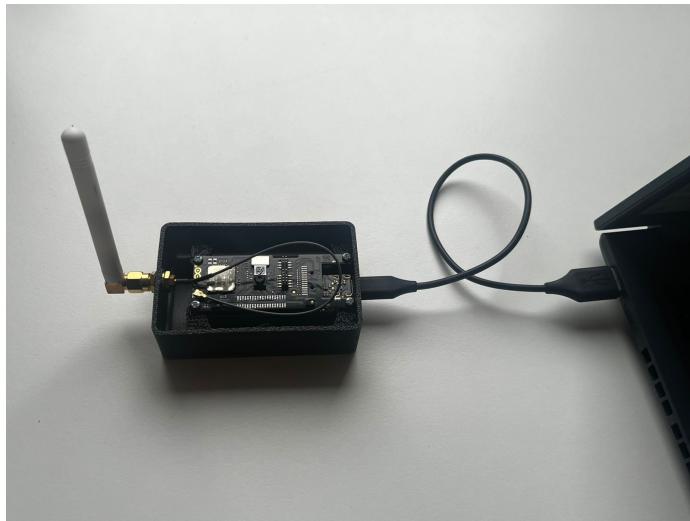


Figure 16.1.: LoRa

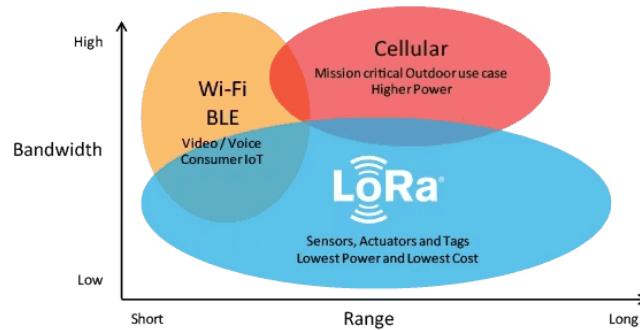


Figure 16.2.: Bandwidth vs. range of short distance, cellular and LPWA networks. Image credits: The Things Network.

- **Robustness:** Resistant to interference and capable of maintaining reliable communication in challenging environments by operating at a link budget of up to 168 dB. [Net25]
- **Scalability:** Supports large-scale deployments with thousands of devices interconnected within a single network. [Joo25]

16.3. Hardware Components of the LoRa Implementation

The LoRa implementation in the Portenta Vision Shield comprises several essential hardware components (Refer Figure 16.1) that work together to enable seamless communication, data handling, and energy management. Below are the key components [Ard25b].

16.3.1. LoRa Transceiver Module

- **Function:** Provides radio communication using LoRa modulation for transmitting and receiving data [Cor25a].
- **Integrated Module:** The Vision Shield integrates the Murata CMWX1ZZABZ module, which is based on the Semtech SX1276 LoRa transceiver [Man25].
- **Features:**
 1. Configurable frequency bands (868 MHz for Europe and 915 MHz for North America).
 2. Adjustable spreading factors for optimizing communication range and data rates.
 3. Supports adaptive data rates (ADR) to improve efficiency [Cor25a].

16.3.2. Antenna

- **Function:** Facilitates the transmission and reception of LoRa signals [Cor25b].
- **Antenna Type:** External antenna connected via a **U.FL connector**, designed to enhance signal quality and maintain a stable communication range [Man25].
- **Considerations:** Optimized for compatibility with LoRa frequency bands and to maximize the communication range in different environments [Cor25b].

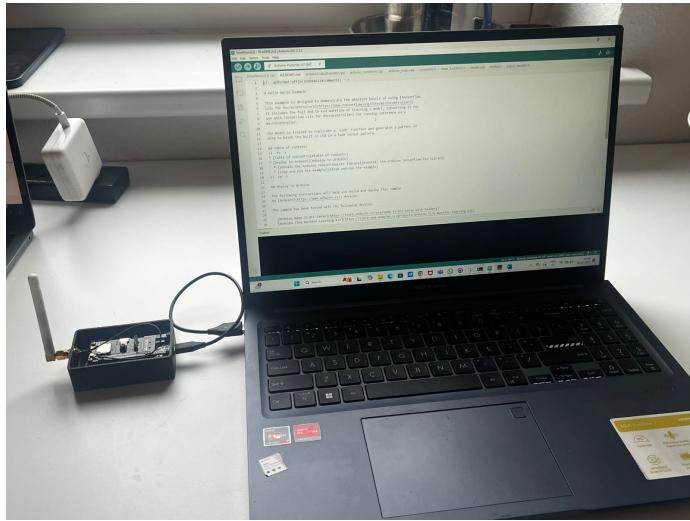


Figure 16.3.: LoRa with Antenna

16.3.3. Microcontroller Unit (MCU)

- **Function:** Serves as the control center for LoRa operations, handling data processing, interfacing with the transceiver, and managing communication protocols [Ard25b].
- **MCU in Portenta H7:**
 1. Dual-core architecture: ARM Cortex-M7 (480 MHz) for high-performance tasks and Cortex-M4 (240 MHz) for low-power operations
 2. Ensures efficient management of LoRa-specific operations like encoding/decoding packets and power optimization.[Lim25]

16.3.4. Power Supply

- **Function:** Supplies power to the LoRa transceiver and its supporting components
- **Sources:** The Vision Shield relies on the Portenta H7's onboard power supply and is compatible with external battery packs for portable LoRa applications
- **Components:** Includes voltage regulators and power management ICs to ensure stable operation of the LoRa transceiver and optimize battery life during transmission [Ard25b].

16.3.5. Interface

- **Function:** The LoRa transceiver communicates with the Portenta H7 via different protocols, which ensures fast and reliable data transfer between the MCU and the LoRa module.
- **Configuration:** Communicates with the MCU using I2C, SPI, or UART protocols [Ard25b].

16.3.6. Memory

- **Function:** Provides storage for firmware, LoRa communication logs, and configuration data.
- **Memory in Portenta H7:**
 1. **Flash Memory:** 16 MB NOR Flash for storing LoRa firmware and application-specific code.
 2. **SRAM:** 8 MB SDRAM for temporary data processing during LoRa communication [Ard25b].

Table 16.1.: Specifications of LoRa Device on Portenta Vision Shield
[Doc25f]

Parameter	Description
Module Size	Compact, integrated into the Vision Shield
Frequency Range	868 MHz / 915 MHz (region-specific frequencies for LoRa communication)
Modulation	Long Range (LoRa) modulation and Frequency-Shift Keying (FSK)
Data Rate	0.3 kilobits per second to 50 kilobits per second
Communication Range	Up to 10 kilometers in open areas with line of sight
Interface	Supports communication with microcontrollers through SPI
Operating Voltage	Operates at a typical voltage of 3.3 volts
Power Output	Output power up to +22 decibel-milliwatts (dBm)
Operating Temperature	Functional within a range of -20°C to +70°C
Power Consumption (Transmit)	20 to 120 milliamperes, depending on the transmission power
Power Consumption (Receive)	10 to 15 milliamperes during reception
Sleep Mode Current	Less than 1 microampere in sleep mode
Antenna Connector	Uses a U.FL connector, a small and reliable connector for attaching an external antenna

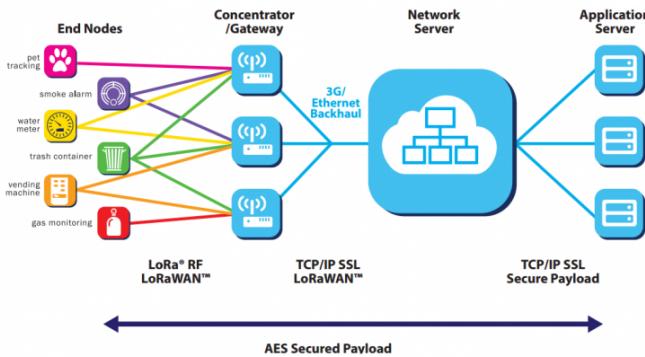


Figure 16.4.: Source: The Things Network

16.3.7. Peer-To-Peer (P2P) LoRa Communication

To understand the unique value proposition of LoRaWAN P2P Communication, we first have to understand the network architecture of LoRaWAN(Refer figure 16.4), which consists of four major components: [Stu21]

1. **End Nodes:** Represents edge devices or sensors
2. **Gateway:** Collects or concentrates data from several end nodes
3. **Network Server:** Consolidates data from gateways for upload to application server
4. **Application Server:** Processes or displays consolidated data [Stu21]

End nodes contain sensors that collect valuable information to be transmitted to the cloud. This is done by broadcasting LR data packets to nearby LoRaWAN gateways, which then forward the data to the cloud. Since all gateways in range of the transmitting end node receive the data, this creates a star-on-star topology which promotes dense interconnectivity and reliability in LoRaWAN networks, shown below:

P2P LR Communication refers to LR communications that occur directly between end node devices (peer-to-peer), instead of those that occur between an end node and the gateway. This communication uses the same long-distance LR modulation between the two devices, and enables numerous possibilities with LR technology! [Stu21]

Benefits of P2P LoRa Communication

The design of the LoRaWAN network architecture has its own purpose and benefits, which we've discussed in depth here. Nonetheless, P2P in

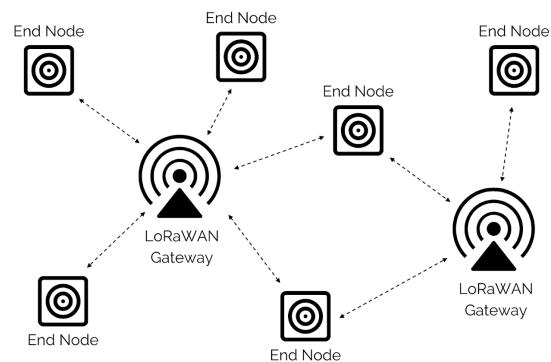


Figure 16.5.: LoRaWAN Topology
[Stu21]

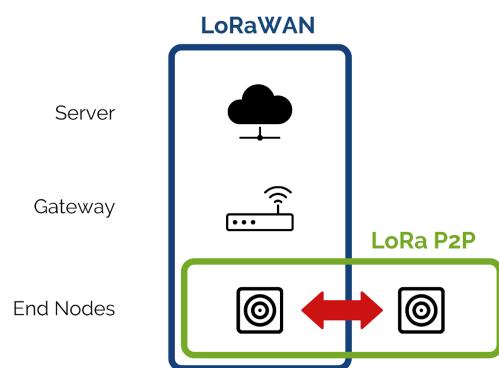


Figure 16.6.: P2P LoRa
[Stu21]

LoRaWAN allows end nodes to communicate directly without the use of a gateway, which brings several benefits. [Stu21]

- **Increase the Range of LoRaWAN Coverage** In some areas, the full LoRaWAN infrastructure may be unavailable or unfeasible to set up. By linking multiple end nodes together, we can create a **chain topology** that extends the range of the LoRaWAN with a limited additional cost!
- **Reduce Costs of Using LR for IoT** Although LoRaWAN gateways are relatively affordable, small-scale applications may benefit from bypassing the use of an additional device to save further costs. This is a great advantage for beginners who are just getting started and want to simply experiment with LoRa.
- **Increased Flexibility with LoRaWAN Networks** Did you know that end nodes engaged in P2P LR communication can continue to interface with nearby LoRaWAN Gateways? When used together, P2P connections can be used to further reinforce the LoRaWAN network, providing greater operational reliability. [Stu21]

For detailed explanation, please refer the following example 16.5.2.

16.4. Low-Power Wide Area Networks(LPWAN)

Low-Power Wide Area Networks (LPWAN) is a group of wireless networks technologies well suited to the specific needs of IoT devices: low-bandwidth and low-power devices, usually battery-powered. This type of networks provide low-bit rates long ranges with a low-power consumption. LPWAN's can accommodate data packets sizes from 10 bytes to 1 kB at uplink speeds up to 200 kbps; long-range connectivity varies from 2 to 1,000 km depending on the network technology. Most LPWAN's technologies have a star topology; this means that each device connects directly to a central access point. [Ard24l]

Some of the important use cases for LPWAN's include the following applications:

1. **Smart cities:** smart parking, intelligent street lighting.
2. **Supply chain management:** asset tracking, condition monitoring.
3. **Smart grids:** electricity, water, and gas metering.
4. **Smart agriculture:** land condition monitoring, animal tracking, geofencing. [Ard24l]

Several LPWAN technologies use licensed or unlicensed frequencies and proprietary or open specifications. LoRa and its Media Access Control (MAC) layer protocol implementation, LoRaWAN, is currently one of the existing LPWAN gaining the most traction to support IoT devices and services.

16.4.1. Difference between LoRa and LoRaWAN?

LoRa is a wireless modulation technique derived from Chirp Spread Spectrum (CSS) technology. CSS uses wideband linear frequency modulated chirp pulses to encode information. LoRa can operate on the following license-free sub-gigahertz ISM (Industrial, Scientific, and Medical) bands: 433 MHz, 868 MHz, and 915 MHz. ISM bands are internationally reserved for industrial, scientific and, medical uses.

Based on LoRa, the LoRaWAN (LoRa for Wide Area Networks) specification extended the LoRa physical communication layer into the Internet by adding a MAC layer(Refer Figure 16.8). The LoRaWAN specification is a software layer that defines how devices must use the LoRa, for example, when they transmit or receive messages. The LoRaWAN specification is open-source; it has been supported and maintained by the LoRa Alliance since 2015. [Ard24l]

16.4.2. LoRaWAN Network Architecture

A typical LoRaWAN network architecture includes the following essential parts: end-devices (usually sensors), a base station or gateway, also known as Long Range Relay (LRR), a network server also known as Long Range Controller (LRC), and the Operation Support System (OSS) for provisioning and management of the network. [Ard24l]

Refer image 16.7, notice there is a fundamental difference between a network server and a gateway. The network server controls the virtualized MAC layer of the LoRaWAN network while gateways are devices pre-integrated with the network server to ease the LPWAN rollout and provisioning. LoRaWAN network servers and gateways access can be public or private.

LoRaWAN networks are usually deployed in a star-of-stars topology; this means that gateways manage data between end-devices and a network server. Gateways are connected to the central network server via the Internet, while end-devices use LoRa to send and receive data to and from the gateways; end-devices are not exclusively tied to a single gateway, end-devices broadcast information to all the gateways in range. Communication in LoRaWAN networks is natively bi-directional, although uplink

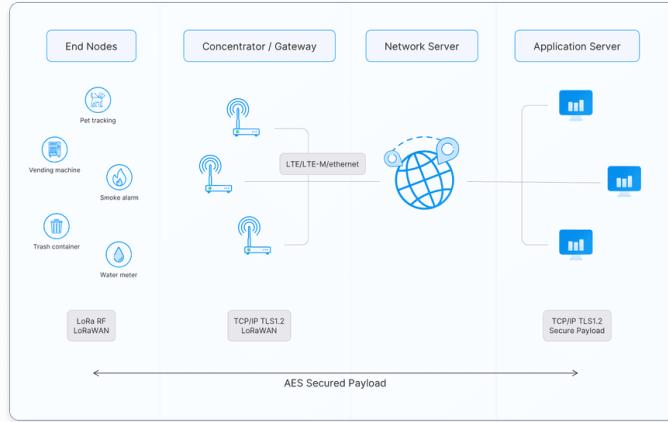


Figure 16.7.: Typical LoRaWAN network architecture example.

communication between end-devices and the central network server is expected to be predominant in the network. [Ard24l]
 Star networks present several advantages compared to other network topologies:

- Gateways can be added to the network anywhere and anytime without prior planning.
- Message delivery is more robust since multiple gateways receive the same data packets during each uplink. [Ard24l]

16.4.3. Data Rates

Communication between end-devices and gateways in LoRaWAN networks is spread out on different frequency channels and data rates (communications using different data rates do not interfere with each other).

LoRa supports data rates ranging from 300 bps to 5 kbps for a 125 kHz bandwidth.

To maximize the battery life of each end-device and the overall capacity available through the network, LoRaWAN uses an Adaptive Data Rate (ADR) mechanism for optimizing data rates, airtime, and power consumption. ADR controls the following transmission parameters on end-devices: [Ard24l]

1. **Spreading factor:** the speed of data transmission. Lower spreading factors mean a higher data transmission rate.
2. **Bandwidth:** the amount of data that can be transmitted from one point to another within the network.

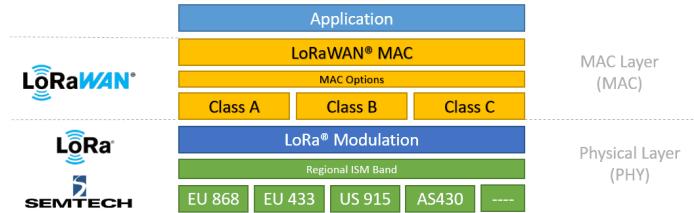


Figure 16.8.: LoRaWAN network layers. Image credits: Semtech.

3. **Transmission power:** the energy that the end-device transmitter produces at its output. [Ard24l]

The table below 16.2 shows compares spreading factor, data rate, and time on-air at a bandwidth of 125 kHz (range is an indicative value, it will depend on the propagation conditions):

Table 16.2.: LoRa Spreading Factor Specifications

Spreading Factor (SF)	Data Rate (bps)	Range (km)	Time on-Air (ms)
SF7	5470	2	56
SF8	3125	4	100
SF9	1760	6	200
SF10	980	8	370
SF11	440	11	40
SF12	290	14	1400

End-devices may transmit on any channel available at any time, using any available data rate, as long as the following rule is respected:

- The end-device changes channel in a pseudo-random fashion for every transmission. The resulting frequency diversity makes the system more robust against interference. [Ard24l]

Also, local regulations must be respected, for example:

- In the EU868 band, the end-device must respect the maximum transmit duty cycle relative to the sub-band used and local regulations (1)
- In the US915 band, the end-device must respect the maximum transmit duration (or dwell time) relative to the sub-band used and local regulations (400ms). [Ard24l]

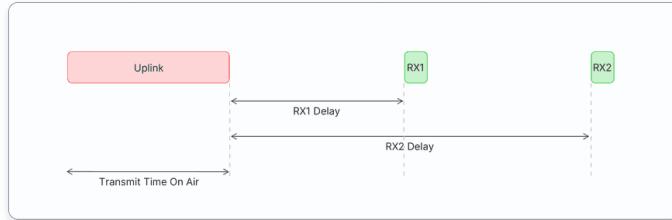


Figure 16.9.: Class A default configuration profile. Image credits: The Things Network.

16.4.4. Classes

The LoRaWAN specification has three different communication profiles between devices and applications: **Class A**, **Class B**, and **Class C**. (Refer figure 16.8) Each class serves different application needs and has optimized requirements for specific purposes. The main difference between the three classes is latency and power consumption; end-devices can always send uplinks when needed, but its class will determine when to receive downlinks. [Ard24l]

Class A: "Aloha"

Class A devices implement a bi-directional communication profile where two short downlinks follow the end-device uplink transmission receive windows, usually referred to as RX1 and RX2. If the server does not respond in either RX1 or RX2 windows, the next opportunity will be after the next uplink transmission(Refer figure 16.9). Class A devices are often battery-powered and spend most of the time in sleep mode; therefore, they have the lowest energy consumption, keep long intervals between uplinks, and have high downlink latency. [Ard24l]

Class B: The "Beaconing" Class

Class B devices extend Class A devices by adding scheduled receive windows for downlinks, and, therefore, they emulate a continuously receiving device by opening receive windows at fixed time intervals(Refer figure 16.10). This class should be implemented when low latency of downlink communication while keeping the power consumption as low as possible is required. [Ard24l]

Class C: Continuous Reception

Class C communication profile is used in applications with enough power available, so there is no need to minimize the time of the reception

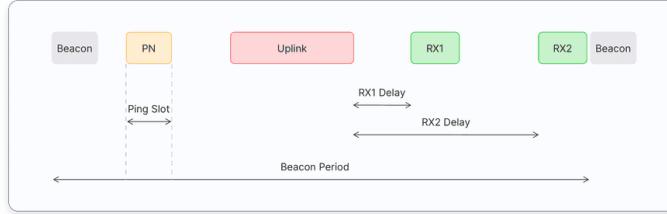


Figure 16.10.: Class B default configuration profile. Image credits: The Things Network.

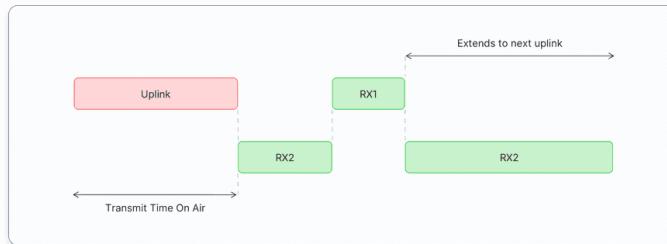


Figure 16.11.: Class C default configuration profile. Image credits: The Things Network

windows(Refer figure 16.11); this is the case of most actuators (e.g., smart plugs, street lights, electrical meters, etc.) Class C devices always listen for downlinks messages unless they transmit an uplink message. This behavior results in the lowest latency between the server and the end-device. [Ard24]

16.4.5. Authentication and Security

Authentication and security are also important in LoRaWAN networks. Any LoRaWAN network has a baseline authentication and security framework based on the AES 128 encryption scheme. Compared to other LPWAN's, which rely on a single key for authentication and encryption, the LoRaWAN framework separates both. Authentication and integrity control use a network session key (NwkSKey) while user data encryption uses an application session key (AppSKey). [Ard24]

NwkSKey and AppSKey are AES-128 root keys specific to the end-device, end-devices manufacturers, or application owners assigned them.

LoRaWAN supports two authentication and activation methods: Over-The-Air-Activation (OTAA) and Activation by Personalization (ABP).

- 1. Over-The-Air Activation (OTAA):** In this method, end-devices are not initialized for any particular network; they send a JOIN

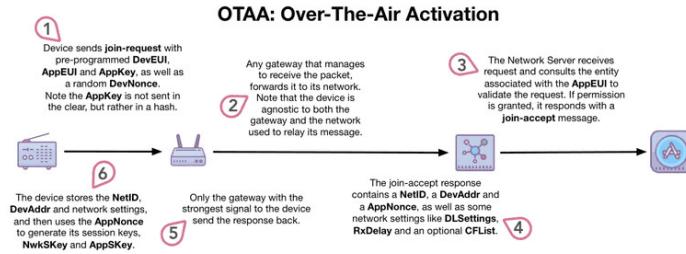


Figure 16.12.: Over-The-Air activation process. Image credits: Heath Raftery.

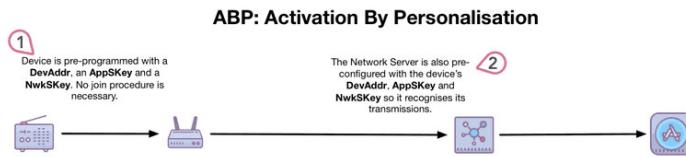


Figure 16.13.: Activation by Personalization process. Image credits: Heath Raftery.

request to a specific LoRaWAN network and then receive a device address and an authorization token from which session keys are derived(Refer figure 16.12); NwkSKey and AppSKey are derived during this procedure from a root AppKey pre-provisioned in the end-devices by its manufacturer. [Ard24l]

2. **Activation by Personalization (ABP):** In this method, end-devices are personalized to work with a given LoRaWAN network. End-devices are pre-provisioned with the NwkSKey and AppSKey and the 32-bits device network address(Refer figure 16.13). [Ard24l]

16.4.6. Connecting to the LoRaWAN-TTN

The Portenta Vision Shield - LoRa can be connected to the TTN and can transmit data to other devices connected to this network through a secure channel. This channel is nothing but an application on the TTN network dedicated for your board. In this tutorial, you will be guided through a step-by-step process of setting up your Portenta board and the Vision Shield - LoRa to communicate with a TTN application. As stated before, to be able to follow this guide, you need to be under coverage of one of the TTN gateways. You can check for the coverage now if you have not done so yet. [Ard24h]

- **1. Setting up the Environment** First choose your region. Next, sign in with your The Things Network account(Refer figure 16.14).

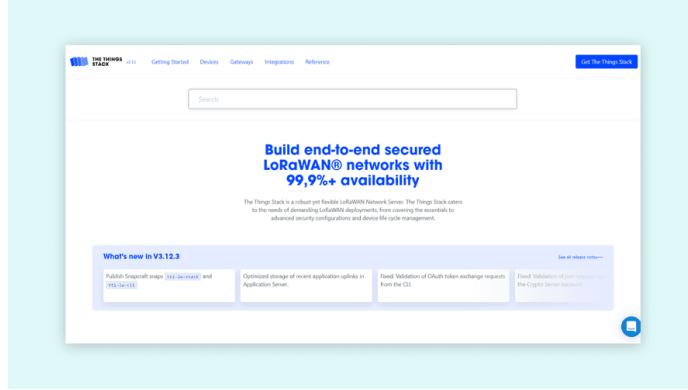


Figure 16.14.: Things Network

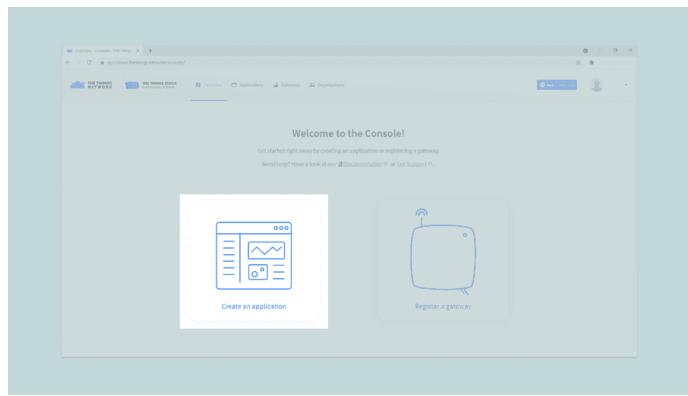


Figure 16.15.: Select Application

If you do not have an account, create a new one on the login page of Things Network. Then fill all the required fields to complete a new registration.

- **2. Creating an App on TTN** Once you have created an account with TTN, you need to create a TTN application(Refer figure 16.15). An application provides a way to aggregate data from different devices, and then use these data with other 3rd party integrations. After signing in, click on Create an application, or Go to applications if you already have created one. [Ard24h]

Here you will have a list of all your applications. Now create your first app by pressing the Create an application button.

You have now to fill only the first two fields:

1. The first one is the Owner of your app, it will automatically have you as the owner.

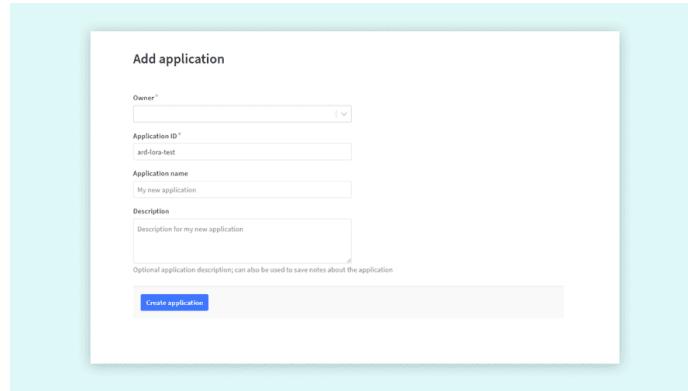


Figure 16.16.: AddingApplication

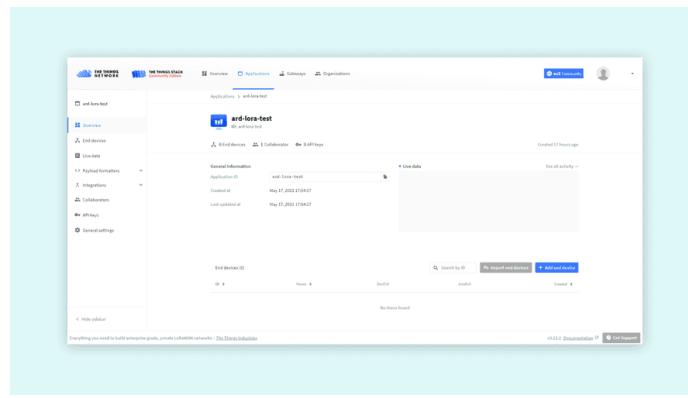


Figure 16.17.: AppParameter

2. The second one is the ID of your app: this must be lowercase and without spaces.

After completing these two fields, press the "Create application" button located at the bottom left corner of the page. The dashboard will then show you an overview of the newly created app. [Ard24h]

Let's take a closer look at these sections:

1. **Application Overview:** in order to use this app, you will need the Application ID and a device specific AppKey. An EUI is a globally unique identifier for networks, gateways applications and devices. The EUIs are used to identify all parts of the LoRaWAN inside the backend server.
2. **End devices:** here you can see and manage all the associated devices (e.g. your Portenta H7 with Portenta Vision Shield - LoRa, Arduino MKR WAN 1300 or MKR WAN 1310), or proceed with the registration of a new one. Registering a new

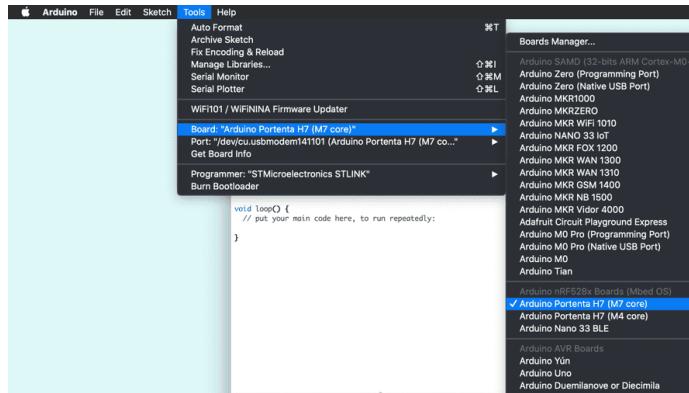


Figure 16.18.: MCORE

device lets you generate an AppEUI and an AppKey.

3. **Collaborators:** here you can see and manage all the app collaborators, to integrate with other collaborative platforms or to manage access rights to the app with other TTN registered profiles.
 4. **API keys:** here you can create an API key, it is the most sensible information. It is basically the key to gain access to your app, so keep it safe. [Ard24h]
- 3. **Configuring the Portenta Vision Shield** It is now time to connect your Portenta H7 and Portenta Vision Shield - LoRa to TTN. You will need to upload code to the board, so, as you probably already know, there are two options:
 - Use the Arduino Cloud Editor
 - Use the Arduino IDE, (this is the option this guide will follow)
 Plug the Portenta Vision Shield - LoRa to the Portenta H7 and them to your PC through the USB port. Be sure to have selected the right board "Arduino Portenta H7 (M7 core)" and the right port.(Refer figure 16.18) [Ard24h]

The LoRa module on the Portenta Vision Shield - LoRa can be accessed by using the **MKRWAN library**(if you cannot find it in your examples list, you can go to **Tools > Library Manager** and type "MKRWAN library" to install it). This library provides all the APIS to communicate with LoRa and LoRaWAN networks and can be installed from the library Manager. The first code you need to upload and run is from the **MKRWAN library**, and its name is **FirstConfiguration**.

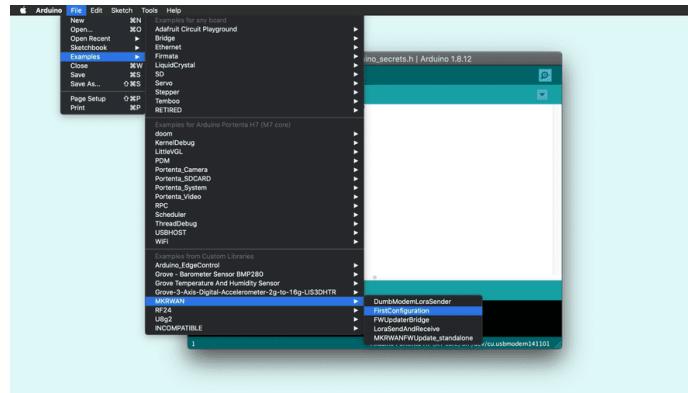


Figure 16.19.: UploadCode
[Ard24h]

The only line you may need to change before uploading the code is the one that sets the frequency. Set the frequency code according to your country if needed. You can find more information about frequency by country at this TTN link.

```

1 // change this to your regional band (eg. US915,
   AS923, ...)
2 if (!modem.begin(EU868)) { ... }
```

Once you have added to the sketch the frequency according to your country, you can upload it to the board. Then, when the upload is completed, open the Serial Monitor. The following details will show up:

```

1 Your module version is: ARD-078 1.2.1
2 Your device EUI is: a8xxxxxxxxxxxxxx
3 Are you connecting via OTAA (1) or ABP (2)?
```

In order to select the way in which the board is going to connect with TTN (OTAA or ABP), you need to configure it on the TTN portal. You will see which option you should select in the following steps. [Ard24h]

- **4. Registering the Portenta on TTN** Before your Portenta H7 can start communicating with the TTN, you need to register the board with an application(Refer figure 16.21). Go back to the TTN portal and scroll to End devices section on your Application dashboard, then click Add end device. [Ard24h]

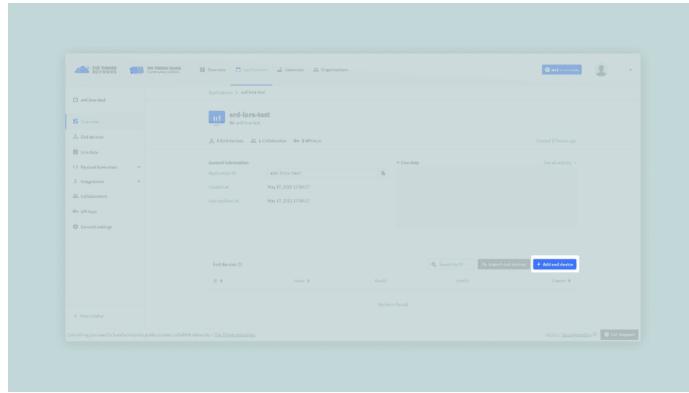


Figure 16.20.: RegisteringDevice

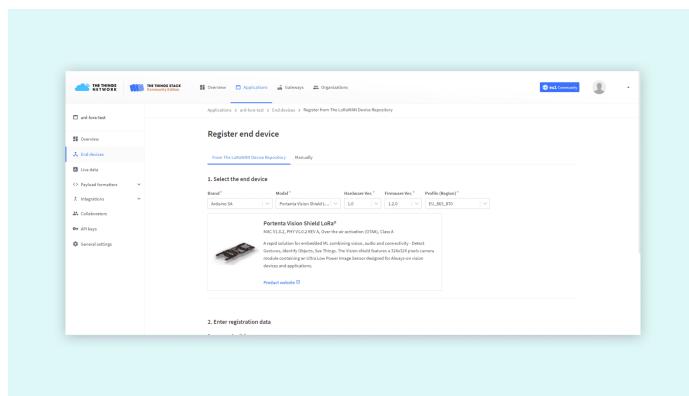


Figure 16.21.: RegisteringDevice

On the registration page, first you have to fill in information about your board(Refer figure 16.21). Select brand Arduino SA, and Portenta Vision Shield - LoRa as the model. Hardware and firmware versions will automatically be set to the newest ones. Then set your preferred region.

In the second step of registering the device, fill in End device ID and End device ID(Refer figure 16.22). You can click the generate button next to the AppKey field to generate an app key for this device. Similarly, you can press the button next to the AppEUI field to make it all zeros, or enter your own AppEUI.

Note: The Device ID must be lowercase and without spaces. The **DevEUI** should be copied from the Serial Monitor.

After pressing the Register button, your board will show up on the **Device Overview** page. You can now see all the information needed to complete the Arduino setup. [Ard24h]

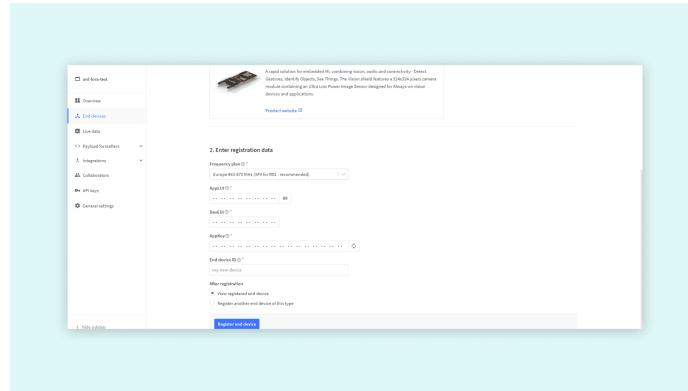


Figure 16.22.: Secondstep

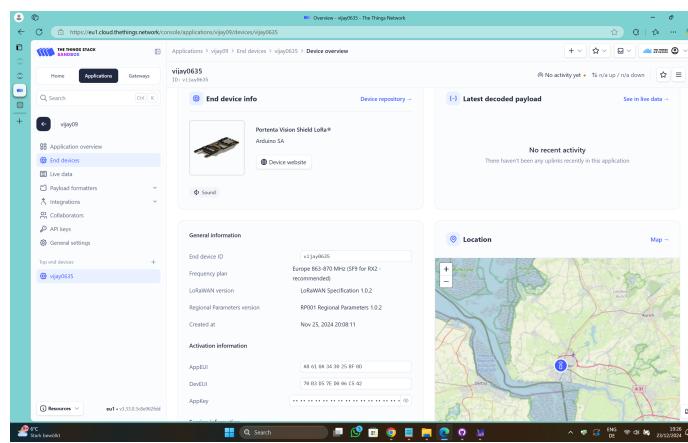


Figure 16.23.: TTN

For detailed explanation, please refer the following example 16.5.1.

16.5. Tests

16.5.1. Connecting the Portenta Vision Shield to TTN Using LoRa

This example explains how to connect your Portenta H7 to The Things Network (TTN) using the the Portenta Vision Shield's LoRa Connectivity feature. [Ard24h]

Overview

This example explains how to connect your Portenta H7 to The Things Network (TTN) using the the Portenta Vision Shield's LoRa Connectivity feature. A data communication channel will be enabled between the H7 and a TTN application that will be configured on your TTN console. [Ard24h]

Required Hardware and Software

1. Portenta H7
2. Portenta Vision Shield - LoRa
3. U.FL Antenna for Portenta H7
4. Arduino IDE 1.8.10+, Arduino IDE 2.0+, or the Arduino Cloud Editor
5. USB-C Cable
6. An Account with The Things Network [Ard24h]

Installing the LoRa Module Firmware

To be able to use the LoRa functionality, we need to first Install the firmware on the LoRa modem. This can be done through Arduino IDE by running a sketch included in the examples from the MKRWAN library.

1. Connect the Portenta H7 and the Portenta Vision Shield - LoRa to your computer and open the Arduino IDE.
2. Install/update the **MKRWAN** library from Arduino IDE menu **Tools > Manage Libraries**[Refer figure 16.24]. Type "MKRWAN" to find the library and click 'Install' or 'Update' if necessary. This library provides all the APIs to communicate with LoRa and LoRaWAN networks.
3. Open the MKRWANFWUpdate standalone sketch from the Arduino IDE menu: **File > Examples > MKRWAN**.(Refer figure 16.25)
4. Upload the sketch. [Ard24h]

Open the Serial Monitor and wait for the update to be confirmed.(Refer figure 16.26)

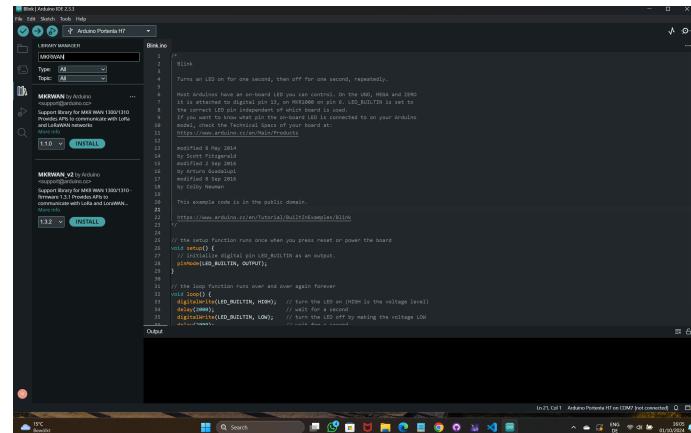


Figure 16.24.: Install MKRWAN

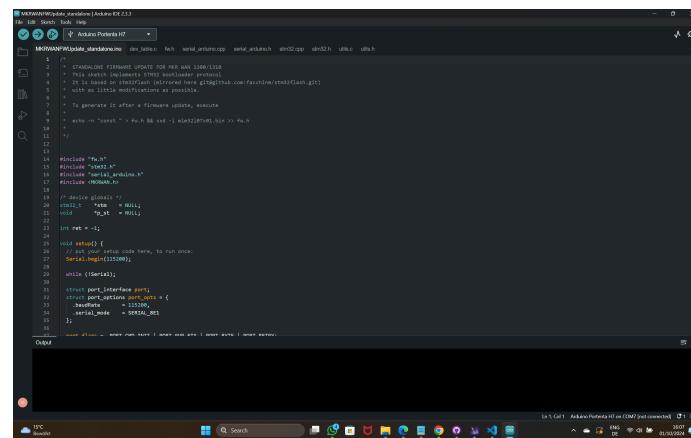


Figure 16.25.: MKRWAN Standalone

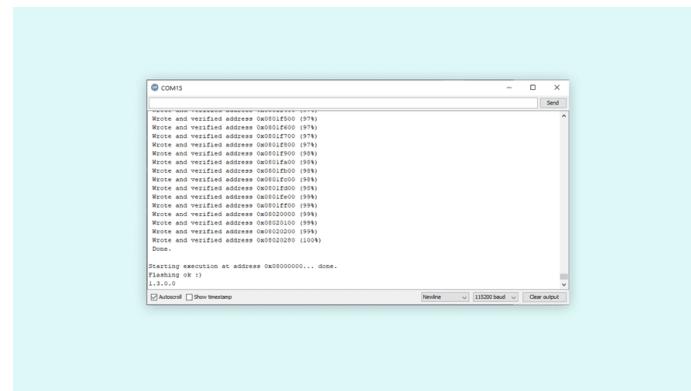


Figure 16.26.: SerialMonitor

Connecting to TTN

Once your board has been registered you can send information to TTN. Let's come back to the Serial Monitor and proceed. It will ask for: (Refer figure 14.4)

1. Activation mode (that, in this case, is OTAA),
2. The Application EUI
3. The App Key.

Lets start by making a connection Over-The-Air (OTA). Enter "1" in the Serial Monitor input box and press ENTER. Then, find the EUI and the App key from TTN [Device Overview](#) page. You can read more into OTA vs ABP activation mode here.

```
1 Your module version is: ARD-078 1.1.9
2 Your device EUI is: a8xxxxxxxxxxxx0a
3 Are you connecting via OTAA (1) or ABP (2)?
4 Enter your APP EUI
5 Enter your APP KEY
```

Next, introduce the APP EUI and the APP KEY in the Serial Monitor. If this process is done successfully, you will see this message: [Ard24h]

```
1 Message sent correctly!
```

else,

- **6. Conclusion:** If you receive this message, you have managed to configure the Portenta H7 and the Portenta Vision Shield - LoRa on TTN.

You have retrieved the device EUI, used it to register the device in the TTN console, and programmed the board using the data provided by TTN. Now, you can send data over the LoRa network which can be viewed from anywhere in the world (as long as we have an Internet connection and your device is in the range of a TTN gateway). [Ard24h]

16.5.2. Designing a Double LoRa Connectivity for the Arduino Portenta H7

Machine learning (ML) is nowadays deployed in ever smaller computing devices including microcontrollers found in Internet of Things (IoT)

```

1 // Firm Configuration
2 // This sketch demonstrates the usage of WiFi module 1808/1810 LoRa module.
3 // This example code is in the public domain.
4 //
5
6 #include <Arduino.h>
7
8 // WiFi module
9 // WiFi module uses the WiFi chip as a module
10 // (so there's no external WiFi antenna)
11
12 // String appID;
13 // String appKey;
14 // String makeID;
15 // String makeKey;
16
17 void setup() {
18     // run your setup code here, to run once
19     // before the loop() runs
20     while (!Serial);
21     Serial.begin(9600); // Connect to WiFi web server
22     // Set regional band (e.g. US/US, Asia/Asia, ...).
23     // Change this to your regional band
24     WiFi.setRegion(WIFI_REGION_ASIA);
25     // Set your favorite WiFi network and we are "ready" to go!
26     // change this to your WiFi network name
27     WiFi.begin("Your WiFi network name");
28     // Print WiFi module version
29     Serial.println("WiFi module version: " + WiFi.firmwareVersion());
30     // Print WiFi module MAC address
31     Serial.print("WiFi module MAC: ");
32     WiFi.macAddress();
33     // Print WiFi module IP address
34     Serial.print("IP Address: ");
35     WiFi.localIP();
36 }
37
38 void loop() {
39     // read message to send message to Arduino Portenta H7 over WiFi
40     if (Serial.available() > 0) {
41         String message = Serial.readStringUntil('\n');
42         Serial.println("Message received from Arduino Portenta H7: " + message);
43     }
44 }

```

Figure 16.27.: Monitor Output

nodes [1]. Signal processing and ML tasks are performed directly at the IoT device, communicating only the result of a classification to remote gateways instead of the raw data [2]. LoRa is a popular communication technology for the IoT. LoRa links can cover several km of distance between the end node, which produces the data, and the gateway. Most IoT applications that use LoRa apply the LoRaWAN architecture [3]. This architecture builds a star topology which establishes a single hop between the end nodes and the gateway. LoRaWAN, however, does not directly interconnect the end nodes between themselves. [Ano22]

LoRa mesh networks have been proposed for diverse scenarios which cannot be addressed well by the LoRaWAN architecture [4]. For geographically spread IoT nodes and low gateway density, multi-hop LoRa can be used where intermediate nodes operate as repeaters that broadcast traffic to other LoRa nodes to finally reach a gateway [5]. Another scenario are gateway-less applications such as Meshtastic [6]. In this real application the nodes communicate only within the LoRa mesh network, without any gateway to the Internet. Given the potential of LoRa mesh networks, in this paper we investigate the possibility of the Arduino Portenta H7 for becoming a node of a LoRa mesh network. The Portenta H7 is a recent microcontroller board which targets high performance industrial machine learning applications, being equipped for this with a dual core and very complete on-board sensors. We focus on the Portenta's LoRa connectivity challenge given that the board in practical applications has already been shown to be well prepared for running embedded machine learning applications [Ano22]

DESIGN

A: Analysis of the Arduino LoRa Vision Shield

The Arduino Portenta H7 is a board with two cores, a Cortex M7 running at 480 MHz and a Cortex M4 running at 240 MHz, to which shields can be connected in order to extend the board with additional features such as sensors(Refer figure 16.28). The shield which provides LoRa connectivity is the Arduino Portenta LoRa Vision Shield2. This shield contains the Murata CMWX1ZZABZ-078 module3, which is internally composed of the SX1276 LoRa radio chip from Semtech, together with other elements for signal modulation, reception and transmission and the STM32L0 microcontroller. [Ano22]

Our aim is to analyze if the Arduino Portenta LoRa Vision Shield supports RadioLib4, a popular communication library which allows to implement LoRa point-to-point communications. From analyzing the Murata datasheet we observed, however, that even though the module has SPI connectivity, it is not physically implemented in the Arduino Portenta Vision Shield and therefore the Arduino Portenta H7 cannot communicate directly with the Murata's SX1276 LoRa radio through this communication protocol. This fact disables the use of RadioLib which does require the SPI communication. While we found that the use of RadioLib was not possible, the MKRWANlibrary is officially compatible with the Portenta LoRa Vision Shield. Using the MKRWAN library we observed that this library communicates with the Murata module over AT commands, which allows to manage the LoRaWAN connectivity of the radio. For operating AT commands, the serial bus named UART8 in the Portenta Vision Shield is connected to the pins named Serial3 of the Portenta board. Therefore, the firmware implemented in the Murata module requires the radio to be connected to a LoRaWAN network. However, with AT commands as the only access to the module,it is not possible to conduct operations below the LoRaWAN layer,which is needed when we want to use the radio in a LoRa mesh network. In addition, it was pointed out that the current implementation of the MKRWAN library does not support some relevant AT commands. [Ano22]

B. The choice of the double LoRa connectivity

Our analysis reveled that while the Murata firmware supports LoRaWAN, it does not allow to operate on the layer below to manage basic LoRa packets. There have been efforts by the community to replace the Murata firmware, but the result is experimental and there is no official support for the Portenta LoRa Vision Shield6, We therefore chose the option to connect a second LoRa radio to the Arduino Portenta H7 and connect

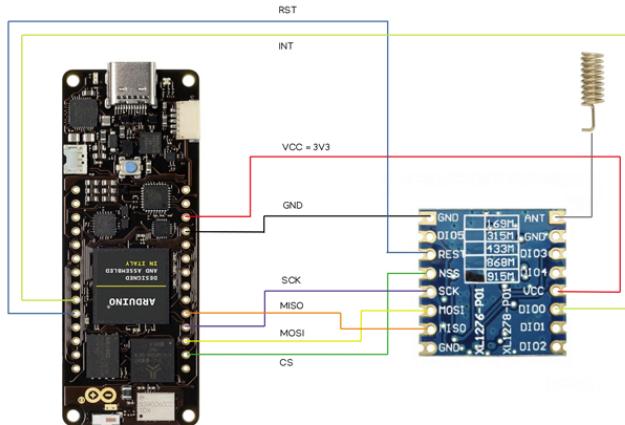


Figure 16.28.: Connection for SX1276 chip to the Arduino Portenta H7

it as shown in Figure 16.29. The SCK, MISO, MOSI and CS signal connections belong to the SPI protocol. In addition, the connection with the SX1276 requires the interrupt and reset signals to correctly handle sending and receiving messages (INT and RST signals, respectively). [Ano22]

III. LIBRARY COMPATIBILITY

In order to validate the design, we first aim to verify that the new LoRa radio added to the Portenta supports the Arduino LoRa library⁷. We flash existing code fragments for LoRa sender and receiver to two Portenta H7 boards and could verify the successful communication between the two boards using the added LoRa radio. The second test aims to validate that RadioLib can be used. With the additional LoRa radio board the SPI interface required by RadioLib is available. It needs to be mentioned that in order to use RadioLib, another interrupt signal has to be added by physically connecting the Portenta H7 with the DIO1 signal of the radio board, which finalizes the reception timer. Using a LoRa sender and receive code the communication between two Portenta boards over the added LoRa radio using RadioLib could be verified. An OLED display was used to verify the successful communication. [Ano22]

IV. CONCLUSIONS AND OUTLOOK

We designed a double LoRa connectivity for the Arduino Portenta H7 to enable this board to become part of a LoRa mesh network. The new design extends the board's application to scenarios in which the device not only reaches the LoRaWAN gateway, but can also be part of a LoRa

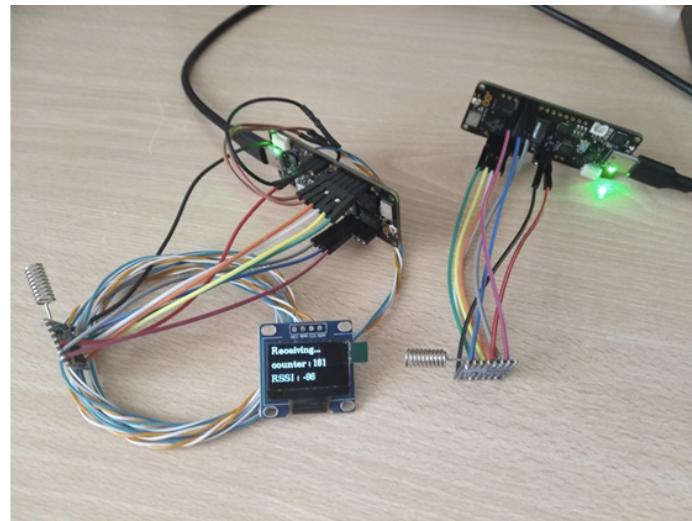


Figure 16.29.: Communication test of two Arduino Portenta H7 by LoRa
[Ano22]

mesh network. In future work we aim to integrate the new communication capacity into the Mbed OS available for the Arduino Portenta H7 and use it for interconnected machine learning applications. [Ano22]

16.5.3. LoRaWAN vision with Edge Impulse and Portenta H7

This is an example application that runs a computer vision model on the Portenta H7 and streams the results over LoRaWAN. The application uses the camera on the Portenta Vision Shield in combination with a machine learning model trained in Edge Impulse to determine when an interesting event happens, then sends this using the LoRa radio on the Portenta Vision Shield back to the network. [Impnd]

Required Hardware and Software

1. Portenta H7
2. Portenta Vision Shield - LoRa
3. U.FL Antenna for Portenta H7
4. Arduino IDE 1.8.10+, Arduino IDE 2.0+, or the Arduino Cloud Editor
5. USB-C Cable
6. An Account with The Things Network [Impnd]

This firmware integrates the Portenta H7 board with the Vision Shield (LoRa) for Edge Impulse-based machine learning and LoRaWAN communication. Below is a breakdown of the code: [Impnd]

Header Files and Dependencies

```
1 #include "ei_main.h"
2 #include "ei_device_portenta.h"
3 #include "porting/ei_classifier_porting.h"
4 #include "ei_portenta_fs_commands.h"
5 #include "ei_microphone.h"
6 #include "ei_config_types.h"
7 #include "at_cmd_repl_mbed.h"
8 #include "ei_run_impulse.h"
9 #include "sensors/ei_camera.h"
10 #include <MKRWAN.h>
```

Listing 16.1: Included Header Files

These headers provide support for:

- Edge Impulse functionalities for AI/ML model inference.

- LoRaWAN communication via `MKRWAN.h`.
- AT command handling for serial communication.

LoRa Modem and LED Initialization

```
1 LoRaModem modem;
2 mbed::DigitalOut led(LED1);
```

Listing 16.2: LoRa Modem and LED Setup

- `LoRaModem modem`; - Initializes the LoRa modem.
- `led(LED1)`; - Controls the onboard LED.

AT Command Interpreter Setup

```
1 EventQueue main_application_queue;
2 static unsigned char repl_stack[8 * 1024];
3 static AtCmdRepl repl(&main_application_queue,
4     ei_get_serial(), sizeof(repl_stack), repl_stack, 5)
5 ;
```

Listing 16.3: AT Command Interpreter Setup

This initializes:

- An event queue for managing background tasks.
- AT Command REPL (Read-Eval-Print-Loop) for serial interactions.

LoRaWAN Credentials

```
1 static String appEui = "70B3D57ED003BFCA";
2 static String appKey = "1
3     F537E0DDCFcff43443004F3A88C4294";
```

Listing 16.4: LoRaWAN Credentials

These credentials authenticate the device with The Things Network (TTN).

Memory Allocation Function

```

1 void fill_memory() {
2     size_t size = 8 * 1024;
3     size_t allocated = 0;
4     while (1) {
5         void *ptr = ei_malloc(size);
6         if (!ptr) {
7             if (size == 1) break;
8             size /= 2;
9         }
10        else {
11            allocated += size;
12        }
13    }
14    ei_printf("Allocated: %u bytes\n", allocated);
15 }
```

Listing 16.5: Memory Allocation

This function tests available RAM by attempting allocations in 8KB chunks.

LoRaWAN Connection and Data Transmission

```

1 void lorawan_main() {
2     ei_printf("Hello from the LoRaWAN thread...\n");
3     ThisThread::sleep_for(5000);
4
5     if (!modem.begin(EU868)) {
6         ei_printf("Failed to initialize modem\n");
7     }
8
9     ei_printf("[LRWN] Device EUI is: %s\n", modem.
10        deviceEUI().c_str());
11
12    bool connected = false;
13    while (!connected) {
14        ei_printf("[LRWN] Joining network...\r\n");
15        connected = modem.joinOTAA(appEui, appKey);
16        if (connected) {
17            ei_printf("[LRWN] Joined network!\r\n");
18        }
19    }
20 }
```

```

19         ei_printf("[LRWN] Joining network failed,
20                     retrying in 10 seconds...\r\n");
21     }
22 }
23 }
```

Listing 16.6: LoRaWAN Connection

This function:

- Connects the device to TTN using Over-The-Air Activation (OTAA).
- Retries every 10 seconds if the connection fails.

Sending AI Inference Results via LoRa

```

1 while (1) {
2     if (is_event_changed && ei_read_timer_ms() -
3         last_message_sent >= 7000) {
4         if (strcmp(last_event, "uncertain") == 0 || strcmp(
5             last_event, last_message_str) == 0) {
6             ThisThread::sleep_for(1000);
7             continue;
8         }
9
10        ei_printf("[LRWN] Sending event: '%s'\n",
11                  last_event);
12        modem.dataRate(5);
13        modem.setPort(3);
14        modem.beginPacket();
15        modem.print(last_event);
16        int err = modem.endPacket(false);
17        ei_printf("[LRWN] Sending event: '%s' returned %d\n"
18                  ", last_event, err);
19
20        memcpy(last_message_str, last_event, 31);
21        last_message_sent = ei_read_timer_ms();
22        is_event_changed = false;
23    }
24    ThisThread::sleep_for(1000);
25 }
```

Listing 16.7: Sending AI Events via LoRa

This transmits detected events over LoRa every 7 seconds, ignoring uncertain classifications.

Running Edge Impulse Model

```

1 void run_nn_at() {
2     ei_at_cmd_handle("AT+RUNIMPULSE");
3 }
```

Listing 16.8: Executing Edge Impulse Model

This function executes the Edge Impulse ML model when triggered via AT command.

Main Execution Flow (ei_main)

```

1 void ei_main() {
2     ei_sleep(1000);
3
4     ei_printf("Hello from Edge Impulse Device SDK.\r\n"
5             "Compiled on %s %s\r\n", __DATE__, __TIME__);
6
7     ei_portenta_fs_init();
8     ei_camera_init();
9
10    ei_at_register_generic_cmds();
11    ei_at_cmd_register("RUNIMPULSE", "Run the impulse",
12                        run_nn_with_cb);
13    ei_at_cmd_register("RUNIMPULSEDEBUG", "Run the
14                        impulse", run_nn_debug_with_cb);
15    ei_at_cmd_register("FILLMEMORY", "Fill memory",
16                        fill_memory);
17
18    Thread lorawan_thread;
19    lorawan_thread.start(&lorawan_main);
20
21 }
```

Listing 16.9: Main Execution Function

This initializes Edge Impulse, registers AT commands, starts the LoRaWAN thread, and enters the main event loop. [Impnd]

Full Code:

Conclusion

This firmware:

The full project and source code are available here: https://github.com/edgeimpulse/example-portenta-lorawan/blob/master/src/ei_main.cpp

- Runs an Edge Impulse ML model for event detection.
- Connects to TTN over LoRaWAN.
- Sends detected events via LoRa.
- Supports AT command interaction. [Impnd]

Part V.

Tools and Packages

17. TensorFlow Lite

17.1. TensorFlow

TensorFlow is a free and open-source software library, tool or a platform for machine learning and artificial intelligence, developed by the Google Brain team. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks. [Goo24o]

TensorFlow uses dataflow graphs to represent computation, shared state, and the operations that mutate that state. It can train and run deep neural networks for applications such as handwritten digit classification, image recognition, word embeddings, recurrent neural networks, sequence-to-sequence models for machine translation, natural language processing, and PDE (partial differential equation) based simulations. TensorFlow maps the nodes of a dataflow graph across many machines in a cluster and within a machine across multiple computational devices, including multicore CPUs, general-purpose GPUs, and custom-designed ASICs known as Tensor Processing Units (TPUs). [Goo24n] [Aba+16]

17.2. pyTorch

PyTorch is an open-source machine learning library, based on the Torch library, developed by Facebook's AI Research lab. Although Pytorch is launched after TensorFlow, it has quickly gained popularity due to its ease of customization. Unlike static computation graphs, PyTorch allows users to define and manipulate the graph dynamically during runtime, which makes debugging and model development more easier. This flexibility is highly beneficial for tasks such as natural language processing, computer vision, and reinforcement learning. PyTorch supports both CPU and GPU computation, offering seamless acceleration using CUDA for high-performance GPU computations. PyTorch's framework is also deeply integrated with Python, making it a preferred choice for researchers and developers. [Wik24] [Met24]

17.3. JAX

JAX is a open source machine learning library developed by Google that is designed for high-performance numerical computing and automatic

differentiation. Unlike traditional deep learning libraries, JAX is built around NumPy and works with various existing frameworks such as TensorFlow and PyTorch with automatic differentiation and it is highly optimised to run on CPUs, GPUs and TPUs. [JAX24]

17.4. Keras

Keras is a high-level deep learning Application Programming Interface (API) written in Python for fast experimentation with neural networks. Initially developed as an independent library, Keras is now tightly integrated into TensorFlow and serves as its official high-level API. Now Keras 3, multi-framework deep learning API is a full rewrite of Keras that enables you to run your Keras workflows on top of either JAX, TensorFlow, or PyTorch, and that unlocks brand new large-scale model training and deployment capabilities. Keras is powerful and can handle a wide variety of tasks such as image classification, natural language processing, and reinforcement learning. It supports both CPU and GPU computation and can be deployed on mobile and web applications. [Cho15]

17.5. TensorFlow Lite

Tensorflow Lite is an optimised environment of TensorFlow models, which is specifically designed for mobile and edge devices, allowing machine learning models to run efficiently on smartphones, embedded systems and other devices with limited computational resources. Essentially it consists of two components, the model converter, which converts pre-trained TensorFlow models into optimised TensorFlow Lite models and the Interpreter, which enables efficient execution on end devices. [Goo24h] TensorFlow Lite has now been officially renamed to LiteRT. Now, LiteRT

(formerly TensorFlow Lite) grown beyond its TensorFlow roots to support models from other frameworks like PyTorch, JAX, and Keras. LiteRT continues to serve as the optimized environment for running AI models on mobile devices, embedded systems, and other edge devices, focusing on reducing model size, latency, and energy consumption. The name change reflects how this high-performance runtime has evolved beyond supporting only TensorFlow models to also include models from other frameworks like PyTorch, JAX, and Keras. LiteRT continues to serve as the optimized environment for running AI models on mobile devices, embedded systems, and other edge devices, focusing on reducing model size, latency, and energy consumption. [Li24]

17.6. Why 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 optimization in terms of size and speed, so that the models can then be run with TensorFlow Lite interpreters on mobile, embedded and IoT devices. [Goo24i]

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. 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. [Goo24b]

17.7. Working Procedure of TensorFlow Lite

TensorFlow Lite workflow consists of four steps as illustrated in the figure 17.1 Here's how TensorFlow Lite works: [Goo24h]

1. Model Creation
2. Model Conversion
3. Model Deployment
4. Model Optimization

17.7.1. Model Creation and Training

The first step involves creating and training a machine learning model using TensorFlow.

TensorFlow Lite uses TensorFlow models converted into a smaller, more

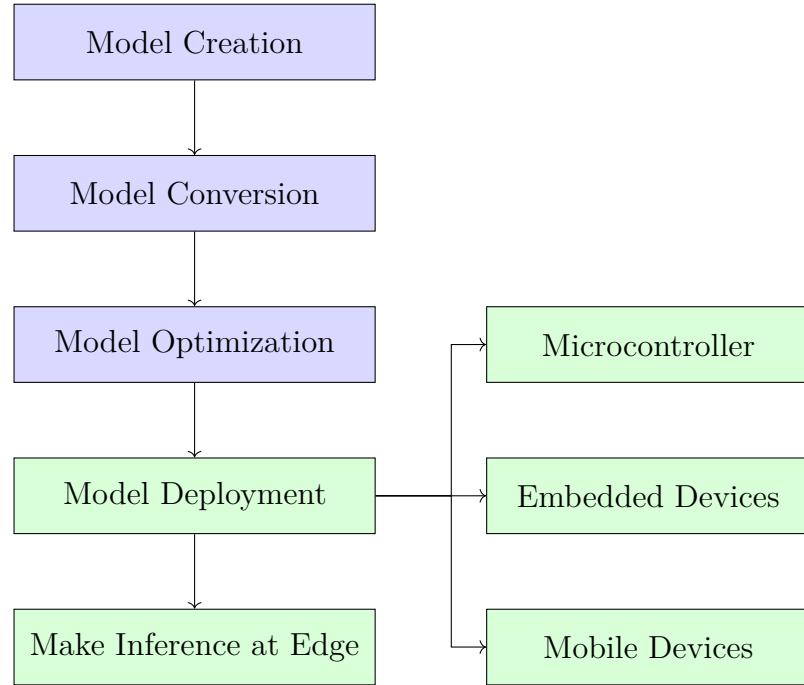


Figure 17.1.: TensorLite Workflow

efficient machine learning (ML) model format. You can use pre-trained models with TensorFlow Lite, modify existing models, or build your own TensorFlow models and then convert them to TensorFlow Lite format. TensorFlow Lite models can perform almost any task a regular TensorFlow model can do: object detection, natural language processing, pattern recognition, and more using a wide range of input data including images, video, audio, and text. [Goo24h]

The listing 17.1 explains the workflow of creating and training a simple neural network model using TensorFlow's Keras API. Also it simplifies a simple example of image classification involving steps in preparing data from MNIST dataset, building, training, evaluating the model for further conversion to TensorFlow Lite format. [Goo24g]

17.7.2. Model Conversion

Once the model is trained, it needs to be converted into a TensorFlow Lite model. The TensorFlow Lite converter takes a TensorFlow model and generates a TensorFlow Lite model, an optimized FlatBuffer format identified by the file extension `.tflite`. [Goo24l]

The model conversion can be done using the Python API or the Command line tool

- Command-line Tool: `tflite convert`, this allows you to integrate the conversion into your development pipeline, apply optimizations,

```
1 import tensorflow as tf
2 from tensorflow.keras.models import Sequential
3 from tensorflow.keras.layers import Dense, Flatten
4 from tensorflow.keras.optimizers import Adam
5
6 # Load the MNIST dataset
7 mnist = tf.keras.datasets.mnist
8 (x_train, y_train), (x_test, y_test) = mnist.load_data()
9
10 # Normalize the images to the range [0, 1]
11 x_train, x_test = x_train / 255.0, x_test / 255.0
12
13 # Define a simple Sequential model
14 model = Sequential([
15     Flatten(input_shape=(28, 28)), # Flatten the 28x28
16     # images into a 1D array of 784 elements
17     Dense(128, activation='relu'), # Hidden layer with 128
18     # neurons and ReLU activation
19     Dense(10, activation='softmax') # Output layer with 10
20     # neurons for classification
21 ])
22
23 # Compile the model with an optimizer, loss function,
24 # and metric
25 model.compile(optimizer=Adam(),
26                 loss='sparse_categorical_crossentropy',
27                 metrics=['accuracy'])
28
29 # Train the model on the training data
30 model.fit(x_train, y_train, epochs=5, validation_data=(
31     x_test, y_test))
32
33 # Evaluate the model on the test data
34 test_loss, test_acc = model.evaluate(x_test, y_test,
35     verbose=2)
36 print(f'\nTest accuracy: {test_acc:.4f}')
37
38 # Save the trained model as a SavedModel format (for
39 # future conversion to TFLite)
40 model.save('saved_model/my_mnist_model')
```

Listing 17.1: Model Creation and Training Example Using TensorFlow

add metadata and many other tasks that simplify the conversion process.

- Python API: `tf.lite.TFLiteConverter`, this only supports basic model conversion.

The process involves following steps:

- **Select the Model:** The converter accepts the following input model formats.
 - `SavedModel`: A TensorFlow model saved as a set of files on disk. `tf.lite.TFLiteConverter.from_saved_model(model)`
 - `Keras model`: A model created using the high level Keras API. `tf.lite.TFLiteConverter.from_keras_model(model)`
 - `Keras H5 format`: A light-weight alternative to SavedModel format supported by Keras API. `tf.lite.TFLiteConverter.from_keras_model(model_loaded_from_model.h5)`
 - `Models built from concrete functions`: A model created using the low level TensorFlow API. `tf.lite.TFLiteConverter.from_concrete_functions([func], model)`

The listing 17.2 provides the APIs for the above mentioned models.
[Goo24f]

- **Optimize the Model:** During conversion, various optimization techniques can be applied to make the model more efficient. These includes quantization, Pruning, Clustering.

The goal of these optimizations is to make the model smaller and faster while maintaining accuracy.

17.7.3. Model Deployment

After converting the model to a file `.tflite`, it can be deployed on a variety of devices including mobile phones, embedded systems, and microcontrollers. TensorFlow Lite supports deployment on both Android and iOS platforms, as well as other systems like Arduino.

17.7.4. Running Inference

The term inference refers to the process of executing a TensorFlow Lite model on-device in order to make predictions based on input data. To perform an inference with a TensorFlow Lite model, you must run it through an interpreter. The TensorFlow Lite interpreter is designed to be lean and fast. The interpreter uses a static graph ordering and a custom

```
1 # Converting a SavedModel to a TensorFlow Lite model.
2 converter = tf.lite.TFLiteConverter.from_saved_model(
3     model)
4 tflite_model = converter.convert()
5
6 # Converting a tf.Keras model to a TensorFlow Lite model.
7 converter = tf.lite.TFLiteConverter.from_keras_model(
8     model)
9 tflite_model = converter.convert()
10
11 # Converting a Keras H5 model to a TensorFlow Lite model.
12 model = tf.keras.models.load_model('model.h5')
13 converter = tf.lite.TFLiteConverter.from_keras_model(
14     model)
15 tflite_model = converter.convert()
16
17 # Converting ConcreteFunctions to a TensorFlow Lite model
18 .
19 converter = tf.lite.TFLiteConverter.
20     from_concrete_functions([func], model)
21 tflite_model = converter.convert()
22
23 # Converting a Jax model to a TensorFlow Lite model.
24 converter = tf.lite.TFLiteConverter.experimental_from_jax(
25     [
26         [func], [[ ('input1', input1), ('input2', input2)]]])
27 tflite_model = converter.convert()
```

Listing 17.2: Python Code for Converting Models to .tflite

```

1 // Include the library
2 #include "tensorflow/lite/micro/all_ops_resolver.h"
3 #include "tensorflow/lite/micro/micro_error_reporter.h"
4 #include "tensorflow/lite/micro/micro_interpreter.h"
5 #include "tensorflow/lite/schema/schema_generated.h"
6 #include "tensorflow/lite/version.h"
7
8 // Load model
9 const tflite::Model* model = ::tflite::GetModel(g_model);
10 if (model->version() != TFLITE_SCHEMA_VERSION) {
11     TF_LITE_REPORT_ERROR(error_reporter,
12         "Model provided is schema version %d not equal "
13         "to supported version %d.\n",
14         model->version(), TFLITE_SCHEMA_VERSION);
15     return;
16 }
```

Listing 17.3: Loading and Validating a TensorFlow Lite Model for Microcontroller

(less-dynamic) memory allocator to ensure minimal load, initialization, and execution latency. [Goo24j] [Goo24d]

Here's the typical process:

- **Load the Model:** The TensorFlow Lite model file `.tflite` is loaded into the memory, which contains the model's execution graph. The listing 17.3 provides the code for Loading and Validating a TensorFlow Lite Model for Microcontroller.
- **Allocate Tensors:** Build an Interpreter based on an existing model, that is allocating memory for the input and output tensors. The listing 17.4 provides the reference code for Allocating Tensors for TensorFlow Lite Model Execution on Microcontroller.
- **Set Input Data:** Set input tensor values (Optionally resize input tensors if the predefined sizes are not desired) and fed into the model. The listing 17.5 gives the reference code for setting Input Data for TensorFlow Lite Inference on a Microcontroller.
- **Invoke the Interpreter:** The interpreter runs the model with the input data to produce the output. The listing 17.6 gives the reference code for invoking TensorFlow Lite Interpreter to perform Inference on a Microcontroller.
- **Get Output Data:** The output data is extracted and post-processed to get the final predictions. The listing 17.7 gives the reference

```
1 // Define the memory needed by the model
2 constexpr int kTensorArenaSize = 2 * 1024;
3 uint8_t tensor_arena[kTensorArenaSize];
4
5 // Instantiate Interpreter
6 tflite::MicroInterpreter interpreter(model, resolver,
7     tensor_arena,
8     kTensorArenaSize, error_reporter);
9
10 TfLiteStatus allocate_status = interpreter.
11     AllocateTensors();
12 if (allocate_status != kTfLiteOk) {
13     TF_LITE_REPORT_ERROR(error_reporter, "AllocateTensors()
14         failed");
15     return;
16 }
```

Listing 17.4: Allocating Tensors for TensorFlow Lite Model Execution

```
1 // Get input tensor and populate with input data
2 TfLiteTensor* input = interpreter.input(0);
3 for (int i = 0; i < input->bytes; i++) {
4     input->data.uint8[i] = your_input_data[i]; // Populate
5         with input data
6 }
```

Listing 17.5: Setting Input Data for TensorFlow Lite Inference

```
1 // Run inference
2 TfLiteStatus invoke_status = interpreter.Invoke();
3 if (invoke_status != kTfLiteOk) {
4     TF_LITE_REPORT_ERROR(error_reporter, "Invoke failed");
5     return;
6 }
```

Listing 17.6: Invoking TensorFlow Lite Interpreter to perform Inference
on a Microcontroller

```
1 // Get output tensor and process the result
2 TfLiteTensor* output = interpreter.output(0);
3 uint8_t* results = output->data.uint8; // Process the
   output data as needed
```

Listing 17.7: Extracting and Processing Output Data from a TensorFlow Lite Model

code Extracting and Processing Output Data from a TensorFlow Lite Model on a Microcontroller

Tensors in TensorFlow Lite

In TensorFlow Lite, a tensor is a multi-dimensional array that represents the basic data structure used by the library. Just like a vector is a one-dimensional array and a matrix is a two-dimensional array, a tensor can be n-dimensional. Tensors can have different shapes and data types, and they are used for both input and output of the model. [Goo24j]

18. Comprehensive TensorFlow Lite Usage Methodologies

This section broadly covers the various contexts in which TensorFlow Lite is used, including its role in microcontrollers (Arduino), as a tool for model conversion and optimization and as a library path in larger projects.

18.1. TensorFlow Lite as a Library for Arduino

TensorFlow Lite for Microcontrollers is a version of TensorFlow Lite designed to run machine learning models on microcontroller devices, such as those based on the Arduino platform. This library is highly optimized to work within the constraints of small devices with limited resources (e.g., very low memory and processing power). [Goo24p]

18.1.1. Usage and Integration

When you use TensorFlow Lite on an Arduino, you typically include the TensorFlow Lite Micro library in your Arduino IDE. This allows you to write programs that can perform tasks like speech recognition, gesture detection, or even basic image classification directly on the Arduino. The library is optimized to run on devices with very limited resources, such as minimal RAM and processing power. The library is installed into the Arduino development environment, and you can include it in your projects by adding the appropriate headers and linking against the library when compiling your code. This allows your Arduino sketches to utilize TensorFlow Lite models. [Goo24p]

18.1.2. TensorFlow Lite Micro Arduino Example

TensorFlow Lite Micro Library for Arduino repository contains examples needed to use Tensorflow Lite Micro on an Arduino. To implement, first clone the TensorFlow Lite Micro Arduino Examples repository from GitHub into a local directory called [Arduino_TensorFlowLite](#). This repository contains examples for using TensorFlow Lite on microcontrollers like Arduino for various machine learning tasks, such as image

```
C:\Users\mselv>git clone https://github.com/tensorflow/tflite-micro-arduino-examples Arduino_TensorFlowLite
Cloning into 'Arduino_TensorFlowLite'...
remote: Enumerating objects: 3207, done.
remote: Counting objects: 100% (1193/1193), done.
remote: Compressing objects: 100% (348/348), done.
remote: Total 3207 (delta 1193), reused 1055 (delta 1055), pack-reused 1804 (from 1)
Receiving objects: 100% (3207/3207), 36.01 MiB | 2.28 MiB/s, done.
Resolving deltas: 100% (2217/2217), done.
Updating files: 100% (592/592), done.
C:\Users\mselv>
```

Figure 18.1.: TFlite Micro github Cloning to local repository

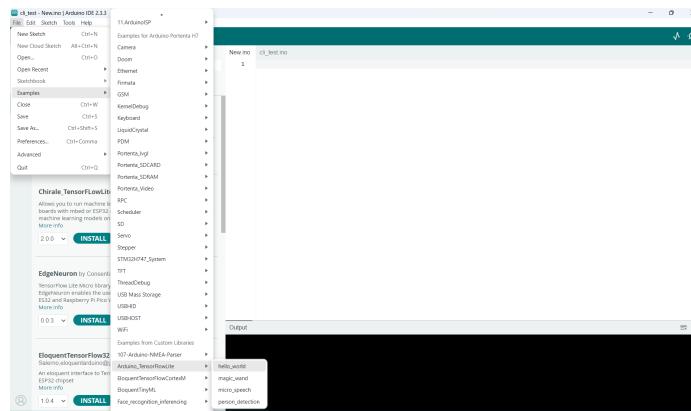


Figure 18.2.: TensorFlow Lite as a Library for Arduino

classification and voice recognition, optimized for low-power devices like Arduino. [Goo24p].

Run the command `git clone https://github.com/tensorflow/tflite-micro-arduino-examples` as per figure 18.1

Once the library is installed, Open Arduino IDE, go to `Files > Examples > Arduino_TensorFlowLite > hello_world` as shown in figure 18.2.

This example program, `hello_world`, is designed to replicate a sine wave using a TensorFlow Lite model. The model generates a pattern that varies the brightness of the built-in LED on an Arduino in a pattern similar to the last predicted sine value. Since the value ranges from -1 to 1, we could represent 0 with a fully off LED, -1 and 1 with a fully lit LED, and all intermediate values with a partially dimmed LED. As the program loops, making inferences, the LED fades in and out repeatedly. To dim our built-in LED, we use a technique called pulse width modulation (PWM). If we turn an output pin on and off extremely quickly, the output voltage of the pin is set to a factor of ratio the time spend between on and off states. If the pin spends half of the time in each state, its output voltage will be half of its maximum (50 percent). With the `constantInferencesPerCycle` the number of inferences performed over a full sine cycle can be varied. Since an inference takes a certain amount of time, by setting the `constantInferencesPerCycle`, can be set how quickly the LED fades out. [Goo24p]

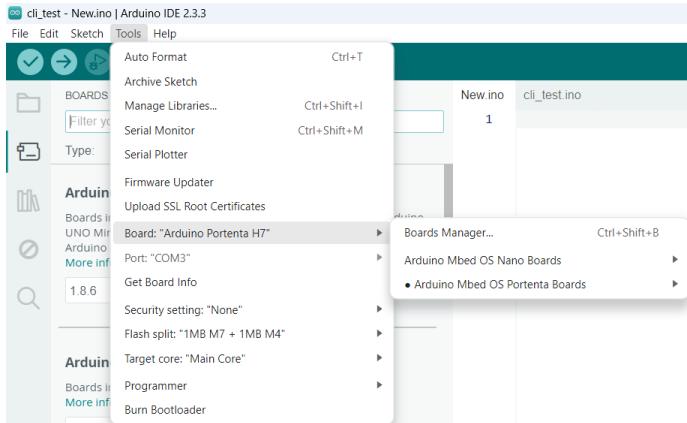


Figure 18.3.: Board and Port selection

First, we include some header files in the source code (refer 18.1). These headers include the essential TensorFlow Lite for Microcontrollers (TFLM) files, allowing the model and interpreter to function.

The global variables define pointers to the model, interpreter, input/output tensors, and a tensor arena for storing intermediate data. The `setup()` function initializes the TensorFlow Lite model and sets up the necessary components such as Model Initialization, Operations Resolver `AllOpsResolver` object used to load all necessary operations for the model, Interpreter setup, Tensor Allocation and Tensor Pointers.

The `loop()` function performs continuous inferences using the sine wave model. The loop function first calculates the current x-value based on `inference_count`, then quantizes the x-value to an integer format, reducing memory and computation requirements. It executes the model inference, updating the output tensor with the result. Following this, it dequantizes the output back to floating-point, generating the sine wave value y. The `HandleOutput()` function is used to update the LED based on the sine wave pattern. Finally, it increments the `inference_count` and resets it after completing a full sine wave cycle.

To run the example, connect your Arduino device via USB. Make sure the correct device type is selected in the Board drop-down list in the Tools menu, **Tools > Board**. If the device not appears in the list, go to **Tools > Board > Board Manager** and in the window appears, install the latest version of the supported board package. Then make sure the device port is selected as shown in the figure 18.3 [Goo24p].

Then finally, in the Arduino window, click the Upload button to compile the code and upload it to your Arduino device. After the upload has completed successfully, you should see the LED on your Arduino board start to either fade in and out or blink on and off, depending on whether the pin it is connected to supports PWM.

```

1 #include <TensorFlowLite.h>
2
3 #include "constants.h"
4 #include "main_functions.h"
5 #include "model.h"
6 #include "output_handler.h"
7 #include "tensorflow/lite/micro/all_ops_resolver.h"
8 #include "tensorflow/lite/micro/micro_interpreter.h"
9 #include "tensorflow/lite/micro/micro_log.h"
10 #include "tensorflow/lite/micro/system_setup.h"
11 #include "tensorflow/lite/schema/schema_generated.h"
12
13 // Globals, used for compatibility with Arduino-style
14 // sketches.
14 namespace {
15 const tflite::Model* model = nullptr;
16 tflite::MicroInterpreter* interpreter = nullptr;
17 TfLiteTensor* input = nullptr;
18 TfLiteTensor* output = nullptr;
19 int inference_count = 0;
20
21 constexpr int kTensorArenaSize = 2000;
22 // Keep aligned to 16 bytes for CMSIS
23 alignas(16) uint8_t tensor_arena[kTensorArenaSize];
24 } // namespace
25
26 // The name of this function is important for Arduino
27 // compatibility.
27 void setup() {
28   tflite::InitializeTarget();
29
30   // Map the model into a usable data structure. This
31   // doesn't involve any
32   // copying or parsing, it's a very lightweight
33   // operation.
34   model = tflite::GetModel(g_model);
35   if (model->version() != TFLITE_SCHEMA_VERSION) {
36     MicroPrintf(
37       "Model provided is schema version %d not equal "
38       "to supported version %d.",
39       model->version(), TFLITE_SCHEMA_VERSION);
40     return;
41   }
42
43   // This pulls in all the operation implementations we
44   // need.
45   // NOLINTNEXTLINE(runtime-global-variables)
46   static tflite::AllOpsResolver resolver;
47
48   // Build an interpreter to run the model with.
49   static tflite::MicroInterpreter static_interpreter(
50     model, resolver, tensor_arena, kTensorArenaSize);
51   interpreter = &static_interpreter;

```

This example demonstrates a simple end-to-end workflow for using TensorFlow Lite as a Library for Arduino Microcontrollers by training and deploying a model to Arduino.

18.2. TensorFlow Lite as a Library Path

In software development, TensorFlow Lite as a library path refers to integrating TFLite as a reusable resource into a broader project, whether it's a mobile application, embedded system, or even a desktop application. The concept of a "library path" in this context refers to how TFLite's library is organized, included, and linked within a project's build system, such as CMake, Bazel, or Makefiles, to streamline access across multiple components [Git24].

When TensorFlow Lite is treated as an external library, it provides a centralized resource that enables different parts of a project to access TFLite's models, functions, and interpreter without duplicating or moving source files.

18.2.1. Usage and Integration:

In building an application with TensorFlow Lite, ensuring that the development environment has access to the TFLite libraries is essential for linking during the build process. This often involves specifying paths in the build configuration files (such as CMakeLists.txt or BUILD files) to include TensorFlow Lite's headers and binary files [Goo24a]. A "library path" refers to the directory where TensorFlow Lite binaries, headers, and other resources are stored. This path is specified in the build configuration so that the compiler and linker can locate TensorFlow Lite components seamlessly.

In large project setup, TensorFlow Lite enables inference on pre-trained models, providing localized AI functionalities directly on the device. By setting TFLite up as a library path, this approach allows each application component to perform machine learning tasks independently, with consistent access to TFLite's resources.

Directory Structure Example

For an organized project, here's an example directory structure 18.4 that illustrates how TensorFlow Lite might be set up as a library path, In this structure:

- `libs/tensorflow_lite` acts as the main library path for TensorFlow Lite.

```

1 set(TFLITE_LIB_PATH "${CMAKE_SOURCE_DIR}/libs/
  tensorflow_lite")
2 include_directories("${TFLITE_LIB_PATH}/include")
3 target_link_libraries(my_project_target "${{
  TFLITE_LIB_PATH}/lib/libtensorflowlite.a"})

```

Listing 18.2: CMake file

```

1 cc_binary(
2   name = "my_project",
3   srcs = ["src/main.cpp"],
4   deps = ["@tensorflow_lite//tensorflow/lite:micro"],

```

Listing 18.3: Bazel BUILD file

- The `include` folder contains the headers needed to access TensorFlow Lite functions, such as `micro_interpreter.h` and `all_ops_resolver.h`, enabling components to directly include these headers.
- `lib/libtensorflowlite.a` is the precompiled TensorFlow Lite library, which can be linked in the build configuration.

Build System Integration

For larger projects, common build systems like CMake, Bazel, or Make can be configured to set up this library path. [Goo24a]

- **CMake:** In your `MakeLists.txt`, define the TensorFlow Lite path using `target_link_libraries` and `include_directories` as shown in 18.2
- **Bazel:** In Bazel’s `BUILD` file, TFLite can be added as an external dependency by adding it to the `deps` list as shown in 18.3
- **Makefile:** For a Makefile-based project, specify paths using `LD-FLAGS` (linker flags) and `CFLAGS` (compiler flags) as shown in 18.4.

```

1 TFLITE_LIB_PATH = ./libs/tensorflow_lite
2 CFLAGS += -I$(TFLITE_LIB_PATH)/include
3 LDFLAGS += $(TFLITE_LIB_PATH)/lib/libtensorflowlite.a

```

Listing 18.4: Make file

18.3. TensorFlow Lite as a Tool

TensorFlow Lite as a tool primarily refers to the package of utilities and functions provided by TensorFlow Lite to convert, optimize, and prepare TensorFlow models for deployment on edge devices, such as mobile phones, microcontrollers and IoT devices. The primary goal of using TensorFlow Lite as a tool is to make machine learning models smaller, faster, and more efficient without compromising much on accuracy. [Goo24h]

- **Model Conversion:** The core of TensorFlow Lite as a tool is the TensorFlow Lite Converter, refer 17.7.2. This tool converts a standard TensorFlow model into a `.tflite` TensorFlow Lite model format.
- **Model Optimization:** TensorFlow Lite offers several optimization techniques that can be applied during the conversion process or as part of a separate step.

18.3.1. Relationship to Other TensorFlow Lite functionality

The use of TensorFlow Lite as a tool is complementary to using TensorFlow Lite as a library path or as a library for Arduino. While the library path provides the necessary runtime environment for deploying models, and the Arduino library allows integration on microcontrollers [Goo24e], the tools ensure that the models are optimized for these environments. The conversion and optimization steps are typically performed once and the resulting model format `.tflite` is then deployed across various platforms and devices. [Goo24i]

In a typical development workflow, a developer would first train a model using TensorFlow, convert and optimize it using TensorFlow Lite tools, and then deploy it using the appropriate TensorFlow Lite library, whether on a mobile device, an Arduino board, or another embedded system. This sequence ensures that the model is not only effective but also efficient and suitable for real-world constraints.

18.4. TensorFlow Lite for Microcontrollers

Arduino microcontrollers are widely used in IoT, prototyping, and educational projects due to their simplicity and versatility. Although traditionally used for basic sensor and actuator tasks, Arduino boards can be extended into machine learning applications through the use of TensorFlow Lite, which optimizes machine learning models to run on devices with limited computational resources. Machine learning models are trained using

TensorFlow on more powerful platforms such as PCs or cloud-based TPUs. The trained models are then converted into a TensorFlow Lite format, which is optimized for edge devices. The TensorFlow Lite models are deployed onto Arduino microcontrollers, particularly those that support AI tasks, such as Arduino Portenta H7. Once deployed, the microcontroller can run real-time inference tasks, enabling on-device decision-making for applications like object detection, face and gesture recognition and sensor data analysis. [Goo24e]

18.4.1. Integrating Edge TPU with Arduino

For applications requiring more computational power than a microcontroller alone can provide, Google Coral devices equipped with Edge TPUs provide a significant performance boost. Google Coral is a platform that features Edge TPUs, specialized accelerators designed for running high-performance AI inference at the edge. By integrating Coral devices, such as the Coral USB Accelerator into an Arduino-based setup, we can achieve powerful AI capabilities with minimal latency and power consumption. [Goo24c]

The Coral USB Accelerator is a plug-and-play device that connects via USB to a host system, such as a Raspberry Pi or a PC, that can interface with Arduino boards. This setup allows the microcontroller to handle control logic and interfacing tasks, while the Edge TPU executes complex AI inferences, significantly enhancing performance.

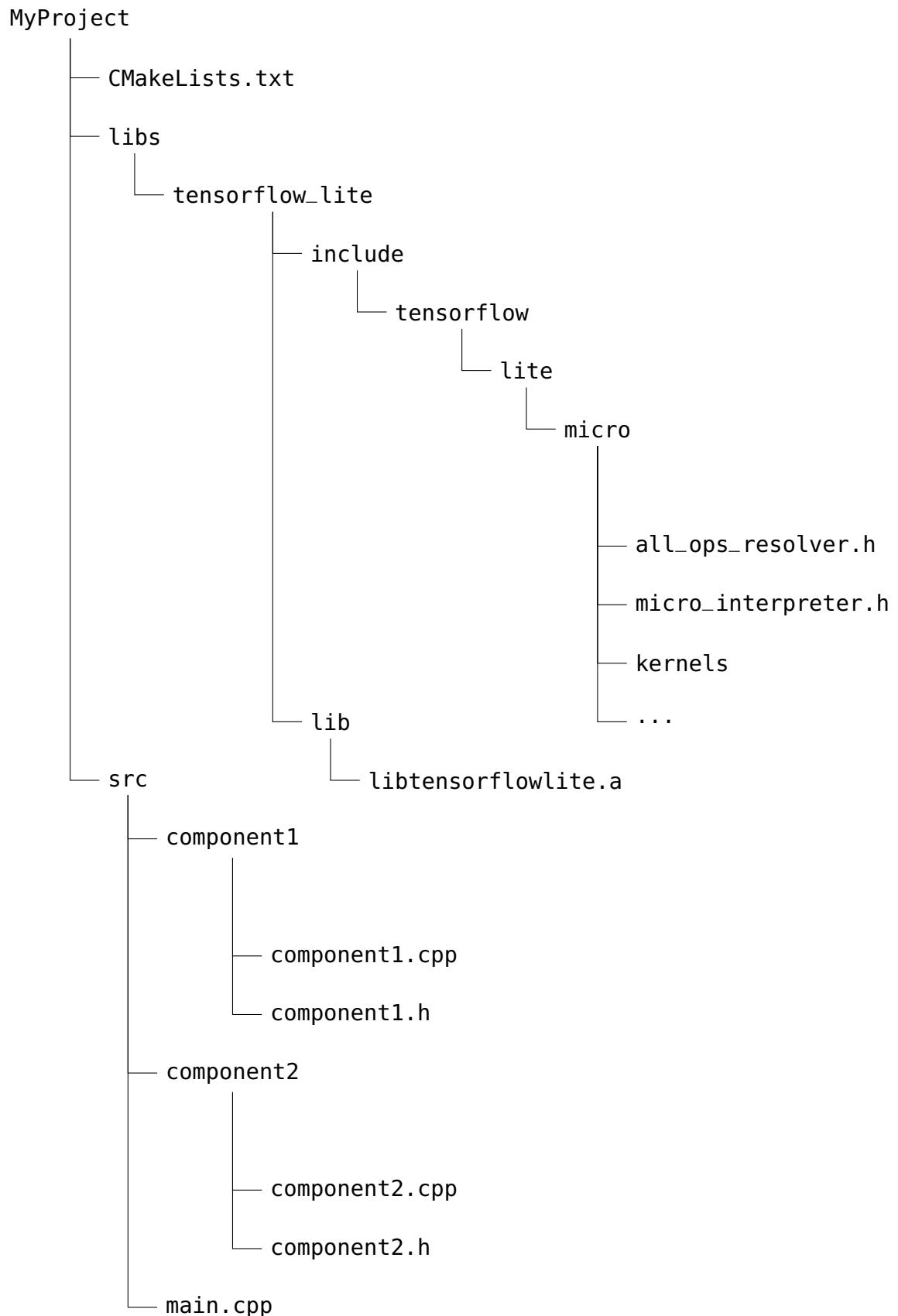


Figure 18.4.: Directory structure with TensorFlow Lite as Library Path

19. Model Optimization

Edge devices often have limited memory or computational power. Various optimizations can be applied to models so that they can be run within these constraints. In addition, some optimizations allow the use of specialized hardware for accelerated inference.

It's recommended that you consider model optimization during your application development process. This document outlines some best practices for optimizing TensorFlow models for deployment to edge hardware.

19.1. Why models should be optimized?

Over the last few years, machine learning models have seen two seemingly opposing trends. On the one hand, the models tend to get bigger and bigger, culminating in what's all the rage these days: the large language models. Nvidia's Megatron-Turing Natural Language Generation model has 530 billion parameters! On the other hand, these models are being deployed onto smaller and smaller devices, such as smartwatches or drones, whose memory and computing power are naturally limited by their size. How do we squeeze ever larger models into increasingly smaller devices? The answer is model optimization: the process of compressing the model in size and reducing its latency. [Shu24]

There are several main ways model optimization can help with application development. [Goo24m]

19.1.1. Size reduction

- **Smaller storage size:** Smaller models occupy less storage space on your users' devices. For example, an Android app using a smaller model will take up less storage space on a user's mobile device.
- **Smaller download size:** Smaller models require less time and bandwidth to download to users' devices.
- **Less memory usage:** Smaller models use less RAM when they are run, which frees up memory for other parts of your application to use, and can translate to better performance and stability. [Goo24m]

19.1.2. Latency reduction

Latency is the amount of time it takes to run a single inference with a given model. Some forms of optimization can reduce the amount of computation required to run inference using a model, resulting in lower latency. Latency can also have an impact on power consumption.

Currently, quantization can be used to reduce latency by simplifying the calculations that occur during inference, potentially at the expense of some accuracy. [Goo24m]

19.1.3. Accelerator compatibility

Some hardware accelerators, such as the Edge TPU, can run inference extremely fast with models that have been correctly optimized.

Generally, these types of devices require models to be quantized in a specific way. See each hardware accelerator's documentation to learn more about their requirements. [Goo24m]

19.1.4. Trade-offs

Optimizations can potentially result in changes in model accuracy, which must be considered during the application development process.

The accuracy changes depend on the individual model being optimized, and are difficult to predict ahead of time. Generally, models that are optimized for size or latency will lose a small amount of accuracy. Depending on your application, this may or may not impact your users' experience. In rare cases, certain models may gain some accuracy as a result of the optimization process. [Goo24m]

19.2. Types of Optimization

TensorFlow Lite currently supports optimization via quantization, pruning and clustering.

These are part of the TensorFlow Model Optimization Toolkit, which provides resources for model optimization techniques that are compatible with TensorFlow Lite.

19.2.1. Quantization

Quantization works by reducing the precision of the numbers (number of bits used to represent a number in memory) used to represent a model's parameters, which by default are 32-bit floating point numbers. This results in a smaller model size and faster computation. [Goo24m]

By default, TensorFlow stores model biases, weights, and activations as 32-bit floating points. Typically, most model weights and activations are not that far away from zero; otherwise, the gradients would explode preventing us from training a model in the first place. Converting these float32s into a lighter data structure such as int8 could go a long way toward reducing the model size, while not necessarily impacting its accuracy.

Quantization is very convenient to use since it operates on an already trained model and only converts its internal data structures. In TensorFlow, it can be done while converting the model to the TF Lite format by setting the optimizations attribute in the converter. [Shu24]

The following types of quantization are available in TensorFlow Lite:

1. Post-training float16 quantization
2. Post-training dynamic range quantization
3. Post-training integer quantization
4. Quantization-aware training

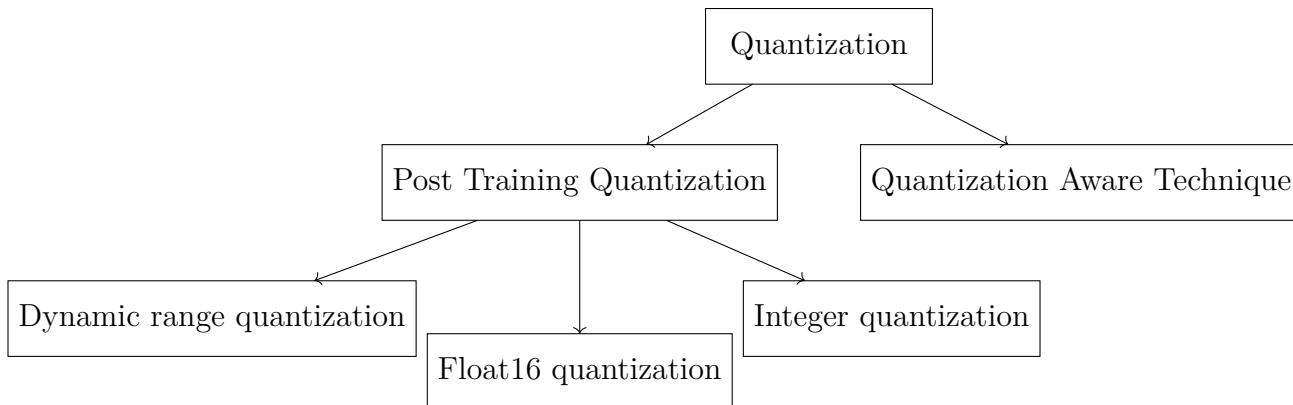


Figure 19.1.: Quantization Techniques

Post-training float16 quantization

In Float-16 quantization, weights are converted to 16-bit floating-point values. This results in a 2x reduction in model size. There is a significant reduction in model size in exchange for minimal impacts to latency and accuracy. [Ope23] [Goo24m]

Post-training dynamic range quantization

In Dynamic Range Quantization, weights are converted to 8-bit precision values. Dynamic range quantization achieves a 4x reduction in the model

size. There is a significant reduction in model size in exchange for minimal impacts to latency and accuracy. [Ope23] [Goo24m]

Post-training integer quantization

Integer quantization is an optimization strategy that converts 32-bit floating-point numbers (such as weights and activation outputs) to the nearest 8-bit fixed-point numbers. This resulted in a smaller model and increased inferencing speed, which is valuable for low-power devices such as microcontrollers. [Ope23] [Goo24m]

Quantization Aware Training

Quantization reduces the precision of the weights and activations to lower bits. Quantizing a model after training once usually leads to lower performance in smaller models. For increasing its performance at lower bit width, models can be trained/finetuned with quantized weights and activations.

First, the model architecture is adjusted to store the full precision and quantized copy of elements. A fake/simulated quantization is introduced to the model in the forward pass making it experience the effects of quantization. The gradients are calculated without loss of precision making it robust to quantization. The quantized values of weights and activations are stored in nodes and gradients are passed through them during training. The figure 19.2 represents the Quantization Aware Training technique by simulating the effects of quantization during training.

Once the model is trained only the quantized model is used for inference. [Dec24]

19.2.2. Pruning

Pruning works by removing parameters within a model that have only a minor impact on its predictions. Pruned models are the same size on disk, and have the same runtime latency, but can be compressed more effectively. This makes pruning a useful technique for reducing model download size. [Goo24m]

Simply, Pruning is an optimization technique that simplifies neural networks by reducing redundancy without significantly impacting task performance. In the figure 19.3, a neural network's structure before and after pruning. On the left is the original dense network with all the connections and neurons intact. On the right, the network has been simplified through pruning: less important connections (synapses) and neurons have been removed.

Pruning is based on the observation that not all neurons contribute equally to the output of a neural network. Identifying and removing

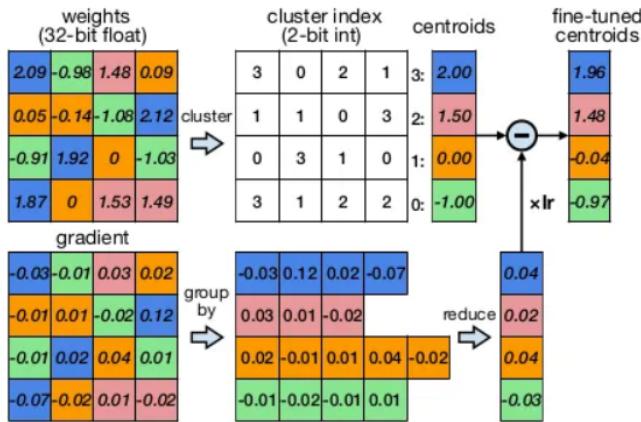


Figure 19.2.: Quantization Aware Training Technique by introducing simulated quantization during model training

the less important neurons can substantially reduce the model's size and complexity without negatively impacting its predictive power. It involves three key phases: identification, elimination, and fine-tuning. [Nep24]

19.2.3. Clustering

Clustering works by grouping the weights of each layer in a model into a predefined number of clusters, then sharing the centroid values for the weights belonging to each individual cluster. This reduces the number of unique weight values in a model, thus reducing its complexity. [Goo24m] As a result, clustered models can be compressed more effectively, providing deployment benefits similar to pruning.

Weight clustering is an optimization algorithm to reduce the storage and network transfer size of your model. The idea in a nutshell is explained in the figure 19.4. For example, that a layer in your model contains a 4x4 matrix of weights (represented by the “weight matrix”). Each weight is stored using a float32 value. When you save the model, you are storing 16 unique float32 values to disk.

Weight clustering reduces the size of your model by replacing similar weights in a layer with the same value. These values are found by running a clustering algorithm over the model’s trained weights. The user can specify the number of clusters (in this case, 4). This step is shown in “Get centroids” in the diagram and the 4 centroid values are shown in the “Centroid” table. Each centroid value has an index (0-3).

Next, each weight in the weight matrix is replaced with its centroid’s index. This step is shown in “Assign indices”. Now, instead of storing

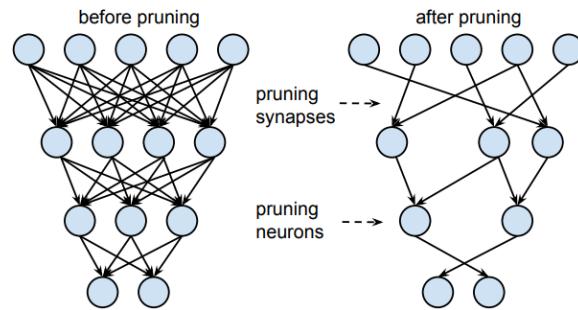


Figure 19.3.: Synapses and neurons in Deep Learning Model before and after pruning technique

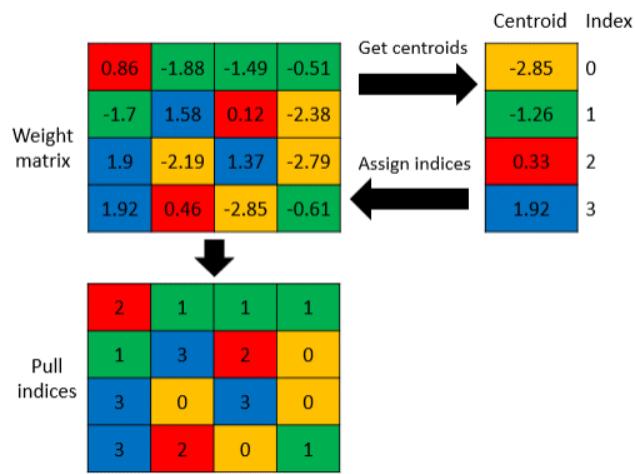


Figure 19.4.: Weight Clustering Optimization technique

the original weight matrix, the weight clustering algorithm can store the modified matrix shown in “Pull indices” (containing the index of the centroid values), and the centroid values themselves.

In this case, we have reduced the size from 16 unique floats, to 4 floats and 16 2-bit indices. The savings increase with larger matrix sizes.

Note that even if we still stored 16 floats, they now have just 4 distinct values. Common compression tools (like zip) can now take advantage of the redundancy in the data to achieve higher compression. [Goo21] [HMD17]

20. TensorFlow Installation on PC

Generally, TensorFlow Lite can be used by installing TensorFlow, as TensorFlow Lite is included as part of the TensorFlow package.

20.1. System Requirements

TensorFlow is tested and supported on the following 64-bit systems: [Goo24k]

- Ubuntu 16.04 or later
- Windows 7 or higher
- macOS 10.12.6 (Sierra) or later
- WSL2 via Windows 10 19044 or higher including GPUs

20.2. Software Requirements

- Python 3.9–3.12
- pip version 19.0 or higher for Linux (requires manylinux2014 support) and Windows. pip version 20.3 or higher for macOS
- Windows Native Requires Microsoft Visual C++ Redistributable for Visual Studio 2015, 2017 and 2019 [Goo24k]

20.3. Installation procedure

- Install the latest version of Python 3.12.3 using the following link <https://www.python.org/downloads/>, refer figure 20.1
- **pip** is the package installer for Python. It should be installed by default with Python
- Verify the version of python and pip using the command **py --version** and **pip --version** as shown in figure 22.3



Figure 20.1.: Python Installation

```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.22631.3880]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\System32>py --version
Python 3.12.3

C:\Windows\System32>pip --version
pip 24.0 from C:\Program Files\Python312\Lib\site-packages\pip (python 3.12)
C:\Windows\System32>
```

Figure 20.2.: Python and pip version

- TensorFlow can be installed using pip. Open Command Prompt or PowerShell and run `pip install tensorflow` 20.3
- To verify the installation, open a Python shell and type `python` and import TensorFlow by typing `import tensorflow as tf`, after importing TensorFlow, you can check the version using `print(tf.__version__)` 20.4 [Goo24k]

20.4. TensorFlow Lite Library Installation on Arduino IDE

For deploying models on microcontrollers (like Portenta H7), TensorFlow Lite Micro libraries are used within Arduino IDE. Open Arduino IDE, navigate to **Sketch > Include Library > Manage Libraries** to include TensorFlow lite library [EloquentTinyML](#), an Arduino library to include TensorFlow Lite Micro for ARM Cortex-M chips (Arduino Portenta H7). 20.5

```
C:\Windows\system32>pip install tensorflow
Collecting tensorflow
  Downloading tensorflow-2.17.0-cp312-cp312-win_amd64.whl.metadata (3.2 kB)
Collecting tensorflow-intel==2.17.0 (from tensorflow)
  Downloading tensorflow_intel-2.17.0-cp312-cp312-win_amd64.whl.metadata (5.0 kB)
Collecting absl-py==1.0.0 (from tensorflow-intel==2.17.0->tensorflow)
  Using cached absl_py-1.0.0-py3-none-any.whl.metadata (2.3 kB)
Collecting astunparse==1.6.0 (from tensorflow-intel==2.17.0->tensorflow)
  Using cached astunparse-1.6.0-py2.py3-none-any.whl.metadata (4.4 kB)
Collecting flatbuffers==24.3.25 (from tensorflow-intel==2.17.0->tensorflow)
  Using cached flatbuffers-24.3.25-py2.py3-none-any.whl.metadata (856 bytes)
Collecting gast<0.5.0,>=0.5.1,!=0.5.2,>0.2.1 (from tensorflow-intel==2.17.0->tensorflow)
  Downloading gast-0.6.0-py3-none-any.whl.metadata (1.3 kB)
Collecting grpcio==1.47.0 (from tensorflow-intel==2.17.0->tensorflow)
  Using cached grpcio-1.47.0-py3-none-any.whl.metadata (814 bytes)
Collecting h5py==3.10.0 (from tensorflow-intel==2.17.0->tensorflow)
  Using cached h5py-3.10.0-cp312-cp312-win_amd64.whl.metadata (2.5 kB)
Collecting libclang==13.0.0 (from tensorflow-intel==2.17.0->tensorflow)
  Using cached libclang-18.1.1-py2.py3-none-win_amd64.whl.metadata (5.3 kB)
Collecting m1_dtypes<0.5.0,>=0.3.1 (from tensorflow-intel==2.17.0->tensorflow)
  Downloading m1_dtypes-0.4.0-cp312-cp312-win_amd64.whl.metadata (20 kB)
Collecting opt-einsum==2.3.2 (from tensorflow-intel==2.17.0->tensorflow)
  Using cached opt_einsum-2.3.2-py3-none-any.whl.metadata (1.8 kB)
Collecting packaging (from tensorflow-intel==2.17.0->tensorflow)
  Downloading packaging-24.1-py3-none-any.whl.metadata (3.2 kB)
Collecting protobuf<4.21.0,>=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 (from tensorflow-intel==2.17.0->tensorflow)
  Downloading protobuf-4.25.4-cp310-abi3-win_amd64.whl.metadata (541 bytes)
```

Figure 20.3.: TensorFlow Installation on PC

```
C:\Windows\System32>python
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr  9 2024, 14:05:25) [MSC v.1938 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import tensorflow as tf
2024-07-27 21:11:20.566100: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You
2024-07-27 21:11:20.566100: I different numerical results due to floating-point round-off errors from different computation orders.
2024-07-27 21:11:20.566100: I the environment variable `TF_ENABLE_ONEDNN_OPTS=0` .
2024-07-27 21:11:28.024618: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You
2024-07-27 21:11:28.024618: I different numerical results due to floating-point round-off errors from different computation orders.
2024-07-27 21:11:28.024618: I the environment variable `TF_ENABLE_ONEDNN_OPTS=0` .
>>> print(tf.__version__)
2.17.0
>>>
```

Figure 20.4.: TensorFlow Version

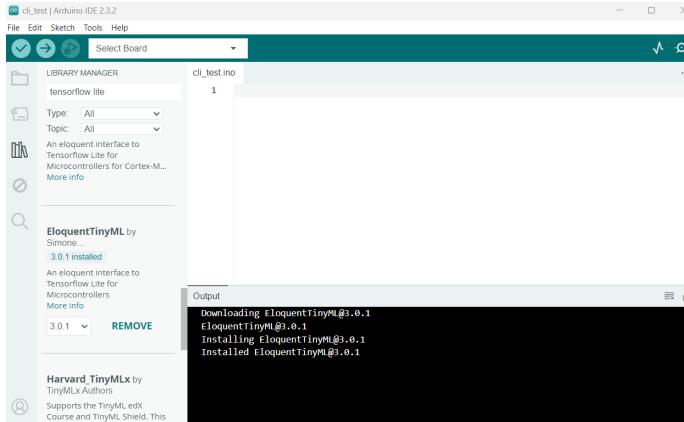


Figure 20.5.: Arduino TensorFlowLite Library Installation

21. MicroPython & OpenMV

21.1. Introduction

In this chapter, we will explore MicroPython and OpenMV in detail. We will discuss key features of the MicroPython programming language and how to run MicroPython scripts on the Arduino Portenta H7 using the OpenMV IDE. As a practice, we will be creating a basic MicroPython script in the OpenMV IDE to blink LED's on the Arduino Portenta H7 board.

21.2. MicroPython

MicroPython is a open source, tiny Python programming language interpreter that works on small embedded development boards. Instead of using complex low-level languages like C or C++, MicroPython allows us to write clean and simple Python code to control hardware (what Arduino uses for programming). It is a lean and efficient implementation of the Python 3 programming language that includes a small subset of the Python standard library and is optimised to run on microcontrollers and in constrained environments. MicroPython runs on a wide range of microcontrollers, as well as on Unix-like (including Linux, BSD, macOS, WSL) and Windows systems and it is compact enough to fit and run within just 256k of code space and 16k of RAM. [Mic24a] [Mic24b] Key features of MicroPython such as :

- **Interactive REPL(Read-Evaluate-Print-Loop):** MicroPython includes a REPL, which allows us to interact with the system in real time, testing and running code without the need for compilation. It is a four step process in which it first reads the user input, evaluate the code, print any results and loop back to first step.
- **External Libraries:** MicroPython supports a wide range of external libraries (e.g., machine learning, network communication, sensor drivers) that can be added to a project to extend its functionality.
- **Extendability:** We can write custom modules in C or C++ to extend the capabilities of MicroPython, providing optimized code for performance-critical parts or interfacing with specific hardware.

- **Cross-Platform:** It can be deployed on a variety of platforms, including ARM Cortex-M microcontrollers, ESP8266/ESP32, Raspberry Pi Pico, Portenta H7 and more.

The MicroPython programming language implements the core python 3 language but it can't implement the entire standard library of Python 3 because trying to fit big libraries into tiny boards with just kilobytes of memory is not possible. One thing to note is that MicroPython is only a programming language interpreter whereas Arduino is an entire ecosystem which has the Arduino IDE and the hardware which in this case is Arduino Portenta H7.

21.3. OpenMV

OpenMV is an open-source platform designed to simplify the integration of computer vision and machine learning into embedded systems. It allows us to run real-time vision algorithms on embedded devices using MicroPython. Although the OpenMV IDE is traditionally used with the OpenMV Camera, it can also be effectively utilized with other hardware platforms, such as the Arduino Portenta H7 and the Vision Shield.

The OpenMV IDE provides a user-friendly environment for writing MicroPython scripts, flashing the firmware onto the Portenta H7, and visualizing results from the Vision Shield's sensors in real time.

By using the OpenMV IDE with the Portenta H7 and Vision Shield, we can leverage the power of real-time vision processing and machine learning, streamlining the development process and enabling sophisticated embedded vision applications. [Ope24]

Here's the updated content with the key features of OpenMV included, tailored to your use of the Vision Shield and Portenta H7:

OpenMV is an open-source platform designed to simplify the integration of computer vision and machine learning into embedded systems. It allows developers to run real-time vision algorithms on embedded devices using MicroPython. Although the OpenMV IDE is traditionally used with the OpenMV Camera, it can also be effectively utilized with other hardware platforms, such as the Arduino Portenta H7 and the Vision Shield.

The OpenMV IDE provides a user-friendly environment for writing MicroPython scripts, flashing the firmware onto the Portenta H7, and visualizing results from the Vision Shield's sensors in real time. This IDE facilitates script development and hardware control, making it a versatile tool for creating and managing vision applications even when not using the OpenMV Camera. [Ope24]

Key Features of OpenMV such as:

- **Machine Vision Algorithms:** OpenMV supports a variety of built-in algorithms, including color tracking, object detection, QR

code recognition, and edge detection. These algorithms can be easily accessed and controlled via MicroPython scripts.

- **MicroPython Integration:** OpenMV runs MicroPython natively, allowing developers to write Python scripts to perform vision tasks. This makes it accessible to those familiar with Python, without needing to delve into lower-level languages.
- **Low Power and Real-Time Processing:** OpenMV is optimized for real-time computer vision tasks while maintaining a low power profile, making it ideal for embedded and battery-operated systems.
- **TensorFlow Lite Support:** OpenMV includes support for running TensorFlow Lite models, enabling machine learning on the edge. This feature allows users to run neural networks for tasks such as image classification and object detection directly on the Portenta H7 with the Vision Shield.
- **Extensive Hardware Support:** The OpenMV platform supports various hardware interfaces such as I2C, SPI, UART, and GPIO, enabling it to interface with other sensors and microcontrollers, including the Portenta H7 and the Vision Shield.

21.4. Implementing MicroPython on Portenta H7 with OpenMV IDE

MicroPython scripts can be run on the Arduino Portenta H7 with the help of OpenMV IDE. OpenMV comes with its own firmware which is built on MicroPython. [Ope24]

We can explore the features of the Arduino Portenta H7 by using different classes and modules provided in MicroPython. In order to implement MicroPython on the Arduino Portenta H7, we first need to connect the Arduino Portenta H7 with the computer using a USB-C cable and flash it by double pressing the reset button. After the flashing is done the bootloader gets updated while flashing the green LED. Now we open the OpenMV IDE and click on connect option provided at the bottom of the screen as done in the earlier chapters. This will install the latest OpenMV firmware to the Arduino Portenta H7 board. After the firmware is updated, we need to click on the play button to connect the board with OpenMV IDE and start programming in the IDE. To be able to check whether MicroPython program can run on the Arduino Portenta H7 we write a simple program in MicroPython to blink LED's on the board using the OpenMV IDE.

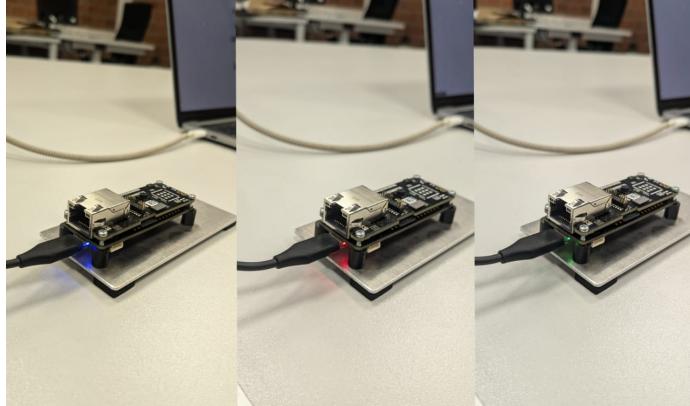


Figure 21.1.: RGB LED blinking blue, red and green in Portenta H7

21.4.1. Example script for blinking builtin LED on the Arduino Portenta H7 with OpenMV

In the program we first import `pyb` module which contains board related functionality such as PIN handling. Then we create variables to control the built in LEDs using `pyb.LED()` function. Then we switch on the red,blue and green LEDs with a delay of 1000ms using `redLED.on()` function and off the LED using `redLED.off()`. Now after the program is complete we hit the play button in the bottom to upload the script. We can see from the figure 21.1 that the output is flashing red,blue and green LEDs at a time interval of 1000ms. Therefore we can say that the MicroPython scripts can be run on the Arduino Portenta H7 with the use of OpenMV IDE.

22. Edge Impulse

22.1. Edge Impulse

Edge Impulse is an integrated cloud platform designed that enables end-to-end development and deployment of machine learning models for edge devices, particularly those with limited computational resources. It provides tools for data collection, model training, testing, verification and deployment specifically tailored for microcontrollers and other edge devices. [Imp24d] [Imp24c]

The figure 22.1 gives the workflow of Edge Impulse Platform from Data Collection, Training, Testing and Deployment to Edge Devices.

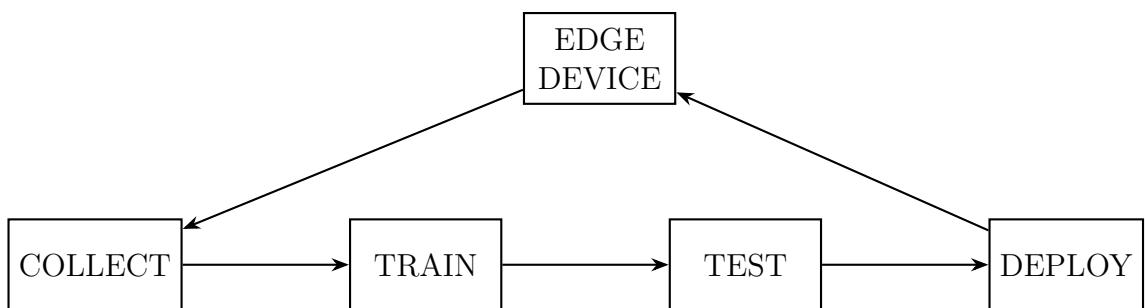


Figure 22.1.: Workflow of Edge Impulse Platform

22.1.1. Features of the Edge Impulse Platform

This section explore the key steps involved in building a machine learning model using Edge Impulse. This includes methods for gathering data, processing techniques, model training options, testing strategies, and deploying the model to edge devices. [Imp24c]

1. **Data Collection:** Edge Impulse platform can perform gathering of data from devices like cameras, microphones, or other sensors. We can collect data for the machine learning model directly from your device or upload it from files to Edge Impulse or through mobile app.
2. **Data Pre-Processing:** Using Edge Impulse, pre-processing of data can be done, helps improve the quality of the data used for training. This includes Filtering, Feature Extraction, Signal Analysis.

3. **Model Training:** Once the data is ready, Edge Impulse helps to train machine learning models without requiring deeper programming languages. It acts as a platform to build and train models without the need of deep programming knowledge. The platform supports various types of models such as classification of models, convolutional neural networks (CNNs), recurrent neural networks (RNNs) and transfer learning, depending on the type of data and the specific needs of the project.
4. **Model Testing:** After training, Edge Impulse allows you to test the model to ensure it performs well. We can use test data to evaluate the model's accuracy, or perform live testing by feeding real-time data from sensors. This helps in identifying any issues and fine-tuning the model for better results.
5. **Model Deployment:** After training, we can deploy the model directly onto edge devices like microcontrollers, IoT devices and other mobile phones. It provides code and libraries to run your trained model on these devices. It also generates and flash firmware that includes the trained model to edge devices.

22.1.2. Edge Impulse Framework Installation

The steps for installing and setting up the edge impulse platform, [Imp24b]

1. Go to the Edge Impulse website EdgeImpulse and click on Sign Up and create a new account.
2. Install the latest version of python from the official website Python.
3. Download and install Node.js from the official website Nodejs. Ensure to include the necessary tools during installation by selecting the check box as shown in figure 22.2
4. After installation, verify that Python, Node.js and npm are installed by running the following commands in your terminal `python -version, node -version, npm -version` as shown in figure 22.3.
5. Then, install Edge Impulse CLI which allows us to collect data from the hardware, train models locally, and upload datasets to the platform. Open a terminal and run the following command to install the CLI globally `npm install -g edge-impulse-cli`.
6. Verify the Edge Impulse CLI by using the command `edge-impulse-daemon -version` as shown in the figure 22.4.

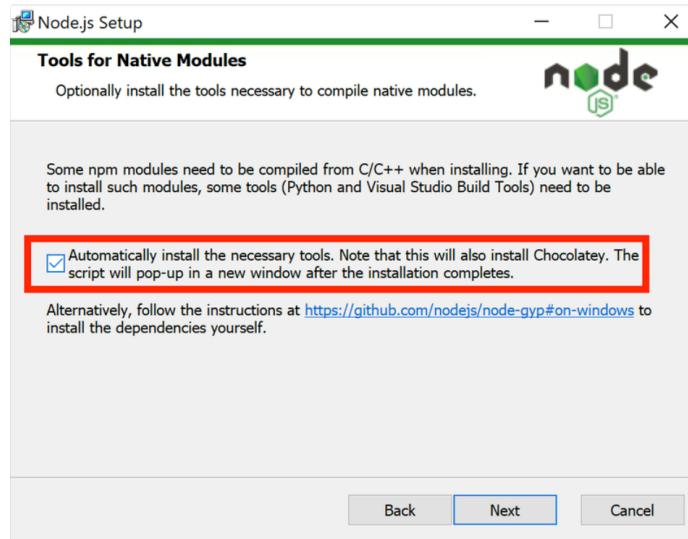


Figure 22.2.: Node.js Installation

22.1.3. Configuring Arduino Portenta H7 to Edge Impulse

These steps includes how to connect Arduino Portenta H7 along with Vision Shield to be connected to Edge Impulse. [Imp24a]

- To setup Portenta H7, we need to ensure Edge Impulse CLI and Arduino IDE are installed. Install Arduino IDE from the official website Arduino IDE
- Open Arduino IDE and navigate to **Tools > Board > Board Manager** and type Portenta H7 in search bar and install the **Arduino Mbed OS Portenta** board package as shown in figure 22.5.
- After installation, go to **Tools > Board** and select **Arduino Portenta H7**.
- Attach the Vision Shield to the Portenta H7 by alighning the pins, providing camera and microphones for image and audio data collection as shown in the figure 22.6.
- Use a USB-C cable to connect the development board to your computer. Then, double-tap the RESET button to put the device into bootloader mode. You should see the green LED on the front pulsating.
- Update the Firmware onto Portenta H7 board

```
C:\Users\mselv>node --version  
v20.17.0  
  
C:\Users\mselv>python --version  
Python 3.12.6  
  
C:\Users\mselv>node --version  
v20.17.0  
  
C:\Users\mselv>npm version  
{  
  npm: '10.8.2',  
  node: '20.17.0',  
  acorn: '8.11.3',  
  ada: '2.9.0',  
  ares: '1.32.3',  
  base64: '0.5.2',  
  brotli: '1.1.0',  
  cjs_module_lexer: '1.2.2',  
  cldr: '45.0',  
  icu: '75.1',  
  llhttp: '8.1.2',  
  modules: '115',  
  napi: '9',  
  nghttp2: '1.61.0',  
  nghttp3: '0.7.0',  
  ngtcp2: '1.1.0',  
  openssl: '3.0.13+quic',  
  simdutf: '5.3.0',  
  tz: '2024a',  
  undici: '6.19.2',  
  unicode: '15.1',  
  uv: '1.46.0',  
  uvwasi: '0.0.21',  
  v8: '11.3.244.8-node.23',  
  zlib: '1.3.0.1-motley-209717d'  
}
```

Figure 22.3.: CLI commands for Version

```
C:\Users\mselv>edge-impulse-daemon --version  
1.27.1
```

Figure 22.4.: Edge Impulse CLI version

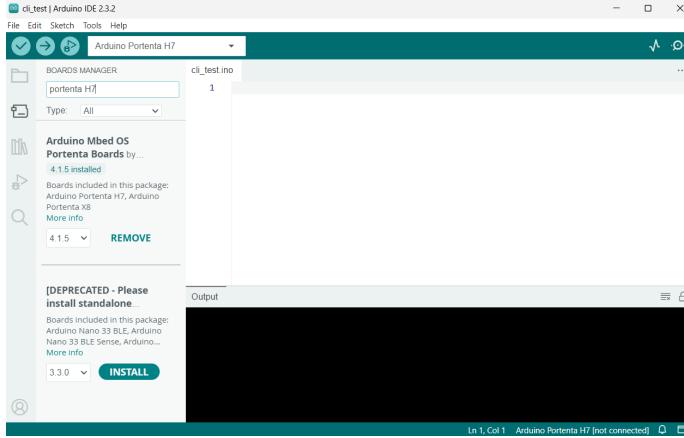


Figure 22.5.: Installing Portenta H7 board in Arduino IDE

1. Download the latest Edge Impulse firmware for Portenta H7 from the following website [PortentaH7 Firmware](#) and unzip the file.
 2. Double press on the RESET button on your board to put it in the bootloader mode.
 3. Open the flash script for the operating system which is in windows bat file [flash_windows](#) to flash the firmware as shown in 22.7.
 4. Wait until flashing is complete, and press the RESET button once to launch the new firmware.
- After flashing the firmware, to connect Portenta H7 with the Vision Shield to Edge impulse, run the command [edge-impulse-daemon](#).
 - Now, we will be prompted to log into the Edge Impulse account. After logging in, create a new project and select the Project to which the Portenta H7 to be connected.

22.1.4. Edge Impulse GUI Description

The Edge Impulse user interface (UI) includes the following sections as shown in the figure 22.8:

- **Dashboard:** Provides a summary of the project, showing its current status and recent activities.
- **Devices:** Lists the device which are connected to the Edge Impulse Platform for data transfer and deployment.

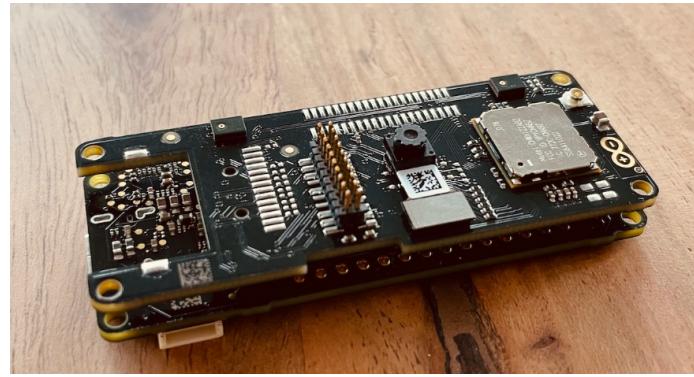


Figure 22.6.: Vision Shield connected to Portenta H7

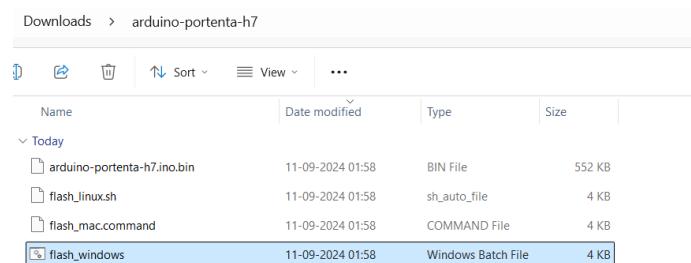


Figure 22.7.: Edge Impulse Firmware

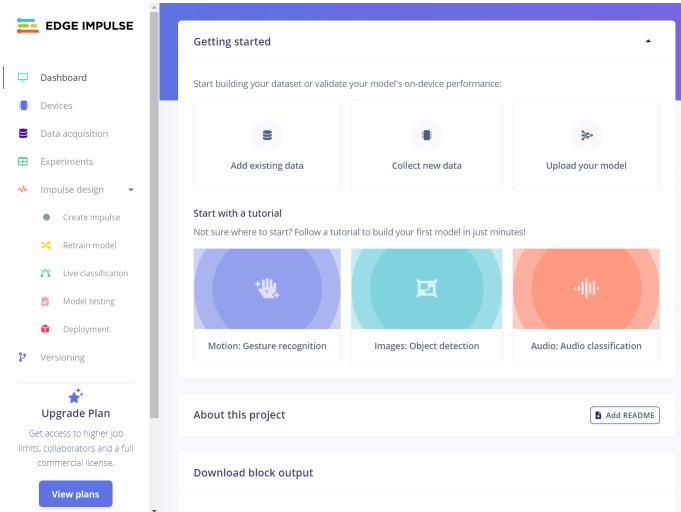


Figure 22.8.: Edge Impulse User Interface

- **Data Acquisition:** Offers tools to collect and organize the data for use.
- **Impulse Design:** Allows us to set up the machine learning process, including steps like preparing your data (pre-processing), feature extraction and training the model.
- **Model Testing:** Provides tools to evaluate the performance of trained model and make any necessary improvements.
- **Deployment:** Provide options to deploy and run our model on different edge devices.

Part VI.

Domain Machine Learning

23. Algorithms

23.1. Description

The algorithm used in our Face Recognition project is Convolutional Neural Networks (CNNs), a lightweight Image Classification algorithm designed for resource-constrained devices. Convolutional Neural Networks form the core of the recognition model. CNNs are highly effective in processing image data due to their ability to detect spatial hierarchies in visual patterns. CNN leverages the strengths of depthwise separable convolutions while ensuring low computational requirements and high efficiency. This algorithm is optimized for real-time applications on embedded systems like the Portenta H7. [Vas24]

23.2. Application

CNN can be applied in various scenarios where real-time Image Classification is required on low-power, low-compute devices. Typical applications include:

- **Facial Recognition:** Identifying or verifying a person based on their facial features. [Tea24]
- **Social Media Tagging:** Social media platforms use CNNs for tagging friends in photos. [Cel24]
- **Autonomous Vehicles:** CNNs are utilized to detect pedestrians, traffic signs, and other vehicles on the road. [All24]
- **Medical Imaging:** CNNs are used in medical imaging applications to segment and identify tumors in MRI or CT scans. [Tec24]
- **Navigation Assistance:** CNNs help in recognizing road signs, lane markings, and obstacles, enabling safer navigation. [All24]

23.3. Hyperparameter

The key hyperparameters for the CNNs algorithm include:

- **Learning Rate:** Controls the step size during the optimization process. [Bro23]
- **Batch Size:** Number of samples processed before the model's internal parameters are updated. [Bro23]
- **Number of Epochs:** Number of times the learning algorithm will work through the entire training dataset. [Bro23]
- **Confidence Threshold:** Minimum confidence score for a detected object to be considered valid.

23.4. Requirements

- Arduino Vision Shield module with Vision Shield camera. [Ard23]
- Edge Impulse account and CLI tools. [Com23]
- TensorFlow Lite for Microcontrollers
- Arduino IDE or PlatformIO for deploying the model

23.5. Input

The input to the CNNs algorithm consists of images captured by the Vision Shield. These images are preprocessed to grayscale and resized to a standard input size required by the model.

23.6. Output

The outputs of the CNN algorithm is the predicted class label for the input image, indicating which person it corresponds to based on the trained model. Additionally, the CNN provides confidence scores for each class, reflecting the model's certainty about the prediction as a probability value between 0 and 1.

23.7. Example with a Program

Below is an example program 23.1 demonstrating the use of the CNN algorithm for Face recognition on the PortentaH7:

Listing 23.1.: Algorithm exmaple code

```
1000 #include <Arduino.h>
1001 #include <EloquentTinyML.h> // Library for TinyML
1002 #include <TensorFlowLite.h>
1003 #include "model.h" // The trained model file from Edge
1004     Impulse
1005 #include <Wire.h>
1006 #include <SPI.h>
1007 #include <Adafruit_GFX.h>
1008 #include <Adafruit_SSD1306.h>
1009
1010 // Define OLED display dimensions and I2C pins (optional, for
1011 // debugging)
1012 #define SCREEN_WIDTH 128
1013 #define SCREEN_HEIGHT 64
1014 Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire,
1015 -1);
1016
1017 // Define model parameters
1018 #define FEATURE_SIZE 128 // Input image size (e.g., 128
1019     x128 grayscale pixels)
1020 #define NUM_CLASSES 3 // Number of classes in the
1021     trained model
1022 #define CONFIDENCE_THRESHOLD 0.8 // Minimum confidence score
1023     to accept prediction
1024
1025 // Instantiate the TensorFlow Lite model
1026 Eloquent::TinyML::TfLite<FEATURE_SIZE, NUM_CLASSES, 10>
1027     mlModel;
1028
1029 // Placeholder for input image data (grayscale image)
1030 float inputImage[FEATURE_SIZE];
1031
1032 // Class labels
1033 const char* classLabels[NUM_CLASSES] = {"Person A", "Person B"
1034     ", "Unknown"};
1035
1036 void setup() {
1037     // Initialize serial communication for debugging
1038     Serial.begin(115200);
1039
1040     // Initialize OLED display (optional)
1041     if (!display.begin(SSD1306_I2C_ADDRESS, OLED_RESET)) {
1042         Serial.println(F("OLED initialization failed!"));
1043         for (;;) {
1044     }
1045     display.clearDisplay();
1046     display.setTextSize(1);
1047     display.setTextColor(WHITE);
1048
1049     // Load the trained model into memory
1050     mlModel.begin(model_tflite);
```

```

1044     Serial.println(F("Face Recognition System Initialized"));
1045     display.println("System Ready");
1046     display.display();
1047 }
1048
1049 void loop() {
1050     // Capture an image using the Vision Shield (pseudocode)
1051     // Replace this with actual image capture code
1052     captureImage(inputImage);
1053
1054     // Preprocess the image (if needed, such as normalization
1055     // or resizing)
1056     preprocessImage(inputImage, FEATURE_SIZE);
1057
1058     // Run inference
1059     float* output = mlModel.predict(inputImage);
1060
1061     // Identify the class with the highest confidence score
1062     int predictedClass = -1;
1063     float maxConfidence = 0.0;
1064     for (int i = 0; i < NUM_CLASSES; i++) {
1065         if (output[i] > maxConfidence) {
1066             maxConfidence = output[i];
1067             predictedClass = i;
1068         }
1069     }
1070
1071     // Display prediction result
1072     if (maxConfidence > CONFIDENCE_THRESHOLD) {
1073         Serial.print(F("Detected: "));
1074         Serial.print(classLabels[predictedClass]);
1075         Serial.print(F(" (Confidence: "));
1076         Serial.print(maxConfidence, 2);
1077         Serial.println(F(")"));
1078
1079         display.clearDisplay();
1080         display.setCursor(0, 0);
1081         display.print(F("Detected: "));
1082         display.println(classLabels[predictedClass]);
1083         display.print(F("Confidence: "));
1084         display.print(maxConfidence, 2);
1085         display.display();
1086     } else {
1087         Serial.println(F("No confident prediction"));
1088         display.clearDisplay();
1089         display.println(F("No match found!"));
1090         display.display();
1091     }
1092     delay(1000); // Adjust delay as needed
1093 }
1094
1095 // Function to capture an image (implement with Vision Shield

```

```
API)
1096 void captureImage( float* imageBuffer ) {
    // Placeholder for image capture logic
1098     // Fill imageBuffer with grayscale pixel data
}
1100
1102 // Function to preprocess the image
1103 void preprocessImage( float* imageBuffer , size_t size ) {
    // Placeholder for preprocessing logic
1104     // Example: normalize pixel values to 0-1 range
1105     for ( size_t i = 0; i < size; i++ ) {
1106         imageBuffer[ i ] = imageBuffer[ i ] / 255.0;
    }
1108 }
```

..../Code/EdgeImpulse/Algorithms/ExampleAlgorithm.ino

24. Packages

24.1. edge-impulse-sdk, Version 1.3.0

24.1.1. Introduction

The [edge-impulse-sdk](#) is a versatile library designed to run machine learning inferences on edge devices like the Portenta H7. In our face recognition project, this SDK allows us to deploy the model trained using Edge Impulse, simplifying the integration of AI into our system for real-time facial recognition tasks. [Imp23]

24.1.2. Installation

To install the ‘edge-impulse-sdk’, follow these steps:

- Download the C++ library from the Edge Impulse Studio.
- Unzip the library into your project directory.
- Ensure you have a C compiler, a C++ compiler, and Make installed on your system.

For detailed instructions, refer to the Edge Impulse Documentation.

24.1.3. Example - Description

This example demonstrates how to set up and use the ‘edge-impulse-sdk’ to perform inference on raw sensor data. The setup involves creating a C++ project, integrating the SDK, and running the impulse to get the model’s output.

24.1.4. Example - Manual

1. **Create a Project:** Set up a directory for your project and unzip the C++ library.
2. **Write Code:** Create a main application file (`main.cpp`) that includes the necessary headers and defines the main function to run the classifier.

3. **Build the Project:** Use **Make** to compile the project and generate the executable.
4. **Run Inference:** Execute the program to run inference on the test data.

24.1.5. Example - code

Here is a simple example 24.1 of a main.cpp file:

Listing 24.1.: Package exmaple code

```

1000 #include <stdio.h>
1002
1004 // Callback function declaration
1005 static int get_signal_data(size_t offset, size_t length,
1006                           float *out_ptr);
1007
1008 // Raw features copied from test sample (Edge Impulse > Model
1009 // testing)
1010 static float input_buf[] = {
1011   /* Paste your raw features here! */
1012 };
1013
1014 int main(int argc, char **argv) {
1015
1016   signal_t signal;           // Wrapper for raw input
1017   buffer_t buffer;
1018   ei_impulse_result_t result; // Used to store inference
1019   output_t output;
1020   EI_IMPULSE_ERROR res;      // Return code from inference
1021
1022   // Calculate the length of the buffer
1023   size_t buf_len = sizeof(input_buf) / sizeof(input_buf[0])
1024   ;
1025
1026   // Make sure that the length of the buffer matches
1027   // expected input length
1028   if (buf_len != EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE) {
1029     printf("ERROR: The size of the input buffer is not
1030           correct.\r\n");
1031     printf("Expected %d items, but got %d\r\n",
1032           EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE,
1033           (int)buf_len);
1034     return 1;
1035   }
1036
1037   // Assign callback function to fill buffer used for
1038   // preprocessing/inference
1039   signal.total_length = EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE;

```

```
1032     signal.get_data = &get_signal_data;  
1034     // Perform DSP pre-processing and inference  
1035     res = run_classifier(&signal, &result, false);  
1036  
1037     // Print return code and how long it took to perform  
1038     // inference  
1039     printf("run_classifier returned: %d\r\n", res);  
1040     printf("Timing: DSP %d ms, inference %d ms, anomaly %d ms  
1041             \r\n",  
1042             result.timing.dsp,  
1043             result.timing.classification,  
1044             result.timing.anomaly);  
1045  
1045     // Print the prediction results (object detection)  
1046 #if EI_CLASSIFIER_OBJECT_DETECTION == 1  
1047     printf("Object detection bounding boxes:\r\n");  
1048     for (uint32_t i = 0; i < EI_CLASSIFIER_OBJECT_DETECTION_COUNT; i++) {  
1049         ei_impulse_result_bounding_box_t bb = result.  
1050         bounding_boxes[i];  
1051         if (bb.value == 0) {  
1052             continue;  
1053         }  
1054         printf(" %s (%f) [ x: %u, y: %u, width: %u, height:  
1055                 %u ]\r\n",  
1056                 bb.label,  
1057                 bb.value,  
1058                 bb.x,  
1059                 bb.y,  
1060                 bb.width,  
1061                 bb.height);  
1062     }  
1063  
1063     // Print the prediction results (classification)  
1064 #else  
1065     printf("Predictions:\r\n");  
1066     for (uint16_t i = 0; i < EI_CLASSIFIER_LABEL_COUNT; i++) {  
1067         printf(" %s: ", ei_classifier_inferencing_categories  
1068             [i]);  
1069         printf("%.5f\r\n", result.classification[i].value);  
1070     }  
1071 #endif  
1072  
1073     // Print anomaly result (if it exists)  
1074 #if EI_CLASSIFIER_HAS_ANOMALY == 1  
1075     printf("Anomaly prediction: %.3f\r\n", result.anomaly);  
1076 #endif  
1077  
1078     return 0;  
1079 }
```

```
1078 // Callback: fill a section of the out_ptr buffer when
1079 // requested
1080 static int get_signal_data(size_t offset, size_t length,
1081     float *out_ptr) {
1082     for (size_t i = 0; i < length; i++) {
1083         out_ptr[i] = (input_buf + offset)[i];
1084     }
1085 }
```

..../Code/EdgeImpulse/Packages/ExampleCodeSDK.ino

24.1.6. Future Reading

For more information on deploying your model as a C++ library, check the Edge Impulse Documentation

24.2. TensorFlow Lite, Version 2.12.0

24.2.1. Introduction

TensorFlow Lite is a lightweight solution for deploying machine learning models on mobile and embedded devices. It is designed to run TensorFlow models on resource-constrained devices with low latency and high efficiency. [Ten]

24.2.2. Installation

To install TensorFlow Lite, run `pip install tensorflow` in command prompt window, refer 20

24.2.3. Example - Description

This example demonstrates how to use TensorFlow Lite to perform image classification on a mobile device. The steps include loading a model, preprocessing input data, running inference, and interpreting the results.

24.2.4. Example - Manual

The manual steps for running a TensorFlow Lite model include:

1. Convert the TensorFlow model to TensorFlow Lite format.
2. Load the TensorFlow Lite model on the device.
3. Preprocess the input data to match the model's requirements.
4. Use the TensorFlow Lite interpreter to run the model.
5. Postprocess the results to interpret the model's output.

24.2.5. Example - Code

Listing 24.2.: Example Code for running Tensorflow Lite model.

```
1000 import tensorflow as tf
1001 import numpy as np
1002 from PIL import Image
1003
1004 # Load the TFLite model and allocate tensors
1005 interpreter = tf.lite.Interpreter(model_path="model.tflite")
1006 interpreter.allocate_tensors()
1007
1008 # Get input and output tensors
1009 input_details = interpreter.get_input_details()
```

```

1010 output_details = interpreter.get_output_details()
1012 # Prepare the input image
1013 img = Image.open('image.jpg').resize((224, 224))
1014 input_data = np.expand_dims(np.array(img, dtype=np.float32),
1015                             axis=0)
1016 # Set the tensor to point to the input data to be inferred
1017 interpreter.set_tensor(input_details[0]['index'], input_data)
1018 # Run the model
1019 interpreter.invoke()
1020
1021 # Get the output data
1022 output_data = interpreter.get_tensor(output_details[0]['index'])
1023 print('Output:', output_data)

```

..../Code/EdgeImpulse/Packages/ExampleCodeTensorFlow.ino

24.2.6. Example - Files

The example files include:

- **model.tflite** - The TensorFlow Lite model file.
- **image.jpg** - The input image file for inference.
- **tflite_example.py** - The Python script to run the model.

24.2.7. Futher reading

For more detailed information, refer to the TensorFlow Lite documentation:

- TensorFlow Lite Guide.
- TensorFlow Lite Models.
- TensorFlow Lite Conversion.

Part VII.

Methodology

25. Methodology

25.1. KDD Process

The Knowledge Discovery in Databases (KDD) is a process in data science and analytics focused to discover and make explicit knowledge available in extensive large data sets. [Win24]

The KDD process typically includes several key stages, refer figure 25.1:

1. **Data Selection:** Identifying and gathering relevant data from various sources. In this project, we are gathering our own datasets with three distinct labels.
2. **Data Preprocessing:** Cleaning and transforming the data to improve its quality. In this project, the preprocessing of the face images includes image resizing and converting to grayscale, reducing complexity to fit to the model.
3. **Data Transformation:** The data is transformed to suitable format to focus on key attributes that are significant for analysis. Techniques like normalization, aggregation, and feature extraction are often used. For this project, the selected images were converted into a suitable dimensions necessary for the machine learning model.
4. **Data Mining:** The core of KDD, where algorithms are defined, trained and optimized to the prepared data to extract patterns. The MobileNetV2 CNN algorithm is used for face detection.
5. **Interpretation/Evaluation:** This step involves analyzing and assessing the discovered patterns to determine their validity and usefulness. Here, Model performance was evaluated using metrics such as F1 score and accuracy
6. **Knowledge Presentation:** Presenting the discovered knowledge in a way that users can understand using Visualization techniques, reports, or dashboards. Here the results are visualized and deployed to Portenta H7 device.

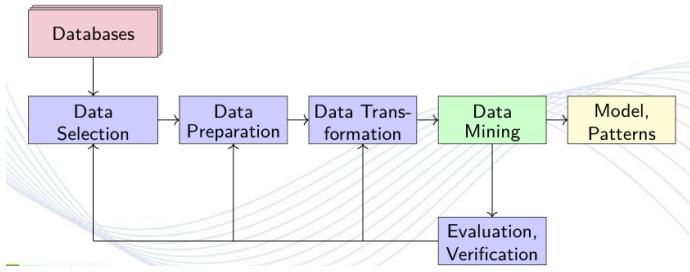


Figure 25.1.: KDD Process
[Win24]

25.2. Justification

The KDD process was selected for its clear and systematic workflow, ensuring each step from data preparation to deployment is effectively managed. It integrates well with Edge Impulse Studio, simplifying tasks like preprocessing and transformation of images for the model.

MobileNetV2, a type of Convolutional Neural Network (CNN) was selected for this project, which introduces inverted residual blocks, which expand the input channels, perform lightweight computations. MobileNetV2's architecture uses Linear Bottlenecks, which is optimized for extracting features from images, such as facial contours, expressions and unique attributes, even when images are preprocessed to lower dimensions. MobileNetV2 supports INT8 quantization, reducing memory usage and speeding up inference can fit into the 8 MB flash memory of the Portenta H7.

Part VIII.

KDD

26. Database

In this chapter, the selection and processing of the project's database are discussed in detail, figure 26.1 [FPSS97].

26.1. Data Selection

Data selection is a critical step, as it directly impacts the project's outcomes. This section discusses the origin, features, types, quality, quantity and fairness of the data.

26.1.1. Origin

The dataset used in this project consists of images of our own faces, refers to measurable physical characteristics that can uniquely identify individuals and facial features, that are specifically collected to meet the requirements of the face detection task. The images were captured using Vision Shield Camera, ensuring realistic scenarios under various conditions.

26.1.2. Features

The datasets includes image samples characterized by:

- Face images corresponds to three distinct images: Person1, Person2 and Person3.
- Every image has consistent features includes proper dimension (160x120 pixels) and grayscale images, represent pixel intensities using values ranging from 0 to 255, where
 - 0 indicates black
 - 255 indicates white
 - Values in between represent varying shades of gray, with 128 being a neutral gray

26.1.3. Quality

The datasets cover a wide range of facial expressions, under different lightning conditions and background.

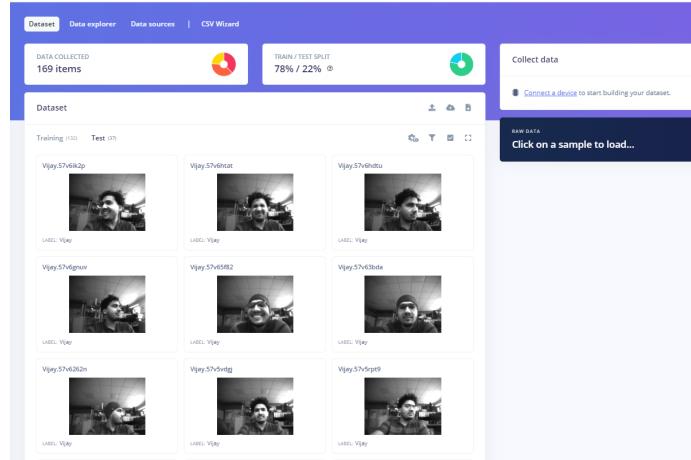


Figure 26.1.: Dataset dashboard

26.1.4. Quantity

In this project, the total size of the datasets used is 163 images, which includes:

- Person1: 56 images
- Person2: 57 images
- Person3: 56 images

These data were split into training and testing sets randomly contributes to 132 and 37 images respectively, ensuring that each of the three image labels (Person1, Person2, Person3) is represented in both the training and testing sets.

26.1.5. Properties

The dataset exhibits the following properties:

- All images captured from Vision Shield with fixed resolution 160x120 pixels.
- Image format used is JPG.
- Consistent labeling used with three unique datasets.
- Image size less than 100KB.

26.1.6. Fairness/Bias

To ensure fairness and reduce bias, the dataset was carefully prepared to include a balanced mix of different facial expressions, lighting conditions and angles. Duplicate or similar images were removed to avoid having too many samples of the same conditions.

26.2. Data Processing

The processing of the datasets involved several steps to prepare the data for training the face recognition model. This section details the steps for merging the datasets, handling outliers and anomalies and augmenting the data. [MFH00]

26.2.1. Outliers

Outliers were identified like detecting images with similar relations, duplicates and background to maintain integrity.

26.2.2. Anomalies

Anomalies such as mislabeled data or overlapping faces were resolved and validated.

26.2.3. Augmentation

Data augmentation techniques such as rotations, flips, scaling and lighting adjustments were applied to make the model robust to varied scenarios. These methods increases the diversity without changing its core characteristics.

27. Data Transformation and Data Mining

This chapter outlines the processes involved in transforming data for model training and mining insights using machine learning techniques. [FPSS97]

27.1. Data Transformation

Data transformation involves converting raw data into a structured and standardized format suitable for model training. In this project, the transformation process included:

1. **Data Labeling:** Labeling the datasets into three classes: Person1, Person2, Person3
2. **Resizing:** Images were resized to 96x96 pixels to ensure compatibility with the model architecture.

27.2. Data Mining

In this process, data mining involves applying machine learning techniques to extract features and patterns from the dataset.

27.2.1. Hyperparameters

The MobileNetV2 model, used in this project employs several hyperparameters that control the learning process. These hyperparameters are critical in defining how the model learns from the training data and adjusts over time.

- Number of training cycles: 100
- Learning rate: 0.001
- Training processor: CPU
- Validation set size: 20%
- Batch size: 32

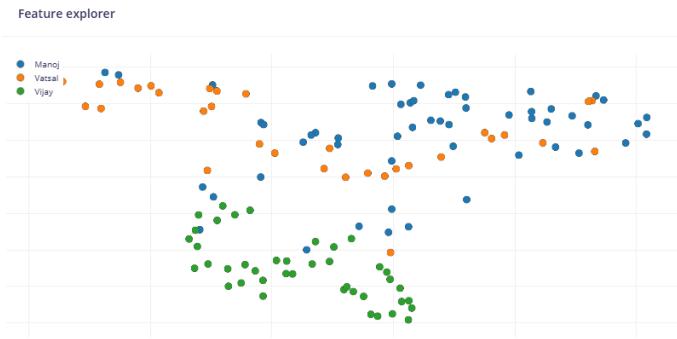


Figure 27.1.: Feature Explorer

27.2.2. Input

The input data consists of raw facial images. These preprocessed images are then used to extract features for the face recognition model.

Feature Explorer

The Feature Explorer in Edge Impulse is used to visualize and cluster the extracted feature vectors generated from the raw input data on the edge impulse platform

The Feature Explorer image as shown in figure 27.1 shows three distinct classes, each in a different color, with features from the facial images clustered into groups

27.2.3. Training

The training phase involved feeding the input data into the machine learning model using following setup

1. **Training Model:** MobileNetV2
2. **Training settings:** Input layer(25600 features), Dropout rate (0.1)
3. **Use Learning Optimizer:** TensorFlow Lite
4. **Processor:** CPU

27.2.4. Interpretation

The Model performance was assessed through the following metrics:

- **Last Training Performance (Validation Set)**
 - Accuracy: 85.2%

- Loss: 0.41
- Confusion Matrix (Validation Set):
 - * F1 Score: Person1(Manoj) 0.60
 - * F1 Score: Person1(Vatsal) 0.82
 - * F1 Score: Person1(Vijay) 1.00
- Metrics (Validation Set)
 - Precision: 0.90
 - Recall: 0.85
 - F1 Score: 0.84
- On-Device Performance
 - Engine: Custom
 - Inferencing Time: 1798 ms
 - Peak RAM Usage: 253.7K
 - Flash Usage: 305.8K

27.2.5. Output

The output consists of the predicted classes and their probabilities, refer 27.2

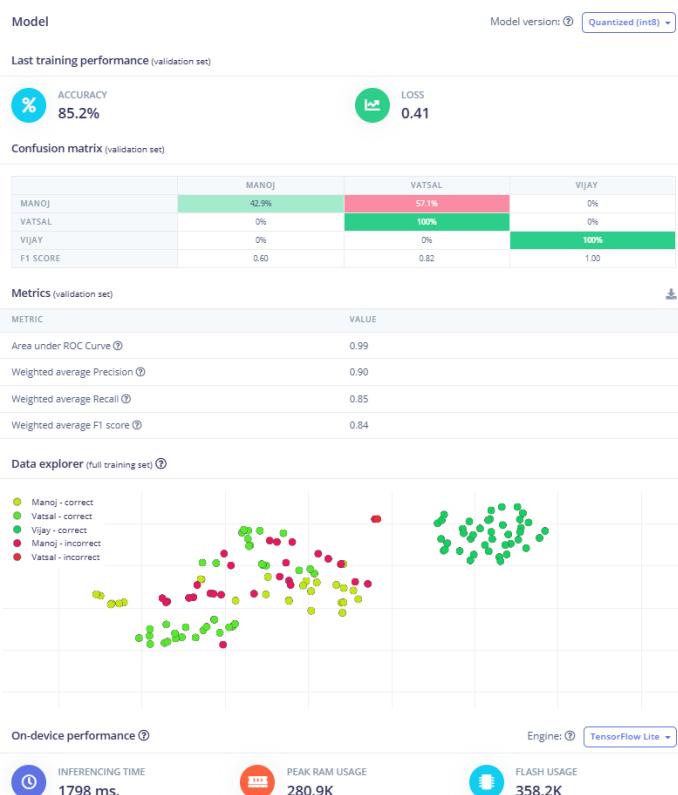


Figure 27.2.: Model Interpretation and Output

28. Evaluation and Verification

This chapter focuses on assessing the performance of the face detection model. The evaluation section breaks down the concepts, process flow and results, while the validation section ensures the model's performance on unseen data and varied scenarios. [FPSS97]

28.1. Evaluation

This section describes the detailed evaluation involved in the project, refer figure 28.1.

28.1.1. Concept

Evaluation involves analyzing the performance of the face detection model across key stages of its workflow. This concept includes assessing the quality of captured images, the effectiveness of preprocessing techniques like resizing and normalization and the accuracy of feature extraction and the machine learning model.

1. Image Capture: Collecting images using the Vision Shield Camera in varied lighting conditions.
2. Image Data Processing: Applying preprocessing techniques such as resizing, normalization, and augmentation.
3. Feature Extraction: Generating meaningful representations (features) from input data, i.e, images for face detection.
4. Model Training: Training the model with labeled data, fine-tuning parameters, and monitoring performance using training and validation metrics.
5. Model Evaluation: Assessing the trained model on a test dataset to measure metrics like accuracy, precision, and recall.
6. Visualization: Displaying results using metric graphs and confusion matrices to provide insights.

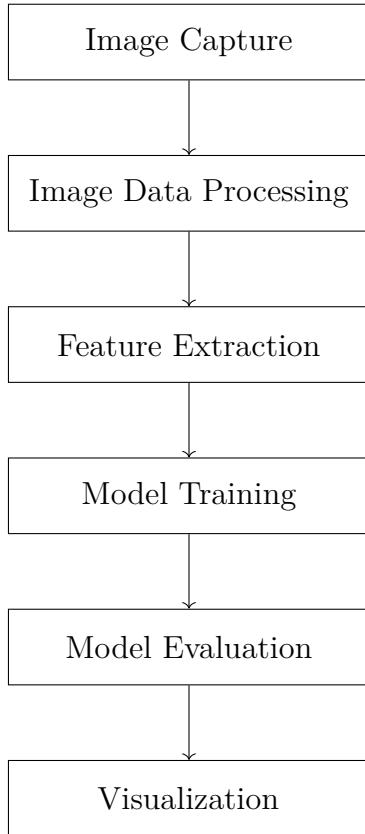


Figure 28.1.: ML Pipeline for Face Detection Evaluation Workflow

28.1.2. Application

The evaluation was applied to ensure robust model performance at each stage includes:

1. **Image Capture:** Images captured by Vision Shield were tested for quality consistency under varied conditions.
2. **Image Data Processing:** Validating the impact of preprocessing steps such as resizing, resolution check and augmentation.
3. **Feature Extraction:** Ensuring the features extracted were meaningful and consistent across samples.
4. **Evaluation Results:** Evaluating the machine learning model's performance using metrics such as accuracy, precision, recall, F1 score, and inference time
5. **Model Testing:** Deploying the model in real-time with the Vision Shield Camera to test its performance to environmental changes.

28.1.3. Result

Results from the evaluation phase, refer table 28.1, figure 28.2 include:

1. **Accuracy:** The overall correctness of face detections.
2. **Precision:** Proportion of true positive detections among all detections.
3. **Recall:** Proportion of true positive detections among actual faces in the dataset.
4. **F1 Score:** Balances precision and recall for an overall performance metric.
5. **Inference Time:** Time taken for the model to process and predict for a single image.

Metric	Value
Accuracy	85.2%
Precision	90%
Recall	85%
F1 Score	84%
Inference Time	1798ms

Table 28.1.: Performance Metrics

28.2. Validation

28.2.1. Concept

General validation ensures that the face detection system performs consistently and reliably across diverse conditions, including different lighting scenarios, facial orientations and environmental challenges.

28.2.2. Ideas

Cross-Validation

Purpose: Ensure the model performs consistently across different data splits.

Method: Dividing the dataset into several folds, iteratively train on some while validating on others and average the results.



Figure 28.2.: Model Evaluation Metrics

Real-Time Testing

Purpose: Assess the model's functionality and robustness in real-world scenarios.

Method: Deploy the model on the Portenta H7 along with the Vision Shield Camera and test it under various lighting conditions, angles and environment.

User Feedback:

Purpose: To gather insights from users to identify any user issues and pinpoint areas for improvement in the face detection model.

Method: Deploy the model in live settings and collect feedback on detection accuracy, speed and usability.

28.2.3. Conclusion

By performing cross-validation, real-time testing and user feedbacks, we confirmed that the model achieves its goals of accurate and efficient face detection. These validation make sure that the system works well technically and is practical enough to be useful in everyday situations.

28.2.4. Future Enhancements

While the project has achieved significant success, there are still several opportunities for further enhancements and exploration, particularly in leveraging advanced communication methods like LoRa for remote access management and real-time monitoring.

LoRa-Based Communication for Prediction Data

The Portenta Vision Shield features built-in LoRa (Long Range) capabilities, offering a low-power, long-range wireless communication method. Future work could involve utilizing LoRa to transmit recognized face data to a remote server, enabling centralized monitoring and decision-making. This would be particularly valuable for applications in smart access control, security systems, and IoT-based facility management.

To implement LoRa communication efficiently, different approaches can be considered:

- **LoRa Peer-to-Peer Communication:** This method allows direct data transmission between two devices without requiring additional infrastructure. This could be used to send face recognition results directly from the Portenta Vision Shield to a local receiver or security system. (Refer 16.5.2)

- **LoRaWAN (LoRa Wide Area Network):** LoRaWAN enables data transmission over a broader network, allowing face recognition results to be sent to cloud-based systems or remote monitoring platforms. This would be ideal for large-scale security applications where multiple access points need to be synchronized in real-time. (Refer 16.5.1)

29. Application Deployment

29.1. Description

Model deployment is the process of integrating the trained face recognition model into a functional environment to perform real-time image recognition and transmission. This involves setting up the hardware, ensuring compatibility between components, and implementing the necessary software for smooth operation. The deployment is guided by the knowledge discovery in databases (KDD) process to ensure an efficient transition from development to application. [FPSS97]

29.2. Deployment Architecture

The deployment architecture consists of three primary components: the Edge Device (Arduino Portenta H7), LoRa Communication Network, and the Centralized System for result processing and storage. Below is a breakdown of each component and their interactions:

29.2.1. Edge Device (Arduino Portenta H7 + Vision Shield)

The Edge Device is responsible for capturing images and processing them through the face recognition model. The steps involved are:

- **Face Recognition Model:** The core model (converted to TensorFlow Lite) runs on the Portenta H7 microcontroller. It processes images captured by the Vision Shield.
- **Image Capture and Preprocessing:** The Vision Shield captures images and preprocesses them to meet the input requirements of the face recognition model.
- **Inference:** The preprocessed image is fed into the model for real-time face recognition.
- **Model Output:** The output (e.g., recognition result, identified face) is generated and prepared for transmission.

29.3. Flow Chart of Each Step in the Deployment

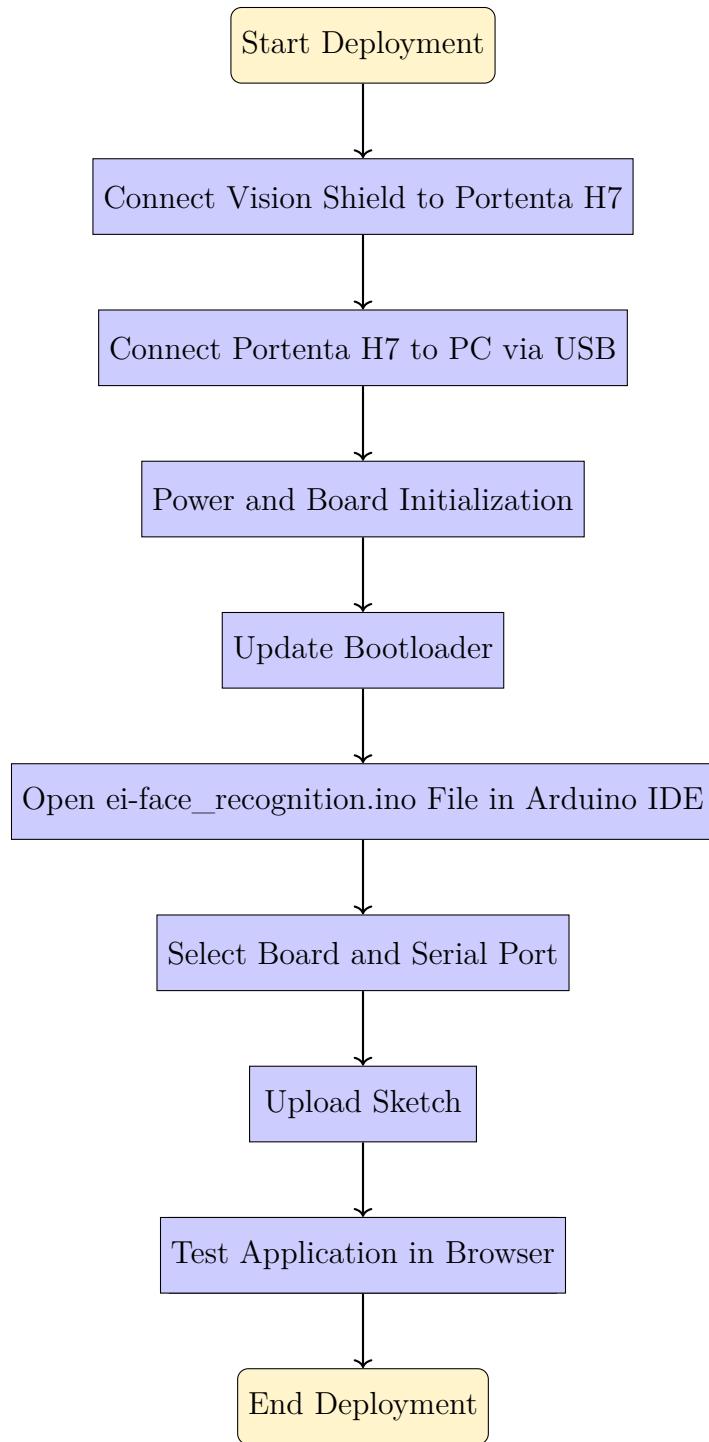


Figure 29.1.: Deployment Process Flowchart

29.4. ML Pipeline



Figure 29.2.: Machine Learning Pipeline

The machine learning pipeline in the deployment process includes the following steps:

- **Data Collection:** Capturing images using the Arduino Portenta H7 with Vision Shield.
- **Preprocessing:** Resizing, converting to grayscale, and cleaning images.
- **Feature Extraction:** Extracting relevant features from the pre-processed images.
- **Model Training:** Training the machine learning model with the extracted features.
- **Evaluation:** Assessing the model's performance and accuracy.
- **Deployment:** Implementing the model into the Arduino Portenta H7 with Vision Shield.
- **Maintenance:** Regularly updating the model and ensuring its continuous performance.

29.5. Process

The deployment process for the face recognition application using the Arduino Portenta H7, Vision Shield, and LoRa device involves multiple detailed steps to ensure seamless functionality. Below is a comprehensive guide:

29.5.1. Connect the Vision Shield to the Arduino Portenta H7

Hardware Connection:

- Attach the Vision Shield to the Arduino Portenta H7 via the designated connectors.
- Ensure the Vision Shield is firmly seated to avoid loose connections.

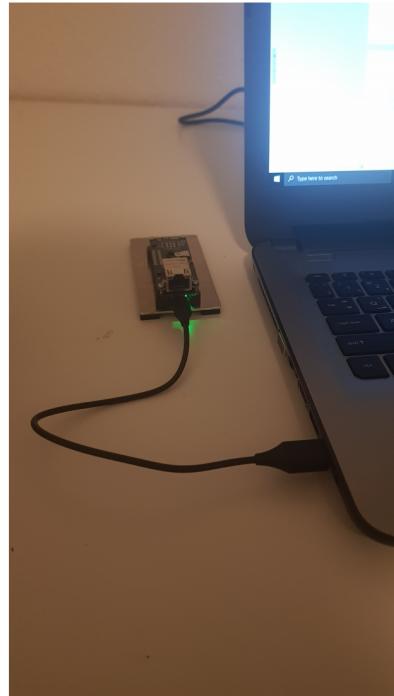


Figure 29.3.: Arduino PortentaH7 Connected to a Laptop

- Verify the Vision Shield's orientation to prevent incorrect installation.

29.5.2. Connect the Arduino Portenta H7 to the System

USB Connection:

- Connect the Portenta H7 to your computer using a USB-C cable.
- Ensure the cable is in good condition to avoid connection interruptions.
- Confirm the connection by checking for a detected COM port in the Arduino IDE or using `ls /dev/tty*` in Linux/Mac.

29.5.3. Check Power Supply and Board Initialization

Power Supply:

- Ensure the Portenta H7 receives a stable power supply (5V via USB-C or external battery).
- Check the onboard LED to confirm power delivery as shown in the Figure 29.3.

Board Initialization:

- Open the Arduino IDE and launch the Serial Monitor.
- Set the baud rate to 115200 and verify the initialization messages to ensure the board starts correctly.

29.5.4. Load the Face Recognition Sketch**Open Project:**

- Launch the Arduino IDE and open the file `.ino` for the face recognition project.
- Ensure the sketch includes:
 - Image capture and preprocessing from the Vision Shield.

Board and Port Selection:

- Select the correct board (Arduino Portenta H7) and corresponding serial port from the Arduino IDE.

29.5.5. Compile and Upload the Sketch**Compilation:**

- Click on the ‘Verify’ button to compile the code and check for errors.

Upload:

- Click the ‘Upload’ button to flash the sketch onto the Portenta H7.
- Monitor the Serial Monitor to confirm successful upload and initialization.

29.5.6. Run and Test the Model**Launch Impulse Runner:**

- Open a terminal and run the Edge Impulse command to initiate the model as shown in the figure 29.4:
`edge-impulse-run-impulse`
- This will execute the face recognition inference on the Portenta H7.

Debug Mode and Real-Time Monitoring:

```
C:\Users\mselv>edge-impulse-run-impulse
Edge Impulse impulse runner v1.30.4
[SER] Connecting to COM3
[SER] Serial is connected, trying to read config...
[SER] Retrieved configuration
[SER] Device is running AT command version 1.8.0

[SER] To get a live feed of the camera and live classification in your browser, run with --debug
[SER] Started inferencing, press CTRL+C to stop...
LSE
Inferencing settings:
  Image resolution: 160x160
  Frame size: 25600
  No. of classes: 3
Starting inferencing in 2 seconds...
Predictions (DSP: 3 ms., Classification: 143 ms., Anomaly: 0 ms.):
#Classification results:
  Manoj: 0.101563
  Vatsal: 0.039063
  Vijay: 0.859375
```

Figure 29.4.: Running the inference

```
C:\Users\mselv>edge-impulse-run-impulse --debug
Edge Impulse impulse runner v1.30.4
[SER] Connecting to COM3
[SER] Serial is connected, trying to read config...
[SER] Retrieved configuration
[SER] Device is running AT command version 1.8.0

Want to see a feed of the camera and live classification in your browser? Go to http://192.168.0.106:4915
[SER] Started inferencing, press CTRL+C to stop...
Predictions (DSP: 3 ms., Classification: 143 ms., Anomaly: 0 ms.):
  Manoj: 0.109375
  Vatsal: 0.039063
  Vijay: 0.851562
```

Figure 29.5.: Debug Mode

- Enable debug mode for real-time inference monitoring by running the below command in the terminal as shown in the figure 29.5: **edge-impulse-run-impulse -debug**
- Open the provided URL to visualize the live camera feed and check predictions in real-time.

30. Monitoring and Maintenance

This chapter outlines the strategies for monitoring the deployed face detection model and maintaining its performance over time.

30.1. Idea

The concept of monitoring and maintenance of the face detection system is designed to allow dynamic updates such as adding or removing individuals from the dataset by incorporating new data, updating the pipeline and performing regular checks.

30.2. Plan/Description

The concept for the face detection system is focused on dynamic dataset management includes setting up a system for regular model evaluation, data collection, and retraining and system adaptability. The primary goals are:

1. **Dataset Updates:** Add new users by collecting and processing their face data or remove users to revoke access, ensuring the dataset stays current.
2. **Model Retraining:** Retrain the model to reflect the updated dataset, to maintain accuracy and reliability.

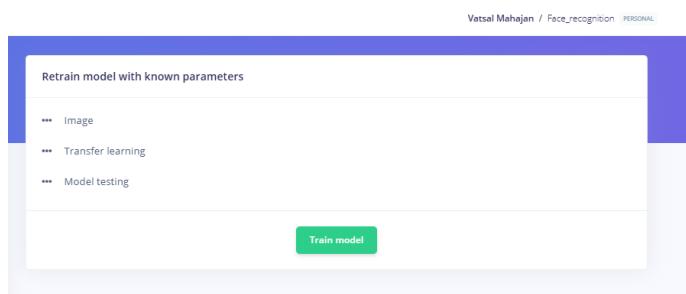


Figure 30.1.: Model Retraining

3. **Evaluate the Retrained Model:** Assess the updated model's performance on metrics like accuracy, precision, and recall to ensure quality.
4. **Deploy the Model:** Deploy the system with the retrained model for consistent real-time performance.

30.3. Updating new Data

The dataset can be updated using Edge impulse data collection tool. This involves capturing multiple face images for new users under various conditions using Vision Shield Camera or removing the existing datasets in the Edge Impulse Dataset explorer.

New face data is added when new users need access. This data is then used for retraining the machine learning model.

30.4. Data updating in the ML Pipeline

Data updation in the ML Pipeline includes the following steps:

1. Add the new faceset data or remove the existing faceset in the Edge Impulse data collection tool
2. Preprocess the updated data to align with existing datasets.
3. Retraining the model with the updated dataset to incorporate changes

30.5. Checks

To ensure the system remains reliable:

1. **Performance Metrics:** Regularly monitor accuracy, precision, recall, and inference times.
2. **Dataset Integrity:** Review the dataset to avoid duplicates or inconsistencies.
3. **Testing:** Perform routine testing in real-world conditions to identify potential issues early.

30.6. Privacy

To ensure privacy in the face access system:

1. **Anonymization:** All collected data is anonymized to protect user identity.
2. **Secure Access Control:** Data is securely stored, and access is restricted to authorized personnel through strict permissions.

30.7. Robustness

To maintain the system's effectiveness:

1. Test the model under varying conditions such as lighting changes or occlusions.
2. Redundant systems to ensure data collection and model evaluation continue uninterrupted.
3. Regularly update the model to adapt to changing user requirements and scenarios.

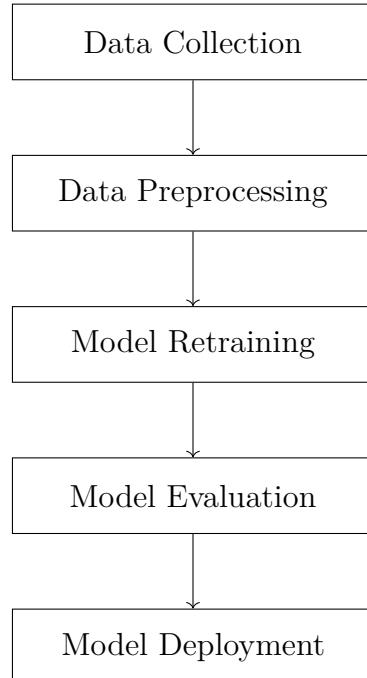


Figure 30.2.: Process Workflow for Face Access System Monitoring

30.8. Process

The overall workflow for the face access system includes the following steps, refer figure 30.2:

1. **Data Collection:** Capture new face images from the deployed system for adding new facesets.
2. **Data Preprocessing:** Clean and preprocess the new data to ensure it aligns with the model's input requirements.
3. **Model Retraining:** Retrain the model using the updated dataset to incorporate new faces and improve performance.
4. **Model Evaluation:** Evaluate the performance of the retrained model using metrics like accuracy, precision, and recall to ensure it meets the required standards.
5. **Model Deployment:** Deploy the updated model back into the system, ensuring that it operates smoothly with the updated data.

Part IX.

Appendix

31. Application Appendix

31.1. List of Materials

Please refer to Table 31.1.

Table 31.1.: Material List

Title	Version	Description	Price	Source
Laptop	Windows 11	Lenovo 15.6 Inch Full HD Notebook	\$366	Amazon
Portenta H7 Board	Latest	Development Board with Wi-Fi and LoRa support	\$130	Arduino
Vision Shield	OV5640 Camera + LoRa	Camera module for Portenta H7	\$65	Arduino
USB Cable	USB C	Cable for connecting Portenta H7 to laptop	\$3	Amazon
Micro SD Card	8 GB	Storage for Vision Shield	\$5	Amazon

31.2. Software Bill of Materials

Please refer to Table 31.2.

Table 31.2.: Software Bill of Materials

Title	Version	Description	Source
Arduino IDE	2.3.2	IDE for Arduino boards	arduino.cc
Edge Impulse CLI	1.15.4	CLI for Edge Impulse models	edgeimpulse.com
Python	3.10	Python programming language	python.org

31.3. List of Packages

Please refer to Table 31.3.

Table 31.3.: List of Packages

Title	Version	Price	Description	Source
TensorFlow Lite	2.12.0	Free	Lightweight version of TensorFlow for embedded devices	tensorflow.org
edge-impulse-sdk	1.3.0	Free	SDK for deploying Edge Impulse models	edgeimpulse.com
arduino-libraries	Latest	Free	Arduino libraries for Vision Shield and LoRa	github.com/arduino

31.4. Requirement File

Here is the path to the requirements file, `requirements.txt`:

`Project/requirements.txt`

The content of the requirements file is:

```
# requirements.txt
tensorflow-lite==2.12.0
numpy==1.24.3
edge-impulse-cli==1.15.4
```

31.5. Programming Languages

Please refer to Table 31.4.

Table 31.4.: Programming Languages

Language	Version
Python	3.10
C++	Latest

32. doxygen - Example

Doxygen is an open-source documentation generator that creates detailed and structured documentation from annotated source code. It supports various programming languages such as C, C++, Python, and Java.

In this project, Doxygen was utilized to document the source code and provide a clear explanation of the functionality of each module, class, and function as shown in the figure 32.1.

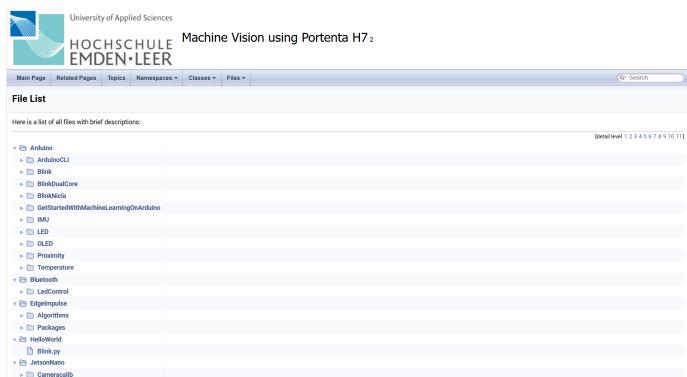


Figure 32.1.: Doxygen

Bibliography

- [AAO21a] C. Author1, D. Author2, and Others. “Comparison of Tensorflow and PyTorch in Convolutional Neural Network”. In: *IEEE Transactions on Neural Networks and Learning Systems* (2021). URL: <https://ieeexplore.ieee.org/document/9515098>.
- [AAO21b] E. Author1, F. Author2, and Others. “Study on real-time image processing applications with edge computing”. In: *IEEE Transactions on Information Forensics and Security* (2021). URL: <https://ieeexplore.ieee.org/document/9576139>.
- [Aba+16] M. Abadi et al. *TensorFlow: A system for large-scale machine learning*. 2016. URL: <https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi>.
- [Ada24] Adafruit. *Adafruit LSM9DS1 Class Reference*. 2024. URL: https://adafruit.github.io/Adafruit_LSM9DS1/html/class_adafruit__l_s_m9_d_s1.html.
- [All24] AllianceTek. *Exploring the Power of Convolutional Neural Networks*. 2024. URL: <https://www.alliancetek.com/blog/post/2024/08/23/exploring-the-power-of-convolutional-neural-networks.aspx>.
- [Ano22] Anonymous. *Portenta Vision Shield: An Overview of LAN-MAN Applications*. Accessed: 2024-12-23. 2022. URL: https://upcommons.upc.edu/bitstream/handle/2117/379227/2022_CR_LANMAN_Portenta.pdf?sequence=1.
- [Ard23] Arduino. *Getting Started with the Portenta Vision Shield Camera*. 2023. URL: https://docs.arduino.cc/tutorials/portenta-vision-shield/getting-started-camera?utm_source=chatgpt.com.
- [Ard24a] Arduino, ed. *Arduino CLI 0.36*. 2024. URL: <https://arduino.github.io/arduino-cli/0.36/commands/arduino-cli/> (visited on 05/11/2024).
- [Ard24b] Arduino, ed. *Arduino CLI (Command Line Interface) Application*. 2024. URL: <https://github.com/arduino/arduino-cli>.

- [Ard24c] Arduino, ed. *Arduino CLI Configuration*. 2024. URL: <https://arduino.github.io/arduino-cli/0.36/configuration/>.
- [Ard24d] Arduino, ed. *Arduino CLI*. 2024. URL: <https://arduino.github.io/arduino-cli/0.36/>.
- [Ard24e] Arduino. *Arduino LSM9DS1 Library Documentation*. 2024. URL: https://docs.arduino.cc/libraries/arduino_lsm9ds1/.
- [Ard24f] Arduino. *Arduino Portenta Cat. M1/NB IoT GNSS Shield Documentation*. 2024. URL: <https://docs.arduino.cc/hardware/portenta-cat-m1-nb-iot-gnss-shield/>.
- [Ard24g] Arduino. *Arduino Portenta Cat. M1/NB IoT GNSS Shield online store*. 2024. URL: <https://store.arduino.cc/products/portenta-catm1/>.
- [Ard24h] Arduino, ed. *Connecting to TTN / Portenta Vision Shield*. Accessed: 2024-12-23. 2024. URL: <https://docs.arduino.cc/tutorials/portenta-vision-shield/connecting-to-ttn/>.
- [Ard24i] Arduino. *Getting Started with Arduino*. 2024. URL: <https://www.arduino.cc/en/Guide>.
- [Ard24j] Arduino. *IMU Gyroscope - Arduino Tutorial*. 2024. URL: <https://docs.arduino.cc/tutorials/nano-33-iot imu-gyroscope/>.
- [Ard24k] Arduino. *Kalman Library*. 2024. URL: <https://docs.arduino.cc/libraries/kalman/>.
- [Ard24l] Arduino, ed. *LoRaWAN 101 / Arduino Documentation*. Accessed: 2024-12-23. 2024. URL: <https://docs.arduino.cc/learn/communication/lorawan-101/>.
- [Ard24m] Arduino. *Portenta H7 Documentation*. 2024. URL: <https://docs.arduino.cc/hardware/portenta-h7/>.
- [Ard24n] Arduino. *Portenta Vision Shield - Arduino Docs*. 2024. URL: <https://docs.arduino.cc/hardware/portenta-vision-shield/#tech-specs>.
- [Ard24o] Arduino. *Wire Library Documentation*. 2024. URL: <https://docs.arduino.cc/language-reference/en/functions/communication/wire/>.
- [Ard24p] Arduino, ed. *arduino-cli*. 2024. URL: <https://www.arduino.cc/pro/software-pro-cli/>.

- [Ard24q] ArduinoPortenta. *Arduino Portenta Cat. M1/NB IoT GNSS Shield Sketch*. 2024. URL: <https://docs.arduino.cc/tutorials/portenta-cat-m1-nb-iot-gnss-shield/getting-started/>.
- [Ard25a] Arduino. *Arduino IDE Environment Guide*. 2025. URL: <https://www.arduino.cc/en/Guide/Environment>.
- [Ard25b] Arduino. *Portenta H7 and Vision Shield LoRa Implementation*. 2025. URL: <https://www.arduino.cc/pro/portenta-h7>.
- [Ardnd] Arduino. *Portenta Vision Shield LoRa - Datasheet*. n.d. URL: <https://docs.arduino.cc/resources/datasheets/ASX00021-ASX00026-datasheet.pdf>.
- [Bro23] J. Brownlee. *Learning Rate for Deep Learning Neural Networks*. 2023. URL: https://www.machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/?utm_source=chatgpt.com.
- [CL+23] Y. Chen, T. Li, et al. “Edge AI: A Systematic Review and Guidelines for Future Research”. In: *ACM Computing Surveys* (2023). URL: <https://dl.acm.org/doi/full/10.1145/3555802>.
- [Cel24] CelerData. *Glossary: Convolutional Neural Network (CNN)*. 2024. URL: <https://celerdata.com/glossary/convolutional-neural-network-cnn>.
- [Cha24] A. Chandrasekaran. *Why Edge Computing is so Important for AI*. 2024. URL: <https://towardsdatascience.com/why-edge-computing-is-so-important-for-ai-e4695d4e7960>.
- [Cho15] F. Chollet. *Keras*. 2015. URL: https://keras.io/keras_3/.
- [Com23] I. Community. *TinyML Image Recognition With Edge Impulse Nano 33*. 2023. URL: https://www.instructables.com/TinyML-Image-Recognition-With-Edge-Impulse-Nano-33/?utm_source=chatgpt.com.
- [Com25] S. O. Community. *How can I reset an Arduino board?* 2025. URL: <https://stackoverflow.com/questions/5290428/how-can-i-reset-an-arduino-board>.
- [Cor25a] S. Corporation. *SX1276 LoRa Transceiver Datasheet*. 2025. URL: <https://www.semtech.com/products/wireless-rf/lora-transceivers/sx1276>.

-
- [Cor25b] S. Corporation. *What is LoRa?* 2025. URL: <https://www.semtech.com/lora/what-is-lora>.
- [Cyb24] CyberLink. “Facial Recognition in Smart Medical Healthcare”. In: *CyberLink Insights* (2024). URL: <https://www.cyberlink.com/faceme/insights/articles/747/facial-recognition-smart-medical-healthcare>.
- [Dec24] C. Deck. *Quantization Aware Training*. 2024. URL: <https://medium.com/@curiositydeck/quantization-aware-training-b80272380098>.
- [Dev24] A. Devices. *Inertial Measurement Units*. 2024. URL: <https://www.analog.com/en/product-category/inertial-measurement-units.html>.
- [Doc24a] A. Documentation. *Creating a Basic Face Filter with the Portenta Vision Shield*. 2024. URL: <https://docs.arduino.cc/tutorials/portenta-vision-shield/creating-basic-face-filter/>.
- [Doc24b] A. Documentation. *Creating a Wi-Fi Access Point with Portenta H7*. 2024. URL: <https://docs.arduino.cc/tutorials/portenta-h7/wifi-access-point/>.
- [Doc24c] A. Documentation. *Datasheet for PortentaH7*. 2024. URL: <https://docs.arduino.cc/resources/datasheets/ABX00042-ABX00045-ABX00046-datasheet.pdf>.
- [Doc24d] A. Documentation. *Getting Started with Arduino IDE 2*. 2024. URL: <https://docs.arduino.cc/software/ide-v2/tutorials/getting-started-ide-v2/>.
- [Doc24e] A. Documentation. *Getting Started with the Portenta Vision Shield Camera*. 2024. URL: <https://docs.arduino.cc/tutorials/portenta-vision-shield/getting-started-camera/>.
- [Doc24f] A. Documentation. *Portenta H7: BLE Connectivity*. 2024. URL: <https://docs.arduino.cc/tutorials/portenta-h7/ble-connectivity/>.
- [Doc24g] A. Documentation. *Saving Bitmap Camera Images to the SD Card*. 2024. URL: <https://docs.arduino.cc/tutorials/portenta-vision-shield/camera-to-bitmap-sd-card/>.
- [Doc24h] A. Documentation. *Setting up Portenta H7*. 2024. URL: <https://docs.arduino.cc/tutorials/portenta-h7/setting-up-portenta/>.

- [Doc24i] A. Documentation. *The Arduino Web Editor*. 2024. URL: <https://docs.arduino.cc/learn/startng-guide/the-arduino-web-editor/>.
- [Doc25a] A. Documentation. *Arduino Cloud*. Arduino. 2025. URL: <https://cloud.arduino.cc/>.
- [Doc25b] A. Documentation. *Installing Libraries*. Arduino. 2025. URL: <https://docs.arduino.cc/software/ide-v1/tutorials/installing-libraries/>.
- [Doc25c] A. Documentation. *Installing and Using Arduino IDE 1.x on Windows*. Ed. by Arduino. 2025. URL: <https://docs.arduino.cc/software/ide-v1/tutorials/Windows/>.
- [Doc25d] A. Documentation. *Installing and Using Arduino IDE 1.x on macOS*. Ed. by Arduino. 2025. URL: <https://docs.arduino.cc/software/ide-v1/tutorials/macOS/>.
- [Doc25e] A. Documentation. *Portenta H7 Datasheet (ABX00042, ABX00045, ABX00046)*. 2025. URL: <https://docs.arduino.cc/resources/datasheets/ABX00042-ABX00045-ABX00046-datasheet.pdf>.
- [Doc25f] A. Documentation. *Specifications of LoRa Device on Portenta Vision Shield*. 2025. URL: <https://www.arduino.cc/pro/tutorials/portenta-vision-shield-lora/specifications>.
- [Doc25g] M. Documentation. *Windows Commands Reference*. 2025. URL: <https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/windows-commands>.
- [Ele24] M. Electronics. *MEMS Sensors Overview*. 2024. URL: https://eu.mouser.com/applications/sensor_solutions_mems/?srsltid=AfmB0opiq7CaYeKLpDajiR0FZjCcLi4kivPkCQFlvXnFXZ3GM
- [Exc25] E. S. Exchange. *What Actually Causes Gyroscope Drift in IMU?* 2025. URL: <https://electronics.stackexchange.com/questions/445787/what-actually-causes-gyroscope-drift-in-imu>.
- [FPSS97] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. “Data Mining and Knowledge Discovery in Databases: Implications for Scientific Databases”. In: *9th International Conference on Scientific and Statistical Database Management* (1997). DOI: [10.1109/SSDM.1997.621141](https://doi.org/10.1109/SSDM.1997.621141). URL: <https://ieeexplore.ieee.org/document/621141>.
- [For24] A. Forum. *How Does Reset Work?* 2024. URL: <https://forum.arduino.cc/t/how-does-reset-work/190779>.

-
- [Git24] Github. *TensorFlowLite: GitHub*. 2024. URL: <https://github.com/tensorflow/tflite-micro-arduino-examples>.
- [Goo21] Google. *TensorFlow Model Optimization Toolkit Weight Clustering API*. 2021. URL: <https://blog.tensorflow.org/2020/08/tensorflow-model-optimization-toolkit-weight-clustering-api.html>.
- [Goo24a] Google. “Build TensorFlow Lite for ARM”. In: *Google for Developers* (2024). URL: <https://ai.google.dev/edge/litert/build/arm>.
- [Goo24b] Google. *Coral AI*. 2024. URL: <https://coral.ai/>.
- [Goo24c] Google. *Coralai doc accelerator*. 2024. URL: <https://coral.ai/docs/accelerator/get-started/#requirements>.
- [Goo24d] Google. *TFL: Inference*. 2024. URL: https://ai.google.dev/edge/litert/microcontrollers/get_started.
- [Goo24e] Google. *TFL: Microconnrtrollers*. 2024. URL: <https://www.tensorflow.org/lite/microcontrollers>.
- [Goo24f] Google. *TFL: ModelConversoinAPI*. 2024. URL: https://www.tensorflow.org/api_docs/python/tf/lite/TFLiteConverter.
- [Goo24g] Google. *TFL: ModelCreationExample*. 2024. URL: <https://www.tensorflow.org/tutorials/keras/classification>.
- [Goo24h] Google. *TensorFlow Lite: Deploy machine learning models on mobile and IoT devices*. 2024. URL: <https://www.tensorflow.org/lite>.
- [Goo24i] Google. *TensorFlow Lite Guide, 2024*. 2024. URL: <https://www.tensorflow.org/lite/guide>.
- [Goo24j] Google. *TensorFlow Lite: Inference*. 2024. URL: <https://www.tensorflow.org/lite/guide/inference>.
- [Goo24k] Google. *TensorFlow Lite: Installation*. 2024. URL: <https://www.tensorflow.org/install/pip>.
- [Goo24l] Google. *TensorFlow Lite: Model Conversion*. 2024. URL: <https://www.tensorflow.org/lite/models/convert>.
- [Goo24m] Google. *TensorFlow Lite: Model Optimization*. 2024. URL: https://www.tensorflow.org/lite/performance/model_optimization.
- [Goo24n] Google. *TensorFlow: Tutorial*. 2024. URL: <https://www.tensorflow.org/tutorials>.

- [Goo24o] Google. *TensorFlow: Wikipedia*. 2024. URL: <https://en.wikipedia.org/wiki/TensorFlow>.
- [Goo24p] Google. *TensorFlowLite: Library*. 2024. URL: <https://github.com/tensorflow/tflite-micro-arduino-examples>.
- [HMD17] S. Han, H. Mao, and W. J. Dally. *Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding*. 2017. URL: <https://arxiv.org/pdf/1510.00149>.
- [Imp23] E. Impulse. *Deploy your model as a C++ library*. 2023. URL: <https://docs.edgeimpulse.com/docs/run-inference/cpp-library/deploy-your-model-as-a-c-library>.
- [Imp24a] E. Impulse. *Arduino Portenta H7*. 2024. URL: <https://docs.edgeimpulse.com/docs/edge-ai-hardware/mcu/arduino-portenta-h7>.
- [Imp24b] E. Impulse. *Edge Impulse CLI Installation*. 2024. URL: <https://docs.edgeimpulse.com/docs/tools/edge-impulse-cli/cli-installation>.
- [Imp24c] E. Impulse. *Edge Impulse Documentation*. 2024. URL: <https://docs.edgeimpulse.com>.
- [Imp24d] E. Impulse. *Edge Impulse*. 2024. URL: <https://www.edgeimpulse.com>.
- [Impnd] E. Impulse. *Example Portenta LoRaWAN*. n.d. URL: <https://github.com/edgeimpulse/example-portenta-lorawan/tree/master>.
- [Ins24] Instructables. *Two Ways to Reset Arduino in Software*. 2024. URL: <https://www.instructables.com/two-ways-to-reset-arduino-in-software/>.
- [JAX24] JAX. *JAX Documentation*. 2024. URL: <https://jax.readthedocs.io/en/latest/>.
- [JKL+20] W. Jiang, J. Konečný, H. Li, et al. “Federated Learning in Edge Computing: A Comprehensive Survey”. In: *IEEE Communications Surveys & Tutorials* (2020). URL: <https://ieeexplore.ieee.org/document/9083958>.
- [Joo25] Jooby. *Unlocking IoT Success: Discover the 8 Key Features of LoRa Technology*. 2025. URL: https://jooby.eu/blog/unlocking-iot-success-discover-the-8-key-features-of-lora-technology/?utm_source=chatgpt.com.

-
- [LLL+22] B. Y. Lin, S. Lee, D.-H. Lee, et al. “The Future of Edge AI: A Comprehensive Review of Current Research and Applications”. In: *arXiv preprint arXiv:2212.03332* (2022). URL: <https://arxiv.labs.arxiv.org/html/2212.03332>.
- [LZW18] F. Liu, Y. Zhou, and D. Wang. “A Research on Machine Learning Methods and Its Applications”. In: *Journal of Physics: Conference Series* (2018). URL: https://www.researchgate.net/publication/327547625_A_Research_on_Machine_Learning_Methods_and_Its_Applications.
- [Li24] A. Li. *TensorFlow Lite rebranded as LiteRT for on-device AI*. 2024. URL: <https://9to5google.com/2024/09/04/tensorflow-lite-liter/>.
- [Lim25] A. Limited. *ARM Cortex-M7 and Cortex-M4 Technical Overview*. 2025. URL: <https://developer.arm.com/ip-products/processors/cortex-m>.
- [MFH00] A. Mockus, R. T. Fielding, and J. Herbsleb. *The KDD process for extracting useful knowledge from volumes of data*. 2000. URL: <https://dl.acm.org/doi/10.1145/240455.240464>.
- [Man25] M. Manufacturing. *CMWX1ZZABZ LoRa Module*. 2025. URL: <https://www.murata.com/en-us/products/communicationmodule/lpwa/lora>.
- [Met24] Meta. *PyTorch*. 2024. URL: <https://pytorch.org/>.
- [Mic24a] MicroPython. *MicroPython GitHub Repository*. 2024. URL: <https://github.com/micropython/micropython>.
- [Mic24b] MicroPython. *MicroPython Official Website*. 2024. URL: <https://micropython.org/>.
- [Nav25] A. Navigation. *Inertial Measurement Unit (IMU) - An Introduction*. 2025. URL: <https://www.advancednavigation.com/tech-articles/inertial-measurement-unit-imu-an-introduction>.
- [Nep24] Neptune.ai. *Deep Learning Model Optimization Methods*. 2024. URL: <https://neptune.ai/blog/deep-learning-model-optimization-methods>.
- [Net25] T. T. Network. *What is LoRaWAN?* 2025. URL: https://www.thethingsnetwork.org/docs lorawan/what-is-lorawan/?utm_source=chatgpt.com.

- [Nor24] Norden. “Facial Recognition System to Improve Security and Access Control”. In: *Norden Communication Blog* (2024). URL: <https://www.nordencommunication.com/en-nl/blog/facial-recognition-system-to-improve-security-and-access-control>.
- [Nov25] H. NovAtel. *APN-064 - Inertial Measurement Unit (IMU) Errors and Calibration*. 2025. URL: <https://hexagondownloads.blob.core.windows.net/public/Novatel/assets/Documents/Bulletins/APN064/APN064.pdf>.
- [Ope23] OpenCV. *TensorFlow Lite Model Optimization for On-Device Machine Learning*. 2023. URL: <https://learnopencv.com/tensorflow-lite-model-optimization-for-on-device-machine-learning/>.
- [Ope24] OpenMV. *OpenMV Documentation*. 2024. URL: <https://docs.openmv.io/>.
- [(PM25)] P. C. (PMC). *Kalman Filter Applications in IMU Sensor Fusion for Accurate Estimation*. 2025. URL: <https://pmc.ncbi.nlm.nih.gov/articles/PMC3812568/>.
- [Pie24] P. Piezotronics. *MEMS Accelerometers*. 2024. URL: <https://www.pcb.com/resources/technical-information/mems-accelerometers>.
- [Res25] ResearchGate. *A Comprehensive Overview of Inertial Sensor Calibration Techniques*. 2025. URL: https://www.researchgate.net/publication/306298761_A_Comprehensive_Overview_of_Inertial_Sensor_Calibration_Techniques.
- [SMS21] G. K. Saha, A. Mandal, and S. Shit. “Real-Time Face Recognition-based Attendance Management System”. In: *IEEE Xplore* (2021). DOI: [10.1109/ICICCS51141.2021.9431473](https://doi.org/10.1109/ICICCS51141.2021.9431473). URL: <https://ieeexplore.ieee.org/document/9431473>.
- [STM24a] STMicroelectronics. *LSM6DSO32X Datasheet*. 2024. URL: <https://www.st.com/resource/en/datasheet/lsm6ds032x.pdf>.
- [STM24b] STMicroelectronics. *LSM9DS1 - 9-axis IMU with digital output*. 2024. URL: <https://www.st.com/en/mems-and-sensors/lsm9ds1.html>.
- [STM24c] STMicroelectronics. *LSM9DS1 Inertial Sensor Datasheet*. 2024. URL: <https://www.st.com/resource/en/datasheet/lsm9ds1.pdf>.

-
- [STM25a] STMicroelectronics. *LSM9DS1 - iNEMO inertial module: 3D accelerometer, gyroscope, and magnetometer*. 2025. URL: <https://www.st.com/en/mems-and-sensors/lsm9ds1.html#overview>.
- [STM25b] STMicroelectronics. *LoRa Connectivity Applications*. 2025. URL: https://www.st.com/en/applications/connectivity/lora.html?utm_source=chatgpt.com.
- [Seg24] S. Segu. *Understanding and Mastering IMU Intrinsic Calibration*. 2024. URL: <https://medium.com/%40sugunsegu/understanding-and-mastering-imu-intrinsic-calibration-ab1e4a4dd3ea>.
- [Shu24] G. Shuklin. *Model Optimization with TensorFlow*. 2024. URL: <https://towardsdatascience.com/model-optimization-with-tensorflow-629342d1a96f>.
- [Stu21] S. Studio. *What is Peer-to-Peer (P2P) LoRa Communication?* Accessed: 2024-12-23. 2021. URL: <https://www.seeedstudio.com/blog/2021/04/26/what-is-peer-to-peer-p2p-lora-communication/>.
- [Sur+22] S. Surve et al. “AttenFace: A Real-Time Attendance System Using Face Recognition”. In: *arXiv* (2022). URL: <https://arxiv.labs.arxiv.org/html/2211.07582>.
- [Tea24] R. Team. *Computer Vision Answer: CNN Applications in AI*. 2024. URL: <https://www.restack.io/p/computer-vision-answer-cnn-applications-cat-ai>.
- [Tec24] TechTarget. *Definition: Convolutional Neural Network (CNN)*. 2024. URL: <https://www.techtarget.com/searchenterpriseai/definition/convolutional-neural-network>.
- [Ten] *TensorFlow Lite Guide*. TensorFlow. 2024. URL: <https://www.tensorflow.org/lite/guide>.
- [Vas24] F. Vascellari. *Lightweight Convolutional Neural Networks for Embedded Systems*. 2024. URL: https://thesis.unipd.it/retrieve/c80af4b1-2d8f-4878-b690-9c4b6ae76fda/Vascellari_Filippo.pdf.
- [Vec25] VectorNav. *Inertial Navigation Primer - Specifications and Error Budgets*. 2025. URL: <https://www.vectornav.com/resources/inertial-navigation-primer/specifications--and--error-budgets/specs-imuspecs>.
- [WHW+23] X. Wang, R. Han, C. Wang, et al. “Edge Devices Inference Performance Comparison”. In: *arXiv preprint arXiv:2306.12093* (2023). URL: <https://arxiv.labs.arxiv.org/html/2306.12093>.

- [Wh24] Wings-hub. *PortentaH7/PortentaH7*. 2024. URL: <https://github.com/Wings-hub/240411PortentaH7/tree/main/PortentaH7/PortentaH7>.
- [Wik24] Wikipedia. *PyTorch*. 2024. URL: <https://en.wikipedia.org/wiki/PyTorch>.
- [Win24] P. E. Wings. *Lecture in Data Science and Analytics: Knowledge Discovery in Database (KDD) Process*. Master in Industrial Informatics, Hochschule Emden/Leer University of Applied Science. 2024.
- [YXW+20] Y. Yang, Y. Xu, W. Wang, et al. “Smart Edge Computing for Autonomous Driving: A Review of the Edge Intelligence Framework”. In: *IEEE Internet of Things Journal* (2020). URL: <https://ieeexplore.ieee.org/document/9263894>.
- [You25] YouTube. *How to Use Watchdog Timers in Microcontrollers*. 2025. URL: <https://www.youtube.com/watch?v=hT24h0TrbEI>.
- [ZCL+19] Z. Zhou, X. Chen, E. Li, et al. “Edge Intelligence: The Confluence of Edge Computing and Artificial Intelligence”. In: *Proceedings of the IEEE* (2019). URL: https://www.researchgate.net/publication/335650663_Edge_Intelligence_The_Confluence_of_Edge_Computing_and_Artificial_Intelligence.

Index

File
.bat, 47
.cmd, 47
.tflite, 218, 220, 222
Arduino LSM9DS1.h, 130, 131
arduino-cli, 31
arduino-cli.exe, 26
arduino-cli.yaml, 27, 43
arduino-cli_1.0.4_Windows_64bit.zip,
 26
arduino-ide_2.3.2_macOS_64bit.dmg,
 10
Arduino_data, 27
BUILD, 230
C:/Users/user.name, 44
cli_test, 44
directories.downloads, 37
flash_windows, 255
Kalman.h, 130, 131, 145
Keras H5 format, 220
Keras model, 220
MakeLists.txt, 230
Models built from concrete func-
 tions, 220
monitor_log.bat, 51
README.md, 32
SavedModel, 220
sketchbook_path, 27
Wire.h, 130, 132
wire.h, 131

CLI, 25
Command Line Interface
 see CCI, 25