

Implementation of a Data Acquisition Tool for Sensor Measurements

Christoffer Nylund

Degree Thesis for Master of Engineering

Degree Programme in Automation Technology

Vaasa, 2023

DEGREE THESIS

Author: Christoffer Nylund

Degree Programme and place of study: Automation Technology, Vaasa

Specialization: Intelligent systems

Supervisor: Jan Berglund

Title: Implementation of a Data Acquisition Tool for Sensor Measurements

Date: 26.11.2023 Number of pages: 95 Appendices: IX

Abstract

It has become more common with the use of sensors to collect different types of data from the environment and motion in connection with product development and research. This thesis work was done for the process control lab at Novia UAS, which wanted a microcontroller that collects data and sends it wirelessly to a graphical user interface that the students could use for visualizing gauges and graphs and use that data for later processing.

The work has been divided into different programming languages for different parts of the data transmission. Programming languages JavaScript, CSS, and HTML are used for developing a graphical user interface. The microcontroller board is programmed in the programming language C / C++. A single-board computer acts as an intermediate communication between the microcontroller and the graphical user interface, where a programming tool based on JavaScript named Node-Red is used. The wireless communication is based on Bluetooth Low Energy and the internet communication uses a standard-based messaging protocol named MQTT. The work also includes planning and 3D construction of a protective box for the microcontroller with the use of a software named Rhinoceros.

The result of this thesis has made it possible for the students at Novia to easily inspect and collect data from environment sensors and motion sensors and then decide to download it for later processing.

Language: English

Key Words: Arduino, Raspberry Pi, Node-Red, Internet of Things, 3D Drawing

EXAMENSARBETE

Författare: Christoffer Nylund

Utbildning och ort: Automationsteknik, Vasa

Inriktning: Intelligent system

Handledare: Jan Berglund

Titel: Implementering av datainsamlingsverktyg för sensormätningar

Datum 26.11.2023 Sidantal 95

Bilagor IX

Abstrakt

Det har blivit allt vanligare att med sensorer samla in olika typer av data om omgivningen och rörelse i samband med bland annat produktutveckling och forskning. Det här examensarbetet är gjort för yrkeshögskolan Novias processkontrollslaboratorium. Högskolan behövde en mikrokontroll som samlar upp data och skickar det trådlöst till ett användargränssnitt där studerande kan se data i mätare och grafer. Datat kan också sparas för att senare användas i olika typer av analyser.

Inom examensarbetet används det i olika skeden av dataöverföringen olika programspråk. Programmeringsspråken som används för det grafiska användargränssnittet är JavaScript, CSS och HTML. I mikrokontrollerkortet används C / C++ programspråk. En enkelkorts dator används som en mellanliggande kommunikator mellan mikrokontrollen och användargränssnittet.

Enkelkorts datorn använder sig av ett programmeringsverktyg kallat Node-Red som baserar sig på JavaScript. Den trådlösa kommunikationen sköts med hjälp av Bluetooth på låg energi. Sedan skickas data över från ett standardiserat meddelandeprotokoll som heter MQTT. Utöver programmeringen består slutarbetet också av en skyddande låda för mikrokontrollen. Den skyddande lådan har skapats som en 3d-konstruktion i ett program som heter Rhinoceros.

Det här slutarbetet har gjort det möjligt för studerande vid yrkeshögskolan Novia att ta del av, ladda ner och senare processa data från olika typer av omgivnings- och rörelsesensorer.

Språk: Engelska

Nyckelord: Arduino, Raspberry Pi, Node-Red, Sakernas Internet, 3D ritning

OPINNÄYTETYÖ

Tekijä: Christoffer Nylund

Koulutus ja paikkakunta: Automaatiotekniikka, Vaasa

Suuntautumisvaihtoehto: Systeemiälylliset järjestelmät

Ohjaaja: Jan Berglund

Nimike: Datankeruutyökalun hyödyntäminen sensorimittauksissa

Päivämäärä 26.11.2023

Sivumäärä 95

Liitteet IX

Tiivistelmä

Tuotekehityksessä ja tutkimuksessa erilaisten sensoreiden hyödyntäminen ympäristön ja liikkeen datankeruussa on yleistynyt. Tämä opinnäytetyö on tehty Novian prosessikehityslaboratoriolle. Korkeakoulu tarvitsi opiskelijoita varten mikrokontrollerin, joka kerää dataa ja lähettää tämän langattomasti käyttöliittymään. Tämän käyttöliittymän avulla opiskelijat voivat nähdä kerätyn datan erilaisissa mittareissa ja kaavioissa. Data voidaan tätä kautta myös tallentaa myöhempää käyttöä varten.

Opinnäytetyön tiedonsiirron eri vaiheissa hyödynnetään eri koodikieliä. Graafisessa käyttöliittymässä hyödynnetään JavaScriptiä, CSS:ää ja HTML:ää. Mikrokontrollerin kortissa taas hyödynnetään C / C++ ohjelmakieltä. Yksilevyinen tietokone hoitaa tiedonvälityksen mikrokontrollerin ja käyttöliittymän välillä. Tiedonvälityksessä hyödynnetään Node-Red ohjelmointityökalua, joka perustuu JavaScriptiin ja tiedonvälitys toimii pienellä energialla toimivan Bluetoohtin sekä MQTT-nimisen viestintäprotokollan avulla.

Ohjelmointiosuuden lisäksi myös 3d-tulostettu laatikko on osa opinnäytetyötä. Laatikko on osana opinnäytetyötä suunniteltu suojaamaan mikrokontrolleria. Laatikko on suunniteltu Rhinoceros-nimisessä ohjelmassa.

Tämä opinnäytetyö on mahdollistanut ammattikorkeakoulu Novian opiskelijoille ympäristö- ja liikesensoreista tulevan datan lataamisen ja tallentamisen myöhempää käsittelyä varten.

Kieli: Englanti

Avainsanat: Arduino, Raspberry Pi, Node-Red, Esineiden internet, 3D piirustus

LIST OF ABBREVIATIONS

AR	Augment Reality
AI	Artificial Intelligence
API	Application Programming Interface
BSEC	Bosch Software Environmental Cluster
BR	Basic Rate
BLE	Bluetooth Low Energy
bVOC	Biogenic Volatile Organic Compounds
CAD	Computer-Aided Design
DLE	Data Length Extension
EDR	Enhanced Data Rate
GNSS	Global Navigation Satellite System
IDE	Integrated development environment
IMU	Inertial Measurement Unit
IoT	Internet of Things
LWT	Last Will and Testament
MR	Mixed Reality
MQTT	Message Queuing Telemetry Transport
NaN	Not a number
npm	Node Package Manager
NURBS	Non-Uniform Rational B-Splines
ODR	Output Data Rate
OS	Operation System
OSI	Open Systems Interconnection
RPI	Raspberry Pi
SBC	Single Board Computer
UAS	University of Applied Sciences
VDI	Virtual Desktop Infrastructure
VOCs	Volatile Organic Compounds
VR	Virtual Reality
VSCs	Volatile Sulfur Compounds
QoS	Quality of Service

Table of Content

1	Introduction	1
1.1	Novia University of Applied Sciences	2
1.1.1	Technobothnia.....	3
1.2	Internet of Things.....	4
2	Hardware.....	5
2.1	Arduino.....	5
2.1.1	Nicla Sense ME.....	6
2.2	Raspberry Pi.....	14
3	Software	16
3.1	Arduino IDE	16
3.2	Raspberry Pi OS	18
3.3	Rhinoceros	19
3.4	UltiMaker Cura	23
4	Communication protocols.....	25
4.1	Bluetooth	26
4.2	Message Queuing Telemetry Transport	32
5	Programming & markup languages/tool	34
5.1	C / C++	34
5.2	JavaScript.....	37
5.3	HyperText Markup Language	38
5.4	Cascading Style Sheets	39
5.5	Node-Red	39
5.5.1	Node-Red editor.....	40
5.5.2	Nodes.....	41
6	Implementation	43
6.1	Implementation of Nicla Sense ME.....	43
6.2	Implementation of Raspberry Pi	49
6.2.1	Installation of Raspberry Pi OS	49
6.2.2	Monitor Raspberry Pi from VNC Viewer	50
6.2.3	Installing MQTT broker on RPI	51
6.2.4	Install and configure BlueZ.....	55
6.2.5	Installing and implementation of Node-Red on RPI	57
6.3	Implementation of a graphical user interface.....	68
6.3.1	HTML (Hyper Text Markup Language)	69
6.3.2	CSS (Cascading Style Sheet).....	71
6.3.3	JS (JavaScript).....	73

6.4	Construction of Nicla Sense protection box.....	78
6.4.1	3D drawing of protection box.....	78
6.4.2	3D printing of protection box.....	81
7	Discussion and results.....	83
7.1	Further research.....	88
7.2	Acknowledgments.....	89
8	References.....	90

1 Introduction

This thesis work is made for Novia University of Applied Sciences, more accurately the process control lab in Technobothnia. The developed tool is a measuring equipment, where a microcontroller named Nicla Sense ME collects data and communicates wirelessly with a Raspberry Pi, which in turn communicates with a graphical user interface.

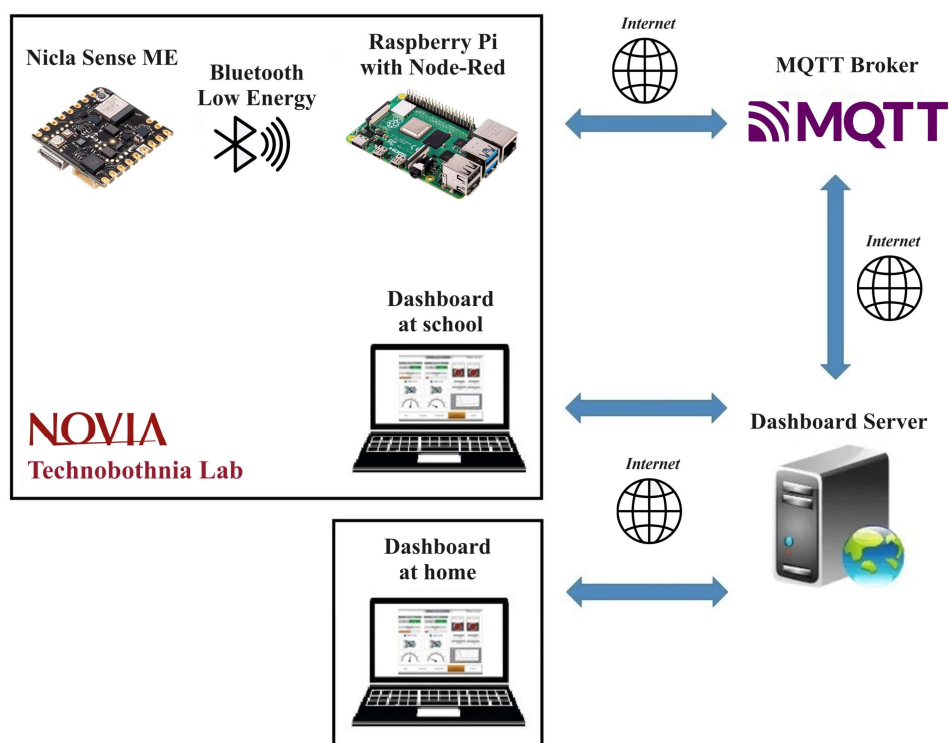


Figure 1. View of sensor data acquisition tool communication.

The thesis is delimited to the following tasks:

- Construct a 3D-printed protection box for a Nicla Sense ME board including a battery.
- Create and develop a graphical user interface that will read and visualize real-time sensor data from the Nicla Sense microcontroller:
 - Sensor data included: Accelerometer, Gyroscope, Temperature, Humidity, Pressure, Gas and Co2.

- Possibility to select sensors that should be logged and set the sampling time for the environment sensors and motion sensors.
- Possibility to save the logged sensor data measurements to CSV file to the local computer.
- Set up a Bluetooth and internet communication bridge module between the microcontroller and the graphical user interface with the use of a single-board computer.
- Use an MQTT broker
- Nicla Sense led indicator (connected or disconnected).
- Manual with instructions for usage.

After the thesis work is completed, the students in the process control lab will have a device that can be connected to and from a webpage. The webpage includes a graphical user interface to visualize and inspect real-time sensor data with graphs and gauges from different experiments. It is also possible to collect and save the data from the graphical user interface to a local computer for further processing. The students will also be able to control the Nicla Sense from the graphical user interface.

The tool makes it possible for students to read and process the data outside the lab, for example at home or in school.

1.1 Novia University of Applied Sciences

Novia University of Applied Sciences is Finland's largest Swedish-speaking UAS where the area of operations can be found along the Finnish West Coast. The University of Applied Sciences has five campuses: Vaasa Campus, Turku Campus, Aboa Mare (Turku), Raasepori Campus, and Campus Allegro (Pietarsaari) (A High-Class and Dynamic University of Applied Sciences, u.d.).

To mention Subic Bay in the Philippines, Novia UAS owns stock in Giga Mare Inc., a Philippine company that provides education for the maritime industry. The ownership is seen as a tactical way to advance globalization and strengthen its position in Asia (A High-Class and Dynamic University of Applied Sciences, u.d.).

At Novia University of Applied Sciences, there are about 4800 students and the workforce consists of approximately 320 people. There are five departments in Novia: Technology and Shipping, Health and Welfare, Business Administration, Art and Culture, and Bioeconomy (A High-Class and Dynamic University of Applied Sciences, u.d.).

Novia offers exclusive education in the mentioned subjects above where it is possible to choose a bachelor's degree or master's degree program, continuing education studies, open UAS studies, or open path studies (A High-Class and Dynamic University of Applied Sciences, u.d.).



Figure 2. Novia UAS logo (Product logos, u.d.).

1.1.1 Technobothnia

Technobothnia is a laboratory in Palosaari Campus in Vaasa that was founded in the year 1996 and is co-owned by three universities: Novia University of Applied Sciences, VAMK University of Applied Sciences, and The University of Vaasa (Technical education with a royal lineage, u.d.).

At Technobothnia students have the opportunity to get practical experiences in engineering and to improve collaboration between the institutions and serve as a platform for improved collaboration between the Vaasa region's technical sector and technical education (Technical education with a royal lineage, u.d.).

Technobothnia strives to form a basis for technology-related research and education of the highest class, play an intermediary role for collaboration between businesses, organizations, and other research institutes and technology institutes, and provide teaching, measuring, testing, research, product development, and other services to the public/private sectors (Technical education with a royal lineage, u.d.).



Figure 3. Technobothnia co-owned schools, VAMK, Vaasan yliopisto & Novia (Technobothnia, u.d.).

1.2 Internet of Things

The Internet of Things also known as IoT could be described as a system of devices that are related to each other and connected to a network to transfer data information without necessarily involving human-to-machine interaction. From other aspects, it is a set of tech devices that can interact with one another. It could be used for example smart homes and smart factories (Science & Technology , 2019).

The devices of IoT are usually called smart devices due to the use of sensors and complex analysis programs. The IoT devices use sensors to gather data, analyze the data, and provide services to the user depending on user-defined criteria (Science & Technology , 2019).

IoT devices can be connected to the internet directly from the device or be connected through another device. Information sharing and user interaction take place across the network. By adding software applications, the IoT establishes links and connections between actual physical objects. With the use of IoT devices, users can get information or manage devices from any location (Science & Technology , 2019).

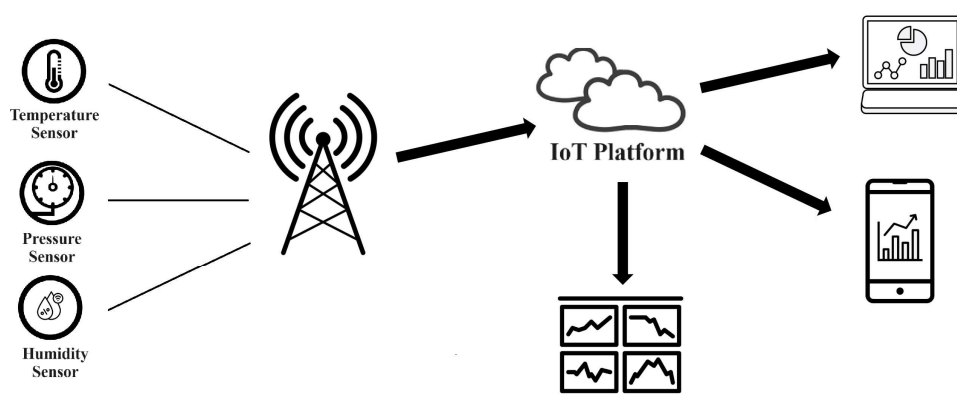


Figure 4. Overview of the Internet of Things.

2 Hardware

This chapter includes information about the hardware that is used in the thesis work. Chapter 2.1 introduces the Arduino board Nicla Sense ME and the different sensors. Chapter 2.2 basic information about the Raspberry Pi and the hardware of version 4.

2.1 Arduino

An open-source electronics platform called Arduino is built on simple hardware and software. The Arduino involves both a programmable microcontroller circuit board and software that runs on a PC to write and upload code to the physical board. Sending a set of commands to the Arduino board's microcontroller will instruct the board on what to do depending on the code instructions uploaded to the board (About Arduino, n.d.).

There are already a bunch of Arduino boards developed but these differ from each other in hardware components and functionalities. For example what kind of sensors are included and if there are built-in Bluetooth or Wi-Fi. (About Arduino, n.d.).

Arduino is used by a worldwide community of developers such as students, hobbyists, programmers, and experts where thousands of different projects have been created and the experience between each other could be shared in the community (About Arduino, n.d.).

The main idea with Arduino was from the beginning to have a device for fast prototyping for students without a background in programming and electronics but after this was found by a wider community the concept had changed to suit other needs and challenges as it could offer simple 8-bit boards to products for IoT applications, wearables, embedded environments, and more on (About Arduino, n.d.).



Figure 5. Arduino company logo (About Arduino, n.d.).

2.1.1 Nicla Sense ME

Nicla Sense ME is a small microcontroller developed by Arduino in September 2021, which includes four different integrated sensors manufactured by Bosch Sensortec where BHI260AP is a motion sensor with integrated AI, BME688 is a 4-in-1 gas sensor with AI and integrated high linearity and high accuracy pressure, humidity, and temperature, BMP390 is a high-pressure sensor and BMM150 is a magnetometer sensor (Sensing and intelligence at the edge become accessible to all, with Nicla Sense ME by Arduino Pro and Bosch Sensortec, u.d.). Figure 6 demonstrates all the mentioned sensors and where they are placed on the Nicla Sense microcontroller chip. ME in the name of Nicla Sense stands for **Motion and Environment** (Nicla Sense ME, u.d.).

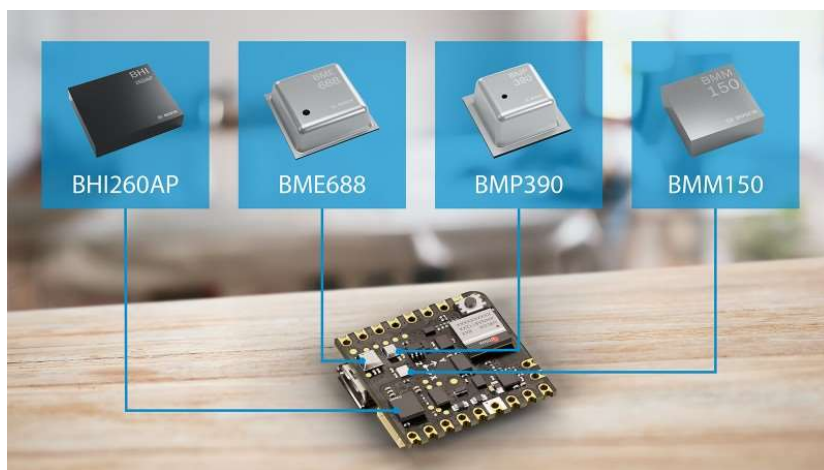


Figure 6. Arduino Nicla Sense ME with four Bosch Sensortec sensors (Arduino Nicla Sense ME, n.d.).

The Nicla Sense is a development board that requires ultra-low power consumption and with its compact form and Bluetooth Low Energy communication, it can be used in many cases for IoT (Nicla Sense ME, u.d.).

Features of the Nicla Sense Bluetooth specification can be seen in Table 1 and operating conditions in Table 2.

Table 1. Features of Nicla Sense ME BLE.

Features	Description
Bluetooth Module	ANNA-B112
nRF52832 System-on-chip	64 MHz ARM Cortex-M4F microcontroller, 64 KB SRAM, 512 KB Flash, RAM mapped FIFOs using EasyDMA, 2 x SPI, 2 x I2C, 12-bit/200 ksps ADC, 2.400 – 2.4835 GHz Bluetooth (5.0 via cordio stack, 4.2 via Arduino BLE)
Internal antenna	
Internal 32 MHz oscillator	
1.8 V Operating Voltage	

(Nicla Sense ME, 2023).

Table 2. Recommended operating condition for Nicla Sense.

Symbol	Description	Min	Type	Max	Unit
V_{IN}	Input voltage from VIN pad	3.5	5.0	5.5	V
V_{USB}	Input voltage from USB connector	4.8	5.0	5.5	V
T_{OP}	Operating Temperature	-40	25	85	°C

(Nicla Sense ME, 2023).

ANNA-B112 Bluetooth module works with BLE version 5.0 and it includes an internal antenna that provides a communication range of up to 160 meters (ANNA-B112-00BU-BLOX, 2021).

Arduino has performed a test by measuring the power consumption in standby mode, a blink sketch, and when advertising with sensors polling. The sensors that were activated during the test were the temperature sensor, accelerometer, and gyroscope which have been configured at 1 Hz, 1 ms latency (Nicla Sense ME, 2023).

Table 3. Power consumption test performed by Arduino.

Description	Typ	Unit
Power consumption on standby	460	uA
Power consumption with blink sketch	960	uA
Power consumption advertising with sensor polling at 1 Hz	2.5	mA

(Nicla Sense ME, 2023).

Battery

Nicla Sense ME can be powered up by using a single-cell 3.7V Li-Po or Li-Ion battery and there are two different ways to connect. The first option uses the header pins that can be seen in Figure 7, the example illustrates Nicla Vision but the pin placements are identical for Nicla Sense. “VBAT” is where the plus wire should be connected and “GND” is where the minus wire should be connected. For some models of batteries, there can be a third wire called NTC which is used to prevent batteries from being charged at temperatures that are too high or too low (Connect a battery to Nicla Sense ME or Nicla Vision, 2022).

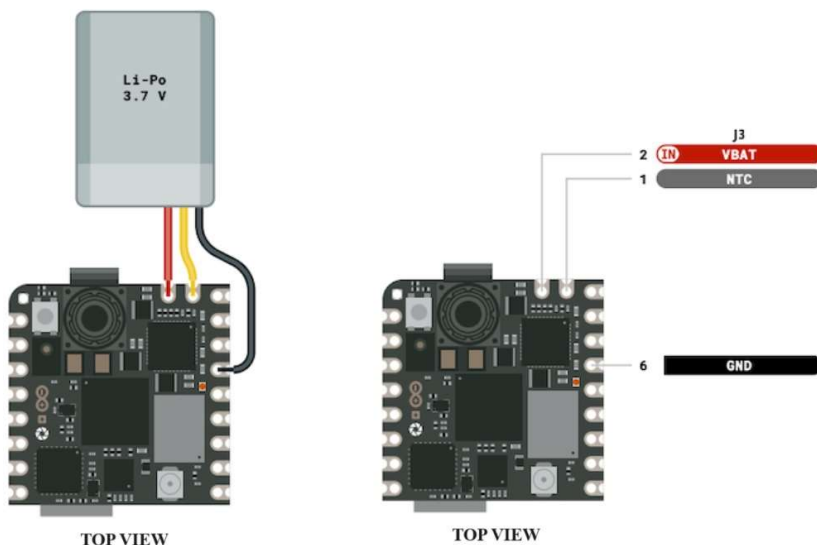


Figure 7. Connect the battery to header pins (Connect a battery to Nicla Sense ME or Nicla Vision, 2022).

The second option to connect a battery is by a connector, which can be found at the bottom of the microchip. The connector is a shrouded header from JST of model “BM03B-ACHSS-GAN-TF”, which is compatible with the ACHR-03V-S connector housing. The positive terminal is connected to the header pin nearest to the USB connector and GND is

connected to the header pin farthest from the USB connector. If there is a NTC wire available for the battery it should be connected to the middle header pin that can be seen in Figure 8 (Connect a battery to Nicla Sense ME or Nicla Vision, 2022).

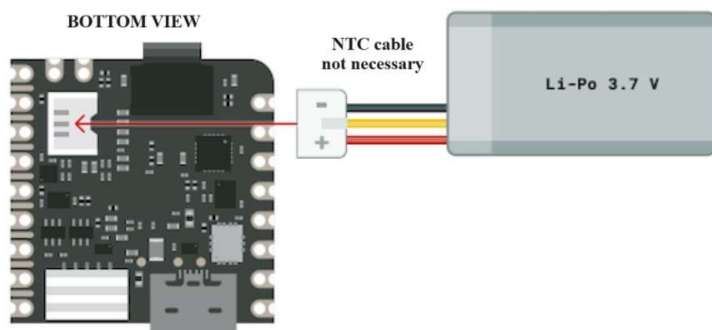


Figure 8. Connect the battery to the connector (Connect a battery to Nicla Sense ME or Nicla Vision, 2022).

BHI260AP (IMU smart sensor)

BHI260AP is an ultra-low power smart sensor developed by Bosch Sensortec. It has a programmable 32-bit microcontroller that contains a 6-axis IMU that will give accelerometer and gyroscope readings (16-bit 3-axis accelerometer + 16-bit 3-axis gyroscope). The sensor also includes self-learning features and an event-driven software framework designed for signal data processing and contains pre-installed sensor fusion which hosts a programmable recognition system and other sensor data processing software (BHI260AP - Ultra-low power, high performance, self-learning AI smart sensor with integrated accelerometer and gyroscope, 2021).

This sensor also handles communication with other sensors on the Arduino Nicla Sense ME board, the interface types are I2C, SPI, and a 2 MB Flash are available used to store and execute in-place code and data storage like the Bosch sensor fusion algorithm (BSX) calibration data. The BHI260AP can collect data that could be trained on a PC (Arduino® Nicla Sense ME, 2023).

The BHI260AP smart sensor is used for fitness tracking, navigation, machine learning analytics, and orientation estimation. This sensor is a useful solution when it comes to always-on sensor processing at extremely low power consumption. Some target applications for this sensor are:

- Wrist wearables for example smartwatches or fitness bands.
- Head-mounted devices such as headsets, smart sunglasses, or wireless in-ear devices.
- Mobile communication devices like smartphones.
- AR/VR/MR headset and controller devices (BHI260AP - Ultra-low power, high performance, self-learning AI smart sensor with integrated accelerometer and gyroscope, 2021).

BHI260AP can be used without additional programming due to its pre-set solution. It consists of over 22 built-in virtual sensors (algorithms). These are independently configurable sensor data processing algorithms that provide software-programmed functionality. For example device orientation in various formations, gesture recognition algorithms, dynamic offset calibration, activity recognition, and step counting (BHI260AP - Ultra-low power, high performance, self-learning AI smart sensor with integrated accelerometer and gyroscope, 2021).

In addition to the libraries available for the previous sensor variant BHI260AB, the BHI260AP can run and use a new set of libraries which includes complex algorithms for a variety of use cases, especially useful software is the pedestrian dead reckoning and the swim analysis library. This sensor can also be used for self-learning AI software for fitness tracking and relative and absolute orientation (BHI260AP - Ultra-low power, high performance, self-learning AI smart sensor with integrated accelerometer and gyroscope, 2021).

The frequency at which the BHI260AP supplies the data from the virtual sensor is defined by the sample rate also known as the Output Data Rate (ODR). The data rates of the accelerometer support up to 1600 Hz and the gyroscope 6400 Hz (BHI260AP - Ultra-low power, high performance, self-learning AI smart sensor with integrated accelerometer and gyroscope, 2021). In Table 4, the features and power consumption according to the technical data of the BHI260AP sensor can be seen.

Table 4. Features and power consumption of sensor BHI260AP.

Features	Description		
Package	3.6 mm x 4.1 mm x 0.83 mm		
Supply Voltage	Min: 1.71V Typ: 1.8V Max: 3.6V		
Operating temperature	-40 ... +85 °C		
Parameter	Technical data		
Game Rotation Vector (Accelerometer & Gyroscope)	ODR	Long Run	Turbo
	25 Hz	1.068 mA	1.085 mA
	400 Hz	1.301 mA	1.372 mA
	800 Hz	-	1.779 mA
Self-learning AI function (25 Hz)	249 uA		

(BHI260AP - Ultra-low power, high performance, self-learning AI smart sensor with integrated accelerometer and gyroscope, 2021).

BME688 (Gas sensor)

The sensor BME688 is the first gas sensor integrated with AI and high-linearity / -accuracy pressure, humidity, and temperature sensors. Size and low power consumption are some advantages. The gas sensor has a part per billion (ppb) detection range for VOCs, VSCs, and other gases like carbon monoxide and hydrogen (Gas Sensor BME688, 2022).

Useful applications of this sensor could be air quality measurement indoors and outdoors, detection of unusual gases and smells example leakage or fire, and bacteria growth detection (Gas Sensor BME688, 2022).

The BSEC generates a range of useful outputs based on complex algorithms, such as the Index for Air Quality (IAQ) between 0 (clean air) and 500 (extremely polluted air), bVOC- & Co₂- equivalents (ppm), and Gas scan results (%) (Gas Sensor BME688, 2022).

In Table 5, the key features of the gas sensor are shown including the size, operating range, gas sensor output, and some test measurements of current consumption. In Table 6 choice of BSEC outputs can be seen with descriptions (Gas Sensor BME688, 2022).

Table 5. Features of BME688 sensor.

Features	Description
Package	3.0 mm x 3.0 mm 0.93 mm metal lid LGA
Current consumption	2.1 uA at 1 Hz humidity and temperature 3.1 uA at 1 Hz pressure and temperature 3.7 uA at 1 Hz humidity, pressure, and temperature 0.15 uA in sleep mode
Operating range	-40 - + 85 °C, 0-100% r.H., 300-1100 hPa
Gas sensor output data processing	IAQ, bVOC- & Co2-equivalents (ppm), Gas scan result (%) & Intensity level

(Gas Sensor BME688, 2022).

Table 6. Description of one part of BSEC outputs.

Output	Description
Sensor-compensated temperature (°C)	Temperature which is compensated for internal cross-influences caused by the BME sensor.
Sensor-compensated relative humidity (%)	The relative humidity is compensated for internal cross-influences caused by the BME sensor.
Sensor-compensated gas resistance (Ω)	Raw gas resistance is compensated by temperature and humidity influences.
CO ₂ equivalents (ppm)	Estimation of the CO ₂ level in ppm. The sensor does not directly measure CO ₂ but derives this from the average correlation between VOCs and CO ₂ in human exhaled breath.

(Gas Sensor BME688, 2022).

BMP390 (Digital pressure sensor)

The BMP390 is a digital sensor that measures temperature and pressure using proven sensing methods. The housing for the sensor module is incredibly small and its compact size and extremely low power consumption allow for integration into battery-operated devices like watches and mobile phones (Pressure sensor BMP390, 2021).

In Table 7, features of the BMP390 sensor can be seen: package size, current consumption, and operating range.

Table 7. Features of sensor BMP390.

Features	Description
Package	2.0 mm x 2.0 mm x 0.75 mm metal lid LGA
Current consumption	3.2 uA at 1 Hz pressure and temperature, 1.4 uA in sleep mode.
Operating range	-40 ... +85 °C, 300 – 1250 hPa

(Pressure sensor BMP390, 2021).

The sensor has greater accuracy than its predecessor BMP380, and it can monitor pressures between 300 and 1250 hPa. The BMP390 can operate in three different modes: sleep mode where no measurements are taken, normal mode which contains an active measurement period and an inactive standby period, and forced mode where one measurement is made and switches back to sleep mode after the measurement is complete. (Pressure sensor BMP390, 2021).

BMM150 (Magnetometer sensor)

The BMM150 is a geomagnetic sensor for everyday uses. It enables magnetic field measurements in three perpendicular axes. The functionality and features of the BMM150 have been carefully developed to meet the high standards of all 3-axis mobile applications, including for example navigation and electronic compasses (Magnetometer BMM150, 2020).

The output of the geomagnetic sensor is converted by an evaluation circuit (ASIC) into digital results that can be read out using SPI or I2C digital interfaces. In Table 8 features of the sensor BMM150 are shown which include the size, power consumption, magnetic field range typical, and temperature range (Magnetometer BMM150, 2020).

Table 8. Key features of sensor BMM150.

Key features	Description
Package	Footprint 1.56 x 1.56 mm ² , height 0.6 mm
Power consumption	170 uA at 10 Hz in low power preset.
Magnetic field range typical	±1300uT (x,y-axis) ±2500uT (z-axis) Magnetic field resolution of -0.3uT
Temperature range	-40 ... +85 °C

(Magnetometer BMM150, 2020).

2.2 Raspberry Pi

The first model of Raspberry Pi Model B, generation 1 was released in February 2012. The low cost of this hardware results in total success. The Raspberry can be used as a working computer just by connecting a keyboard, mouse, and monitor. An SD card is needed also to install example the operating system Raspbian. Raspbian is a Debian-based Linux operating system, and it boots up just by connecting power to it. The Raspberry is usually referred to single-board computer (SBC), which means it runs a full operating system and has enough memory, CPU, and power regulation to run without extra hardware (The Raspberry Pi: A Technology Disrupter, and the Enabler of Dreams, 2017).

Other SBCs that could be found before Raspberry Pi were mostly mentioned for industrial platforms meanwhile the Raspberry Pi developed a board that almost anyone could use (The Raspberry Pi: A Technology Disrupter, and the Enabler of Dreams, 2017).

Raspberry Pi 4

In the year 2019 a new model of the Raspberry Pi computer series was developed named Raspberry Pi 4 Model B. Compared to the previous generation, it provides advancements in CPU speed, multimedia performance, memory, and connectivity (Ltd., Raspberry Pi Trading, 2019).

The format size for RPI4 is the same as the previous model RPI3 but the new model has upgraded and better performance. In Figure 9 a picture of the hardware can be seen, and in Table 9 the specifications of the RPI4.

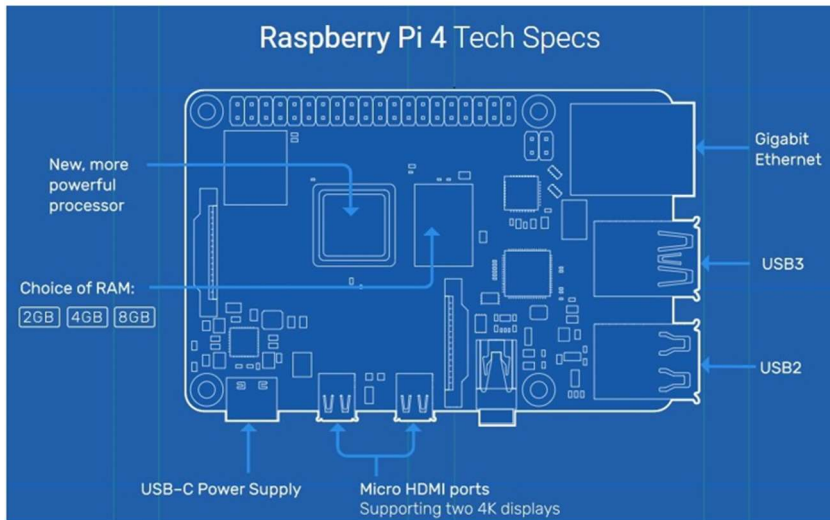


Figure 9. Raspberry Pi 4 hardware. (Raspberry Pi 4 Tech Specs, u.d.).

Table 9. Specifications of RPI4.

Specifications
Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC 1.8 GHz 1 – 8 GB LPDDR4-3200 SDRAM
2.4 GHz and 5.0 GHz IEEE 802.11 ac wireless, Bluetooth 5.0, BLE Gigabit Ethernet
2 USB 3.0 ports; 2 USB 2.0 ports
2 x micro-HDMI ports (4kp60 supported)
Micro-SD card slot for loading OS and data storage
5V DC via USB-C connector (minimum 3 A)
Operating temperature 0 – 50 degrees

(Raspberry Pi 4 Tech Specs, u.d.).

3 Software

This chapter explains the different software programs that have been used. Insight into the Arduino software program Arduino IDE, after that the Raspberry Pi's operating system, and then explaining the 3D programming software Rhinoceros and the printing program UltiMaker Cura.

3.1 Arduino IDE

The Arduino Integrated Development Environment is a software tool that is used to program the Arduino board. It includes a text editor for writing code, a message area, a text console, a toolbar with buttons for common functions, and several menus (Getting Started with Arduino IDE 2.0, 2022).

The Arduino IDE software establishes a connection with the Arduino board and communicates to upload programs. Programs that are written in Arduino IDE are with other words called sketches and these are created and saved with the file extension *.ino*. The text editor has quite similar features as other editors, for example cutting/pasting and searching/replacing text. When saving and uploading to the board, the message area provides feedback and shows errors if there are any. The configured board and serial port are visible in the bottom right corner. The toolbar buttons enable to create, open, save sketches, validate, and upload programs and open the serial monitor to follow what is going on the Arduino board (Getting Started with Arduino IDE 2.0, 2022).

Figure 10 shows an overview of the Arduino IDE software. The numbers are shown with their descriptions below.

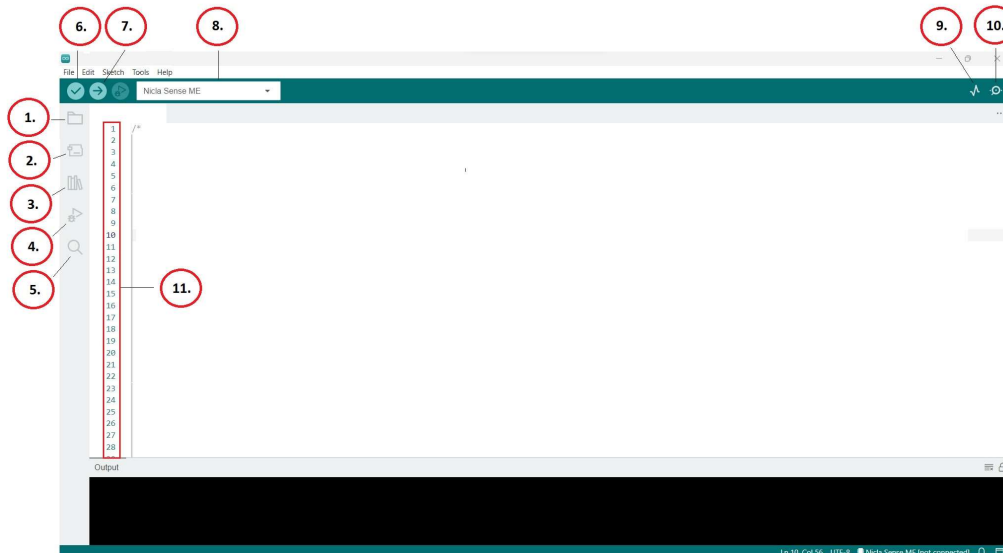


Figure 10. Explanation of Arduino IDE functions.

1. **Sketchbook:** On this tab, it is possible to find all the sketches that are locally stored on the computer where the code file is kept. It is even possible to sync with the Arduino Cloud to receive the sketches from the internet environment. Arduino sketches are saved as .ino files and must be kept in a folder with the same name. For instance, in a folder called *Arduino_test*, the sketch must be saved as *Arduino_test.ino* and should not be changed.
2. **Boards Manager:** Used to browse through Arduino and third-party packages that could be installed to the software. For example, when using the Nicla Sense ME it is required to install the Nicla boards package intended for the board otherwise it will not be possible to compile and upload the code for the specific board.
3. **Library Manager:** There are thousands of Arduino libraries here to browse between, developed by Arduino and from its community. Libraries are extensions of the Arduino API that make it simpler to do things like the use of the MQTT module and read sensors.
4. **Debug:** Used to test and debug programs, it can be applied to controllably go through a program's execution.
5. **Search:** If it is difficult to find something that has been written in the code, then a search command could be used.
6. **Verify:** Verify button compile and check for errors.
7. **Upload:** Upload button assembles the program code before uploading it to the chosen board.

8. **Select Board and Port:** From here it finds the Arduino boards that are available along with the port number.
9. **Serial Plotter:** Serial Plotter opens a new tab in the console that checks outgoing values from *Serial.print()* command to inspect for example temperature values. The tool is open in a separate window but integrated with the editor.
10. **Serial Monitor:** The Serial Monitor tool enables data streaming from the board with for example *Serial.print()* command. This tool was previously located in a separate window, but it is now a part of the editor. This makes it simple to run numerous instances concurrently.
11. **Number column:** When errors occur, it will describe the problem and refer to a number to find the fault faster (Getting Started with Arduino IDE 2.0, 2022).

3.2 Raspberry Pi OS

The recommended operating system for normal use on an RPI device is the Raspberry Pi OS which is a free operating system built on Debian OS. This operating system includes more than 35000 packages of pre-compiled programs for quick installations on the Raspberry Pi. With a focus on improving the reliability and performance of as many Debian packages as possible, the Raspberry Pi OS is actively being developed (Raspberry Pi OS, u.d.).

It is essential to update the Raspberry Pi OS gradually. Security is the first and primary factor. There are millions of lines that keep up Raspberry Pi OS on an RPI device and this code will eventually get common vulnerabilities that are simple to exploit because it is documented in databases that are open to the public and the only way to avoid these attacks is to keep the software updated, another reason to regularly update the software is that there could be bugs that not need to be security risks but necessary to run the software properly. To upgrade the software, it is possible to open a terminal window and use the advanced packaging tool (apt tool) by commands *sudo apt update* and *sudo apt full-upgrade*. (Raspberry Pi OS, u.d.).

3.3 Rhinoceros

Rhinoceros mostly known as Rhino is a professional 3D drawing software program / CAD application software that was founded by Robert McNeel & Associates year 1980 (About McNeel, u.d.).

Rhinoceros is designed from NURBS which is a mathematical representation of geometry. From basic shapes as 2D line, circle, arc, or curve to the most complicated 3D organic free-form surface or solid can be properly described by NURBS. NURBS geometry in Rhinoceros is utilized in the fields of architecture, industrial design, product design, multimedia, and graphic design. It is also used for rapid prototyping, 3D printing, and reverse engineering (What are NURBS?, u.d.).

The operating systems that could be used for Rhinoceros are Microsoft Windows and macOS. For Windows with the latest version of Rhino 7, the hardware recommendation is a 64-bit Intel or AMD processor, RAM 8 GB or more, 600 MB disk space, OpenGL 4.1 video card, 4GB Video RAM, or more. For the Windows Operating Systems alternative versions 11, 10, 8.1, or Windows VDI. Not supported for Rhino 7 is Windows 8 or older version, Windows Server direct login, Virtualization systems like VMWare, and Remote Desktop also should note Linux is not supported operating system. ARM processors include the Microsoft SQ® 1 and 2, Chromebooks (System Requirements, u.d.).

Rhino 7 with the use of Mac, the hardware recommendations are Intel Mac or Apple Silicon Mac, RAM of 8 GB or more, AMD graphics processor is recommended on Intel Macs, and 8 GB free disk space. MacOS operating systems supported are macOS 13 (Ventura), macOS 12.4 (Monterey), macOS 11.6 (Big Sur), macOS 10.15.7 (Catalina), and macOS 10.14.6 (Mojave). Operating systems not supported are macOS 10.13.6 (High Sierra) or older, Digitizers (Faro, Microscribe), and iPad / iPad Pro (System Requirements, u.d.).



Figure 11. Logo of Rhinoceros software (Rhinoceros, u.d.).

Rhino 7

Rhinoceros version Rhino 7 was released in November year 2020 (The History of Rhino, 2020).

An overview of the work field of the Rhino 7 program can be seen in Figure 12 with a lot of options. On the top, "File" could be found where the basic things like creating new files, open, save, printing, and more commonly things. From "Dimension" it is possible to measure lines and objects in different directions. Below these options, there is a shortcut command where it is possible to directly mention a command to use, here also the parameters should be adjusted for the command that has been set.

There are four different workspaces available to make it easier to process the object that working with (*top, perspective, front, and right*).

At the bottom of the program, the user can decide where the mouse should focus on a line or object, for example, *End, Near, Point, Mid, Cen, and Int*. Below these options, the scale of the drawing is shown.

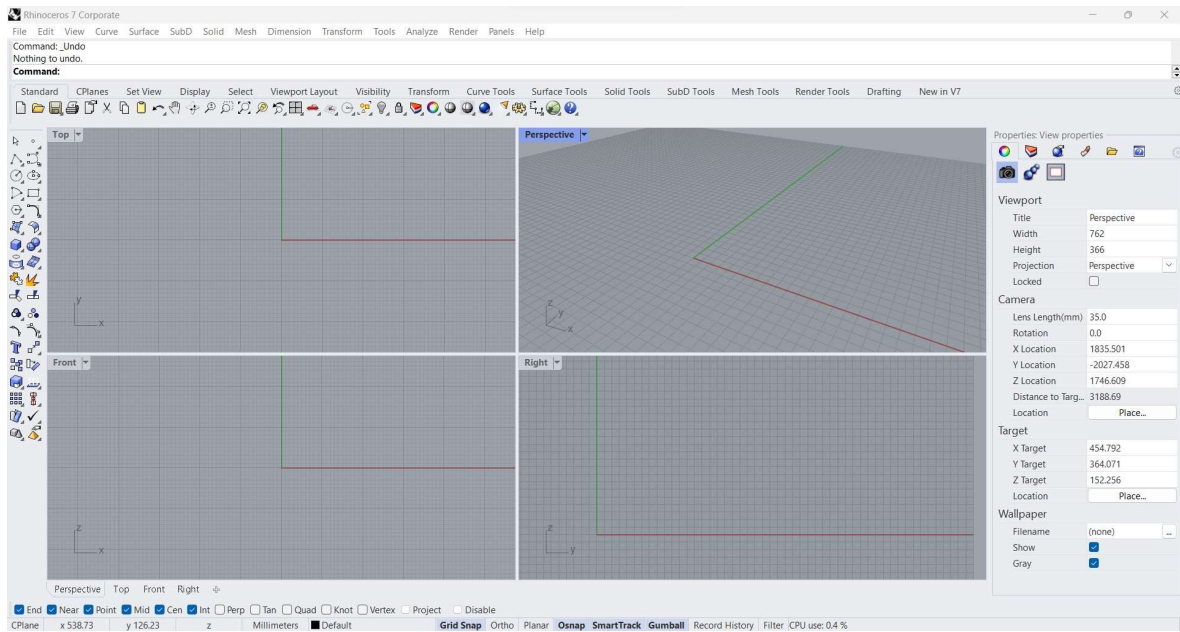


Figure 12. Overview of the program Rhino 7.

Below the command area are many different worktabs, when one tab is selected the tools become available and pre-defined options to the left side will appear.

At tab **Standard**, it is possible to use the pre-defined tools on the left side like text objects, and draw different lines, circles, rectangles, etc.

The display tab has many kinds of options for how to display an object.

Solid Tools is important when constructing 3D models, it is used when 3D blocks/figures are processed. Some options commonly used in this project are described below and the symbols are seen in Figure 13.

BooleanUnion is used when having two or more objects to combine. Click on the tool symbol and then select the objects and then press enter (*BooleanUnion*, u.d.).

BooleanDifference is used to subtract the volume of the object. Click on the tool symbol and then select the first object that will be modified and then click on the part that will be subtracted and then press enter (*BooleanDifference*, u.d.).

FilletEdge is used when the edges of the object need to be softer. Click on the tool symbol and then select the edges that should be modified and decide the radius (*FilletEdge*, u.d.).

MergeAllCoplanarFaces is used when the object has been modified and wants to combine all the planar region's faces into a single face on a mesh, polysurface, or SubD (*MergeAllCoplanarFaces*, u.d.).

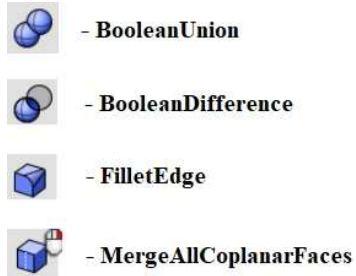


Figure 13. Important 3D model options in Solid Tools.

In the pre-defined column on the left side when selecting the Solid Tools tab, there can be found many different blocks for 3D drawing for example a box or cylinder. It is easy to use by entering the decided values for example millimeters or centimeters depending on what the scale is on.



Figure 14. Blocks for 3D drawing.

File formats

The file format for Rhinoceros is (.3dm) which is essential for the exchange of NURBS geometry. Rhinoceros supports several CAD file types for importing and exporting, making it compatible with other programs. In Rhino when opening a CAD file format that is not .3dm format the program will convert the geometry to its natural format (Rhinoceros 3D, 2023).

Table 10. Common supported file formats for Rhino 7.

Program	File extension
Rhino 3D Model	.3dm
ACIS	.sat (Save/Export)
Adobe Illustrator	.ai
Adobe Cult3D	.cd (Save/Export)
AutoCAD Drawing	.dwg
DirectX	.x
Google Earth	.kmz (Save/Export)
PDF	.pdf
STEP	.stp, .step
STL (Stereolithography)	.stl

(Supported File Formats, u.d.).

3.4 UltiMaker Cura

UltiMaker Cura is a 3D printing software tool, which is free to use and constructed as simple for fine-tuning the 3D model with 400+ parameters for best printing and slicing results. The software works on PC machines with operating systems Mac OS 64-bit, Windows 64-bit, Linux 64-bit, and Linux-Modern 64-bit (UltiMaker Cura, u.d.).

This software is a lightweight program that works on low-end systems. The size of the 3D models that are loaded has an important effect on the amount of RAM, CPU, and GPU that are required. The system minimum requirements are OpenGL 2 compatible graphics card, OpenGL 4.1 for 3D layer view, Display resolution 1024 x 768, Intel Core 2 or AMD Athlon 64, 550 MB available hard disk space, and 4GB RAM (What are the system requirements for Ultimaker Cura?, u.d.).

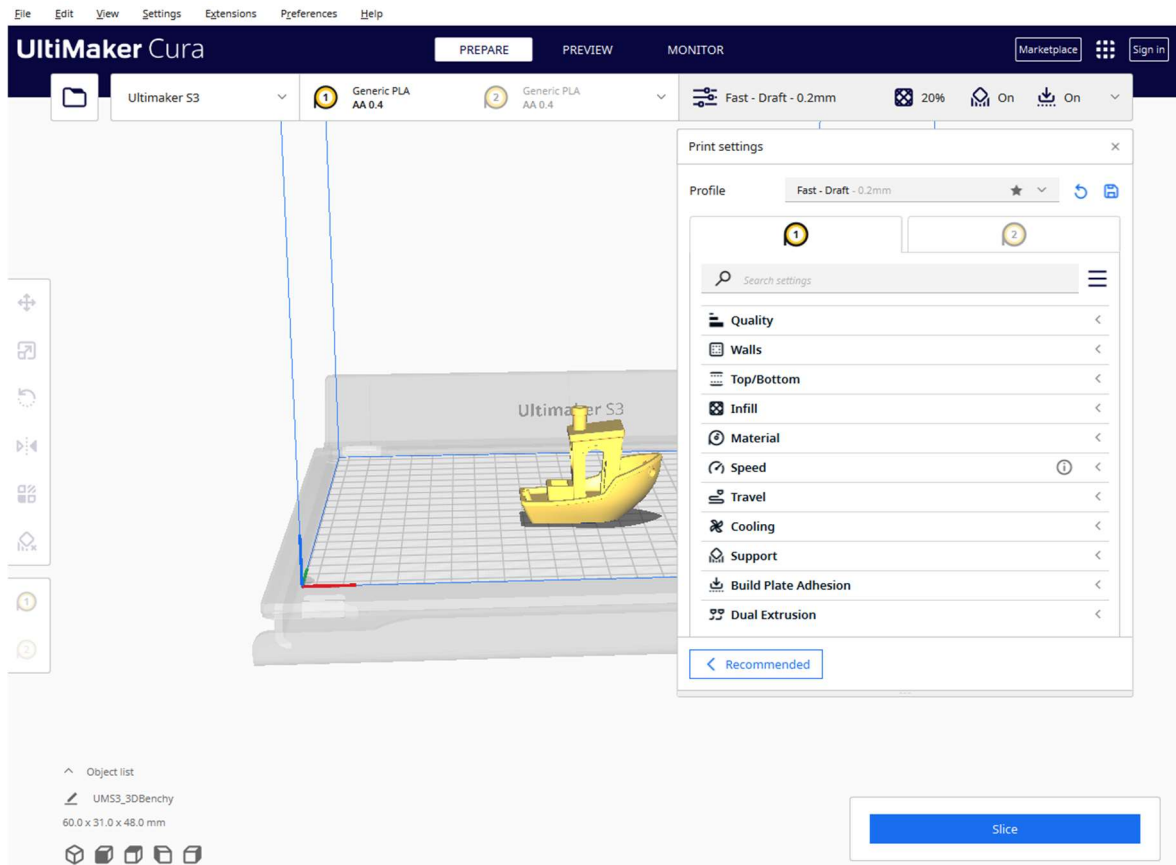


Figure 15. UltiMaker Cura software.

There are a lot of settings that can be adjusted in UltiMaker Cura software, in Figure 15 can be seen at the top from left to right: open new project, choose/add printer, 1st extruder, 2nd extruder, and print settings. When clicking on the extruders it is possible to enable both extruders or disable one, this of course depends on the 3D printer.

Behind the print settings, a 3D model of a boat can be seen. The model is placed on the middle point of the plate in this case the UltiMaker S3 printer bed. The plate size changes according to the printer that has been chosen.

From the settings, it is possible to use different configurations for both extruders. Settings could include **Quality** adjustments which could affect the resolution and the printing time. **Walls** and **Top/Bottom** adjustments for example the thickness. **Infill** means what will be filled in the walls for example the pattern of triangles, grids, or other constructions, and the density in percent. **Material** settings are the printing temperature and build plate temperature, **Speed** settings are possible to adjust the print speed and travel speed. **Travel** is how the nozzle will move when retracted. When enabling **Cooling**

it is possible to adjust the fan speed. **Support** is important for ensuring to prevent part deformation. **Build Plate Adhesion** can be used when needed to support the model at the build plate. **Dual Extrusion** can be used when both extrusions are enabled, this prints a tower next to the 3D model which serves to prime the material after each nozzle switch.

The toolbar can be found on the left side, when selecting the 3D model and using this toolbar it is possible to move, rotate, mirror, scale, and change mesh type. From there it is also possible to choose a support blocker which is good to use when it is not required to have support material in specific places on the model.

When satisfied with the settings, press the slice button which will then calculate the printing time and how much of the filaments in grams will be used. Then it is possible to preview the model to see how the program has set the support material to the 3D model. If everything is okay, then click save to removable... and choose the USB that will be inserted into the 3D printer.

4 Communication protocols

This chapter will describe about communication protocols used in this project. The OSI model (Open Systems Interconnection) consists of seven layers which describe a standardization of the functionalities in a communication system via abstract layers.

<i>Layer</i>	<i>Name</i>	<i>Example protocols</i>
7	Application Layer	MQTT, HTTP, FTP, DNS
6	Presentation Layer	SSL, SSH, TLS
5	Session Layer	NetBIOS, PPTP
4	Transport Layer	TCP, UDP
3	Network Layer	IP, ARP, ICMP
2	Data Link Layer	ATM, Ethernet
1	Physical Layer	Bluetooth, USB

Figure 16. OSI Model layers and examples of protocols.

Layer 1 (Physical Layer) is responsible for the physical cable or wireless connection between network nodes. This layer is liable for the transmission of the raw data bits which is a series of 0s and 1s. Bluetooth is included in layer 1 which will be described first (OSI Model, u.d.).

Layer 7 (Application Layer) is used by the end-users for example web browser clients. This provides protocols that accept software to send and receive information and show relevant data to the user. MQTT is included in layer 7 which will be described after the Bluetooth part (OSI Model, u.d.).

4.1 Bluetooth

This chapter will explain the history of Bluetooth and describe the common and differences between the Bluetooth Classic and Bluetooth Low Energy and later go more into the Bluetooth Low Energy part which the Arduino Nicla Sense ME and Raspberry Pi will use to communicate with each other.

Bluetooth Classic

Bluetooth technology has been available since about the year 2000 when its purpose was to let two devices exchange data wirelessly without any need for networking equipment and it found a great position in products for example: wireless keyboards, mice, and hands-free devices (Woolley, 2022).

There are two different versions of Bluetooth Classic. One version is the Bluetooth BR which was the start of Bluetooth technology and used in the first Bluetooth products that provided a raw data rate of one million bits per second (1 mb/s) (Woolley, 2022).

Bluetooth BR/EDR was later developed as a faster version of Bluetooth technology. With Bluetooth BR/EDR it was possible to get a raw data rate of two million bits per second (2 mb/s), but it was just to communicate and exchange data with only two devices directly between them (Woolley, 2022).

A summary of the Bluetooth version history is shown in Table 11.

Table 11. History of Bluetooth versions.

Bluetooth version	1.0	2.0	2.1	3.0	4.0	4.1	4.2	5.0	5.1
Year released	1999	2004	2007	2009	2010	2013	2014	2016	2019

(Bagur, 2023).

Bluetooth Low Energy

Bluetooth Low Energy was introduced in the year 2010 at version 4.0 of the Bluetooth Core Specification and the BLE specifications will also be available in the later versions. Version 4.0 was developed as another alternative to its predecessor Bluetooth BR/EDR. BLE has new capabilities and qualities that are great for the new generation of products that could fulfill complicated and new technical and functional requirements for example where power consumption is necessary to hold down and also to communicate with small data/information from example sensors and control systems (Woolley, 2022).

In addition to point-to-point communication between two devices, the BLE offers additional structures with a broadcast mode that enables one device to send data to an endless number of recipients at once (Woolley, 2022).

The main goal of this new Bluetooth technology BLE was as the name says to reduce the power consumption of wireless communication. The advantages of this are that the battery can be chosen with less ampere and smaller size within the same time frame as when using Bluetooth Classic with a battery of more ampere and bigger size depending on how long time the device should operate before the need of charging (Woolley, 2022).

The Bluetooth Low Energy is backward-compatible, which means if someone using the latest BLE version it is possible to interact with other BLE devices that use older BLE versions (Afeneh, 2020).

Bluetooth Core Specification is one important thing that defines the architecture, procedures, and protocols of the BLE. Profiles and Services are special types of collections and specifications where the products use Bluetooth to describe that they are compatible (Woolley, 2022).

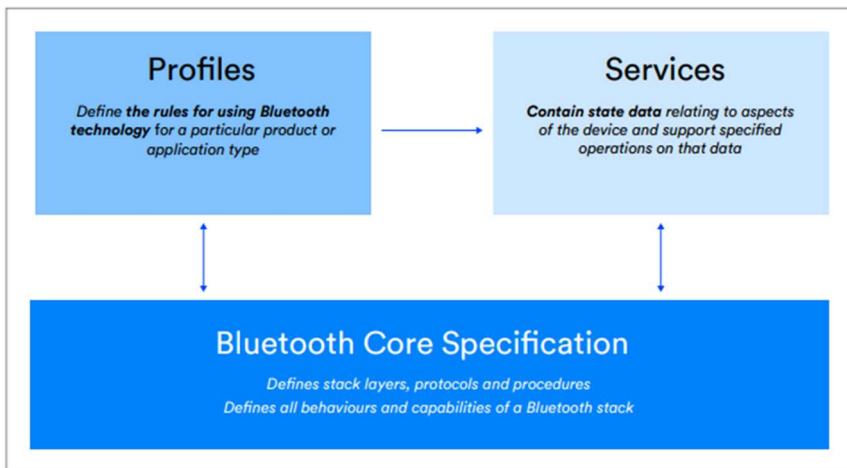


Figure 17. Bluetooth LE specifications (Woolley, 2022).

Bluetooth Core Specification:

The primary specification for both Bluetooth Classic and Bluetooth Low Energy is the Bluetooth Core Specification which defines the architecture of the technology and its layers. It defines and describes the key features of the important operations and protocols that the devices can communicate at a given layer of the stack (Woolley, 2022).

The Attribute Protocol (ATT)

The attribute protocol specifies the structure and methods of two devices, which a server makes its data available to a client. The server can be for example a sensor showing a set of compounded data items also known as attributes. The server arranges attributes in an indexed list known as the attribute table (Woolley, 2022).

Every attribute includes a handle, a Universally Unique Identifier most known as UUID, a value, and a set-up of permissions. An ATT Client can identify a specific object in the database using the handle with its unique index value. The type of the attribute is identified by the UUID. The permissions field is a collection of flags that specify the types of access that are allowed that could be read, written, or both forms of access (Woolley, 2022).

Generic Attribute Profile (GATT)

Generic Attribute Profile is meaningful once a connection between the devices has been established and explains the concepts of services and their characteristics, which are used to describe how two BLE devices exchange data back and forth (Woolley, 2022).

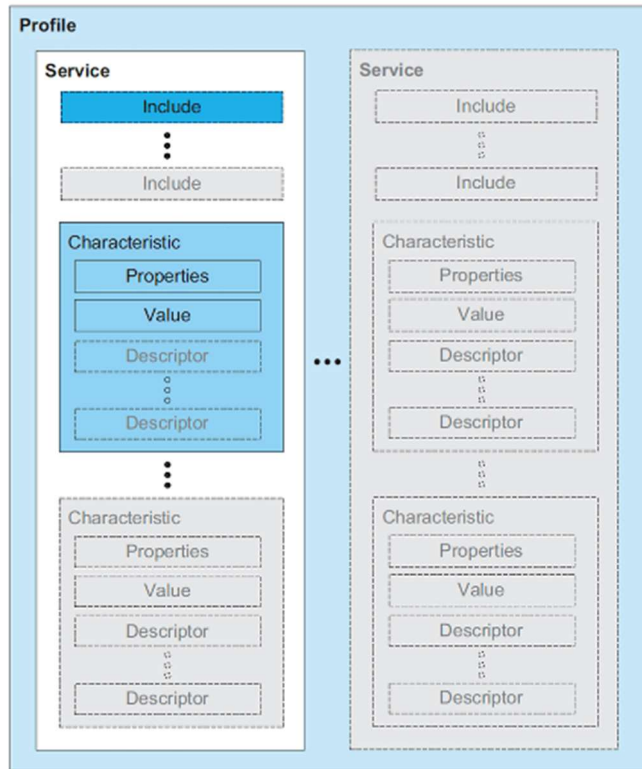


Figure 18. Example of Profile, Services, and Characteristics (Afaneh, Bluetooth GATT: How to Design Custom Services & Characteristics, u.d.).

Profile Specifications:

The client/server relationship has been established when two devices communicate over a Bluetooth Low Energy connection. With a server, it means a sensor with some kind of information data, and the client uses that data in some way. An example could be a key fob that communicates with a smartphone where the key fob acting as a server and the smartphone act as the client. When pressing the button of the application on the smartphone, the key fob's state should be changed and can send out a sound so it will be easier to find the keys (Woolley, 2022).

Profile specifications define the roles of related devices (such as the key fob and smartphone) and assume and define the behavior of the client and the data from the connected server as communicating (Woolley, 2022).

Service

A service is a collection of one or more characteristics, and it brings together relevant properties that perform a specific server functionality. For instance, if there is a battery it could have one battery service that contains one characteristic named Battery Level. The Device information service, Environment sensing service, and Motion sensing service are other examples of services. The data included in a service may also contain additional features that help to structure the data such as service declarations, and characteristics declarations (Afaneh, Bluetooth GATT: How to Design Custom Services & Characteristics, u.d.)

Characteristic

A characteristic is a part of any service and represents some information or data that the server wants to be visible to the client. For instance, the battery level characteristic shows the amount of battery power left in a BLE device which a client wants to know. The characteristic includes other attributes that help to define the value it holds: *Properties* can be defined by how the characteristic values can be used for example read, write, notify, write without response, and indicate. The other attribute *descriptors* are utilized to include important details about the characteristic value type for example, extended properties, user description, the field used for subscribing to notifications and indications, and a field that specifies how the values are presented (Afaneh, Bluetooth GATT: How to Design Custom Services & Characteristics, u.d.).

Universally Unique Identifier (UUID)

The official Bluetooth specification services can be 16-bit or 32-bit long which are predefined UUIDS and listed by the Bluetooth SIG while non-official Bluetooth services are 128-bit long that users can create UUIDs customly (Bagur, 2023). Shortened (pre-configured) UUIDs are available from Bluetooth SIG for different types of services and

characteristics. If the list of predefined UUIDs doesn't meet the demands, it is possible to generate custom UUIDs (Townsend, Cufi, Akiba, & Davidson, 2014).

Operating range and data throughput

Bluetooth Low Energy is primarily focused on short-range communication. The actual range of BLE wireless devices depends on a wide range of parameters, such as operating environment, antenna design, enclosure, device orientation, etc. The range of the frequency spectrum is between 2.400 – 2.4835 GHz (Townsend, Cufi, Akiba, & Davidson, 2014).

The data throughput for the BLE version 4.2 is limited to 1 Mbps while version 5.0 extends up to 2 Mbps when using the high-speed feature, but noted the rate could drop to 500 or 125 Kbps when the long-range capability is used (Afaneh, Intro to Bluetooth Low Energy, 2018).

A connection event is any instance of communication that takes place between two devices and these devices exchange packets during a connection interval. Multiple packets can be transmitted within a connection interval, this process could proceed until a package fails to get delivered, the maximum connection event duration is reached or neither has more data to send (Derhgawen, 2020).

A connection's throughput increases with the number of packets it can deliver. The maximum number of packets that can be sent in an interval is usually restricted to the maximum connection event duration (Derhgawen, 2020).

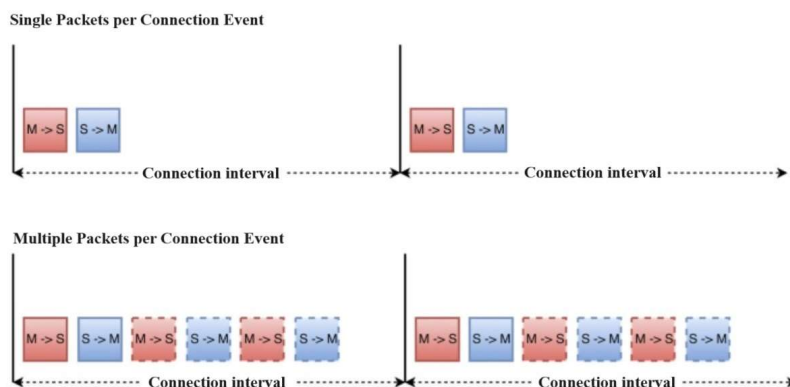


Figure 19. Connection event over BLE (Derhgawen, 2020).

4.2 Message Queuing Telemetry Transport

Message Queuing Telemetry Transport (MQTT) is constructed and used as a messaging protocol that uses publish and subscribe functions with clients between an MQTT server normally called a broker. The broker is the central role that hosts any topics which usually is formed as a folder structure for example “Arduino/Nicla/Temperature”, but any utf-8 string name with at least one character can be used as a topic (Gustafsson & Jarefors, 2022).

For a topic to exist it needs a subscriber or publisher to connect to the broker and the topics are just visible for a short time when sending the message and not stored forever. The client can both subscribe and/or publish topics on the broker server. When a client publishes a message on a topic, all subscribers (clients) who are connected to the specific topic can receive the information from that message (Gustafsson & Jarefors, 2022).

There are “wildcard” characters that can be used for a client to subscribe to any number of topics:

- Multi-level subscriptions are used with the symbol “#”, for example topic *Arduino/Nicla/#* means that it subscribes to *Arduino/Nicla/* and all the subtopics like *Arduino/Nicla/Temperature* and *Arduino/Nicla/Humidity* and so on.
- Single-level subscriptions are used with the symbol “+”, for example topic *Arduino+/Temperature* will subscribe to the temperature of the Arduino boards connected with this topic like there could be *Arduino/Nicla/Temperature* and *Arduino/Nano33/Temperature* (Gustafsson & Jarefors, 2022).

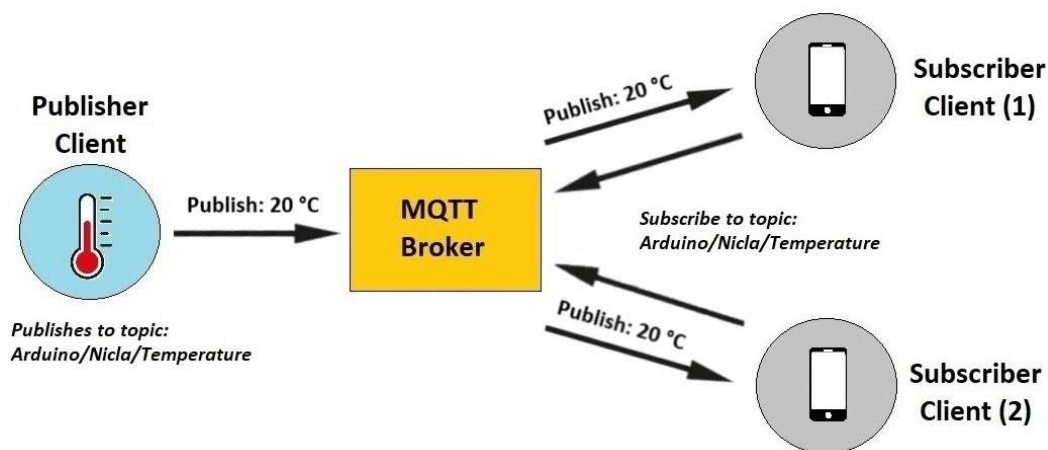


Figure 20. Illustration of publisher and subscriber arrangement.

Figure 20 presents an overview of what the MQTT system could look like; the publisher client (temperature sensor) sends out information of value 20 Celsius to the broker with the topic *Arduino/Nicla/Temperature*. The two subscribers (smartphones) subscribe to the broker with the same topic *Arduino/Nicla/Temperature* to get the data information from the publisher.

Quality of Service

QoS is an agreement between a communication's sender and receiver that defines the delivery promise for a particular message known as the Quality-of-Service level, also abbreviated QoS. There are three different options for QoS levels numbered (0, 1, 2) (Gustafsson & Jarefors, 2022).

- **QoS 0:** This is the lowest and most basic level. At this level, the sender just sends the message without considering if it was properly received and does not attempt to save it or resend it (Gustafsson & Jarefors, 2022).
- **QoS 1:** This level means that a message will be delivered to the sender at least once. Until the recipient sends a "PUBACK" packet confirming delivery of the message, the sender keeps onto the message. With this QoS level, it may be possible that a message could be sent or delivered several times (Gustafsson & Jarefors, 2022).
- **QoS 2:** This is the highest level of QoS to ensure that the message will only be delivered once. This is accomplished by sending and receiving messages back and forth (Gustafsson & Jarefors, 2022).

PUBREC packet from the recipient indicates that the message has been received, the sender can then safely discard the first message. Otherwise, the sender will keep the message after publishing it with QoS 2 and resend it regularly with a duplicate flag until it receives a PUBREC packet. After that, the sender sends a PUBREL packet to the receiver, which clarifies that the PUBREC packet has been received and then the receiver can discard its local stored states of the original publish and send a PUBCOMP packet back to the original sender. This pattern concludes both the sender and the receiver are certain that the message has only been received once (Gustafsson & Jarefors, 2022).

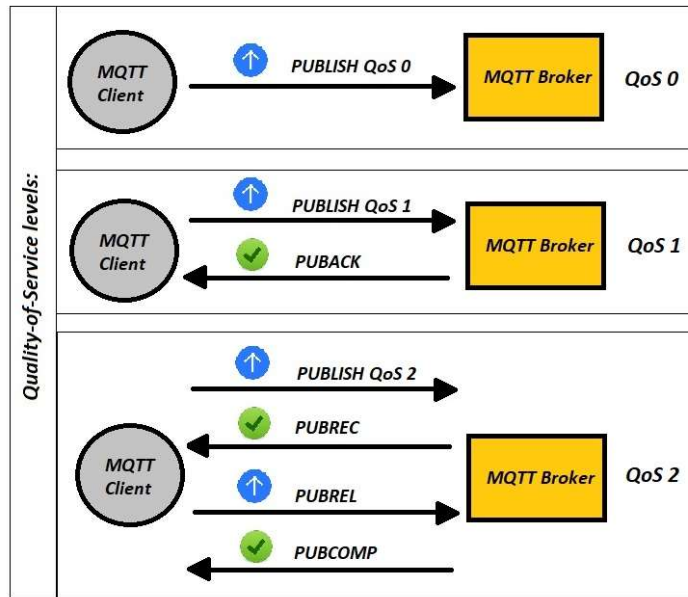


Figure 21. Quality of Service levels.

Last Will and Testament

Clients can keep track of other clients' connection status by using Last Will and Testament (LWT). In case a client disconnects unexpectedly, the LWT system allows clients to automatically provide a message to the broker they are connected to. The LWT message has a similar format as other messages, a topic to be sent to, a QoS level, and so on. (Gustafsson & Jarefors, 2022).

5 Programming & markup languages/tool

In this chapter programming/markup languages and tools used in this project will be explained. The C / C++ programming language has been used for programming the Arduino microcontroller. The JavaScript, HTML, and CSS that have been used to program the webpage will be described, and lastly, the programming tool Node-Red will be explained.

5.1 C / C++

The computer executes with the use of binary numbers (1 and 0) the instructions that define electronic signals that take place inside the computer. Between the years 1940 – 1950, the program was written by using binary digits, the programmers had to learn how the machine processed various combinations of 1 and 0 digits. Making programmers work

with the concepts of the computer's 1 and 0 proved increasingly impractical as the applications programs grew larger. Instead, scientists developed programming languages that enable people to translate computer instructions into human-friendly language (Jamsa, Ph., & Klander, 1998).

Dennis M. Ritchie, who was working at Bell Labs (AT&T) at the time developed in the early 1970s C program which is the predecessor to C++ (The C++ programming language in cheminformatics and computational chemistry, 2020).

Programming languages such as C and C++ are constructed to handle collections of instructions that the computer follows to complete a task. When writing a program, the instructions are saved in an ASCII file that typically ends with an extension .C for C program and .CPP for C++ programs (Jamsa, Ph., & Klander, 1998).

Programmers place their instructions in a file referred to the source file, which is then compiled in a second program into a programming language with 1 and 0 binary digits known as machine code that the computer can understand (Jamsa, Ph., & Klander, 1998). A sketch of the described can be seen in Figure 22.

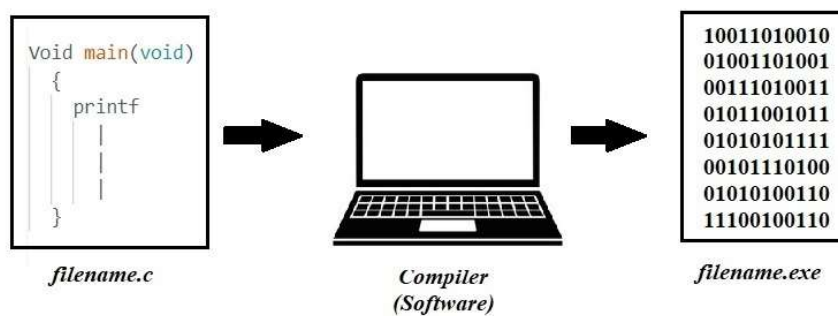


Figure 22. The instructions of the source code are compiled into machine code.

There are differences between C and C++, to get a better overview it is listed some of these differences are in Table 12.

Table 12. Differences between C and C++.

C	C++
Procedural programming language follows methods that put procedure before data.	Object-oriented programming language, data is given priority.
Supports only pre-defined data types.	In addition to the pre-defined data types, it supports user-defined data types.
Does not support classes and objects.	Classes and objects are supported.
Does not support Inheritance, polymorphism, data encapsulation, or abstraction.	Supports object-oriented features such as inheritance, data encapsulation, polymorphism, and abstraction.
Supports only pointers, no concept of reference variables.	Has both pointers and reference variables.
32 keywords.	63 keywords.
Does not support function and operator overloading.	Supports function and operator overloading.

(Ravikiran , 2022).

Advantages of using C++

Compared to interpreted languages like Python, C++ is a high-performance language that can be compiled into native code. Due to its minimal memory footprint and ability to enable low-level programming, it can be a great option for applications like system programming.

C++ includes features like classes, objects, inheritance, polymorphism, encapsulation, and abstraction. This allows the creation of modular code structures that are simpler to update as well as the writing of complex applications and code reuse.

It is possible to compile C++ code with Windows, Mac OS, and Linux among other platforms. There is a large development community and a large of information available for example from online forums, libraries, and platforms.

C++ and C are backward compatible, C++ programs can use existing C code and vice versa which can be helpful if needed to reuse legacy code (SSDN Technologies, 2023).

5.2 JavaScript

In the year 1996, JavaScript was released by Netscape. LiveScript was the original name, but changed in part because one of its initial goals was to provide users with some control over Java Applets within the browser (Connolly & Hoar, 2018).

JavaScript is one of the most widely used programming languages nowadays which allows the widespread possibility of different functionalities. JavaScript is often utilized to adjust HTML and CSS to update a user interface. The fundamental programming constructs in the client-side JavaScript language perform things like storing useful values inside variables and executing code in reaction to specific events on a website (What is JavaScript?, 2023).

In web development, client-side scripting becomes an important concept. Instead of relying on the server to run code and return the output, it refers to the client computer executing code locally (Connolly & Hoar, 2018).

Variables in JavaScript can easily be changed from one data type to another since it uses dynamic typing. In JavaScript, a variable's data type is set at runtime and can change at any time (Connolly & Hoar, 2018).

Node.js

A cross-platform, open-source JavaScript runtime system and library named Node.js is used to run web applications outside the client's browser. It was developed in the year 2009 by Ryan Dahl. Since Node.js is an asynchronous, event-driven framework, developers can use it to create server-side web apps that are ideal for processing large amounts of data (Sufiyan, 2023).

Node.js is an excellent solution for many common web development challenges and was created to optimize throughput and scalability in web applications for example real-time web apps (Express/Node introduction, 2023).

Node package manager (NPM) is a manager for Node.js packages/modules which hosts thousands of free packages that can be downloaded and used. When installing Node.js on the computer, the NPM program will be included. All the files required for a module in Node.js are contained in a package and the modules are JavaScript libraries that are possible to include in a project (Node.js NPM, u.d.).

Node.js Express is a web application framework for Node.js that offers a wide range of functionalities for developing web applications. It is a layer that assists in controlling servers and routes and is built on top of Node.js (Sufiyan, 2023).

5.3 HyperText Markup Language

The most significant element of the webpage is HyperText Markup Language also known as HTML. It describes the purpose and organization of web content and the markup language code is used to structure a web page and its content. The appearance (CSS) and functionality (JS) of a webpage are typically described using technologies other than HTML (HTML: HyperText Markup Language, 2023).

The explanation of HyperText Markup Language could be described as follows:

Hyper: means that a text is non-linear. Hyper refers to the fact that the text is linked to other texts, which allows the connection between different texts (Faraon & Holmberg, 2022).

Text: Characters, symbols, and numbers that are combined to form words and sentences (Faraon & Holmberg, 2022).

Markup: Content can be structured with HTML elements, for instance, specific words become clickable as links (Faraon & Holmberg, 2022).

Language: Relates that HTML is a computer language. Syntax and semantics differ from each other in both natural languages and computer languages, syntax refers to writing rules while semantics refers to meaning. Computer languages can be identified by the fact that usually have a very strict syntax which means if forgot for example a slash the program will not run. The semantics that is, the meaning or the message to be conveyed or the program is to do is defined by the person who creates a program or a web page (Faraon & Holmberg, 2022).

5.4 Cascading Style Sheets

Cascading Style Sheets (CSS) explain how HTML elements should be displayed. A lot of work is saved by using CSS in an external .css file, it can manage the design of several web pages simultaneously and possibly change the look of the whole website. The web page's design, layout, and differences in display for different devices and screen sizes are some reasons why using CSS (CSS Introduction, u.d.).

Some benefits of using CSS are less work and easier updating, faster downloading due to less code needed, and platform independence which means CSS works in every web browser (Faraon & Holmberg, 2022).

Many different configuration options could be used to design an HTML webpage with the use of CSS, some options could be to define font sizes or the color of text, change the background color, or general structure the content from the CSS file. By using tags on the HTML file for example id or class with a specific name could have a specific configuration to its tag (CSS Introduction, u.d.).

5.5 Node-Red

Node-Red is a flow-based programming software that was released in year 2013 by Nick O'Leary and Dave Conway-Jones. Node-Red was originally developed by IBM's Emerging Technology Services team but is now a part of the OpenJS Foundation.

Flow-based programming is a method of representing an application's behavior as a network of nodes. Each node serves a specific job, it receives data, processes it, and then transmits the results to other nodes. Data transmission between nodes is handled by the network.

This sort of model allows better visualization of the workflow and gives more accessibility to a broader group of users. One advantage is that it is not needed to interpret each line of code, someone who can break an issue down into discrete steps can study a flow and get a sense of what it is doing (About, u.d.).

5.5.1 Node-Red editor

To access the flow editor, connect a web browser to the Node.js-based runtime that represents Node-Red. Design application in the browser by dragging nodes from the palette into a workspace and then starting to connect them.

Installing new nodes made by the community makes it simple to expand the palette of nodes, and it is possible to share the flows created with others.

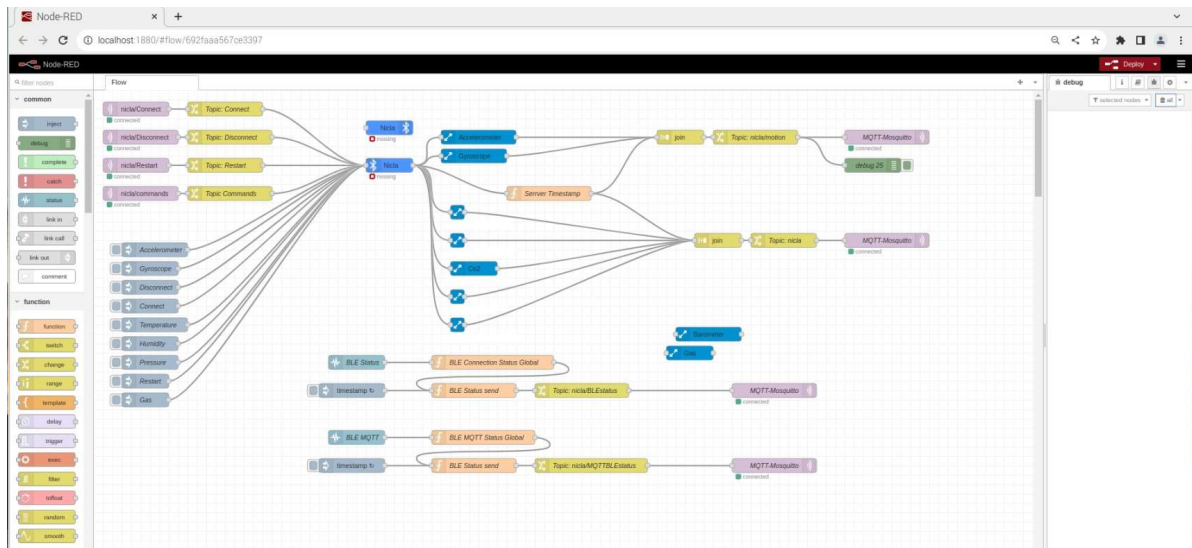


Figure 23. Node-Red editor.

The Node-Red workflow can be seen in Figure 23. On the left the palette with different nodes are listed, it is possible to access more kind of nodes than the primary by downloading from the menu “Manage palette”. In the middle of the website the flow is created, to use different nodes on the workflow it is just to drag the decided node from the palette to the middle. It is possible to have many flows and it is also possible to hide workflows. The deploy button can be seen to the right on the website where to execute the application and next to it a list with options such as import, export, and many more settings. Below the deploy button and the list, the debug option can be found to inspect nodes if a debug node is inserted into the workflow, there are also help, information, and configuration options.

5.5.2 Nodes

The main component of a flow in Node-Red is called a node. Nodes are activated by an external event such as an HTTP request, or a timer, or by receiving a message from a preceding node in a flow. The node responds to that message or event and then might communicate with the following nodes in the flow. A node can only have one input port and several output ports (Node-RED Concepts, u.d.).

To make nodes accessible to the community, they can be uploaded to the Node-Red Flow Library and published as npm modules to the public npm repository (Creating Nodes, u.d.).

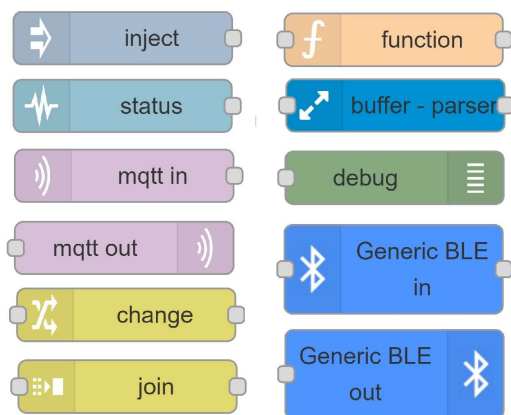


Figure 24. Node-Red nodes used in the project.

In Figure 24 different nodes can be seen that were used in Node-Red for this project. A description of these different nodes can be found below.

- Inject:** By selecting the Inject node's button inside the editor, a flow can be manually triggered. Additionally, it can be used to automatically start flows at a predetermined time interval. The payload and topic properties of the message sent by the Inject node can be configured by double-clicking on the node. Several types of payloads can be selected in an Inject node for example a value for a flow or global context property, a string, number, boolean, buffer or object, and timestamp in milliseconds. The maximum interval that can be chosen is 596 hours (The Core Nodes, u.d.).
- Status:** The status node is used by selecting other nodes that want to read the status. When the desired node updates its status it sends that to the status node.

- **Join:** Join nodes are used when need to merge multiple messages that are coming from different sources into one message by giving each payload a unique msg.topic value (Create a single message from separate streams of messages, u.d.).
- **Debug:** Messages can be shown in the editor's Debug sidebar by wiring a debug node to other nodes where the process flows. With the debug node it is much easier to follow up on what is going on from the flow. From the debug sidebar it is possible to reach details about each message, such as the time it was sent and what values are included in the message. The output of the node could be enabled or disabled using the button on the node (The Core Nodes, u.d.).
- **Change:** A change node can be used to change a message's properties and set context properties. Many operations can be configured for each node and can be set in order. "Set" a property, the value may come from an existing message or context attribute or be of a variety of various types. "Change", look for and swap out specific message property elements. "Move", "rename" or "delete" a property (The Core Nodes, u.d.).
- **Function:** The messages that are passed through the function node let their own written JavaScript code be operated (The Core Nodes, u.d.).
- **Buffer – parser:** Buffer – parser node is used to convert values from buffer. Some functionalities are examples of setting up a specification and converting multiple parts of a buffer to example int or float (node-red-contrib-buffer-parser, u.d.).
- **Generic BLE in/out:** The Generic BLE node gives users access to BLE peripheral GATT features, BLE in node is used to give commands by using UUID and receive some information from the server example a sensor, and with BLE out it is possible to send configuration commands to change some settings on the sensor. Supported operations are start, stop, and restart BLE scanning. Connect to and disconnect from a peripheral device. It can also read, write, write without response, and notify between a peripheral. This node can also give some different

statuses such as connecting, connected, disconnecting, disconnected, missing, and error (node-red-contrib-generic-ble, u.d.).

- **MQTT in/out:** The MQTT out node is used when publishing messages to other clients on a specific MQTT topic. To be able to publish a message to a topic it is needed to enter the MQTT broker server address with its port and password if there are any. Below the server, the topic to which the message be sent should be filled in and optional QoS could be configured (Publish messages to a topic, u.d.).

The MQTT in node is used when subscribing to the broker for messages on an MQTT topic. (Subscribe to a topic, u.d.).

6 Implementation

This chapter goes through the different steps to have a completed data acquisition tool for data collection. It describes how to install and set up the Nicla Sense ME in Arduino IDE software and after that go through Raspberry Pi installation and set up the Raspberry Pi OS, get acquainted with configuring a local MQTT broker, and install and set up Node-Red on the Raspberry Pi. There will also be a description of a graphical user interface that receives data from and send commands to the Nicla Sense. Lastly how to construct a 3D-printed box for the Nicla microcontroller

6.1 Implementation of Nicla Sense ME

To be able to create and run code to the Nicla Sense ME microcontroller, the first step is to download the software Arduino IDE from <https://www.arduino.cc/en/software> and choose the operating system on the PC.

When the software is up and running, the required driver for the Nicla Sense microcontroller needs to be downloaded. In the menu bar navigate to Tools → Board → Boards Manager. Search for Nicla Sense and the drivers for the microcontroller will be visible *Arduino Mbed OS Nicla Boards by Arduino*, install, and wait until the installation is completed.

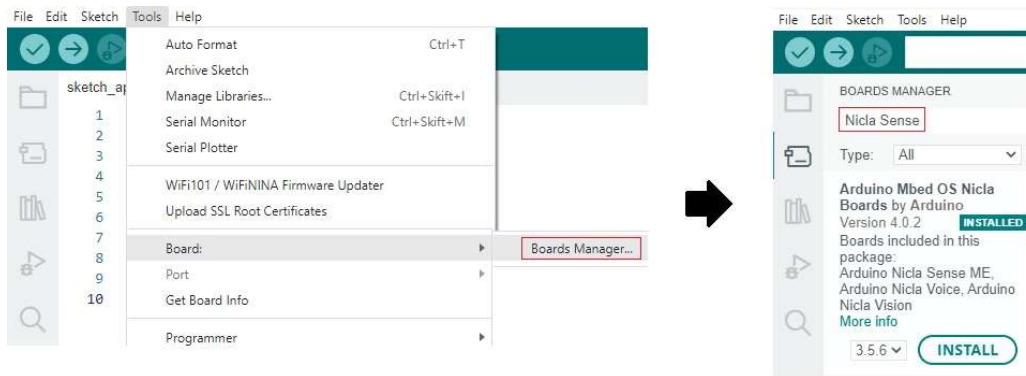


Figure 25. Install driver at Board Manager for Nicla Sense.

When the Nicla Sense driver is installed, the board can be selected at Tools → Board → Arduino Mbed OS Nicla Boards → Arduino Nicla Sense ME (version 4.0.2 was used in this project).

Before transferring the code to the board, different libraries are needed to be installed: *Arduino_BHY2* and *ArduinoBLE*. To find these libraries navigate to Sketch → Include Library → Manage Libraries. In Figure 26 it can be seen how to navigate to the mentioned libraries and after installation is finished it should be shown “INSTALLED” at the specific library.

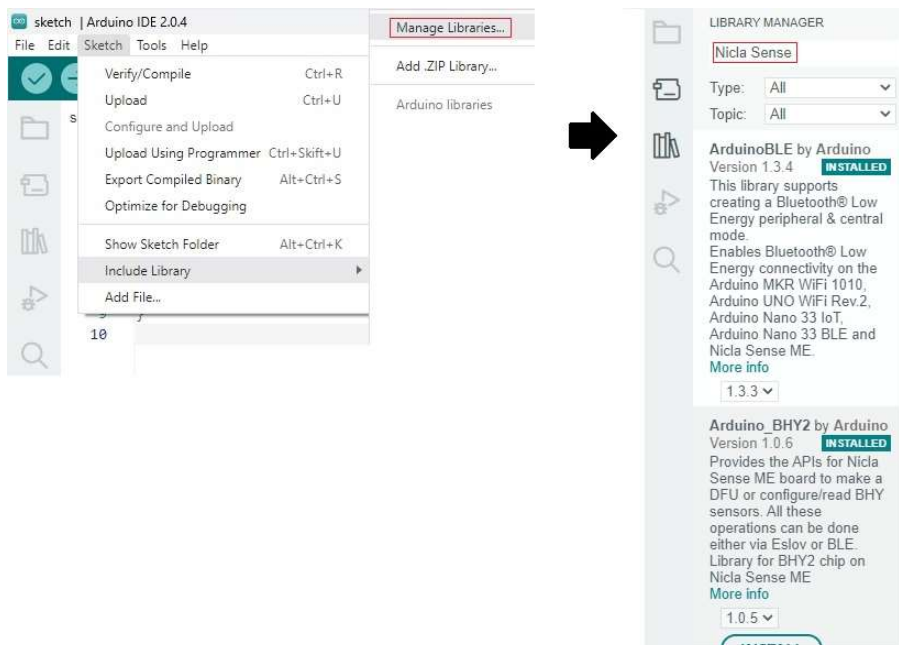


Figure 26. Install libraries at Manage Libraries for Nicla Sense.

When the Nicla Sense driver and libraries are installed, connect the board with the USB cable to the computer and navigate to Tools → Port and select the COM port for the Nicla Sense.

On the first run of the Nicla Sense Board, it may have outdated firmware. Bosh Sensortech has updated a compensated temperature and humidity for this board in April 2022, it is possible to download the firmware from the link: https://github.com/bstbud/nicla-sense-me-fw/tree/fea_temp_compensation/Arduino BHY2/examples/BHYFirmwareUpdate.

The files need to be stored at:

...*Document\Arduino\libraries\Arduino BHY2\examples\BHYFirmwareUpdate*. For safety reasons, it is good to create a backup folder with the existing files before overriding it.

Open the replaced *BHYFirmwareUpdate.ino* file and the Arduino IDE opens with some firmware code. Nicla Sense Board needs to be connected to the computer and the COM port selected, click on the upload button to run the code. During the compiling and uploading the serial monitor could be opened to see when the code has been completed, if the upload of the code was successful something like the below could be seen in the serial monitor:

```
10:31:56.943 -> The computed CRC is 0x7A  
10:31:56.943 -> Writing BHY FW binary into SPIFlash...  
10:32:08.701 -> BHY FW Upload Done!
```

The firmware is finished and now move on to the program code. The Arduino code for this project can be found in Appendix I with some explanations above or beside the code lines.

The project imports library code by using *#include* (Nicla_System.h, Arduino BHY2.h, ArduinoBLE.h) these are the header files which is necessary because of the use of commands that are executed for the project.

The installed library *Arduino BHY2.h* is necessary to be able to read data from sensors on the Nicla Sense. To use the sensors needed it is necessary to know the Sensor IDs and their class which can be found at <https://docs.arduino.cc/tutorials/nicla-sense-me/cheat-sheet>. For example, the class “*Sensor*” belongs to temperature, humidity, gas, and pressure. Class “*SensorXYZ*” belongs to the accelerometer and gyroscope. Class “*SensorBSEC*” belongs to the Bosch Sensortec Environmental Cluster which includes Co2, compensated

temperature, and compensated humidity. The sensor ID always starts with “*SENSOR_ID_*” for example, gas: *SENSOR_ID_GAS*.

The library *Nicla_System.h* is required because of the internal components such as the RGB LED and battery settings. The command *nicla::* is used to call the Nicla Sense for example, to start the Nicla Sense when powered on “*nicla::begin();*” and to activate the RGB led: “*nicla::leds.begin();*” and then set the color to red: “*nicla::leds.setColor(red);*”. The Nicla Sense charge current is limited to 100 mA by using the command “*nicla::enableCharge(100);*”.

The library *ArduinoBLE.h* which is BLE 4.2 version compatible, enables Bluetooth Low Energy connectivity on the Arduino Nicla Sense. A lot of functionalities for this library can be found at <https://www.arduino.cc/reference/en/libraries/arduinoBLE/>.

The code is structured by BLE GATT transactions with the profile, services, and characteristics, in Figure 27 an overview of this can be seen. The services are divided into three different categories, the first service is pre-defined as “Environment Sensing” and consists of five different characteristics: temperature, humidity, pressure, gas, and Co2. All these characteristics have been assigned with the properties of read.

The next service is called “Motion Sensing” with a custom-used UUID and this service has two characteristics including accelerometer and gyroscope which have been assigned with the properties of read and notify. The last service is pre-assigned as “Battery” with just one characteristic called battery level.

The short 16-bit identifiers are pre-assigned UUIDs officially adopted by Bluetooth SIG which is a part of a 128-bit UUID. For example, the temperature has a pre-assigned UUID *2a6e* where the client can easily find the name of that characteristic. To find these pre-assigned services and characteristics, a document called *Assigned Numbers* can be found at <https://bluetooth.com/specifications/assigned-numbers/>.

The custom-designed 128-bit UUIDs are customized and created by the user. These UUIDs are produced by using a “UUID generator”, several tools/utilities are available on the internet. A custom-designed UUID is used to ensure that no one else is using that specific ID.

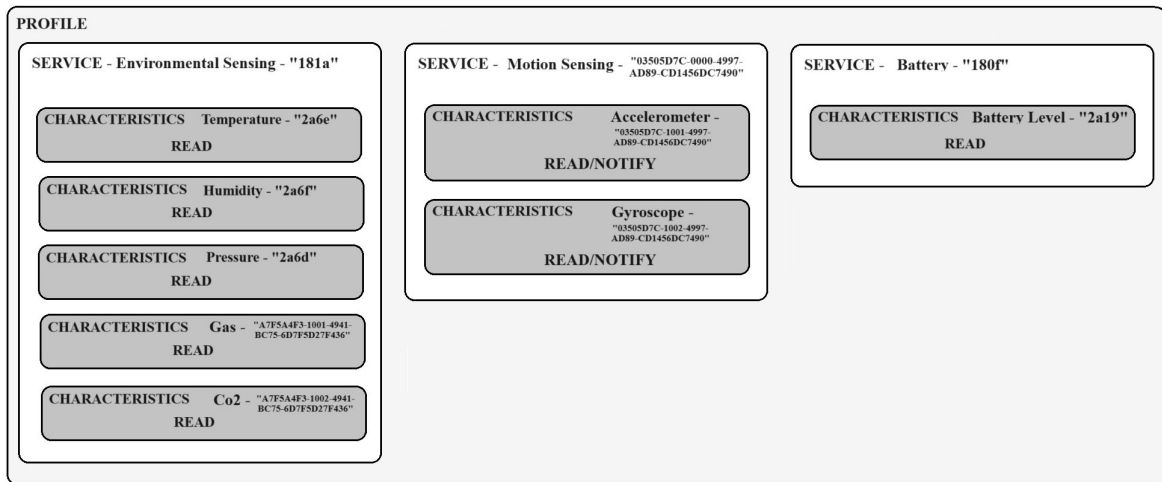


Figure 27. GATT transactions (Profile / Services / Characteristics).

When the Nicla Sense is powered on, the LED indicator color is red and from the ArduinoBLE library *setEventHandler* was used for both connected and disconnected states. When the device is connected to the client, the LED changes to green, and when disconnected the LED changes to red.

The name of the device that should be discovered when scanning has been set to *Nicla*.

To get readings of the battery status "*nicla::getBatteryStatus();*" was used which should update its status according to the numbers below:

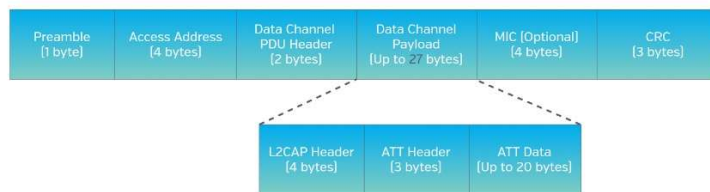
BATTERY_FULL	5
BATTERY_ALMOST_FULL	4
BATTERY_HALF	3
BATTERY_ALMOST_EMPTY	2
BATTERY_EMPTY	1

New battery configuration/settings have been developed by Arduino with the newest Mbed OS Nicla Sense upgrade during this work, for some reason the status will not more update with the correct battery status for the older version either. The battery service will still be left in the code for further research.

To write and send sensor data through BLE, the specific characteristics of UUID need to be requested from the client. For example, if one data packet of temperature is needed, a request of characteristic UUID *2a6e* should be received which means one request sends one data package.

The BLE version 4.0 and 4.1 has a data channel payload limited to 27 bytes. By including Data Length Extension (DLE) from Bluetooth version 4.2 and further, the Data Channel Protocol Data Unit (PDU) payload field can be expanded from the default 27 bytes up to 251 bytes which can be seen in Figure 28. The Data channel payload includes the L2CAP header which reserves 4 bytes and the ATT header reserves 3 bytes, that means by default 20 bytes will be left for the payload, and with the use of DLE a maximum of 244 bytes (Michel, 2019).

Bluetooth 4.0/4.1 Link Layer Packet Structure



Bluetooth 4.2/5.0 Link Layer Packet Structure

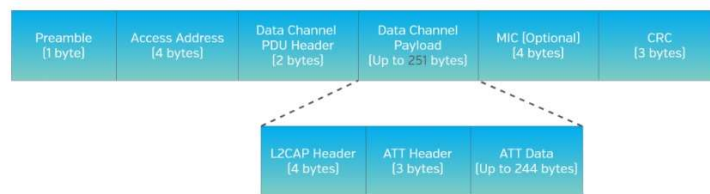


Figure 28. Link Layer Packet Structure (Michel, 2019).

For this project, the default settings are used and one request to get one measurement will be enough for the environment sensors. On the other hand, the motion sensors become limited to transfer with higher sampling frequencies since needed to store more data. For now, the accelerometer and gyroscope start when the device is connected to the BLE but send just one X, Y, and Z measurement per request as the environment sensors.

The compensated temperature and humidity are adjusted values that take into account the impact when the microcontroller is on and heated up a bit. The values have been updated through *SENSORBSEC*, and the sensor data have been followed up on the serial monitor for both the raw and compensated data of temperature and humidity. The compensated temperature value was accurate and did not need to be adjusted, the raw data was 5.2 Celsius above the compensated value which means the compensated temp

value was used. For the humidity sensor, the compensated humidity data was accurate and did not need to be adjusted either. The humidity raw data was approximately 10 percent lower (error marginal) than the compensated value.

6.2 Implementation of Raspberry Pi

First, a short explanation of installing the operating system for the Raspberry Pi, and after that show how to display the Raspberry Pi to a PC by using VNC Viewer. Then investigate how to install and start up an MQTT Mosquitto broker on the Raspberry Pi, but it should be noted that Novia's own MQTT broker will be used in this project. Then after that explanation of how to create Bluetooth Low Energy to MQTT bridge and how to install and set the project flow in Node-Red.

6.2.1 Installation of Raspberry Pi OS

To start with the installation of Raspberry Pi OS (previously called Raspbian OS) a Micro SD card is needed, the recommendation is to use 8 GB or higher capacity. It is also recommended to have a higher number of classes on the Micro SD card for faster loading. In this project, it is used Kingston 32 GB of Class 10.

To install the Raspberry Pi OS, it is recommended by the manufacturer to download the Raspberry Pi Imager, in this case, the Imager was downloaded from the website <https://www.raspberrypi.com/software/> and the OS Windows 11 was used. A file named imager_1.7.5.exe (version could change later) was downloaded and installed, in Figure 29 the Imager program is shown.



Figure 29. Raspberry Pi Imager

The Operation System should be selected from *Choose OS* and the alternative *Raspberry Pi OS (32-bit)* is used. The Micro SD card needs to be inserted into the PC where the Raspberry Pi Imager program is installed, note that the Raspberry Pi uses a Micro SD card so then an adapter is needed.

Before writing the OS to the Micro SD Card, select advanced options by pushing **Ctrl-Shift-X**, this is useful for configuring Raspberry Pi. From here it is possible to set the hostname, enable SSH, set username and password, connect to a Wireless LAN, and set locale settings. There are boxes to check to play sound, eject media, and enable telemetry after completion. The hostname was set, SSH enabled, username and password were set, the wireless LAN configured, and locale settings were changed (note if the advanced options will be used then the configuration wizard will be skipped on the first boot).

When the OS has been written completely to the SD Card, be sure to eject it safely from the PC and switch it to the Raspberry Pi computer. A keyboard, mouse with USB connector, and a monitor display with HDMI is needed. For the power supply, it is recommended 5.1 VDC and 3 Ampere output and when connecting the power source, the Raspberry Pi will start up directly and be ready.

6.2.2 Monitor Raspberry Pi from VNC Viewer

If there is no access to any monitor display, there are possibilities to connect and control the Raspberry Pi directly from the laptop with a program called PuTTY for just terminal window access or VNC Viewer for graphical user view.

The requirements are a PC or laptop and an ethernet cable for direct connection, it is also possible to connect wirelessly, but it is recommended to connect with an ethernet cable for a faster connection.

The programs PuTTY and VNC Viewer need to be downloaded and installed on the local computer, PuTTY can be found at the website:

<https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html>, and VNC Viewer could be found at: <https://www.realvnc.com/en/connect/download/viewer/>.

When PuTTY has been installed it is just to open the program, the default settings should just be kept as they are. When SSH is selected, type `raspberrypi.local` previously set when installing the OS, and port number 22 is standardly used, then click *Open*.

A terminal window field will arise with login where the username should be set and the password required. When the username and password have been entered, the connection status to the Raspberry Pi appears. When the Raspberry Pi is running for the first time it is a good idea to update all the packages to the latest version, with the command (note that this needs an internet connection):

```
sudo apt-get update && sudo apt-get upgrade -y
```

When all the updates are installed, open the Raspberry Pi Software Configuration Tool, this can be done with the command:

```
sudo raspi-config
```

Then navigate with the keyboard's arrows to *Interface Options* and select *VNC*. Here it should ask "Would you like the VNC Server to be enabled?" then select *Yes*.

Go to System Options → Boot / Auto Login choose *Desktop Autologin* and press *Enter* to apply. After that navigate to *Finnish* by pressing *Tab* and then pressing *Enter*.

When everything has been finished and configured with the use of PuTTY, the next step is to open the VNC Viewer program that has been installed. Set a new connection from *File* or push **CTRL-N**. From here in the property's menu, set the VNC Server hostname *raspberrypi.local* and then press OK. A field shows up where the VNC Server credentials need to be set, which means the Raspberry Pi's username and password.

After pressing *OK* the Raspberry Pi's screens should be visible from the local computer in the VNC Viewer program. From here it is possible to do the same things that could be done directly from the Raspberry Pi.

6.2.3 Installing MQTT broker on RPI

To be able to send commands and receive data through the internet a message protocol is needed. Novia has its own MQTT broker which will be used in this project, but now it will be explained how to install an MQTT broker on Raspberry Pi with the use of the MQTT protocol Mosquitto.

The first step is to upgrade and update the system by opening a new terminal window:

```
sudo apt update && sudo apt upgrade
```

Type **Y** and press **Enter**. This will take some time depending on how old the system is and how many new updates are coming.

To be able to install the Mosquitto broker type command:

```
sudo apt install -y mosquitto mosquitto-clients
```

When it has been loaded and the installation is done, then allow the mosquitto broker to automatically run in the background when the Raspberry Pi starts up.

```
sudo systemctl enable mosquitto.service
```

To check that everything works and find the mosquitto version running on the Raspberry Pi by command:

```
mosquitto -v
```

There is also a message that says *“Starting in local mode. Connections will only be possible from clients running on this machine. Create a configuration file that defines a listener to allow remote access.”* That means it is not possible to communicate with this Mosquitto broker from other devices than the configured Raspberry Pi computer. This applies from Mosquitto version 2.0 which could be cited *“In Mosquitto 2.0 and up, you must choose your authentication options explicitly before clients can connect. In earlier versions, the default is to allow clients to connect without authentication.”* from

<https://randomnerdtutorials.com/how-to-install-mosquitto-broker-on-raspberry-pi/>.

Two options could be chosen to be able to have remote access to communicate with other IoT devices. The first option is with no authentication and the second option is authentication with user and password.

The first step is to configure the mosquitto.conf file, this will be done in the terminal window by using the command:

```
sudo nano /etc/mosquitto/mosquitto.conf
```

Use the arrow keys to navigate to the end of the file and type “listener 1883” where the port number 1883 is used, after that it is needed to have one row below also “*allow_anonymous true*” which allows users to interact without user and password. To save the file press **CTRL-X** and after that type, **Y** and press **Enter**.

When everything is configured, restart the Mosquitto broker by typing:

```
sudo systemctl restart mosquitto
```

It is needed to find out the Raspberry Pi’s IP address to be able to use the Mosquitto broker, to do that type in the terminal window `hostname -I`, copy the IP address, and save it to a text file which will be used later.

Testing Mosquitto broker:

There are alternatives to test the Mosquitto broker after the installation and configuration by installing an MQTT client. To do that run the command:

```
sudo apt install -y mosquitto mosquitto-clients
```

After that run the mosquitto broker in the background by typing:

```
mosquitto -d
```

Open terminal window #1 and type the following command to subscribe to an MQTT topic “Test”:

```
mosquitto_sub -d -t Test
```

To publish a message to the topic “Test” a second terminal window #2 needs to be opened and then type:

```
mosquitto_pub -d -t Test -m "Hello World!"
```

When checking the first terminal window that subscribes to the message it can be seen “Hello World!” from the publisher. There could be several subscribers and publishers on this same topic, so if opening a third terminal window and type `mosquitto_sub -d -t Test` and after that again type in the terminal window #2 `mosquitto_pub -d -t Test "Hello World!"`, it could be seen this message in both terminal window #1 and #3 since they both clients are subscribed to the topic Test.

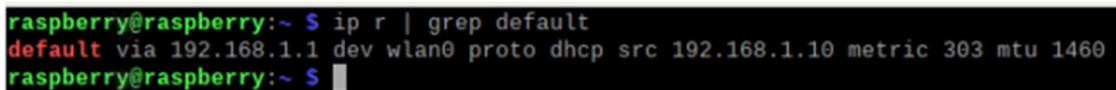
Assign Static IP Address to RPI:

It is recommended to have a static IP address assigned to the Raspberry Pi because the IP address that has been provided to not automatically change later when using the MQTT broker. If the IP address changes later the clients which already are configured will not find the broker anymore.

To start setting up, it needs to get some information about the current network, this could be done by typing:

```
ip r | grep default
```

Something like in Figure 30 should be shown. From here memorize the first IP address, in this example, 192.168.1.1 which is the current router address.



```

raspberrypi@raspberrypi:~$ ip r | grep default
default via 192.168.1.1 dev wlan0 proto dhcp src 192.168.1.10 metric 303 mtu 1460
raspberrypi@raspberrypi:~$

```

Figure 30. “ip r | grep default” command

The next step is to get the current DNS server address from the line “*nameserver <IP address>*” by typing:

```
sudo nano /etc/resolv.conf
```

The next step is to modify “*dhcpcd.conf*” configuration file to modify how the Raspberry Pi handles the network, this could be done by typing:

```
sudo nano /etc/dhcpcd.conf
```

At the bottom of this configuration file the following rows should be applied, see Figure 31. First, choose whether to configure a static IP for Wi-Fi connection “wlan0” or Ethernet connector “eth0”. In this case, use wlan0 instead of <Network>. Next, replace <StaticIP> with the IP address that want to have as static IP for the Raspberry Pi, and this should ensure that this IP cannot simply be attached to another device on the network. After that <RouterIP> should be replaced with the router IP address that has been noted. At last, replace <DNSIP> with example the DNS IP address that had been found earlier or

with Google's "8.8.8.8". Now when everything has been configured, save the file by pushing **CRTL-X** and then type **Y** and **Enter**.

```
interface <Network>
static ip_address=<StaticIP>/24
static routers=<RouterIP>
static domain_name_servers=<DNSIP>
```

Figure 31. Commands in "dhcpcd.conf" file.

When the Raspberry Pi's DHCP file has been configured, the RPI needs to be restarted so the new changes can be loaded. During the reboot, the RPI will try to connect to the router using the static IP address. To restart through the terminal window, type:

```
sudo reboot
```

To check that the static IP address has been assigned, open a terminal window and type:

```
hostname -I
```

6.2.4 Install and configure BlueZ

To be able to communicate with Nicla Sense through Bluetooth Low Energy it is needed to install and set up BlueZ on the Raspberry Pi.

To download the file, check the latest version from <http://www.bluez.org/download>, open a new terminal window, and enter the command:

```
wget http://www.kernel.org/pub/linux/bluetooth/bluez-5.66.tar.xz
```

after that write:

```
tar xvf bluez-5.66.tar.xz (at the moment version 5.66 is the latest).
```

Move to the folder by writing:

```
cd bluez-5.66
```

To install dependencies that the BlueZ library uses is done by typing:

```
sudo apt-get update
```

```
sudo apt-get install -y libusb-dev libdbus-1-dev libglib2.0-dev libudev-dev libicaldev
```

```
libreadline-dev
```

To run the configure script type within the BlueZ source directory:

```
./configure --enable-library
```

Once the configure scripts have been done, next is to compile the BlueZ code by typing:

```
make
```

When BlueZ code finishes compiling, it can be installed by the command:

```
sudo make install
```

After the installation, the next is to set up the BlueZ service, first to check the status type:

```
systemctl status Bluetooth
```

To start the BlueZ service manually, type:

```
sudo systemctl start Bluetooth
```

If checking again the status it should say active. To stop the service manually type:

```
sudo systemctl stop Bluetooth
```

It is good to set the BlueZ service so it starts directly when the Raspberry Pi boots up, this could be done by typing:

```
sudo systemctl enable Bluetooth
```

For information, to disable the autoboot use the command:

```
sudo systemctl disable Bluetooth
```

To enable BlueZ features like BLE, it could be modified in the BlueZ service configuration by typing:

```
sudo nano /lib/systemd/system/bluetooth.service
```

By adding **–experimental** last in the ExecStart line the experimental features are enabled and press “Ctrl-O” → “Enter” → “Ctrl-X”. To reload the system configuration and restart the BlueZ service type following commands:

```
sudo systemctl daemon-reload
```

```
sudo systemctl restart Bluetooth
```

When BlueZ is installed and configured, some more steps should be done. Open a new terminal window and type:

```
sudo apt update && sudo apt upgrade
```

Install the BlueZ Blueman Bluetooth Manager tool with the command:

```
sudo apt install bluetooth pi-bluetooth bluez blueman
```

Reboot the Raspberry Pi, and then it is ready to use.

6.2.5 Installing and implementation of Node-Red on RPI

On Raspberry Pi, it is possible to use a script that will install Node.js, npm (Node Package Manager), and Node-Red. When a new version is available, the script can also be used to upgrade an existing installation. To run the script to ensure npm can fetch and build any binary modules, open the terminal window and run the command:

```
sudo apt install build-essential git curl
```

To install all the needed parts for Node-Red, run the command:

```
bash <(curl -sL https://raw.githubusercontent.com/node-red/linux-installers/master/deb/update-nodejs-and-nodered) (Running on Raspberry Pi, u.d.).
```

Terminal window commands for Node-Red: **node-red-start** / **node-red-stop** / **node-red-restart** / **node-red-log**.

Once Node-Red is running, to access the editor on the web browser type <http://localhost:1880>, since port 1880 is standard and has not changed.

To start Node-Red directly when the Raspberry Pi starts up could be done by opening a terminal window and entering the command:

```
sudo systemctl enable nodered.service
```

The next time the RPI starts up Node-Red will automatically run in the background.

The Node-Red editor is not secured by default, anyone with access to its IP address can use it to deploy adjustments. To create a username/password before having access to the editor, the adminAuth property needs to be uncommented in the settings.js file that can be found in the Node-Red user directory `~/node-red`. To enter a new password type in the terminal window:

```
node-red admin hash-pw.
```


Node-Red code

Node-Red is up and running on the Raspberry Pi with automatic startup on boot. To navigate to the Node-Red Flow editor open a web browser and type the IP address of the Raspberry Pi Node-Red host and then the port number 1880 (for standard).

In Figure 32, the whole flow for this project can be seen and it will be more details explained below.

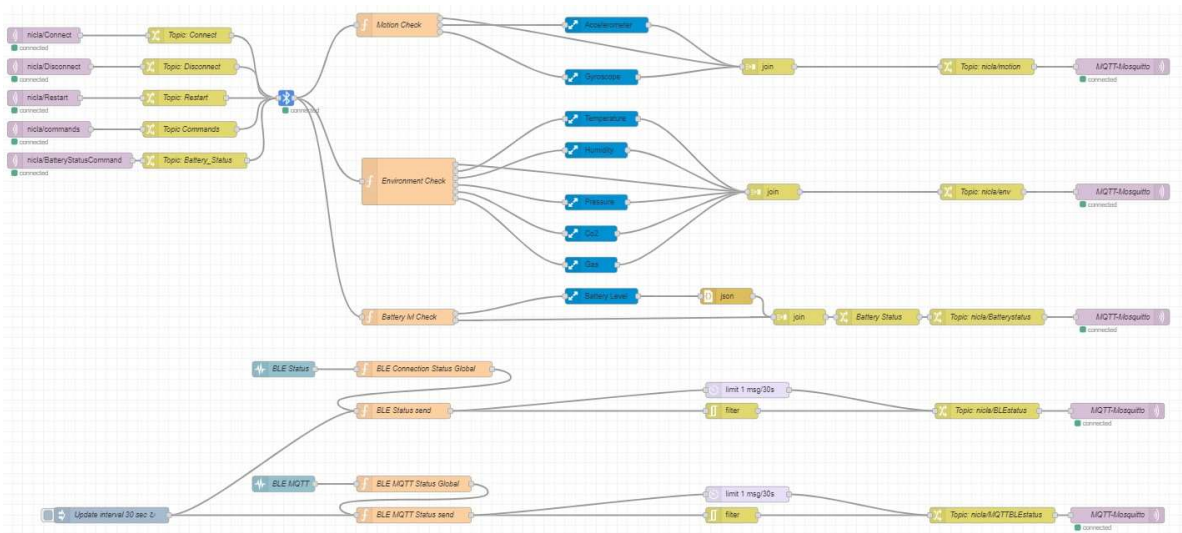


Figure 32. Overview of the Node-Red flow of the project.

There are several mqtt in nodes in the sketch with different topics. These mqtt in nodes communicate with the user interface and collect the commands. In Figure 33 the properties of the mqtt in node could be seen. First picture the topic will be noted, second picture the server address to the mqtt server and its port should be set. The third picture should be filled in with login details for the mqtt broker if there are any. The fourth picture should be filled in with a message and topic if there is a connection, disconnection, or unexpected disconnection. Novia's own MQTT Broker was used here.

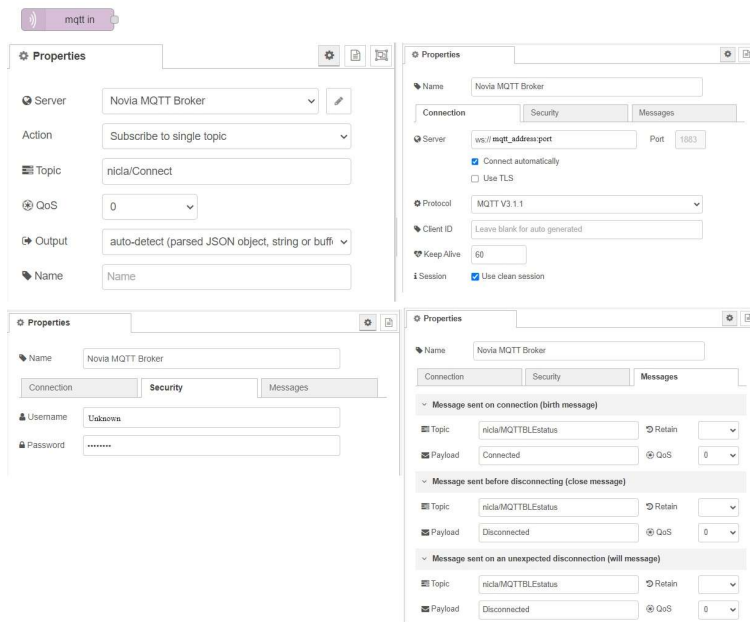


Figure 33. Settings of MQTT in node.

The change node was used. This node will collect the topic from the mqtt in node and then change the topic. For example, in Figure 34 the Nicla/Connect topic comes from the mqtt in node and then the change node will just send this topic as “connect”. This needs to be done because the Bluetooth node would be able to read the commands.

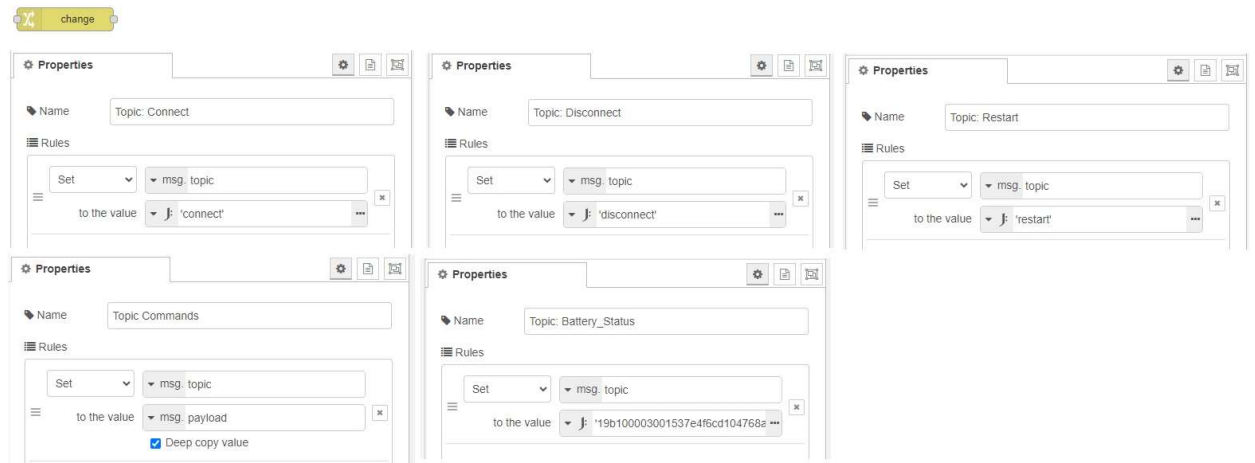


Figure 34. Settings of Change node.

The Generic BLE in node is the Bluetooth communication core. This node collects the command from the user interface and receives the sensor readings from the Nicla Sense. At the properties, it is possible to scan for new BLE devices and find their MAC address,

when connected to the device and then press apply, all the BLE GATT Characteristics can be seen configured at the microcontroller.

To access the BlueZ D-Bus API, the host of the Node-Red process must be a member of the Bluetooth group, otherwise the Bluetooth node will not work at all due to the bluetooth permission issue. In the terminal window at the Raspberry Pi run the command and after that, the OS needs to be rebooted:

```
sudo usermod -G bluetooth -a username
```

```
sudo reboot
```

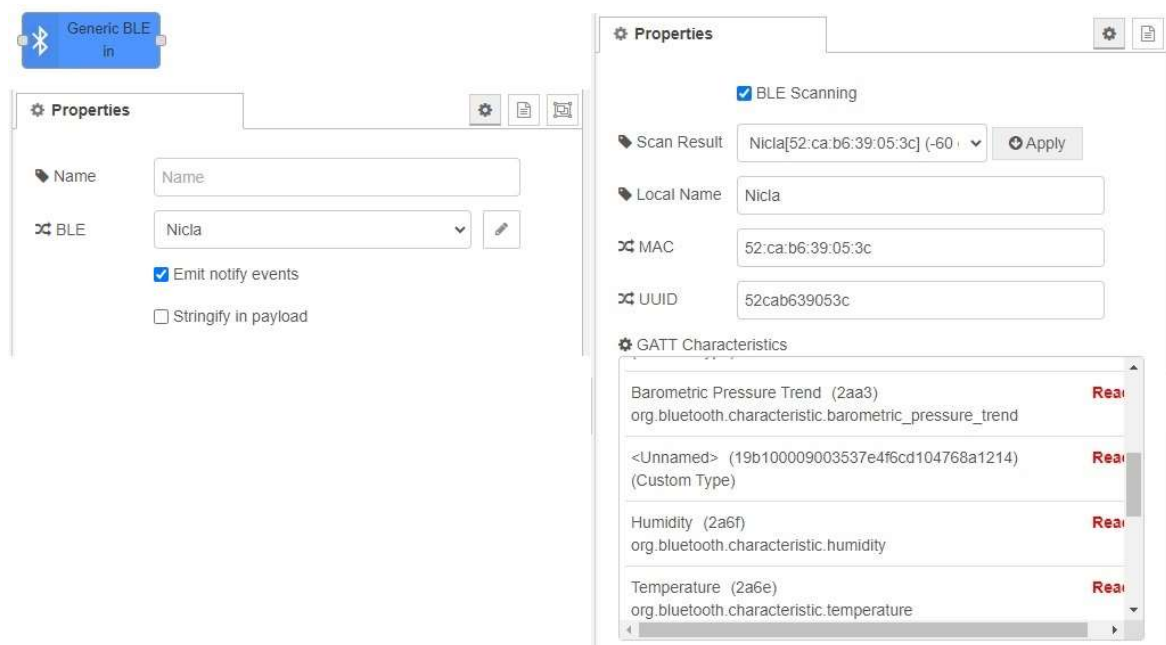


Figure 35. Settings of Generic BLE in node.

The Bluetooth node splits up into three different function nodes one for motion sensors, the second for environment sensors, and last the battery level status. These nodes check when incoming messages arrive if their UUID can be found and then let it pass through the function node otherwise it rejects the output.

Code examples 1 - 3 show three different function node codes, all these three function nodes have in common one input and an output channel for timestamp. Motion Check function has three outputs: timestamp, accelerometer, and gyroscope. Environment Check has six outputs: timestamp, temperature, humidity, pressure, CO2, and gas. Battery lvl Check has two outputs: timestamp and battery level.

Code example 1. Function node (Motion Check).

```

const date = new Date();
var time = date.toLocaleTimeString();

Object.keys(msg.payload.characteristics).forEach(key => {

    // Timestamp
    if (key == ('03505d7c10014997ad89cd1456dc7490') || ('03505d7c10024997ad89cd1456dc7490')) {
        var msg1 = { payload: { "Time": time } };
    };

    // Accelerometer
    if (key == '03505d7c10014997ad89cd1456dc7490') {
        var msg2 = msg;
    };

    // Gyroscope
    if (key == '03505d7c10024997ad89cd1456dc7490') {
        var msg3 = msg;
    };

    node.send([[msg1], [msg2], [msg3]]);
})

```

Code example 2. Function node (Environment Check).

```

const date = new Date();
var time = date.toLocaleTimeString();

Object.keys(msg.payload.characteristics).forEach(key => {

    // Timestamp
    if (key == ('2a6e') || ('2a6f') || ('2a6d') || ('a7f5a4f310014941bc756d7f5d27f436') ||
('a7f5a4f310024941bc756d7f5d27f436')) {
        var msg1 = { payload: { "Time": time } };
    };

    // Temperature
    if (key == '2a6e') {
        var msg2 = msg;
    };

    // Humidity
    if (key == '2a6f') {
        var msg3 = msg;
    };

    // Pressure
    if (key == '2a6d') {

```

```

var msg4 = msg;
};

// Co2
if (key == 'a7f5a4f310024941bc756d7f5d27f436') {
var msg5 = msg;
};

// Gas
if (key == 'a7f5a4f310014941bc756d7f5d27f436') {
var msg6 = msg;
};

node.send([[msg1], [msg2], [msg3], [msg4], [msg5], [msg6]]);
})

```

Code example 3. Function node (Battery lvl Check).

```

const date = new Date();
var time = date.toLocaleTimeString();

Object.keys(msg.payload.characteristics).forEach(key => {
  // Battery Level
  if (key == '2a19') {
    var msg1 = msg;
  };
  // Timestamp
  if (key == '2a19') {
    var msg2 = { payload: time };
  };

  node.send([[msg1], [msg2]]);
})

```

When data is read from the Bluetooth device it reads byte by byte to a data buffer. Data is temporarily stored and sent using the data buffers (Cope, 2023).

The buffer-parser node converts incoming messages from the buffer to float/int/uint values. The node searches for its UUID and if it matches the incoming message, it will convert it and let it pass to the next node. An example of the settings for the accelerometer can be seen in Figure 36.

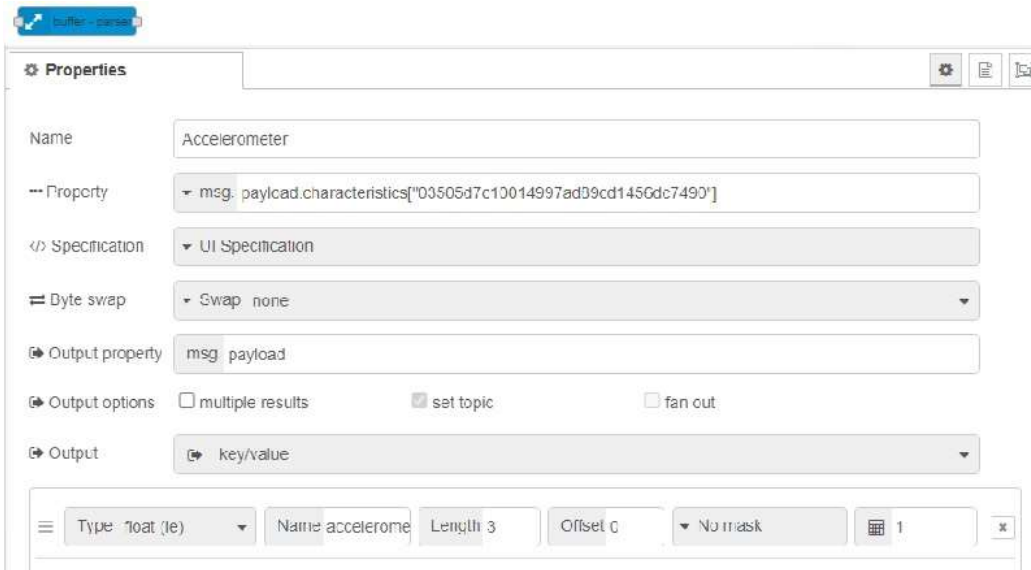


Figure 36. Settings of buffer - parser for the accelerometer.

The join node combines the incoming messages to just one merged object for the sensors and to one string for the battery level. After a message has passed the node, it waits 0.1 seconds for all incoming messages and after that sends it forward to a change node that will add the correct topic for the user interface.

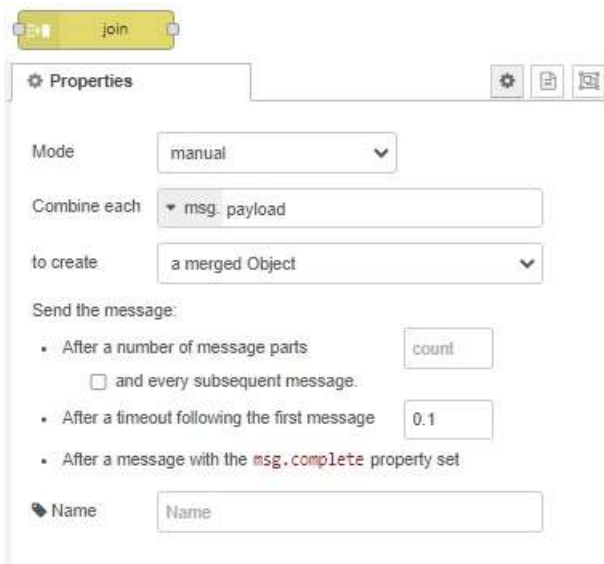


Figure 37. Settings of join node.

The mqtt out node is configured exactly as the mqtt in node but the difference is that it will send out the message.

The JSON node converts incoming object messages to strings.



Figure 38. Settings for JSON node.

When the battery level message has passed the join node and combined the time and battery level status, it will pass a change node that will change a number between one and five. These numbers indicate the status of the battery which will change to a description in the change node which can be seen in Figure 39.

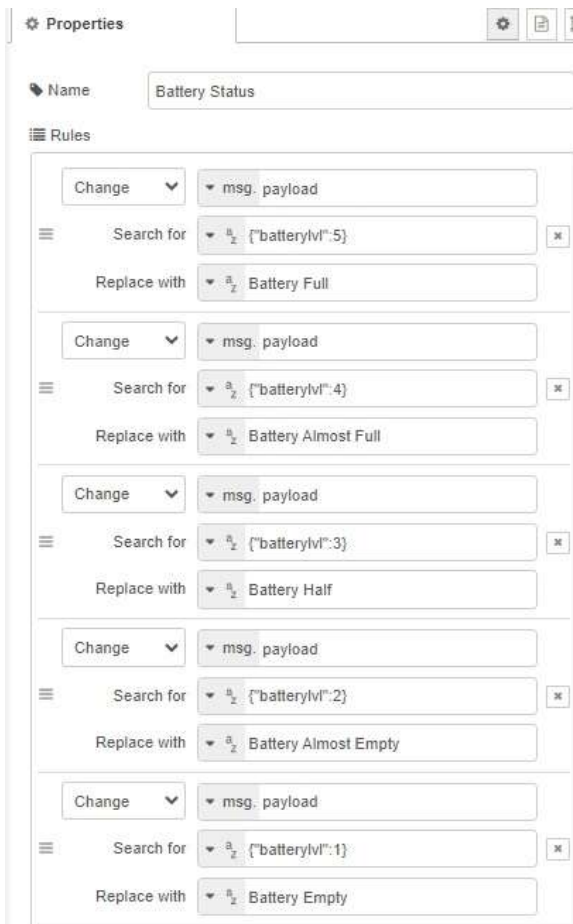


Figure 39. Change node for battery status.

The status node BLE status reports the status from the Bluetooth connection and the BLE MQTT status node reports the mqtt out connection status. If any of the two nodes change status from example disconnected to connected it will send a message to the status node with the change. At the properties for the BLE status node the “Generic BLE in” should be checked to get the status from that node, see Figure 40.

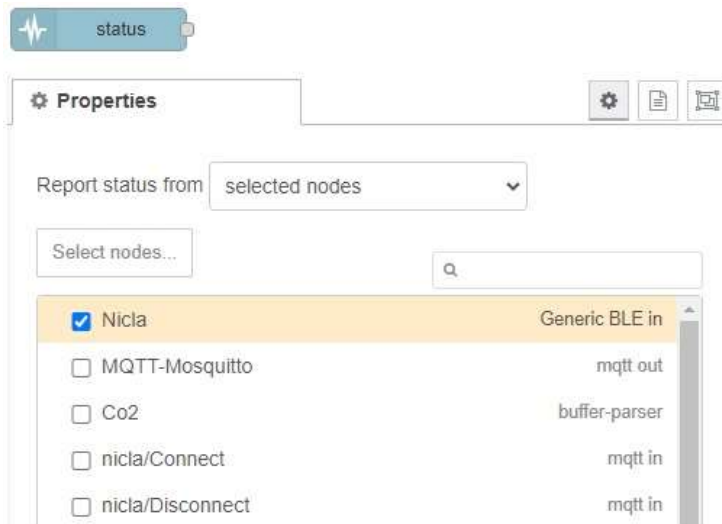


Figure 40. Settings of BLE Status node.

After the BLE Status and BLE MQTT Status node, a short function node for each should be added, this function stores or sets the status object global for the next function with the use of `global.set`.

Code example 4. Function of BLE Connection Status.

```
var ble_status=msg.status;
global.set('ble_status',ble_status);
return msg;
```

Code example 5. Function of BLE MQTT Status.

```
var bleMQTT_status=msg.status;
global.set('bleMQTT_status',bleMQTT_status);
return msg;
```


The next function nodes “BLE Status send” and “BLE MQTT Status send” check for any of the connection statuses which then send the correct message forward. With the use of `global.get`, the status object will be saved to a variable and then search for the correct status from `status.text`, when the correct status is found a `msg.payload` will send a text with example “Connected” or “Connecting” depending on the status.

Code example 6. The function of BLE Status Send.

```
var status=global.get('ble_status');
node.log("status is =" +status.text);
if (status.text == "generic-ble.status.connected")
{
  msg.payload="Connected";
  return msg;
}
if (status.text == "generic-ble.status.connecting") {
  msg.payload = "Connecting";
  return msg;
}
if (status.text == "generic-ble.status.disconnected") {
  msg.payload = "Disconnected";
  return msg;
}
if (status.text == "generic-ble.status.disconnecting") {
  msg.payload = "Disconnecting";
  return msg;
}
if (status.text == "generic-ble.status.missing") {
  msg.payload = "Missing";
  return msg;
}
if (status.text == "generic-ble.status.error") {
  msg.payload = "Error";
  return msg;
}
return null;
```

Code example 7. The function of BLE MQTT Status Send.

```

var status=global.get('bleMQTT_status');
node.log("status is =" +status.text);
if (status.text == "node-red:common.status.connected")
{
  msg.payload="Connected";
  return msg;
}
if (status.text == "node-red:common.status.connecting") {
  msg.payload = "Connecting";
  return msg;
}
if (status.text == "node-red:common.status.disconnected") {
  msg.payload = "Disconnected";
  return msg;
}
return null;

```

An inject node pushes every 30 seconds an update about the status for both the BLE and BLE MQTT status, therefore a filter node has been inserted for both which will block the message if not the status has been changed since the last incoming message. This is a good way to not burden the mqtt broker unnecessarily.

A delay node has been connected in parallel, with a delay of 30 seconds. This node is needed because if something has been stuck, an update should be updated every half minute.

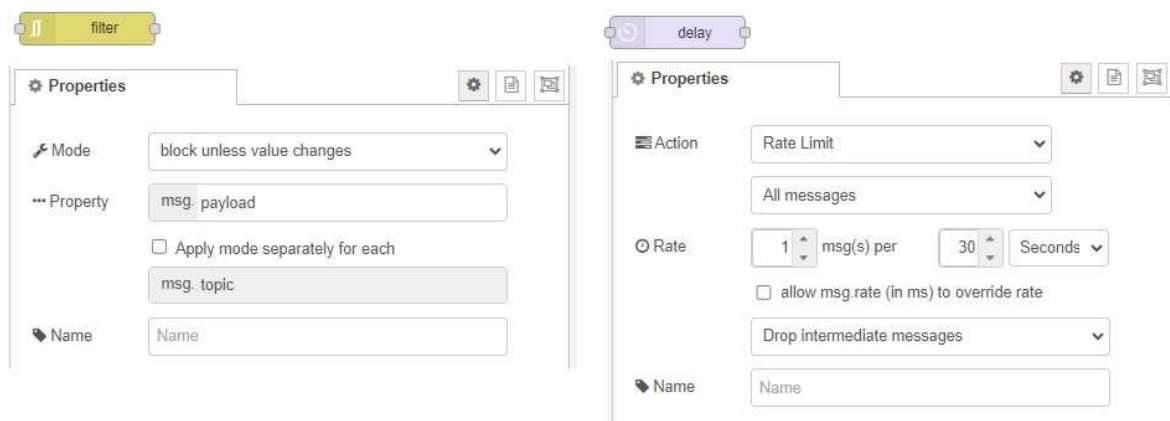


Figure 41. Settings for filter and delay node.

6.3 Implementation of a graphical user interface

In this chapter, a graphical user interface will be developed with the use of JavaScript, HTML, and CSS styling. First The HTML part will be described after that the CSS style and at last the JavaScript functionality. This graphical user interface is inspired and based from a dashboard project at <https://github.com/donskytech/mqtt-custom-dashboard-node-js> and then further developed with own needs of all the included files below in this chapter (Appendix II – VIII).

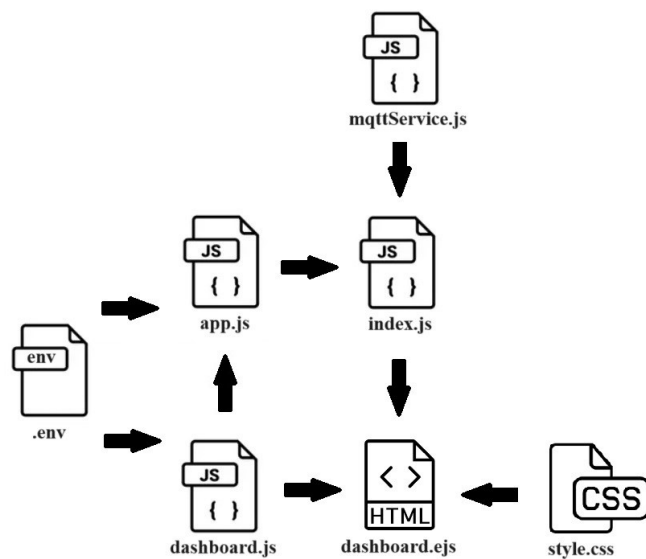


Figure 42. Overview of files used for graphical user interface.

Appendix II: *dashboard.ejs*

Appendix III: *style.css*

Appendix IV: *index.js*

Appendix V: *mqttService.js*

Appendix VI: *app.js*

Appendix VII: *dashboard.js*

Appendix VIII: *.env*

Figure 42 shows an overview of the files for creating a webpage, the env. file consists of configuration information that the app.js and dashboard.js receive. The app.js includes the Express web server application and the dashboard.js is the router that handles the web URL. The mqttService.js handles the code to the MQTT Broker and the index.js calls the mqttService.js to communicate with the MQTT broker and also handles all the other functions of the webpage. The style.css file is used to style the HTML template and the

dashboard.ejs contains the web application and collaborates with the functions and styles from the other files.

Several Node.js modules are stored in a folder node_modules at the project's root directory, some examples are express, ejs, and dotenv. A file package.json is also stored in the project's root directory which contains the Node.JS configurations and dependencies required by the application.

6.3.1 HTML (Hyper Text Markup Language)

Links that join web pages together, either within a single website or between several websites are referred to HTML. In this project, a file named "dashboard.ejs" was used (see Appendix II), .ejs stands for Embedded Javascript Templating which is a templating engine used by Node.js. A template engine can help to create an HTML template with less code. In addition, it can generate the HTML by introducing data into an HTML template on the client side. (Use EJS as Template Engine in Node.js, u.d.). The dashboard.ejs imports parts of CSS and JavaScript from other files to make a complete web page.

Located between the <html> and <body> tag a <head> element is used as a container for metadata. Data about the HTML document is called HTML metadata, there is however no display for this information. Character sets, styles, scripts, document titles, and other meta information are usually defined via metadata (HTML - The Head Element, u.d.).

The <body> tag defines the document's body, and the <body> element includes all the contents of the HTML document for example all the headings, images, graphs, and more. It can only exist one <body> tag in an HTML document (HTML <body> Tag, u.d.).

Class and ID selectors are used to identify different elements of HTML while writing CSS and JavaScript. The ability to present an HTML element differently based on its class or ID is the primary advantage of setting a class or ID. The ID is a unique identifier of the HTML element that has to be utilized when one HTML element on the page has a particular style or function. An example to add the ID to the HTML could be: <div id= "id"> and to identify in CSS or JavaScript the symbol "#" is used before the identifier (Kaur, 2020).

Class is not unique and is used when several HTML elements on the same page need to have for example the same style applied. An example of adding a class to the HTML element

could be: `<div class="class">`, to identify the class in JavaScript or CSS the symbol "." is used before the identifier (Kaur, 2020).

For marking up text, images, and other content for display in a web browser, HTML uses "markup" that has unique elements such as: ``, `<table>`, `<p>`, `<div>`, ``, and more on. One thing to note with all types of tags is that when ending a tag it should be used `</>` for example `<div> "content" </div>` or `<button> id="Type_of_ID"</button>` (HTML: HyperText Markup Language, 2023).

Tags that consist of the element name contained by "<" and ">" are used to separate HTML elements from other content on a page. The letters are not essential when naming an element inside a tag, which means it can be written in uppercase or lowercase as `<title>` or `<TITLE>` but tags are recommended to be written in lowercase which is usual and advised (HTML: HyperText Markup Language, 2023).

The `<div>` tag is frequently used in HTML which defines a section/division. The HTML elements are contained within the `<div>` tag and can be modified using CSS and JavaScript with the use of class or id attributes (HTML `<div>` Tag, u.d.). To mention some of the `<div>` tags in this code there are example classes of temperature, pressure, CO₂, gas, accelerometer, and gyroscope. Other classes are history charts, gauges, and buttons.

The `<input type="..">` tag provides input information by the user. Different input types could be used in this case, type of checkbox has been used for sending out specific commands depending on the sensor for example `"<input id="envCheck1" type="checkbox" value="2a6e"/><label>Temperature</label>"` which means when checkbox temperature has been checked the value 2a6e (Bluetooth command) will be sent for every event dependent on the sampling time. Another input type used is "button" which is assigned to an ID and gives input on click, there are seven different buttons on the user interface: connect, disconnect, battery status, start, stop, download, and clear. One more kind of input type used is "text" and this input reads and sends the sampling time that the user writes in the text box (requires the sampling time, letters will not be recognized).

There are two different tables in this project one for environment sensors and the other for motion sensors. This is done by use of the tag `<table>` and includes all the elements that should be included before the `</table>` tag.

The `` tag is an inline container for marking up a section of a text or document. The class or id attribute of the `` tag makes it simple to customize using CSS or control with JavaScript (HTML `` Tag, u.d.). Inside the `<head>` tag a link is entered “`<link href=https://fonts.googleapis.com/icon?family=Material+Symbols+Sharp rel="stylesheet"/>`” and makes it possible to import symbols from this link, by clicking on the desired symbol from the link and copy from “inserting the icon” for example the temperature gauge: ` device_thermostat `.

Two different images are stored in an images folder at the project, one picture represents the Novia logo, and the second picture a microcontroller Nicla Sense. These pictures are imported by searching the source file by `` and ``.

6.3.2 CSS (Cascading Style Sheet)

A CSS file named `style.css` (see Appendix III) is imported by the HTML file (`dashboard.ejs`) for styling most of the graphical user interface. With the use of tags, classes, and IDs it is possible to set parameters for each group which then will be exported to the dashboard layout.

At the beginning of the file `style.css`, many different backgrounds are assigned for example dark/light mode and background colors for different purposes with the use of color hex code or RGB code, these color codes can be chosen at www.htmlcolorcodes.com. A hex color / RGB color is assigned to a name which will then be used later for the tags below in this file. The hex code starts with “#” with a specific code for example (`#FE1B04`) that represents the red color and the RGB color has three different numbers which represent “Red” “Green” and “Blue” for example (99, 209, 35) which combines into green color.

The style is divided into different categories for example tags of headings `h1`, `h2`, and `h3` which have their specific setting in this case different font sizes. The insights are identified by class which represents in this case the sensor gauges, each of these gauges has one picture/symbol that represents its purpose. Each of these symbols has a specific color assigned, in code example 8 green color is added to the “`—color-insight-1`” and specifies the insight of temperature.

Code example 8. The background color of the symbol for temperature.

```

/* Color of measurement pictures*/
--color-insight-1: rgb(99, 209, 35);

/* Insights different colors*/
aside .insights > div.temperature span {
  background: var(--color-insight-1);
}

```

@media only screen and (max-width: px) is used to apply when the user uses either a PC/laptop or mobile phone for example. This setting could change the style view depending on what device connects to the HTML side which means this tool is even user-friendly with the use of smartphones.

In CSS it is possible to use various units of measure to express the lengths or sizes of elements, alternatives can be for example px, rem, vh, vw, and %. Even though pixels (px) are related to the viewing device's DPI and resolution, it is still considered absolute units. The px unit on the device is fixed and independent of any other element and is a good option if there are elements that should not be scaled. The units rem, vh, vw, and % are relative units where rem is relative to the root element, vh is relative to the viewport's height and vw is relative to the viewport's width. Since relative units scale up or down based on the size of another element, it performs better across a range of devices. The standard font size in the majority of browsers is 16 px and from this basis, the relative units determine the size (Units of measurement, 2023).

Some basic configurations in CSS are for example "right", "left", "top", "height", "width" and "text-align" and the value to decide where to place text, picture, or something else on the webpage. Other options to use could be "font-weight" to decide how the text should look, "font-size" to decide the size of the text, "color" to change the color of the text, or "background-color" to use another color than white as background of the specific place or whole webpage (CSS Introduction, u.d.).

6.3.3 JS (JavaScript)

There are many ways JavaScript could be linked to an HTML page. JavaScript code must first be downloaded to the browser before it can be used to run the scripts. When a web page has a lot of scripts included it can run slowly. The method of placing JavaScript code inside an `<script>` element is referred to “Internal Javascript” for example `<script type="text/javascript"> alert ("Hello World!"); </script>` (Connolly & Hoar, 2018).

“External JavaScript” is often useful and recommended to have JavaScript code in an external file that could communicate with the HTML file, the extension of the JavaScript file is defined by `.js` and by use of for example `<script type="module" src=./filename.js> </script>` (Connolly & Hoar, 2018). For this project, the external JavaScript method is used to import the `index.js` file to the HTML dashboard.

By adding ID or Class identifiers from example buttons or text fields in the HTML file it is possible to link triggers and values from the web page to the JavaScript file by adding the identifiers to the specific JavaScript function.

HTML items of buttons, checkboxes, theme toggles, etc. are assigned with variables in the `index.js` file by use of `querySelector` and `getElementById`. The `querySelector()` is used to extract elements from the document. The first element that matches the given selector (Class, ID, or tag name) is returned. The `getElementById()` is used to get an element from the document by the ID attribute. `getElementById()` will always return one element or null if there is not a matching element because IDs have to be unique. `getElementById()` needs only to search for one element and is faster than the `querySelector()` (Colelevy, 2023).

The `EventTarget` interface `addEventListener()` functionality configures a function to be called each time the designated event is sent to the target (MDN contributors, 2023). For example, `themeTogger.addEventListener` change between day and night background dependent on what mode the button is set to. `window.addEventListener` is another event listener that activates when the page is loaded, in this case, it activates the line chart configurations, MQTT connection, and the `mediaQuery` (format of the device connected).

The plotly line charts will show historical data from the sensors during collection. The environment sensors draw lines with marker dots while the motion sensors just draw

straight lines on the charts. The layout of the graphs is adjusted in this file with settings like sizes, colors, auto-size options, etc.

The line chart max lines/points have been adjusted, the environmental sensors reject the latest point when 50 points have been collected to the line chart, and for the motion sensors, the max points are 100.

The function *updateMotionSensorReadings* is used by a callback function that will retrieve sensor readings and redraw the chart with the latest readings. When a message is received by the callback function *OnMessageMotion*, it checks if any of the motion checkboxes are checked and if any of these are true it will add the values to Plotly.newPlot configuration. This function also checks if the “Clear” button has been pressed and needs to start from the beginning.

The function *updateMotionBoxes* add the latest received X, Y, and Z value for the accelerometer and gyroscope gauges while the *updateMotionCharts* function collects data and checks the sample length to the maximum graph point at the chart and then push the values of X, Y, and Z to the next sample point on the graph. The function *updateEnvBoxes* and *updateEnvCharts* work in the same way but these functions are focused on the environmental sensors. The function *updateXArray* is used to increase by one on the X array for every event on this function.

Windows.matchMedia compares the size of the user screen and if the max width is smaller than 600px addEventListener changes with the use of the *handleDeviceChange* function new settings of the variable updateHistory.

To access the MQTT connections from the server it is essential to use the *fetchMQTTConnection()* function and below this different function connections are used that handle its own MQTT topic.

In code example 9 an if state is used to check if any of the checkboxes for environmental or motion sensors are checked and also notes the sampling time. Then binds together all the Bluetooth commands from the checkboxes and after that sends the commands within the desired time interval until the stop button is pressed which clears the time interval.

Code example 9. Function of environmental sensor checkboxes.

```

// Reference the Environment Table.
if (envCheck1.checked == true || envCheck2.checked == true || envCheck3.checked == true ||
envCheck4.checked == true || envCheck5.checked == true) {
    var timeInterval = document.querySelector('#Time_Interval_Env').value;
    var chks = tblSensors.getElementsByTagName("INPUT");
}

// Reference the Motion Table.
if (motionCheck1.checked == true || motionCheck2.checked == true) {
    var chks = motion_tblSensors.getElementsByTagName("INPUT");
    var timeInterval = document.querySelector('#Time_Interval_Motion').value;
}
var mqttService = new MQTTService(mqttServer);
mqttService.connect();

// Loop and push the checked CheckBox value in Array.
for (var i = 0; i < chks.length; i++) {
    if (chks[i].checked) {
        selected.push(chks[i].value);
    }
}

// Display the selected CheckBox values.
if (selected.length > 0) {
    var commands = (" " + selected.join(", "));
    let timerId = setInterval(function() {
        mqttService.publish(mqttSensorCommandsTopic, commands);
    }, timeInterval);

    // Stopping the timer:
    stop.addEventListener('click', function() {
        console.log("stop")
        clearInterval(timerId);
    });
}
});
}

```

To store the data from the sensor records, the sessionStorage is used, which means it stores the data for just one session on the client-side web storage, and the data is deleted when the browser is closed, refreshed, or in this case by pressing the “Clear” button. Data that are saved to the sessionStorage uses key-value pairs and these both are strings. The getItem(key) retrieves the value for the given key, setItem(key, value) sets the value for the given key, and clear() removes all key-value pairs. The sessionStorage is limited to 5 Mb storage (Zakas, 2009). An investigation has been executed to check how many samples are possible to collect during one session if all the sensors are used (environment or motion) 50 samples correspond to approximately 2 Kb memory. $5000 \text{ Kb} / 2 \text{ Kb} \rightarrow 2500 * 50 \rightarrow 125\,000$ samples with the 5 Mb limitation, so if the sampling time is set to one second it would be possible to collect data for approximately 34.5 hours.

Code example 10. sessionStorage setItem or clear.

```
function sensorValues(temperature, humidity, co2, gas, pressure, time) {
  if (restoreKey > 0) {
    sessionStorage.clear();
    console.log("clear records");
    restoreKey = 0;
    sessionRow = 0;
  } else {
    sessionRow;
  }

  sessionRow++;

  var sensorData = [sessionRow, time, temperature, humidity, pressure, co2, gas];
  var sensorDataString = JSON.stringify(sensorData);

  sessionStorage.setItem(sessionRow, sensorDataString);
}
```

In code example 11, when the download button is pressed, the `sessionStorage.getItem` is used to collect the data from `sessionStorage.setItem`. The if states are used to define the first column for headings on the CSV file. Environment sensors will always have the same headings if any of the sensors are used cause the NaN values will be included for the sensors not measured. The motion sensors have three different options when using both sensors, or either sensor. `sessionForEach` function adding a new line for each new sample.

Code example 11. sessionStorage gets data and download function.

```
// Create a user-defined function to download CSV file
downloadCSVbtn.addEventListener('click', function(){

// Iterate sessionStorage
for (var i = 0; i < sessionStorage.length; i++) {

// Set iteration key name
var key = sessionStorage.key(i);

// Use key name to retrieve the corresponding value
var value = sessionStorage.getItem(key);
sessionData.push(JSON.parse(value));
console.log(sessionData);
}

// Define the heading for each row of the data
if (envCheck1.checked == true || envCheck2.checked == true || envCheck3.checked == true ||
envCheck4.checked == true || envCheck5.checked == true) {
  var csv = 'Sample,Timestamp,Temperature,Humidity,Pressure,Co2,Gas\n';
}

if (motionCheck1.checked == true && motionCheck2.checked == true) {
  var csv = 'Sample,Timestamp,accX,accY,accZ,gyrX,gyrY,gyrZ\n';
}
}
```

```

else {

    if(motionCheck1.checked == true) {
        var csv = 'Sample,Timestamp,accX,accY,accZ\n';
    }

    if(motionCheck2.checked == true) {
        var csv = 'Sample,Timestamp,gyrX,gyrY,gyrZ\n';
    }
}

// Merge the data with CSV
sessionData.forEach(function(row) {
    csv += row.join(',');
    csv += "\n";
});

var hiddenElement = document.createElement('a');
hiddenElement.href = 'data:text/csv;charset=utf-8,' + encodeURIComponent(csv);
hiddenElement.target = '_blank';

// Provide the name for the CSV file to be downloaded
hiddenElement.download = 'SensorData.csv';
hiddenElement.click();

sessionData = [];
});

```

The .env file includes the title Nicla Sense Dashboard, and also the title Novia UAS on the webpage. The dashboard.js renders the view and sends the titles from the .env file to the dashboard. The env. file also includes the different MQTT topics and the MQTT broker address which will be forwarded to app.js. The app.js starts the webpage server on port: 4000, and also gets and processes the mqtt topics. The mqttService.js file connects to the MQTT broker, includes publish and subscribe, and uses callback functions when messages arrive from different topics. Index.js as previously stated contains most functions and processes from the mqtt topics and takes in data from the sensors which are included in functions that are then transferred to the web page.

6.4 Construction of Nicla Sense protection box

In this chapter, the steps of how to produce a protection box for the Nicla Sense will be explained. First, the steps of the 3D drawing in Rhino 7 will be explained, and after that the steps of how to get the 3D model printed with the use of UltiMaker Cura software.

6.4.1 3D drawing of protection box

Before starting to draw, it is necessary to decide what battery should be included in the box to estimate the size. In this project, a battery marked LIPO-603048 was used which has a size of 48x30x6 millimeters (L x W x H).

The next thing is to decide the thickness of the walls, in the calculation both side walls are included. The length of the box has been calculated with 48 mm (battery) + 6 mm (walls) + 4 mm (airspace at sides of the battery) = 58 millimeters. The width has been calculated quite similarly, 30 mm (battery) + 6 mm (walls) + 4 mm (airspace for the battery). The height was a bit more difficult to determine but estimated to be 6 mm (battery) + upper wall thickness 2 mm + 2 mm recess for the upper hatch and the bottom wall to 3 mm. Then 13 millimeters were reserved for airflow between the battery and the Nicla Sense and reclining support for the battery and mounting brackets for the Nicla Sense, the sum for the height was then set to 26 millimeters.

When all the sides are determined a solid box could be drawn with these sizes and the tool could be found at "Solid Tools" → "Box: Corner to Corner, Height". After that, one more box should be drawn with a size of 55 x 34 x 23 mm, this box should be inside the first box that was created, then go to "Solid Tools" → "Boolean difference" and then do according to the command. Figure 43 shows the before and after view of the box.

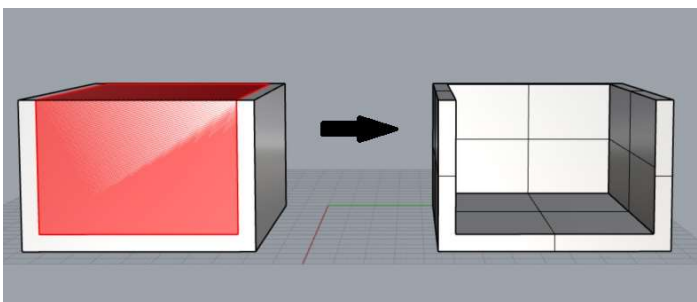


Figure 43. Before and after the use of the Boolean difference tool.

The next step is to make a 1.5 x 2 mm recess inside the box along where the hatches will be able to slide in. Two millimeters were reserved from the outer edges. This will be done with the same steps as previously.

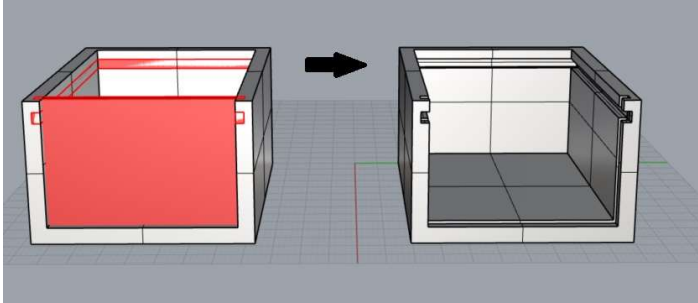


Figure 44. Before and after recess for hatches.

After that, an opening for the Nicla Sense was made. The dimensions for this were considered by where the sensors are on the microchip so they have an open surface for readings. Also, the attachment point was considered so the Nicla Sense could not slip through the box and the USB port was also noticed that the USB cable could fit into the port.

Then after that the shelf for the battery was constructed and then the mounting rail for the Nicla Sense. To merge the shelf with the Nicla Sense Box, design the shelf then place it in the desired field along the Nicla Sense box. To merge the shelf with the Nicla Sense box go to "Solid Tools" → "Boolean Union" select the objects to merge and then press enter. For the Nicla Sense, there was also an LED indicator light and reset button which should be able to reach so at the same time when creating the mounting rail the holes for this were considered.

A switch was added to be able to turn off the Nicla Sense when not in use. A small switch marked with *1K2-S (1 x ON – ON 0,5 A)* was used, this switch was mounted above the Nicla Sense.

Six air gaps were made at suitable distances on each side of the battery and three gaps were made at each side to be able to insert cable ties to hold the box steady for the project's different purposes. Text with "Nicla Sense" was also engraved along the longest sides, this could be done with the use of "Text object" and note that the solids box is

ticked to create 3D text. The text was placed into the box sides and then used “Solid Tools” → “Boolean difference”.

When the box and all the details are completed, all the adjustments and edits could be compounded by first marking the box and then typing in the command field “MergeAllCoplanarFaces” and then pressing enter. The last step that should be done is to create softer edges of the box, this could be done by going to “Solid Tools” → “Fillet edges” and then clicking on all the outer lines of the box which should be rounded then press enter and decide the radius and then enter again, note that this could be done after the hatches have been created. In Figure 45 a clearer view of the result is shown.

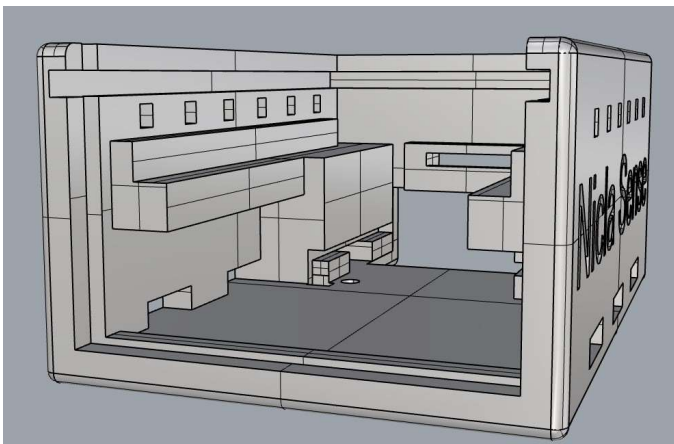


Figure 45. Nicla Sense box details.

When the box has been completed, the hatches should be created. The hatches have been done in quite a similar way as the box, the one above the battery has some ducts for the airflow. The other hatch has been engraved with the text Novia. In Figure 46 the completed parts of the Nicla Sense box can be seen with the outer dimensions.

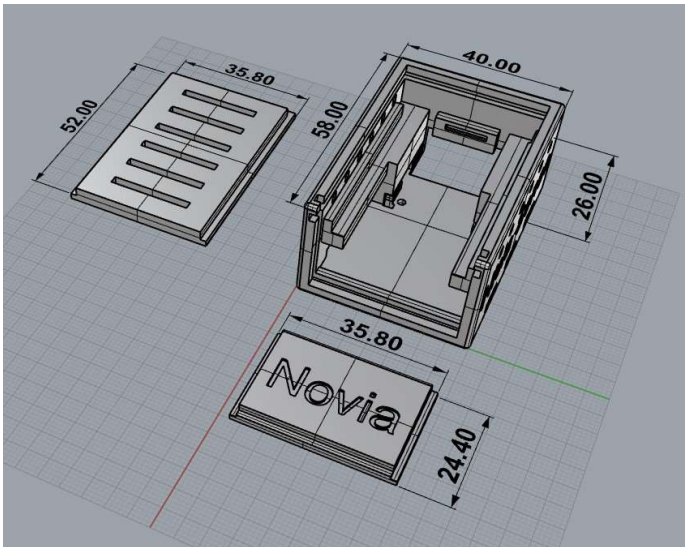


Figure 46. The final version of the Nicla Sense box with dimensions.

6.4.2 3D printing of protection box

Before saving the STL file, the three parts (the box, upper hatch, and side hatch) should be considered in how to place the different parts to avoid the 3D printer requiring a lot of support material. That means, not having a lot of air below something that should be 3D printed, Figure 46 shows a good way to place these parts.

The file is saved to an STL (Stereolithography) file. STL Mesh Export Options Tolerance was set to 0.01 millimeters and in ASCII format. Next, the UltiMaker Cura program was opened, and the 3D printer used was selected from the list then a layout of the 3D printer will appear where the max size of printer volume could be seen.

The STL file should be imported into the program, In Figure 47 the Nicla Sense protection box is imported to the UltiMaker Cura program, and to the right side, the necessary settings are listed on the extruder 1. When all the settings are set, the slice button should be pressed and then the estimated time will be shown, in this case, 2 hours and 27 minutes. It is also possible to click on the preview button to check all the supports that will be inserted into the 3D object. When everything is good click “save to disk” and save it to a USB flash drive which then will be inserted into the 3D printer.

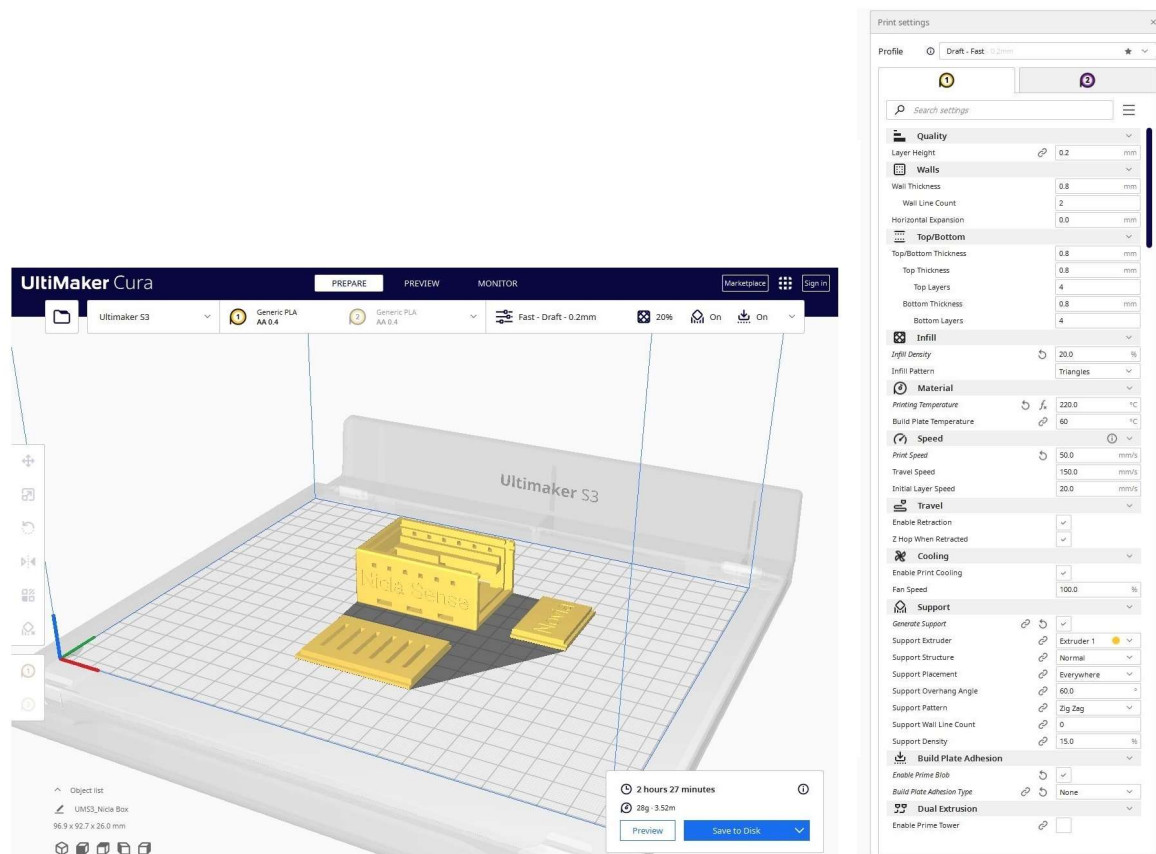


Figure 47. UltiMaker Cura with settings of the print.

PLA filament will be used, so first verify that the correct filament is in the 3D printer. Then after that insert the USB flash drive and choose the correct file, this file is saved as UFP (UltiMaker Format Package). Then it should just be to start the printing. It will take some time during the heat up of the nozzle and bed, and the auto-level bedding before the printer starts to print the model.

When the 3D-printed model is finished it is good to wait a while to get it cooled down because it is easier to remove the object from the printing bed. The support material on the model is quite easy to remove with the use of example long nose plier and a small-sized screwdriver. Sandpaper was used to make the surface smooth, firstly use size 240 to get the rougher surface sharpened, and last with size 800 to get the “polished” surface. In Figure 48, the picture to the left shows directly after the model was printed, and the picture to the right when the protection box is finished.

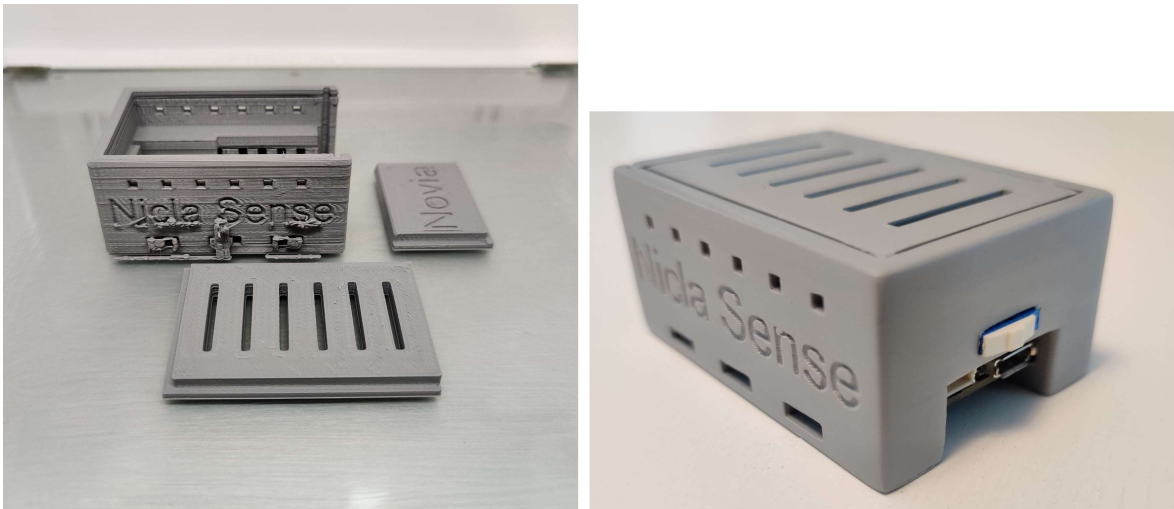


Figure 48. Right: 3D printed box with supports. Left: complete protection box.

7 Discussion and results

This thesis was intended to have wireless measuring equipment that could collect environment and motion sensor data. A small microcontroller was used that had all the necessary sensor readings included in the controller. Other requirements were that the controller would send the data wirelessly and this microcontroller included Bluetooth Low Energy features. The equipment needed for communicating parts was a Raspberry Pi to communicate through Bluetooth to collect data readings from the microcontroller and transmit the data through internet MQTT protocol to the user interface and the other way to send commands from the user interface to the Raspberry Pi and then to the microcontroller.

The thesis work consisted of different results. A thoroughly planned Nicla Sense box was constructed and the sensors were in mind to get airflow directly outside the box. Around the box, air gaps have been created so the air can be changed due to both the battery and the microcontroller getting heated up. At the bottom of the box, holes exist for the built-in reset button and the LED indicator which shows the connection status. A switch can be found above the microcontroller which breaks the power to the controller when not in use and if there are connection issues it is good to shut it off just for a few seconds and power it on again. A small connector designed for the Nicla Sense has been used so it would be easier to disconnect the microcontroller from the battery if necessary. Three

openings on both sides have also been constructed with considering holding the Nicla Sense in place with the use of cable ties. Nicla Sense has also been engraved on both sides of the box to easier see what microcontroller it is inside, and the hatch has been engraved with Novia to indicate who the controller belongs to.

The Nicla Sense microcontroller is pre-configured for the user interface, which means for now it is not possible to change to another Nicla Sense except if not going into the Raspberry Pi's Node-Red configuration and configuring to another one.

The dashboard is constructed so the screen view is adaptable with both computer and smartphone screens.

On the user interface a connect and disconnect button for the connection of the Nicla Sense and the battery status can be found next to it.

At the stage when the battery status part was added to the project it worked as it should, but at a later stage it was noticed that the battery status was only kept between "Battery empty" when the USB cable was used and "Battery Full" when the Nicla Sense was in battery operation. It is difficult to predict what causes this problem but one thought may be the updates that have been made during the work. It was decided to keep the function for further research with the newest Nicla Sense Mbed OS version.

The user interface consists of one part of environment sensors and the second part of motion sensors. It is only possible to choose one of these categories when starting to collect data, but it is possible to choose several sensors from the selected category at one record. When selecting the sensor boxes a sampling time is needed to be chosen in milliseconds, the fastest sampling time that should be used is 250 ms which means 4 samples per second. If selected faster than that some records could be lost or the communication could be lost due to too much traffic.

Below the three buttons, there are four different statuses, one for the dashboard MQTT connection status, and the next is the sensor MQTT connection which means the Raspberry Pi MQTT bridge is up and running. The Bluetooth connection shows the status of the microcontroller connected through Bluetooth to the Raspberry Pi. The last is the battery status. There is also a button up to the right corner which changes the user interface between day and night mode.

During the data collection, the latest received data is shown on the gauges, and graphs that draw the history of the received data. When collecting environment sensor data when some of the boxes are not checked, the value “NaN “will be shown (not a number).

If needed to start over and enter new data records, it is possible to press the button “Clear” which will delete all previous data received and proceed with new data from the Nicla Sense. There is also a “Stop” button that is used when a break is needed during the collection and when pressing start it continues to collect data. When satisfied with the data collected, the “Download” button should be pressed which will download the collected data to the local computer for future processing in a CSV file.

An overview of the user interface can be seen in Appendix IX, which is a user manual in Swedish on how to use this data acquisition tool.

Wi-Fi has a much longer connection distance to be connected to the internet than Bluetooth can be connected between devices, therefore the Raspberry Pi needs to be placed near enough to the Nicla Sense. If needed to collect data outside the school, the Raspberry Pi needs to be taken with and if there is another internet connection outside, the Raspberry Pi needs probably to be connected to that connection. Worth noting is that the Raspberry Pi starts up automatically when receiving power from the 230V outlet and if it finds an accessible internet connection, the device can be used immediately.

For now, a specific dashboard server is not possible due to security risks, which means that the webpage should be developed with security functions, for example, username and password logins.

If a computer runs the dashboard application through the internet at school, the student at home cannot use that host IP address and port to access the Nicla Sense dashboard but if the student has access to the files to run the dashboard on the local computer it is possible to control and receive data from the Nicla Sense at school through the MQTT.

What should be noted is that when the dashboard server is running on a computer at school and connected to the school’s network. The student could connect to the same network and use the host's internet address and port number 4000 and then have access to the Nicla Sense dashboard.

A discovery made was the difference in temperature and humidity measurements when using the battery or USB cable. A longer period of tests has been done with both options of power supplied. In the Arduino code for the Nicla Sense, the temperature and humidity have been adjusted to the battery option, when connected USB cable the temperature measurement rose three Celsius more, and the humidity measurements fell by three percent. The temperature was hard to get a completely correct measurement value due to the heating of the microcontroller in operation. Lastly, it was noted that Co2 measurement was stuck to 500 ppm for the first approximately four to five minutes when the Nicla Sense was switched on, and after that started measuring.

A part of different projects with other hardware in communication to user interface could be found, but usually only one point communication, which means it has only Wi-Fi inbuilt and is directly connected to an MQTT broker, or if the microcontroller has built-in Bluetooth it communicates with just Bluetooth. Since the Nicla Sense microcontroller was released late in the year 2021 there has been limited previous project done but noticed that there are new projects made with this microcontroller gradually.

Using the tool Node-Red has its pros and cons, the positive thing is that it is easy to follow up and get an overview of sketches to see how the data is processed. It is easy to use pre-programmed nodes that can easily be set by their configurations. The downside is there may be no updated nodes that do not work correctly. Another thing is that a prebuilt node is determined by how it works and may not suit the user's needs.

An experiment has also been carried out indoors to check the Bluetooth connection range between the Raspberry Pi and the Nicla Sense. In Figure 49 an overview of the test could be seen. The wireless Bluetooth transmission has a great impact on the material of the walls and this test was performed with plasterboards. The brown mark represents a cabinet which also has the impact of the connection.

In the two nearest rooms to the Raspberry Pi, no problem with the connection was noticed, in the farthest room it worked okay but could sometimes drop the connection.

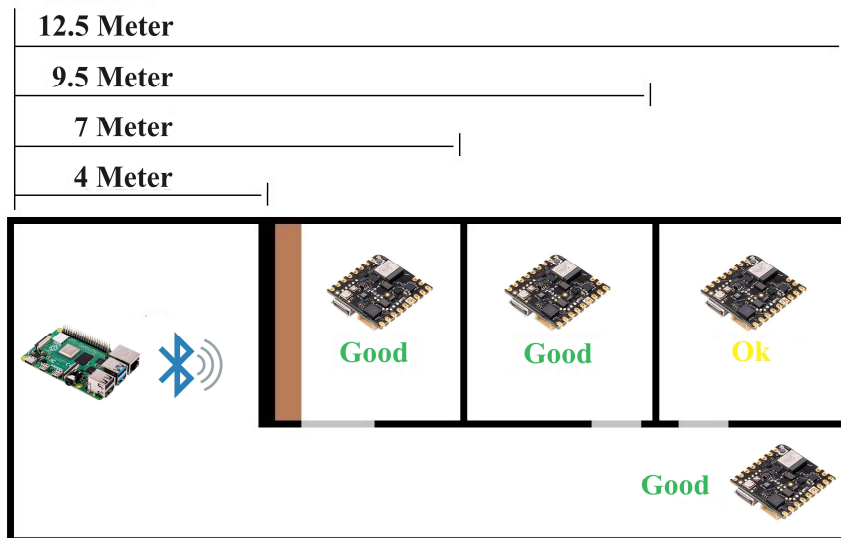


Figure 49. BLE connection range test.

Considering Bluetooth Low Energy, it has exceeded my expectations of connection range when thinking about the wall interference. But as was said before, the wall material has a big impact and does not have to work as well on a concrete wall for example.

A test has been performed by measuring the sensors inside and outside the protection box until the values have stabilized. The result shows that the temperature sensor is a little bit affected by the box, when taking the microcontroller out of the box, the degrees drop by 1.2 Celsius. Worth noting is that the sensor readings have been taken into account in the protection box and not outside.

One thing discovered was that several updates for the Mbed OS Nicla Board were released during the thesis work. When updating to the latest version it will complain about different errors since the Arduino community has changed some code to the Nicla Sense library. The latest version that could be compiled for this project was the version Arduino Mbed OS Nicla Board version 4.0.2.

7.1 Further research

This thesis investigated only one Nicla Sense microcontroller, one suggestion could be the subject of connecting several Nicla Sense microcontrollers into the same user interface so several measurement projects could be handled at the same time.

The accelerometer and gyroscope readings were included in the scope and for now, these sensors could handle four samples per second to the user interface but should be able to transfer much faster. The suggestion could be to do further research on how to do this in other ways to send more samples in one second. Some thought was giving just one command to the Nicla Sense from the user interface, and then the Nicla Sense should bunch all the accelerometer or gyroscope sensor readings for example during 30 seconds, and send it to the user interface. But there should also be a noticeable amount of the Bluetooth Low Energy data limitation.

Another further research suggestion is to see if there is the possibility to use the connect button on the user interface to change to other Nicla Sense microcontrollers because now the specific Nicla Sense controller is pre-programmed and cannot be changed directly from the user interface.

In a later stage, it was noted that the battery status level did not work as it should, and at least it was found that new commands had been added to the Nicla Sense library for the battery which may be the cause. The older Nicla Sense OS version has been uploaded but the lack of time in the end has not made it possible to check this further. The button is left on the application and all the other functions, and a further development may be to update to the latest Nicla Sense OS version and use the new commands to get a working battery status information.

And last suggestion is to check if any other sensors could be included in the user interface (there are also smart functionalities for this sensor).

7.2 Acknowledgments

I have very little coding experience from my background, so this thesis has allowed me to be more familiar with not just one, but three different ways of coding. I am sure that this experience will be needed many times in the future.

Last but not least, I want to thank Hans Lindén, Jan Berglund, and Ray Pörn for the support I have received during this thesis work.

8 References

- A High-Class and Dynamic University of Applied Sciences.* (n.d.). Retrieved from Novia: <https://www.novia.fi/en/about-us/>
- About.* (n.d.). Retrieved from Node-Red: <https://nodered.org/about/>
- About Arduino.* (n.d.). Retrieved February 05, 2018, from Arduino: <https://www.arduino.cc/en/Guide/Introduction>
- About McNeel.* (n.d.). Retrieved from Rhinoceros: <https://www.rhino3d.com/mcneel/about/>
- Afaneh, M. (2018). *Intro to Bluetooth Low Energy*. Novel Bits, LLC.
- Afaneh, M. (n.d.). *Bluetooth GATT: How to Design Custom Services & Characteristics*. Retrieved from NovelBits: Bluetooth GATT: How to Design Custom Services & Characteristics [MIDI device use case]
- Afeneh, M. (2020, April 28). *How Bluetooth Low Energy Works: 21 Interesting Facts*. Retrieved from NovelBits: <https://novelbits.io/how-bluetooth-low-energy-works-21-interesting-facts/>
- American psychology association. (n.d.). *Style and Grammar*. Retrieved January 21, 2022, from APA Style: <https://apastyle.apa.org/style-grammar-guidelines>
- ANNA-B112-00BU-BLOX.* (2021, August 10). Retrieved from TME.eu: https://www.tme.eu/Document/728cd946bd63e740e669ed0616162faf/ANNA-B112_DataSheet.pdf
- Arduino. (2022, December 01). *Arduino*. Retrieved from Nicla Sense ME: <https://store.arduino.cc/products/nicla-sense-me>
- Arduino Nicla Sense ME.* (n.d.). Retrieved 2022, from Bosch Sensortec: <https://www.bosch-sensortec.com/software-tools/tools/arduino-nicla-sense-me/>
- Arduino® Nicla Sense ME.* (2023, October 17). Retrieved from Arduino: <https://docs.arduino.cc/resources/datasheets/ABX00050-datasheet.pdf>
- Bagur, J. (2023, May 16). *Connecting Nano 33 BLE Devices over Bluetooth®*. Retrieved from Arduino: <https://docs.arduino.cc/tutorials/nano-33-ble-sense/ble-device-to-device>
- BHI260AP - Ultra-low power, high performance, self-learning AI smart sensor with integrated accelerometer and gyroscope.* (2021, April 15). Retrieved from Bosch Sensortec: <https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bhi260ap-ds000.pdf>
- BooleanDifference.* (n.d.). Retrieved from Robert McNeel & Associates: <https://docs.mcneel.com/rhino/7/help/en-us/commands/booleanunion.htm#BooleanDifference>

- BooleanUnion*. (n.d.). Retrieved from Robert McNeel & Associates:
<https://docs.mcneel.com/rhino/7/help/en-us/commands/booleanunion.htm>
- Colelevy. (2023, April 6). *querySelector()* vs. *getElementById()*. Retrieved from Dev Community: <https://dev.to/colelevy/querySelector-vs-getelementbyid-166n>
- Connect a battery to Nicla Sense ME or Nicla Vision*. (2022, December 13). Retrieved from Arduino: <https://support.arduino.cc/hc/en-us/articles/4408893476498-Connect-a-battery-to-Nicla-Sense-ME?queryID=undefined>
- Connect to an MQTT Broker*. (n.d.). Retrieved from Node-Red: <https://cookbook.nodered.org/mqtt/connect-to-broker>
- Connolly, R., & Hoar, R. (2018). *FUNDAMENTALS OF Web Development*. 221 River Street, Hoboken, NJ 07030: Pearson Education.
- Cope, S. (2023, October 23). *Understanding and Using Buffers In Node-Red*. Retrieved from Steve's Node-Red Guide: <https://stevesnoderedguide.com/understanding-buffers-node-red>
- Create a single message from separate streams of messages*. (n.d.). Retrieved from Node-Red: <https://cookbook.nodered.org/basic/join-streams>
- Creating Nodes*. (n.d.). Retrieved from Node-Red: <https://nodered.org/docs/creating-nodes/>
- CSS Introduction*. (n.d.). Retrieved from Learn to Code: https://www.w3schools.com/css/css_intro.asp
- Derhgawen, A. (2020, November 16). *Maximizing BLE Throughput Part 4: Everything You Need to Know*. Retrieved from PunchThrough: <https://punchthrough.com/ble-throughput-part-4/>
- Express/Node introduction*. (2023, September 27). Retrieved from Resources for Developers: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction
- Faraon, M., & Holmberg, L. (2022). *Introduktion till HTML och CSS*. Lund : Studentlitteratur AB.
- FilletEdge*. (n.d.). Retrieved from Robert McNeel & Associates:
<https://docs.mcneel.com/rhino/7/help/en-us/commands/filletedge.htm>
- Gas Sensor BME688*. (2022, July). Retrieved from Bosch Sensortec: <https://www.bosch-sensortec.com/products/environmental-sensors/gas-sensors/bme688/#documents>
- Getting Started with Arduino IDE 2.0*. (2022, 12 7). Retrieved from Arduino: <https://docs.arduino.cc/software/ide-v2/tutorials/getting-started-ide-v2>
- Gustafsson, E., & Jarefors, R. (2022). Data collection in IoT: A comparison of MQTT implementations.

- HTML - The Head Element*. (n.d.). Retrieved from W3 Schools - Learn to Code:
https://www.w3schools.com/html/html_head.asp
- HTML <body> Tag*. (n.d.). Retrieved from W3 Schools:
https://www.w3schools.com/tags/tag_body.asp
- HTML <div> Tag*. (n.d.). Retrieved from W3 School - Learn to Code:
https://www.w3schools.com/tags/tag_div.ASP
- HTML Tag*. (n.d.). Retrieved from W3 Schools - Learn to Code:
https://www.w3schools.com/tags/tag_span.asp
- HTML: HyperText Markup Language*. (2023, March 13). Retrieved from Resources for Developers by Developers: <https://developer.mozilla.org/en-US/docs/Web/HTML>
- Jamsa, K., Ph., D., & Klander, L. (1998). *C/C++ PROGRAMMER'S BIBLE, The Ultimate Guide to C/C++ Programming*. 2975 S. Rainbow Blvd., Las Vegas, NV 89102: Jamsa Press.
- Kaur, M. (2020, May 8). *The Difference Between ID and Class*. Retrieved from Medium: <https://medium.com/@mandeepkaur1/the-difference-between-id-and-class-be2a8322b82c>
- Ltd., Raspberry Pi Trading. (2019, June). Raspberry Pi 4 Computer Model B.
- Magnetometer BMM150*. (2020, April). Retrieved from Bosch Sensortec:
<https://www.bosch-sensortec.com/products/motion-sensors/magnetometers-bmm150/#documents>
- MDN contributors. (2023, September 24). *EventTarget: addEventListener() method*. Retrieved from Resources for Developers, by Developers:
<https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/addEventListener>
- MergeAllCoplanarFaces*. (n.d.). Retrieved from Robert McNeel & Associates:
<https://docs.mcneel.com/rhino/7/help/en-us/commands/mergectoplanarface.htm#MergeAllCoplanarFaces>
- Michel, Z. (2019, April 26). *Maximizing BLE Throughput Part 3: Data Length Extension (DLE)*. Retrieved from PunchThrough:
<https://punchthrough.com/maximizing-ble-throughput-part-3-data-length-extension-dle-2/>
- Nicla Sense ME*. (n.d.). Retrieved from Arduino:
<https://docs.arduino.cc/hardware/nicla-sense-me>
- Nicla Sense ME*. (2023, May 16). Retrieved from Arduino:
<https://docs.arduino.cc/hardware/nicla-sense-me>
- Node.js NPM*. (n.d.). Retrieved from W3 schools:
https://www.w3schools.com/nodejs/nodejs_npm.asp
- Node-RED Concepts*. (n.d.). Retrieved from Node-Red:
<https://nodered.org/docs/user->

guide/concepts#:~:text=A%20Node%20is%20the%20basic,timer%20or%20GPIO%20hardware%20change.

node-red-contrib-buffer-parser. (n.d.). Retrieved from Node-Red:

<https://flows.nodered.org/node/node-red-contrib-buffer-parser>

node-red-contrib-generic-ble. (n.d.). Retrieved from Node-Red:

<https://flows.nodered.org/node/node-red-contrib-generic-ble>

OSI Model. (n.d.). Retrieved from Imperva:

<https://www.imperva.com/learn/application-security/osi-model/>

Pressure sensor BMP390. (2021, March). Retrieved from Bosch Sensortec:

<https://www.bosch-sensortec.com/products/environmental-sensors/pressure-sensors/bmp390/#documents>

Product logos. (n.d.). Retrieved from Novia: <https://www.novia.fi/en/about-us/graphic-profile/logo-and-seal/product-logos>

Publish messages to a topic. (n.d.). Retrieved from Node-Red:

<https://cookbook.nodered.org/mqtt/publish-to-topic>

Raspberry Pi 4 Tech Specs. (n.d.). Retrieved from Raspberry Pi:

<https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/>

Raspberry Pi OS. (n.d.). Retrieved from Raspberry Pi:

<https://www.raspberrypi.com/documentation/computers/os.html>

Ravikiran , A. (2022, December 12). *The Difference Between C and C++*. Retrieved from

simplilearn: https://www.simplilearn.com/tutorials/cpp-tutorial/difference-between-c-and-cpp#difference_between_c_and_c

Rhinoceros. (n.d.). Retrieved from Rhinoceros: <https://www.rhino3d.com/>

Rhinoceros 3D. (2023, April 13). Retrieved from Wikipedia:

https://en.wikipedia.org/wiki/Rhinoceros_3D

Romero, S. (n.d.). *Arduino Nicla Sense ME Cheat Sheet*. Retrieved from Arduino:

<https://docs.arduino.cc/tutorials/nicla-sense-me/cheat-sheet>

Running on Raspberry Pi. (n.d.). Retrieved from Node-Red:

<https://nodered.org/docs/getting-started/raspberrypi#installing-and-upgrading-node-red>

Science & Technology. (2019, June 4). Retrieved from Congressional Research Service:

<https://crsreports.congress.gov>

Sensing and intelligence at the edge become accessible to all, with Nicla Sense ME by

Arduino Pro and Bosch Sensortec. (n.d.). Retrieved from Bosch Sensortec:

<https://www.bosch-sensortec.com/news/sensing-and-intelligence-at-the-edge-become-accessible-to-all-with-nicla-sense-me-by-arduino-pro-and-bosch-sensortec.html>

SSDN Technologies. (2023, May 11). *Advantage & Disadvantages of C++ Programming*. Retrieved from Medium: <https://ssdntechnologies.medium.com/advantage-disadvantages-of-c-programming-dc1eb9af2d72>

Subscribe to a topic. (n.d.). Retrieved from Node-Red: <https://cookbook.nodered.org/mqtt/subscribe-to-topic>

Sufiyan, T. (2023, May 16). *What is Node.js: A Comprehensive Guide*. Retrieved from Simplilearn: https://www.simplilearn.com/tutorials/nodejs-tutorial/what-is-nodejs#what_is_nodejs

Supported File Formats. (n.d.). Retrieved from Rhinoceros: <https://www.rhino3d.com/features/file-formats/>

System Requirements. (n.d.). Retrieved from Rhinoceros: <https://www.rhino3d.com/7/system-requirements/>

Technical education with a royal lineage. (n.d.). Retrieved from Technobothnia: <https://www.technobothnia.fi/technical-collaboration/about-technobothnia/>

Technobothnia. (n.d.). Retrieved from Technobothnia: <https://www.technobothnia.fi/>

The C++ programming language in cheminformatics and computational chemistry. (2020). *Journal of Cheminformatics*, 1.

The Core Nodes. (n.d.). Retrieved from Node-Red: <https://nodered.org/docs/user-guide/nodes#inject>

The History of Rhino. (2020, December 4). Retrieved from Robert McNeel & Associates Wiki: <https://wiki.mcneel.com/rhino/rhinohistory>

The Raspberry Pi: A Technology Disrupter, and the Enabler of Dreams. (2017, July 12). Southampton, United Kingdom: mdpi.

Townsend, K., Cufi, C., Akiba, & Davidson, R. (2014). *Getting Started with Bluetooth Low Energy*. 1005 Gravenstein Highway Nort, Sebastopol, CA 95472: O'Reilly Media, Inc.

UltiMaker Cura. (n.d.). Retrieved from UltiMaker: <https://ultimaker.com/software/ultimaker-cura/>

Units of measurement. (2023, July 25). Retrieved from elementor: <https://elementor.com/help/whats-the-difference-between-px-em-rem-vw-and-vh/>

Use EJS as Template Engine in Node.js. (n.d.). Retrieved from geeksforgeeks: Use EJS as Template Engine in Node.js

What are NURBS? (n.d.). Retrieved from Rhinoceros: <https://www.rhino3d.com/features/nurbs/>

What are the system requirements for Ultimaker Cura? (n.d.). Retrieved from UltiMaker: https://support.makerbot.com/s/article/1667410778209?_gl=1*6m2m7z*_ga*MzYwOTAxMTIyLjE2ODMyMDI0Njk.*_ga_JHX8W909G8*MTY4MzI3ODAwNi

4yLjEuMTY4MzI3ODU2Mi41OS4wLjA.*_ga_CJM2DTBWYF*MTY4MzI3ODAwNi
4yLjEuMTY4MzI3ODU2Mi41OS4wLjA.

What Is IoT? (2021, June 16). Retrieved from Renke: <https://www.renkeer.com/iot-sensors-applications/>

What is JavaScript? (2023, March 05). Retrieved from Resources for Developers: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript

Woolley, M. (2022, June 6). *Introducing: The Bluetooth*. Retrieved from Bluetooth: <https://www.bluetooth.com/blog/introducing-the-bluetooth-low-energy-primer/>

Zakas, N. C. (2009, July 21). *Introduction to sessionStorage*. Retrieved from Human Who Codes: <https://humanwhocodes.com/blog/2009/07/21/introduction-to-sessionstorage/>

APPENDICES

Appendices with a lot of code that is too long and too extensive to be included in the main text will be visible here, these codes will be attached below with different references. There is also one appendix included of the user manual in Swedish.

Appendix I – Arduino Code

```
/* Arduino Nicla Sense ME BLE Sense Dashboard */

// Include outside libraries in this sketch.
#include "Nicla_System.h"
#include "Arduino BHY2.h"
#include <ArduinoBLE.h>

// Define gives a name to a constant value before the program is compiled.
#define ACC_SENSOR_UPDATE_INTERVAL      (1000)
#define GYRO_SENSOR_UPDATE_INTERVAL    (1000)

// Store different data types in a single definition
union sensor_data {
  struct __attribute__((packed)) {
    float values[3]; // float array for data (it holds 3)
    bool updated = false;
  };
  uint8_t bytes[3 * sizeof(float)]; // size as byte array
};

union sensor_data accData;
union sensor_data gyroData;

// Environment UUIDS
BLEService Env_service("181a"); // Service UUID for Environment sensors.
// Pre-designed UUID for temperature: 2a6e with the properties of read.
BLEFloatCharacteristic temperatureCharacteristic("2a6e", BLERead);
// Pre-designed UUID for humidity: 2a6f with the properties of read.
BLEUnsignedIntCharacteristic humidityCharacteristic("2a6f", BLERead);
// Pre-designed UUID for pressure: 2aa3 with the properties of read.
BLEFloatCharacteristic pressureCharacteristic("2a6d", BLERead);
// Custom-designed UUID for gas: 19b10000-1001-537e-4f6c-d104768a1214 with the properties of read.
BLEUnsignedIntCharacteristic gasCharacteristic("a7f5a4f3-1001-4941-bc75-6d7f5d27f436", BLERead);
// Custom-designed UUID for co2: 19b10000-1002-537e-4f6c-d104768a1214 with the properties of read.
BLEIntCharacteristic co2Characteristic("a7f5a4f3-1002-4941-bc75-6d7f5d27f436", BLERead);

// Motion UUIDS
BLEService Motion_service("03505D7C-0000-4997-AD89-CD1456DC7490"); // Service UUID for Motion sensors.
```

```

// Custom UUID for accelerometer and gyroscope sensors with properties of read and notify.
BLECharacteristic accCharacteristic("03505d7c-1001-4997-ad89-cd1456dc7490", BLERead | BLENotify,
sizeof accData.bytes);
BLECharacteristic gyroCharacteristic("03505d7c-1002-4997-ad89-cd1456dc7490", BLERead | BLENotify,
sizeof accData.bytes);

// Battery level UUID
BLEService Battery_service("180f"); // Service UUID for Battery level.
// Custom UUID for Battery level with properties of read.
BLEUnsignedIntCharacteristic batterylvlCharacteristic("2a19", BLERead);

// Get (Class | sensor name | Sensor Id Macro) of the needed sensors from Nicla Sense
Sensor temperature(SENSOR_ID_TEMP);
Sensor humidity(SENSOR_ID_HUM);
Sensor gas(SENSOR_ID_GAS);
Sensor pressure(SENSOR_ID_BARO);
SensorBSEC bsec(SENSOR_ID_BSEC);
SensorXYZ accelerometer(SENSOR_ID_ACC);
SensorXYZ gyroscope(SENSOR_ID_GYRO);

// Setup required at the beginning of the program.
void setup() {
  Serial.begin(115200); // Sets the data rate in bits per second (baud rate) for serial monitor
  Serial.println("Start"); // Prints "Start" when compiling is finished

  // Start Nicla Sense and set the led indicator colour to red.
  nicla::begin();
  nicla::leds.begin();
  nicla::leds.setColor(red);

  pinMode(p25, OUTPUT); // Configure the specified pin as an output.

  nicla::enableCharge(100); // Enable charge function of 100 mA

  //Sensors initialization
  BHY2.begin();
  temperature.begin();
  humidity.begin();
  gas.begin();
  pressure.begin();
  bsec.begin();
  accelerometer.begin();
  gyroscope.begin();

  // write initial value
  for (int i = 0; i < 3; i++) {
    accData.values[i] = i;
  }

```



```

    gyroData.values[i] = i;
}

// Set the advertised service UUID used when advertising to the value of the BLEservice provided.
BLE.setAdvertisedService(Env_service);
BLE.setAdvertisedService(Motion_service);
BLE.setAdvertisedService(Battery_service);

// Add all the previously defined Characteristics to the specific services
Env_service.addCharacteristic(temperatureCharacteristic);
Env_service.addCharacteristic(humidityCharacteristic);
Env_service.addCharacteristic(gasCharacteristic);
Env_service.addCharacteristic(pressureCharacteristic);
Env_service.addCharacteristic(co2Characteristic);

Motion_service.addCharacteristic(accCharacteristic);
Motion_service.addCharacteristic(gyroCharacteristic);

Battery_service.addCharacteristic(batterylvlCharacteristic);

// Connect event handler
BLE.setEventHandler(BLEConnected, blePeripheralConnectHandler);

// Disconnect event handler
BLE.setEventHandler(BLEDisconnected, blePeripheralDisconnectHandler);

// Sensors event handlers
temperatureCharacteristic.setEventHandler(BLERead, onTemperatureCharacteristicRead);
humidityCharacteristic.setEventHandler(BLERead, onHumidityCharacteristicRead);
gasCharacteristic.setEventHandler(BLERead, onGasCharacteristicRead);
pressureCharacteristic.setEventHandler(BLERead, onPressureCharacteristicRead);
co2Characteristic.setEventHandler(BLERead, onCo2CharacteristicRead);
batterylvlCharacteristic.setEventHandler(BLERead, onBatterylvlCharacteristicRead);

BLE.setLocalName("Nicla"); // Add bluetooth name to the device
BLE.addService(Env_service); // Add a BLE service to the set of services the BLE device provides.
BLE.addService(Motion_service); // Add a BLE service to the set of services the BLE device provides.
BLE.addService(Battery_service); // Add a BLE service to the set of services the BLE device provides.
BLE.advertise(); // Start advertising.
}

// When device connected change led to green.
void blePeripheralConnectHandler(BLEDevice central){
    nicla::leds.setColor(green);
}

// After the device connected once and after that disconnects the led changes to blue.
void blePeripheralDisconnectHandler(BLEDevice central) {
    nicla::leds.setColor(red);
}

```

```

}

// Loops consecutively, allowing the program to change and respond.
void loop(){

  // Activate tasks while BLE is connected.
  while (BLE.connected()){
    BHY2.update();

    bleAccTask();
    if (accSensorTask()) {
      }
    bleGyroTask();
    if (gyroSensorTask()) {
      }
    }
  }

  // Bool holds one of two values (true or false).
  bool accSensorTask() {
    static long previousMillis2 = 0;
    unsigned long currentMillis2 = millis(); // Returns the number of milliseconds passed since the board began
    running the current program.
    static float x = 0.00, y = 0.00, z = 0.00;
    if (currentMillis2 - previousMillis2 < ACC_SENSOR_UPDATE_INTERVAL) {
      return false;
    }
    previousMillis2 = currentMillis2; // Writing over the time to currentMillis2.
    if(accelerometer.begin()){
      float x, y, z;
      x = accelerometer.x();
      y = accelerometer.y();
      z = accelerometer.z();

      accData.values[0] = x;
      accData.values[1] = y;
      accData.values[2] = z;
      accData.updated = true;
    }
    return accData.updated;
  }

  // Same procedure as accSensorTask.
  bool gyroSensorTask() {
    static long previousMillis2 = 0;
    unsigned long currentMillis2 = millis();
    static float x = 0.00, y = 0.00, z = 0.00;
    if (currentMillis2 - previousMillis2 < GYRO_SENSOR_UPDATE_INTERVAL) {
      return false;
    }
    previousMillis2 = currentMillis2;

```

```

if(gyroscope.begin()){
    float x, y, z;
    x = gyroscope.x();
    y = gyroscope.y();
    z = gyroscope.z();

    gyroData.values[0] = x;
    gyroData.values[1] = y;
    gyroData.values[2] = z;
    gyroData.updated = true;
}
return gyroData.updated;
}

// Request read and write value from sensor.
void onTemperatureCharacteristicRead(BLEDevice central, BLECharacteristic characteristic){
    float temperatureRaw = temperature.value(); // Read raw data temperature value.
    temperatureCharacteristic.writeValue(bsec.comp_t()); // Write and send compensated temperature value.
    Serial.println(String("Temperature Raw: ") + String(temperatureRaw)); // Print raw data temperature to
    serial monitor.
    Serial.println(String("Temperature Compensated: ") + String(bsec.comp_t())); // Print compensated
    temperature value
}

void onHumidityCharacteristicRead(BLEDevice central, BLECharacteristic characteristic){
    float humidityRaw = humidity.value(); // Read raw data humidity value.
    humidityCharacteristic.writeValue(bsec.comp_h()); // Write and send compensated humidity value.
    Serial.println(String("Humidity Raw: ") + String(humidityRaw)); // Print raw data humidity to serial monitor.
    Serial.println(String("HumidityComp Compensated: ") + String(bsec.comp_h())); // Print compensated
    humidity value
}

void onGasCharacteristicRead(BLEDevice central, BLECharacteristic characteristic){
    unsigned int gasValue = gas.value(); // Read gas value uint.
    gasCharacteristic.writeValue(gasValue); // Write and send gas value
    Serial.println(String("Gas: ") + String(gasValue));
}

void onPressureCharacteristicRead(BLEDevice central, BLECharacteristic characteristic){
    float pressureValue = pressure.value(); // Read float pressure value.
    pressureCharacteristic.writeValue(pressureValue); // Write and send pressure value.
    Serial.println(String("Pressure: ") + String(pressureValue));
}

void onCo2CharacteristicRead(BLEDevice central, BLECharacteristic characteristic){
    uint32_t co2Value = bsec.co2_eq(); // Read uint co2 value.
    co2Characteristic.writeValue(co2Value); // Write and send co2 value.
    Serial.println(String("Co2: ") + String(co2Value));
}

void onBatterylvlCharacteristicRead(BLEDevice central, BLECharacteristic characteristic){

```

```

digitalWrite(p25, HIGH); // Write high value to digital pin.
static int batteryValue = nicla::getBatteryStatus(); // Get the current battery status.
batteryIvCharacteristic.writeValue(batteryValue); // Write and send battery value.
digitalWrite(p25, LOW); // Write low value to digital pin.
}

```

```

void bleAccTask()
{
    const uint32_t BLE_UPDATE_INTERVAL = 1000;
    static uint32_t previousMillis = 0;
    uint32_t currentMillis = millis();
    if (currentMillis - previousMillis >= BLE_UPDATE_INTERVAL) {
        previousMillis = currentMillis;
        BLE.poll();
    }
    if (accData.updated) {
        // When bluetooth connected update values and write on request
        int16_t accelerometer_X = round(accData.values[0]);
        int16_t accelerometer_Y = round(accData.values[1]);
        int16_t accelerometer_Z = round(accData.values[2]);
        accCharacteristic.writeValue(accData.bytes, sizeof accData.bytes);
        int16_t helpme = (accData.bytes, sizeof accData.bytes);

        Serial.println(String(accelerometer_X));
        accData.updated = false;
    }
}

```

```

void bleGyroTask()
{
    const uint32_t BLE_UPDATE_INTERVAL = 1000;
    static uint32_t previousMillis = 0;
    uint32_t currentMillis = millis();
    if (currentMillis - previousMillis >= BLE_UPDATE_INTERVAL) {
        previousMillis = currentMillis;
        BLE.poll();
    }
    if (gyroData.updated) {
        // When bluetooth connected update values and write on request
        int16_t gyroscope_X = round(gyroData.values[0]);
        int16_t gyroscope_Y = round(gyroData.values[1]);
        int16_t gyroscope_Z = round(gyroData.values[2]);
        gyroCharacteristic.writeValue(gyroData.bytes, sizeof gyroData.bytes);
        gyroData.updated = false;
    }
}

```

Appendix II – dashboard.ejs

```
<!DOCTYPE html>
<html lang="en">
<head>
<!-- Defines metadata (information) about the HTML document -->
<meta charset="UTF-8" />
<meta http-equiv="X-UA-Compatible" content="IE=edge" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<title><%=dashboardTitle%></title>
<link
href="https://fonts.googleapis.com/icon?family=Material+Symbols+Sharp"
rel="stylesheet"
/>
<!-- Scripts points to external script files -->
<script src="https://cdn.plot.ly/plotly-2.16.1.min.js"></script>
<script src="https://unpkg.com/mqtt/dist/mqtt.min.js"></script>
<script src = "https://code.jquery.com/jquery-2.1.4.min.js" ></script>
<!-- Import style.css document -->
<link rel="stylesheet" href="/style.css" />
</head>
<!-- Body element defines the document's body.-->
<body>
<div class="container">
<!-- Start of contents on the left side-->
<aside>
<div class="top">
<div class="logo">

<h2><%=name%></h2>
</div>
<div class="close" id="close-btn">
<span class="material-symbols-sharp"> close </span>
</div>
</div>
<div class="sidebar">
<br /><br />

<div class="insights">
<br /><br /><br />
<h3><b>Environment Sensors:</b></h3>
<div class="temperature">
<div class="middle">
<div class="left">
<div class="icon">
<span class="material-symbols-sharp"> device_thermostat </span> <b>Temperature</b>
</div>
<h2 id="temperature"></h2>
</div>
</div>
</div>
<!-- End of temperature -->
<br />
<div class="humidity">
<div class="middle">
<div class="left">
</div>
</div>
<div class="icon">
```

```

<span class="material-symbols-sharp"> humidity_percentage </span> <b>Humidity</b>
</div>
<h2 id="humidity"></h2>
</div>
</div>
<!-- End of humidity -->
<br />
<div class="pressure">
<div class="middle">
<div class="left">
</div>
<div class="icon">
<span class="material-symbols-sharp"> partly_cloudy_day </span> <b>Pressure</b>
</div>
<h2 id="pressure"></h2>
</div>
</div>
<!-- End of barometer -->
<br />
<div class="co2">
<div class="middle">
<div class="left">
</div>
<div class="icon">
<span class="material-symbols-sharp"> speed </span> <b>Co2</b>
</div>
<h2 id="co2"></h2>
</div>
</div>
<!-- End of Co2 -->
<br />
<div class="gas">
<div class="middle">
<div class="left">
</div>
<div class="icon">
<span class="material-symbols-sharp"> gas_meter </span> <b>Gas</b>
</div>
<h2 id="gas"></h2>
</div>
</div>
<!-- End of gas -->
<br />
<h3><b>Motion Sensors:</b></h3>
<div class="accelerometer">
<div class="middle">
<div class="left">
</div>
<div class="icon">
<span class="material-symbols-sharp"> open_with </span> <b>Accelerometer</b>
</div>
<h2 id="accelerometer"></h2>
</div>
</div>
<!-- End of Accelerometer -->
<br />
<div class="gyroscope">
<div class="middle">
<div class="left">

```

```

</div>
<div class="icon">
<span class="material-symbols-sharp">explore</span> <b>Gyroscope</b>
</div>
<h2 id="gyroscope"></h2>
</div>
</div>
<!-- End of Gyroscope -->
</div>
</aside>
<!-- End of contents on the left side-->
<!-- Start of contents on center-->
<main>
<h1><%=dashboardTitle%></h1>
<!-- Start of contents on the top of right side-->
<div class="right">
<div class="top">
<button id="menu-btn">
<span class="material-symbols-sharp"> menu </span>
</button>
<div class="theme-toggler">
<span class="material-symbols-sharp active"> light_mode </span>
<span class="material-symbols-sharp"> dark_mode </span>
</div>
</div>
<!-- End of top -->
<div class="inline">
<div>
<div class="bluetooth-btn">
<button id="BLEconnect-btn">Connect</button>
<button id="BLEdisconnect-btn">Disconnect</button>
<button id="Batterystatus-btn"><font size="0,1">Battery Status<br>(not in use)</font></button>
</div>
<table id="connection-status" border="1">
<tr><h3><td><center><center>Description:</center></td> <td><center>Status:</center></td></h3></tr>
<tr><h3><td>Dashboard MQTT Connection:</td> <td><center><span
class="status">Disconnected</center></span></td></h3></tr>
<tr><h3><td>Sensor MQTT Connection:</td> <td><center><span
class="BLEMQTTstatus">Unknown</center></span></td></h3></tr>
<tr><h3><td>Bluetooth Connection:</td> <td><center><span
class="BLEstatus">Unknown</center></span></td></h3></tr>
<tr><h3><td>Battery Status - Updated: <span class="NiclaBatterystatusTime"></span></td>
<td><center><span class="NiclaBatterystatus">Unknown</center></span></td></h3></tr>
</table>
</div>
<div class="Env">
<p><b>Environment Sensors</b></p>
<table id="Environment" border="1">
<div class="Sensor_Select">
<div class="Time_Interval">
<tr><td>
<label id="TimeInterval" for="TimeInterval">Sampling Time (ms)</label><br>
<input type="text" id="Time_Interval_Env" value=>
</div>
<br>
<!-- Checkboxes-->
<div id="tblSensors" type="checkbox">
<input id="envCheck1" type="checkbox" value="2a6e"/><label>Temperature</label>
<br>

```



```
</div>
</div>
</main>
<!-- End of contents on center-->
</div>
</div>
<script type="module" src="./index.js"></script>
<script type="module" src="./mqttService.js"></script>
<script type="text/javascript"></script>
</body>
</html>
```

Appendix III – Style.CSS

```
/*Style dashboard*/
@import
url("https://fonts.googleapis.com/css2?family=Poppins:wght@300;400;500;600;700;800&display=swap");

:root {
  --color-primary: #7380ec;
  --color-danger: #ff7782;
  --color-success: #41f1b6;
  --color-warning: #ffbb55;
  --color-white: #fff;
  --color-info-dark: #7d8da1;
  --color-info-light: #dce1eb;
  --color-dark: #363949;
  --color-light: rgba(132, 139, 200, 0.18);
  --color-primary-variant: #111e88;
  --color-dark-variant: #636363;
  --color-background: #f6f6f9;

  /* Color of measurement pictures*/
  --color-insight-1: rgb(99, 209, 35);
  --color-insight-2: rgb(233, 245, 59);
  --color-insight-3: rgb(255, 21, 1);
  --color-insight-4: rgb(56, 183, 238);
  --color-insight-5: rgb(255, 62, 150);
  --color-insight-6: rgb(238, 159, 238);
  --color-insight-7: rgb(211, 211, 211);

  --card-border-radius: 2rem;
  --border-radius-1: 0.4rem;
  --border-radius-2: 0.8rem;
  --border-radius-3: 1.2rem;

  --card-padding: 1.8rem;
  --padding-1: 1.2rem;

  --box-shadow: 0 2rem 3rem var(--color-light);

  /* Plotly Chart Color */
  --chart-background: #fff;
  --chart-font-color: #444;
  --chart-axis-color: #444;
  --color-commands: #e5e4e2;
}

/* Dark theme color variables */
.dark-theme-variables {
  --color-background: #090d3e;
  --color-white: #0b0f4a;
  --color-primary: #fff;
  --color-dark: #edeffd;
  --color-dark-variant: #fff;
  --color-light: rgba(0, 0, 0, 0.4);
  --box-shadow: 0 2rem 3rem var(--color-light);

  --chart-background: #0d1256;
  --chart-font-color: #fff;
}
```

```
--chart-axis-color: #fff;
--color-commands: #D3D3D3;

}

/* Select of all element */
* {
  margin: 0;
  padding: 0;
  outline: 0;
  text-decoration: none;
  list-style: none;
  box-sizing: border-box;
}

html {
  font-size: 14px;
}

/* Define the document's body */
body {
  width: 100vw;
  height: 100vh;
  font-family: poppins, san-serif;
  font-size: 0.88rem;
  background: var(--color-background);
  user-select: none;
  overflow-x: hidden;
  color: var(--color-dark-variant);
}

/* Settings on class container */
.container {
  display: grid;
  width: 96%;
  margin: 0 auto;
  gap: 1.8rem;
  grid-template-columns: 14rem auto 30rem;
}

a {
  color: var(--color-dark);
}

/* Images settings */
img {
  display: block;
  width: 100%;
}

/* Headings */
h1 {
  font-weight: 800;
  font-size: 1.8rem;
}

h2 {
  font-size: 1.4rem;
}
```

```
h3 {
  font-size: 0.87rem;
}

/* Color info */
p {
  color: var(--color-dark-variant);
}

b {
  color: var(--color-dark-variant);
}

/** Sidebar **/
aside {
  height: 100vh;
}

aside .top {
  display: flex;
  align-items: center;
  justify-content: space-between;
  margin-top: 1.4rem;
}

aside .logo {
  display: flex;
  gap: 2rem;
}

aside .logo img {
  width: 6rem;
  height: 6rem;
}

aside .close {
  display: none;
}

/***** Left sidebar *****/
aside .sidebar {
  display: flex;
  flex-direction: column;
  height: 86vh;
  position: relative;
  top: 3rem;
}

aside h3 {
  font-weight: 500;
}

aside .sidebar a {
  display: flex;
  color: var(--color-info-dark);
  margin-left: 2rem;
  gap: 1rem;
  align-items: center;
```

```
position: relative;
height: 3.7rem;
transition: all 300ms ease;
}

aside .sidebar a span {
font-size: 1.6rem;
transition: all 300ms ease;
}

aside .sidebar a:hover {
color: var(--color-primary);
}

aside .sidebar a:hover span {
margin-left: 1rem;
}

/** Main tag */
main {
margin-top: 1.5rem;
}

/* Insights Measurement Gagues */
aside .insights > div {
background: var(--color-white);
padding: var(--card-padding);
border-radius: var(--card-border-radius);
margin-top: 1rem;
box-shadow: var(--box-shadow);
transition: all 300ms ease;
}

aside .insights > div:hover {
box-shadow: none;
}

aside .insights > div span {
background: var(--color-primary);
padding: 0.1rem;
border-radius: 60%;
color: var(--color-white);
font-size: 1.5rem;
}

/* Insights different colors*/
aside .insights > div.temperature span {
background: var(--color-insight-1);
}

aside .insights > div.humidity span {
background: var(--color-insight-2);
}

aside .insights > div.co2 span {
background: var(--color-insight-3);
}
```

```

aside .insights > div.gas span {
  background: var(--color-insight-4);
}

aside .insights > div.pressure span {
  background: var(--color-insight-5);
}

aside .insights > div.accelerometer span {
  background: var(--color-insight-6);
}

aside .insights > div.gyroscope span {
  background: var(--color-insight-7);
}

/* Font size of heading for gauges and margin */
aside .insights h3 {
  margin: 1rem 0 0.6rem;
  font-size: 1rem;
}

/** End of Insights */

/** History Charts */
main .histories {
  margin-top: 2rem;
}

main .history-charts {
  display: grid;
  grid-template-columns: repeat(2, 1fr);
  gap: 2.5rem;
  background: var(--color-white);
  border-radius: var(--border-radius-1);
  padding: var(--card-padding);
  text-align: center;
  box-shadow: var(--box-shadow);
}

main .history-charts:hover {
  box-shadow: none;
}

main .history-charts .history-divs {
  text-align: center;
}

main .histories h2 {
  margin-bottom: 0.8rem;
}

/** The class RIGHT */
.right {
  margin-top: 1.4rem;
}

.right .top {

```

```

display: flex;
justify-content: end;
gap: 2rem;
}

.right .top button {
display: none;
}

.right .theme-toggler {
background: var(--color-light);
display: flex;
justify-content: space-between;
align-items: center;
height: 1.6rem;
width: 4.2rem;
cursor: pointer;
border-radius: var(--border-radius-1);
}

.right .theme-toggler span {
font-size: 1.2rem;
width: 50%;
height: 100%;
display: flex;
align-items: center;
justify-content: center;
}

.right .theme-toggler span.active {
background: var(--color-primary);
color: white;
border-radius: var(--border-radius-1);
}

/* MEDIA QUERIES (decision based on display screen)*/
@media screen and (max-width: 1200px) {
aside .container {
width: 94%;
grid-template-columns: 7rem auto 23rem;
}
aside .logo h2 {
display: none;
}

aside .sidebar h3 {
display: none;
}
aside .sidebar a {
width: 5.6rem;
}
aside .sidebar a:last-child {
position: relative;
margin-top: 1.8rem;
}
main .insights {
grid-template-columns: 1fr;
}
main .histories {

```

```

width: 94%;
position: absolute;
left: 50%;
transform: translateX(-50%);
margin: 2rem 0 0 8.8rem;
}
main .histories .history-charts {
grid-template-columns: 1fr;
width: 54vw;
}
}

@media only screen and (max-width: 992px) {
.container {
width: 94%;
grid-template-columns: 12rem auto 23rem;
}
main .insights {
grid-template-columns: repeat(2, 1fr);
gap: 1.6rem;
}
main .histories .history-charts {
grid-template-columns: 1fr;
align-items: center;
justify-content: center;
}
}

@media screen and (max-width: 768px) {
.container {
width: 100%;
grid-template-columns: 1fr;
}
aside {
position: fixed;
left: -100%;
background: var(--color-white);
width: 18rem;
z-index: 3;
box-shadow: 1rem 3rem 4rem var(--color-light);
height: 150vh;
padding-right: var(--card-padding);
display: none;
animation: showMenu 350ms ease forwards;
}
@keyframes showMenu {
to {
left: 0;
}
}
img {
width: 80%;
}
aside .logo {
margin-left: 1rem;
}
aside .logo h2 {
display: inline;
}
}

```



```
aside .sidebar h3 {
  display: inline;
}
aside .sidebar a {
  width: 100%;
  height: 3.4rem;
}

aside .close {
  display: inline-block;
  cursor: pointer;
}
main {
  margin: 8rem 2rem 2rem 2rem;
  padding: 0 1rem;
}
main .histories {
  position: relative;
  margin: 3rem 0 0 0;
  width: 100%;
}
main .histories .history-charts {
  width: 140%;
  justify-content: right;
  align-items: right;
  display: flex;
  flex-direction: column;
  justify-content: right;
  align-items: right;
}
.right {
  width: 90%;
  margin: 0 auto 0rem auto;
}
.right .top {
  position: fixed;
  top: 0;
  left: 0;
  align-items: center;
  padding: 0 0.8rem;
  height: 5rem;
  background: var(--color-white);
  width: 100%;
  margin: 0;
  z-index: 2;
  box-shadow: 0 1rem 1 rem var(--color-light);
}
.right .top .theme-toggler {
  width: 4.4rem;
  position: absolute;
  right: 2rem;
}

.right .top button {
  display: inline-block;
  background: transparent;
  cursor: pointer;
  color: var(--color-dark);
  position: absolute;
```

```
    left: 1rem;
  }
  .right .top button span {
    font-size: 2rem;
  }
}

@media screen and (max-width: 600px) {
  .container {
    width: 150%;
    grid-template-columns: 1fr;
    margin: 1rem 0 1rem 0;
  }
  main {
    margin: 5rem 1rem 1rem 1rem;
    padding: 0 1rem;
    width: 90vw;
  }

  main .insights {
    gap: 0.4rem;
  }
  main .insights > div {
    padding: 0.4rem;
  }

  main .history-charts {
    display: grid;
    grid-template-columns: 1fr;
  }

  h1 {
    font-weight: 800;
    font-size: 1.4rem;
  }

  #Start_command{
    margin-left: -100px;
  }

  #Stop_command{
    margin-left: -100px;
  }

  #DownloadCSV-btn{
    margin-left: -100px;
  }

  #clearButton{
    margin-left: -100px;
  }

  .Env{
    width: 100%;
    margin: 1rem -1rem 0rem 1rem;
  }

  .Motion{
    width: 20%;
  }
}
```

```
margin: 19rem 6rem 0rem 2rem;  
}
```

```
#connection-status{  
font-weight: bold;  
table-layout: auto;  
width: 130%;  
margin-top: 60px;  
}
```

```
#BLEconnect-btn{  
background: lightgreen;  
margin: 3rem 0rem 0rem 0rem;  
}
```

```
#BLEdisconnect-btn{  
background: red;  
margin: 1rem 0rem 0rem 0rem;  
}
```

```
#Batterystatus-btn{  
background: lightblue;  
margin: 1rem 0rem 0rem 0rem;  
}
```

```
table {  
width: 140px;  
}
```

```
}
```

```
button{  
border: none;  
padding-top: 10px;  
padding-bottom: 10px;  
color: black;  
font-weight: bold;  
width: 100px;  
margin-bottom: 15px;  
border-radius: 50px;  
cursor: pointer;  
}
```

```
#BLEconnect-btn{  
background: lightgreen;  
}
```

```
#BLEdisconnect-btn{  
background: red;  
}
```

```
#Batterystatus-btn{  
background: lightblue;  
right: 50px;  
}
```

```
#connection-status{  
font-weight: bold;  
table-layout: auto;
```

```
width: 130%;
}

.inline{
display: flex;
}

.control-btn{
justify-content: center;
Position:relative;
right: -40px;
top: 40px;
}

#Start_command{
background: var(--color-commands);
}

#Stop_command{
background: var(--color-commands);
}

#DownloadCSV-btn{
background: var(--color-commands);
}

#clearButton{
background: var(--color-commands);
}

.Time_Interval{
padding: 7px 0;
display: inline;
justify-content: center;
font-weight: bold;
}

#tblSensors{
Position:relative;
right: -10px;
}

#motion_tblSensors{
Position:relative;
right: -10px;
}

input[label] {
width: 100px;
}

label {
padding-left: 5px;
text-align: center;
}

input[type="text"] {
width: 80px;
Position:relative;
```

```
right: -25px;  
}
```

```
.Sensor_Select{  
  Position:relative;  
  right: -200px;  
}
```

```
#TimeInterval{  
  text-align: center;  
  right: 5px;  
  position: center;  
}
```

```
.Env{  
  Position:relative;  
  right: -120px;  
  width: 20%;  
  top: -25px;  
}
```

```
.Motion{  
  Position:relative;  
  right: -50px;  
  width: 20%;  
  top: -25px;  
}
```

Appendix IV – Index.js

```
// Import MQTT service
import { MQTTService } from "./mqttService.js";

// Target specific HTML items
const sideMenu = document.querySelector("aside");
const menuBtn = document.querySelector("#menu-btn");
const closeBtn = document.querySelector("#close-btn");
const themeToggler = document.querySelector(".theme-toggler");

const bleConnect = document.querySelector("#BLEconnect-btn");
const bleDisconnect = document.querySelector("#BLEdisconnect-btn");
const batterystatuscmd = document.querySelector("#Batterystatus-btn");

const clearButton = document.querySelector("#clearButton");
const cb_start = document.querySelector("#start-btn");

// Environment Sensor Checkboxes true or false
var envCheck1 = document.querySelector("#envCheck1");
var envCheck2 = document.querySelector("#envCheck2");
var envCheck3 = document.querySelector("#envCheck3");
var envCheck4 = document.querySelector("#envCheck4");
var envCheck5 = document.querySelector("#envCheck5");

// Motion Sensor Checkboxes true or false
var motionCheck1 = document.querySelector("#motionCheck1");
var motionCheck2 = document.querySelector("#motionCheck2");

var boxCheckEnv = document.getElementById("tblSensors");
var boxCheckMotion = document.getElementById("motion_tblSensors");

boxCheckEnv.addEventListener("change", (event) => {
  if (envCheck1.checked == true || envCheck2.checked == true || envCheck3.checked == true ||
  envCheck4.checked == true || envCheck5.checked == true) {
    document.getElementById("motionCheck1").checked = false;
    document.getElementById("motionCheck2").checked = false;
  }
});

boxCheckMotion.addEventListener("change", (event) => {
  if (motionCheck1.checked == true || motionCheck2.checked == true) {
    document.getElementById("envCheck1").checked = false;
    document.getElementById("envCheck2").checked = false;
    document.getElementById("envCheck3").checked = false;
    document.getElementById("envCheck4").checked = false;
    document.getElementById("envCheck5").checked = false;
  }
});

// Holds the background color of all chart
var chartBGColor = getComputedStyle(document.body).getPropertyValue(
  "--chart-background"
);
var chartFontColor = getComputedStyle(document.body).getPropertyValue(
```

```

"--chart-font-color"
);
var chartAxisColor = getComputedStyle(document.body).getPropertyValue(
"--chart-axis-color"
);

/*
Event listeners for any HTML click
*/
menuBtn.addEventListener("click", () => {
sideMenu.style.display = "block";
});

closeBtn.addEventListener("click", () => {
sideMenu.style.display = "none";
});

themeToggler.addEventListener("click", () => {
document.body.classList.toggle("dark-theme-variables");
themeToggler.querySelector("span:nth-child(1)").classList.toggle("active");
themeToggler.querySelector("span:nth-child(2)").classList.toggle("active");

// Update Chart background
chartBGColor = getComputedStyle(document.body).getPropertyValue(
"--chart-background"
);
chartFontColor = getComputedStyle(document.body).getPropertyValue(
"--chart-font-color"
);
chartAxisColor = getComputedStyle(document.body).getPropertyValue(
"--chart-axis-color"
);
updateChartsBackground();
});

/*
Plotly.js graph and chart setup code
*/

var temperatureHistoryDiv = document.getElementById("temperature-history");
var humidityHistoryDiv = document.getElementById("humidity-history");
var co2HistoryDiv = document.getElementById("co2-history");
var gasHistoryDiv = document.getElementById("gas-history");
var pressureHistoryDiv = document.getElementById("pressure-history");
var accelerometerHistoryDiv = document.getElementById("accelerometer-history");
var gyroscopeHistoryDiv = document.getElementById("gyroscope-history");

const historyCharts = [
temperatureHistoryDiv,
humidityHistoryDiv,
co2HistoryDiv,
gasHistoryDiv,
pressureHistoryDiv,
accelerometerHistoryDiv,
gyroscopeHistoryDiv,

```

```
];
```

```
// History Data
```

```
var temperatureTrace = {  
  x: [],  
  y: [],  
  name: "Temperature",  
  mode: "lines+markers",  
  type: "line",  
};
```

```
var humidityTrace = {  
  x: [],  
  y: [],  
  name: "Humidity",  
  mode: "lines+markers",  
  type: "line",  
};
```

```
var co2Trace = {  
  x: [],  
  y: [],  
  name: "Co2",  
  mode: "lines+markers",  
  type: "line",  
};
```

```
var gasTrace = {  
  x: [],  
  y: [],  
  name: "Gas",  
  mode: "lines+markers",  
  type: "line",  
};
```

```
var pressureTrace = {  
  x: [],  
  y: [],  
  name: "Pressure",  
  mode: "lines+markers",  
  type: "line",  
};
```

```
var accelerometerXTrace = {  
  x: [],  
  y: [],  
  name: "Accelerometer X",  
  mode: "lines",  
  type: "line",  
};
```

```
var accelerometerYTrace = {  
  x: [],  
  y: [],  
  name: "Accelerometer Y",  
  mode: "lines",  
  type: "line",  
};
```

```
var accelerometerZTrace = {  
  x: [],  
  y: [],  
  name: "Accelerometer Z",  
};
```



```
mode: "lines",
type: "line",
};

var gyroscopeXTrace = {
x: [],
y: [],
name: "Gyroscope X",
mode: "lines",
type: "line",
};

var gyroscopeYTrace = {
x: [],
y: [],
name: "Gyroscope Y",
mode: "lines",
type: "line",
};

var gyroscopeZTrace = {
x: [],
y: [],
name: "Gyroscope Z",
mode: "lines",
type: "line",
};

var temperatureLayout = {
autosize: true,
title: {
text: "Temperature",
},
font: {
size: 12,
color: chartFontColor,
family: "poppins, san-serif",
},
colorway: ["#05AD86"],
margin: { t: 40, b: 40, l: 30, r: 30, pad: 0 },
plot_bgcolor: chartBGColor,
paper_bgcolor: chartBGColor,
xaxis: {
color: chartAxisColor,
linecolor: chartAxisColor,
gridwidth: "2",
autorange: true,
},
yaxis: {
color: chartAxisColor,
linecolor: chartAxisColor,
gridwidth: "2",
autorange: true,
},
};

var humidityLayout = {
autosize: true,
title: {
```

```

    text: "Humidity",
  },
  font: {
    size: 12,
    color: chartFontColor,
    family: "poppins, san-serif",
  },
  colorway: ["#05AD86"],
  margin: { t: 40, b: 40, l: 30, r: 30, pad: 0 },
  plot_bgcolor: chartBGColor,
  paper_bgcolor: chartBGColor,
  xaxis: {
    color: chartAxisColor,
    linecolor: chartAxisColor,
    gridwidth: "2",
  },
  yaxis: {
    color: chartAxisColor,
    linecolor: chartAxisColor,
  },
};
var co2Layout = {
  autosize: true,
  title: {
    text: "Co2",
  },
  font: {
    size: 12,
    color: chartFontColor,
    family: "poppins, san-serif",
  },
  colorway: ["#05AD86"],
  margin: { t: 40, b: 40, l: 30, r: 30, pad: 0 },
  plot_bgcolor: chartBGColor,
  paper_bgcolor: chartBGColor,
  xaxis: {
    color: chartAxisColor,
    linecolor: chartAxisColor,
    gridwidth: "2",
  },
  yaxis: {
    color: chartAxisColor,
    linecolor: chartAxisColor,
  },
};
var gasLayout = {
  autosize: true,
  title: {
    text: "Gas",
  },
  font: {
    size: 12,
    color: chartFontColor,
    family: "poppins, san-serif",
  },
  colorway: ["#05AD86"],
  margin: { t: 40, b: 40, l: 30, r: 30, pad: 0 },
  plot_bgcolor: chartBGColor,
  paper_bgcolor: chartBGColor,

```

```

xaxis: {
  color: chartAxisColor,
  linecolor: chartAxisColor,
  gridwidth: "2",
},
yaxis: {
  color: chartAxisColor,
  linecolor: chartAxisColor,
},
};

var pressureLayout = {
  autosize: true,
  title: {
    text: "Pressure",
  },
  font: {
    size: 12,
    color: chartFontColor,
    family: "poppins, san-serif",
  },
  colorway: ["#05AD86"],
  margin: { t: 40, b: 40, l: 30, r: 30, pad: 0 },
  plot_bgcolor: chartBGColor,
  paper_bgcolor: chartBGColor,
  xaxis: {
    color: chartAxisColor,
    linecolor: chartAxisColor,
    gridwidth: "2",
  },
  yaxis: {
    color: chartAxisColor,
    linecolor: chartAxisColor,
  },
};

var accelerometerLayout = {
  autosize: true,
  title: {
    text: "Accelerometer",
  },
  font: {
    size: 12,
    color: chartFontColor,
    family: "poppins, san-serif",
  },
  colorway: ["#b60c26", "#357b3e", "#0000ff"],
  margin: { t: 40, b: 40, l: 30, r: 30, pad: 0 },
  plot_bgcolor: chartBGColor,
  paper_bgcolor: chartBGColor,
  xaxis: {
    color: chartAxisColor,
    linecolor: chartAxisColor,
    gridwidth: "2",
  },
  yaxis: {
    color: chartAxisColor,
    linecolor: chartAxisColor,
  },
};

```

```

};

var gyroscopeLayout = {
  autosize: true,
  title: {
    text: "Gyroscope",
  },
  font: {
    size: 12,
    color: chartFontColor,
    family: "poppins, san-serif",
  },
  colorway: ["#b60c26", "#357b3e", "#0000ff"],
  margin: { t: 40, b: 40, l: 30, r: 30, pad: 0 },
  plot_bgcolor: chartBGColor,
  paper_bgcolor: chartBGColor,
  xaxis: {
    color: chartAxisColor,
    linecolor: chartAxisColor,
    gridwidth: "2",
  },
  yaxis: {
    color: chartAxisColor,
    linecolor: chartAxisColor,
  },
};

var config = { responsive: true, displayModeBar: false };

// Event listener when page is loaded
window.addEventListener("load", (event) => {
  Plotly.newPlot(temperatureHistoryDiv, [temperatureTrace], temperatureLayout, config);
  Plotly.newPlot(humidityHistoryDiv, [humidityTrace], humidityLayout, config);
  Plotly.newPlot(co2HistoryDiv, [co2Trace], co2Layout, config);
  Plotly.newPlot(gasHistoryDiv, [gasTrace], gasLayout, config);
  Plotly.newPlot(pressureHistoryDiv, [pressureTrace], pressureLayout, config);
  Plotly.newPlot(accelerometerHistoryDiv, [accelerometerXTrace, accelerometerYTrace,
  accelerometerZTrace], accelerometerLayout, config);
  Plotly.newPlot(gyroscopeHistoryDiv, [gyroscopeXTrace, gyroscopeYTrace, gyroscopeZTrace],
  gyroscopeLayout, config);

  // Get MQTT Connection
  fetchMQTTConnection();

  // Run it initially
  handleDeviceChange(mediaQuery);
});

// Will hold the arrays we receive from our Nicla Sense
var xArray = [];
var newXArray = [];
var xArrayUpd;

// Temperature
let newTempYArray = [];
// Humidity
let newHumidityYArray = [];
// Co2

```

```

let newCo2YArray = [];
// gas
let newgasYArray = [];
// pressure
let newpressureYArray = [];
// Accelerometer X
let newAccxYArray = [];
// Accelerometer Y
let newAccyYArray = [];
// Accelerometer Z
let newAcczYArray = [];
// Gyroscope X
let newGyrxYArray = [];
// Gyroscope Y
let newGyryYArray = [];
// Gyroscope Z
let newGyrzYArray = [];

// The maximum number of data points displayed on our scatter/line graph

let MAX_GRAPH_POINTS_X = 50;
let MAX_GRAPH_POINTS_Y = 49;

let MAX_GRAPH_POINTS_X_MOTION = 101;
let MAX_GRAPH_POINTS_Y_MOTION = 100;

var ctr = 0;

// Motion Sensors Callback function that will retrieve our latest sensor readings and redraw our Gauge with
the latest readings
function updateMotionSensorReadings(jsonResponse) {
  if(motionCheck1.checked == true || motionCheck2.checked == true) {

    if (hasBeenCleared == true) {
      hasBeenCleared = false;
      Plotly.deleteTraces(accelerometerHistoryDiv, [0]);
      Plotly.deleteTraces(gyroscopeHistoryDiv, [0]);

      var accelerometerXTrace = {
        x: [],
        y: [],
        name: "Accelerometer X",
        mode: "lines",
        type: "line",
        line_color:"red",
      };

      var accelerometerYTrace = {
        x: [],
        y: [],
        name: "Accelerometer Y",
        mode: "lines",
        type: "line",
      };

      var accelerometerZTrace = {
        x: [],

```

```
y: [],  
name: "Accelerometer Z",  
mode: "lines",  
type: "line",  
};
```

```
var gyroscopeXTrace = {  
x: [],  
y: [],  
name: "Gyroscope X",  
mode: "lines",  
type: "line",  
};
```

```
var gyroscopeYTrace = {  
x: [],  
y: [],  
name: "Gyroscope Y",  
mode: "lines",  
type: "line",  
};
```

```
var gyroscopeZTrace = {  
x: [],  
y: [],  
name: "Gyroscope Z",  
mode: "lines",  
type: "line",  
};
```

```
Plotly.newPlot(accelerometerHistoryDiv, [accelerometerXTrace, accelerometerYTrace,  
accelerometerZTrace], accelerometerLayout, config);
```

```
Plotly.newPlot(gyroscopeHistoryDiv, [gyroscopeXTrace, gyroscopeYTrace, gyroscopeZTrace],  
gyroscopeLayout, config);
```

```
xArray = [];  
xArrayUpd = [1];  
ctr = 0;  
}
```

```
var accX;  
var accY;  
var accZ;  
var gyrX;  
var gyrY;  
var gyrZ;
```

```
if(jsonResponse.accelerometer){  
var accX = Number(jsonResponse.accelerometer[0]).toFixed();  
var accY = Number(jsonResponse.accelerometer[1]).toFixed();  
var accZ = Number(jsonResponse.accelerometer[2]).toFixed();  
} else {  
var accX = NaN;  
var accY = NaN;  
var accZ = NaN;  
}
```

```
if(jsonResponse.gyroscope){
```

```

var gyrX = Number(jsonResponse.gyroscope[0]).toFixed();
console.log(gyrX);
var gyrY = Number(jsonResponse.gyroscope[1]).toFixed();
console.log(gyrY);
var gyrZ = Number(jsonResponse.gyroscope[2]).toFixed();
console.log(gyrZ);
} else {
var gyrX = NaN;
var gyrY = NaN;
var gyrZ = NaN;
}
let time = (jsonResponse.Time).toString();

updateMotionBoxes(accX, accY, accZ, gyrX, gyrY, gyrZ);
sensorMotionValues(accX, accY, accZ, gyrX, gyrY, gyrZ, time);
updateXArray(xArray);

```

```

// Update Accelerometer Line Chart

```

```

updateMotionCharts(
  accelerometerHistoryDiv,
  newXArray,
  newAccxYArray,
  newAccyYArray,
  newAcczYArray,
  accX,
  accY,
  accZ
);

```

```

// Update Gyroscope Line Chart

```

```

updateMotionCharts(
  gyroscopeHistoryDiv,
  newXArray,
  newGyrxYArray,
  newGyryYArray,
  newGyrzYArray,
  gyrX,
  gyrY,
  gyrZ
);

```

```

}
}

```

```

function updateMotionBoxes(accX, accY, accZ, gyrX, gyrY, gyrZ) {
  let accelerometerDiv = document.getElementById("accelerometer");
  let gyroscopeDiv = document.getElementById("gyroscope");

  accelerometerDiv.innerHTML = "x: " + accX + " y: " + accY + " z: " + accZ;
  gyroscopeDiv.innerHTML = "x: " + gyrX + " y: " + gyrY + " z: " + gyrZ;
}

```

```

function updateMotionCharts(lineChartDiv, newXArray, yxArray, yyArray, yzArray, sensorReadX,
sensorReadY, sensorReadZ) {
  if (xArrayUpd.length >= MAX_GRAPH_POINTS_X_MOTION) {
    xArrayUpd.shift();
  }
  if (yxArray.length >= MAX_GRAPH_POINTS_Y_MOTION) {
    yxArray.shift();
  }
}

```

```

if (yyArray.length >= MAX_GRAPH_POINTS_Y_MOTION) {
  yyArray.shift();
}

if (yzArray.length >= MAX_GRAPH_POINTS_Y_MOTION) {
  yzArray.shift();
}

yxAarray.push(sensorReadX);
yyArray.push(sensorReadY);
yzArray.push(sensorReadZ);

var data_update = {
  x: [xArrayUpd],
  y: [yxAarray,yyArray,yzArray],
};
Plotly.update(lineChartDiv, data_update);
}

// Environment Sensors Callback function that will retrieve our latest sensor readings and redraw our Gauge
with the latest readings
function updateSensorReadings(jsonResponse) {
  if(envCheck1.checked == true || envCheck2.checked == true || envCheck3.checked == true ||
envCheck4.checked == true || envCheck5.checked == true) {
    if (hasBeenCleared == true) {
      hasBeenCleared = false;
      Plotly.deleteTraces(temperatureHistoryDiv, [0]);
      Plotly.deleteTraces(humidityHistoryDiv, [0]);
      Plotly.deleteTraces(co2HistoryDiv, [0]);
      Plotly.deleteTraces(gasHistoryDiv, [0]);
      Plotly.deleteTraces(pressureHistoryDiv, [0]);

      temperatureTrace = {
        x: [],
        y: [],
        name: "Temperature",
        mode: "lines+markers",
        type: "line",
      };

      humidityTrace = {
        x: [],
        y: [],
        name: "Humidity",
        mode: "lines+markers",
        type: "line",
      };

      co2Trace = {
        x: [],
        y: [],
        name: "Co2",
        mode: "lines+markers",
        type: "line",
      };

      gasTrace = {
        x: [],
        y: [],

```



```

    name: "Gas",
    mode: "lines+markers",
    type: "line",
  };

  pressureTrace = {
    x: [],
    y: [],
    name: "Pressure",
    mode: "lines+markers",
    type: "line",
  };

  // Temperature
  newTempYArray = [];

  // Humidity
  newHumidityYArray = [];

  // Co2
  newCo2YArray = [];

  // gas
  newgasYArray = [];

  // pressure
  newpressureYArray = [];

  Plotly.newPlot(temperatureHistoryDiv, [temperatureTrace], temperatureLayout, config);
  Plotly.newPlot(humidityHistoryDiv, [humidityTrace], humidityLayout, config);
  Plotly.newPlot(co2HistoryDiv, [co2Trace], co2Layout, config);
  Plotly.newPlot(gasHistoryDiv, [gasTrace], gasLayout, config);
  Plotly.newPlot(pressureHistoryDiv, [pressureTrace], pressureLayout, config);

  xArray = [];
  xArrayUpd = [1];
  ctr = 0;
}

let temperature = Number(jsonResponse.temperature).toFixed(1);
let humidity = Number(jsonResponse.humidity).toFixed();
let co2 = Number(jsonResponse.co2).toFixed();
let gas = Number(jsonResponse.gas).toFixed();
let pressure = Number(jsonResponse.pressure).toFixed(1);
let time = (jsonResponse.Time).toString();

updateEnvBoxes(temperature, humidity, co2, gas, pressure);
sensorValues(temperature, humidity, co2, gas, pressure, time);
updateXArray(xArray);

// Update Temperature Line Chart
updateEnvCharts(
  temperatureHistoryDiv,
  newXArray,
  newTempYArray,
  temperature

```

```

    );
    // Update Humidity Line Chart
    updateEnvCharts(
        humidityHistoryDiv,
        newXArray,
        newHumidityYArray,
        humidity
    );

    // Update Co2 Line Chart
    updateEnvCharts(
        co2HistoryDiv,
        newXArray,
        newCo2YArray,
        co2
    );

    // Update gas Line Chart
    updateEnvCharts(
        gasHistoryDiv,
        newXArray,
        newgasYArray,
        gas
    );

    // Update pressure Line Chart
    updateEnvCharts(
        pressureHistoryDiv,
        newXArray,
        newpressureYArray,
        pressure
    );

}
}

function updateEnvBoxes(temperature, humidity, co2, gas, pressure) {
    let temperatureDiv = document.getElementById("temperature");
    let humidityDiv = document.getElementById("humidity");
    let co2Div = document.getElementById("co2");
    let gasDiv = document.getElementById("gas");
    let pressureDiv = document.getElementById("pressure");

    temperatureDiv.innerHTML = temperature + " °C";
    humidityDiv.innerHTML = humidity + "%";
    co2Div.innerHTML = co2 + " ppm";
    gasDiv.innerHTML = gas + " Ω";
    pressureDiv.innerHTML = pressure + " hPa";
}

function updatexArray(xArray) {
    xArray.push(++ctr);
    xArrayUpd = xArray;
    var newXArray = {
        x: [xArray],
    };
}
}

```

```

function updateEnvCharts(lineChartDiv, newXArray, yArray, sensorRead) {
  if (xArrayUpd.length >= MAX_GRAPH_POINTS_X) {
    xArrayUpd.shift();
  }
  if (yArray.length >= MAX_GRAPH_POINTS_Y) {
    yArray.shift();
  }
  yArray.push(sensorRead);
  var data_update = {
    x: [xArrayUpd],
    y: [yArray],
  };
  Plotly.update(lineChartDiv, data_update);
}

```

```

function updateChartsBackground() {
  // updates the background color of historical charts
  var updateHistory = {
    plot_bgcolor: chartBGColor,
    paper_bgcolor: chartBGColor,
    font: {
      color: chartFontColor,
    },
    xaxis: {
      color: chartAxisColor,
      linecolor: chartAxisColor,
    },
    yaxis: {
      color: chartAxisColor,
      linecolor: chartAxisColor,
    },
  };
  historyCharts.forEach((chart) => Plotly.relayout(chart, updateHistory));
}

```

```

const mediaQuery = window.matchMedia("(max-width: 600px)");

```

```

mediaQuery.addEventListener("change", function (e) {
  handleDeviceChange(e);
});

```

```

function handleDeviceChange(e) {
  if (e.matches) {
    console.log("Inside Mobile");
    var updateHistory = {
      width: 323,
      height: 250,
      "xaxis.autorange": true,
      "yaxis.autorange": true,
    };
    historyCharts.forEach((chart) => Plotly.relayout(chart, updateHistory));
  } else {
    var updateHistory = {
      width: 550,
      height: 260,
      "xaxis.autorange": true,
      "yaxis.autorange": true,
    };

```

```

    };
    historyCharts.forEach((chart) => Plotly.relayout(chart, updateHistory));
  }
}

// Bluetooth connection status Message Handling Code

const BLEStatus = document.querySelector(".BLEstatus");

// This is the function which handles subscribing to topics after a connection is made
function BLEConnectionStatus(mqttServer, mqttBLEConnectionStatusTopic){
  console.log(
    `Initializing connection to :: ${mqttServer}, topic :: ${mqttBLEConnectionStatusTopic}`
  );

  var fnCallbacks = { onMessageArrived };
  var mqttService = new MQTTService(mqttServer, fnCallbacks);

  mqttService.connect();
  mqttService.subscribe(mqttBLEConnectionStatusTopic);
};

function updateBLEstatus(stringmessage) {
  console.log(stringmessage);
  var BatteryCheckDisc = stringmessage;
  BLEStatus.textContent = stringmessage;
  if(BatteryCheckDisc == "Disconnected"){
    BatteryStatus.textContent = "Unknown";
    BatteryStatusTime.textContent = "";
  }
}

/*
Bluetooth device MQTT connection status Message Handling Code
*/

const BLEMQTTStatus = document.querySelector(".BLEMQTTstatus");

// This is the function which handles subscribing to topics after a connection is made
function BLEMQTT_Status(mqttServer, mqttBLEStatusTopic){
  console.log(
    `Initializing connection to :: ${mqttServer}, topic :: ${mqttBLEStatusTopic}`
  );

  var fnCallbacks = { onMessageArrivedMQTTBLE };
  var mqttService = new MQTTService(mqttServer, fnCallbacks);

  mqttService.connect();
  mqttService.subscribe(mqttBLEStatusTopic);
};

function updateMQTTBLEstatus(stringmessage) {
  console.log(stringmessage);
  BLEMQTTStatus.textContent = stringmessage;
}

//Nicla Sense Battery Statys Handling Code
const BatteryStatus = document.querySelector(".NiclaBatterystatus");

```

```

const BatteryStatusTime = document.querySelector(".NiclaBatterystatusTime");

// This is the function which updates the Nicla Sense battery status
function NBatteryStatus(mqttServer, mqttBatteryStatusTopic){
  console.log(
    `Initializing connection to :: ${mqttServer}, topic :: ${mqttBatteryStatusTopic}`
  );

  var fnCallbacks = { onMessageArrivedBatteryStatus };
  var mqttService = new MQTTService(mqttServer, fnCallbacks);

  mqttService.connect();
  mqttService.subscribe(mqttBatteryStatusTopic);
};

var StatusDisconnected = "Disconnected";

function updateBatterystatus(stringmessage) {
  var battery = stringmessage.split(',');
  var time = battery[0];
  var batterupdatenow = battery[1];
  BatteryStatus.textContent = batterupdatenow;
  BatteryStatusTime.textContent = time;
}

// MQTT Message Handling Code
const mqttStatus = document.querySelector(".status");

function onConnect(message) {
  mqttStatus.textContent = "Connected";
}

function onMessage(topic, message) {
  var stringResponse = message.toString();
  var messageResponse = JSON.parse(stringResponse);
  updateSensorReadings(messageResponse);
}

function onMessageMotion(topic, message) {
  var stringResponse = message.toString();
  var messageResponse = JSON.parse(stringResponse);
  updateMotionSensorReadings(messageResponse);
}

function onMessageArrived(topic, message) {
  var stringmessage = message.toString();
  updateBLEstatus(stringmessage);
}

function onMessageArrivedMQTTBLE(topic, message) {
  var stringmessage = message.toString();
  updateMQTTBLEstatus(stringmessage);
}

function onMessageArrivedBatteryStatus(topic, message) {
  var stringmessage = message.toString();
  updateBatterystatus(stringmessage);
}

```

```

function onError(error) {
  console.log(`Error encountered :: ${error}`);
  mqttStatus.textContent = "Error";
}

function onClose() {
  console.log(`MQTT connection closed!`);
  mqttStatus.textContent = "Closed";
}

function fetchMQTTConnection() {
  fetch("/mqttConnDetails", {
    method: "GET",
    headers: {
      "Content-type": "application/json; charset=UTF-8",
    },
  })
  .then(function (response) {
    return response.json();
  })
  .then(function (data) {
    MQTTEnvConnection(data.mqttServer, data.mqttTopicEnv);
    MQTTMotionConnection(data.mqttServer, data.mqttTopicMotion);
    BLEConnect_message(data.mqttServer, data.mqttBLEConnectTopic);
    BLEDisconnect_message(data.mqttServer, data.mqttBLEDisconnectTopic);
    BLEConnectionStatus(data.mqttServer, data.mqttBLEConnectionStatusTopic);
    BLEMQTT_Status(data.mqttServer, data.mqttBLEStatusTopic);
    mqttSensorCommands(data.mqttServer, data.mqttSensorCommandsTopic);
    BatteryStatusCommand(data.mqttServer, data.mqttBatteryStatusCommandTopic);
    NBatteryStatus(data.mqttServer, data.mqttBatteryStatusTopic);
  })
  .catch((error) => console.error("Error getting MQTT Connection :", error));
}

function MQTTEnvConnection(mqttServer, mqttTopic) {
  console.log(
    `Initializing connection to :: ${mqttServer}, topic :: ${mqttTopic}`
  );
  var fnCallbacks = { onConnect, onMessage, onError, onClose };

  var mqttService = new MQTTService(mqttServer, fnCallbacks);
  mqttService.connect();
  mqttService.subscribe(mqttTopic);
}

function MQTTMotionConnection(mqttServer, mqttTopicMotion) {
  console.log(
    `Initializing connection to :: ${mqttServer}, topic :: ${mqttTopicMotion}`
  );
  var fnCallbacks = { onConnect, onMessageMotion, onError, onClose };

  var mqttService = new MQTTService(mqttServer, fnCallbacks);
  mqttService.connect();
  mqttService.subscribe(mqttTopicMotion);
}

// Send command connect BLE device
function BLEConnect_message(mqttServer, mqttBLEConnectTopic) {
  console.log(

```

```

    `Initializing connection to :: ${mqttServer}, topic :: ${mqttBLEConnectTopic}`
  );
  bleConnect.addEventListener("click", BLEConnectActivate);

  var mqttService = new MQTTService(mqttServer);
  mqttService.connect();
  function BLEConnectActivate() {
    mqttService.publish(mqttBLEConnectTopic);
  }
}

// Send command disconnect BLE device
function BLEDisconnect_message(mqttServer, mqttBLEDisconnectTopic) {
  console.log(
    `Initializing connection to :: ${mqttServer}, topic :: ${mqttBLEDisconnectTopic}`
  );
  bleDisconnect.addEventListener("click", BLEDisconnectActivate);

  var mqttService = new MQTTService(mqttServer);
  mqttService.connect();
  function BLEDisconnectActivate() {
    mqttService.publish(mqttBLEDisconnectTopic)
  }
}

// Send command to get Nicla Sense Battery Status
function BatteryStatusCommand(mqttServer, mqttBatteryStatusCommandTopic) {
  console.log(
    `Initializing connection to :: ${mqttServer}, topic :: ${mqttBatteryStatusCommandTopic}`
  );
  batterystatuscmd.addEventListener("click", BatteryStatusActivate);

  var mqttService = new MQTTService(mqttServer);
  mqttService.connect();
  function BatteryStatusActivate() {
    mqttService.publish(mqttBatteryStatusCommandTopic)
  }
}

let start = document.querySelector("#Start_command");
let stop = document.querySelector("#Stop_command");
let timerId; // makes the variable global

function mqttSensorCommands(mqttServer, mqttSensorCommandsTopic) {
  start.addEventListener('click', function() {

//Create an Array.
    var selected = new Array();
    var chks;

//Reference the Environment Table.
    if (envCheck1.checked == true || envCheck2.checked == true || envCheck3.checked == true ||
envCheck4.checked == true || envCheck5.checked == true) {
      var timeInterval = document.querySelector("#Time_Interval_Env").value;
      var chks = tblSensors.getElementsByTagName("INPUT");
    }
  }
}

```

```

//Reference the Motion Table.
if (motionCheck1.checked == true || motionCheck2.checked == true) {
    var chks = motion_tblSensors.getElementsByTagName("INPUT");
    var timeInterval = document.querySelector("#Time_Interval_Motion").value;
}
var mqttService = new MQTTService(mqttServer);
mqttService.connect();

// Loop and push the checked CheckBox value in Array.
for (var i = 0; i < chks.length; i++) {
    if (chks[i].checked) {
        selected.push(chks[i].value);
    }
}

//Display the selected CheckBox values.
if (selected.length > 0) {
    var commands = (" " + selected.join(", "));
    let timerId = setInterval(function() {
        mqttService.publish(mqttSensorCommandsTopic, commands);
    }, timeInterval);

    // Stopping the timer:
    stop.addEventListener('click', function() {
        console.log("stop")
        clearInterval(timerId);
    });
}
});
}

```

```

let downloadCSVbtn = document.querySelector('#DownloadCSV-btn');

```

```

var sessionData = [];
var sesssionDataCSV;
var sessionRow = 0;
var restoreKey;
var restoreData;
var restoreGraph;

```

```

//clears the entire sessionStorage
window.onload = function clearStorage() {
    sessionStorage.clear();
}

```

```

var hasBeenCleared = false;

```

```

document.getElementById("clearButton").onclick = function clearData() {
    sessionStorage.clear();
    restoreKey = 1;
    restoreData = 1;
    restoreGraph = 1;

    hasBeenCleared = true;
}

```

```

function sensorValues(temperature, humidity, co2, gas, pressure, time) {

```



```

if (restoreKey > 0) {
  sessionStorage.clear();
  console.log("clear records");
  restoreKey = 0;
  sessionRow = 0;
} else {
  sessionRow;
}

sessionRow++;

var sensorData = [sessionRow, time, temperature, humidity, pressure, co2, gas];
var sensorDataString = JSON.stringify(sensorData);

sessionStorage.setItem(sessionRow, sensorDataString);
}

```

```

function sensorMotionValues(accX, accY, accZ, gyrX, gyrY, gyrZ, time) {
  if (restoreKey > 0) {
    sessionStorage.clear();
    console.log("clear records");
    restoreKey = 0;
    sessionRow = 0;
  } else {
    sessionRow;
  }

  sessionRow++;

  if (motionCheck1.checked == true && motionCheck2.checked == true) {
    var sensorData = [sessionRow, time, accX, accY, accZ, gyrX, gyrY, gyrZ];
  }
  else {
    if (motionCheck1.checked == true) {
      var sensorData = [sessionRow, time, accX, accY, accZ];
    }

    if (motionCheck2.checked == true) {
      var sensorData = [sessionRow, time, gyrX, gyrY, gyrZ];
    }
  }

  var sensorDataString = JSON.stringify(sensorData);

  sessionStorage.setItem(sessionRow, sensorDataString);
}

```

```

//create a user-defined function to download CSV file
downloadCSVbtn.addEventListener('click', function(){

```

```

  // iterate sessionStorage
  for (var i = 0; i < sessionStorage.length; i++) {
    // set iteration key name
    var key = sessionStorage.key(i);
    // use key name to retrieve the corresponding value
    var value = sessionStorage.getItem(key);
    sessionData.push(JSON.parse(value));
  }
}

```

```

    console.log(sessionData);
  }

  //define the heading for each row of the data

  if (envCheck1.checked == true || envCheck2.checked == true || envCheck3.checked == true ||
  envCheck4.checked == true || envCheck5.checked == true) {
    var csv = 'Sample,Timestamp,Temperature,Humidity,Pressure,Co2,Gas\n';
  }

  if (motionCheck1.checked == true && motionCheck2.checked == true) {
    var csv = 'Sample,Timestamp,accX,accY,accZ,gyrX,gyrY,gyrZ\n';
  }
  else {

    if (motionCheck1.checked == true) {
      var csv = 'Sample,Timestamp,accX,accY,accZ\n';
    }

    if (motionCheck2.checked == true) {
      var csv = 'Sample,Timestamp,gyrX,gyrY,gyrZ\n';
    }
  }

  //merge the data with CSV
  sessionData.forEach(function(row) {
    csv += row.join(',');
    csv += "\n";
  });

  var hiddenElement = document.createElement('a');
  hiddenElement.href = 'data:text/csv;charset=utf-8,' + encodeURIComponent(csv);
  hiddenElement.target = '_blank';

  //provide the name for the CSV file to be downloaded
  hiddenElement.download = 'SensorData.csv';
  hiddenElement.click();

  sessionData = [];

});

```

Appendix V – mqttService.js

```
// MQTT Service, Publish, Subscribing, Messagecallback
export class MQTTService {
  constructor(host, messageCallbacks) {
    this.mqttClient = null;
    this.host = host;
    this.messageCallbacks = messageCallbacks;
  }

  // Connect to MQTT Broker
  connect() {
    var options = {username: "****", password: "****"},
    this.mqttClient = mqtt.connect(this.host, options);

    // MQTT Callback for 'error' event
    this.mqttClient.on("error", (err) => {
      console.log(err);
      this.mqttClient.end();
      if (this.messageCallbacks && this.messageCallbacks.onError)
        this.messageCallbacks.onError(err);
    });

    // MQTT Callback for 'connect' event
    this.mqttClient.on("connect", () => {
      console.log(`MQTT client connected`);
      if (this.messageCallbacks && this.messageCallbacks.onConnect) {
        this.messageCallbacks.onConnect("Connected");
      }
    });

    // MQTT Callback function when environment message arrived
    this.mqttClient.on("message", (topic, message) => {
      if (this.messageCallbacks && this.messageCallbacks.onMessage) {
        this.messageCallbacks.onMessage(topic, message);
      }
    });

    // MQTT Callback function when motion message arrived
    this.mqttClient.on("message", (topic, message) => {
      if (this.messageCallbacks && this.messageCallbacks.onMessageMotion) {
        this.messageCallbacks.onMessageMotion(topic, message);
      }
    });

    // MQTT Callback function when connection status message arrived
    this.mqttClient.on("message", (topic, message) => {
      if (this.messageCallbacks && this.messageCallbacks.onMessageArrived) {
        this.messageCallbacks.onMessageArrived(topic, message);
      }
    });

    // MQTT Callback function when mqtt ble status connection message arrived
    this.mqttClient.on("message", (topic, message) => {
      if (this.messageCallbacks && this.messageCallbacks.onMessageArrivedMQTTBLE) {
        this.messageCallbacks.onMessageArrivedMQTTBLE(topic, message);
      }
    });
  }
}
```

```
    // Callback function when battery status message arrived
    this.mqttClient.on("message", (topic, message) => {
    if (this.messageCallbacks && this.messageCallbacks.onMessageArrivedBatteryStatus) {
        this.messageCallbacks.onMessageArrivedBatteryStatus(topic, message);
    }
    });
```

```
    // Callback function when mqtt client disconnected message arrived
    this.mqttClient.on("close", () => {
    console.log(`MQTT client disconnected`);
    if (this.messageCallbacks && this.messageCallbacks.onClose)
        this.messageCallbacks.onClose();
    });
}
```

```
    // Publish MQTT Message
    publish(topic, message, options) {
    this.mqttClient.publish(topic, message, {
        qos: 0,
        retain: false,
    });
}
```

```
    // Subscribe to MQTT Message
    subscribe(topic, options) {
    this.mqttClient.subscribe(topic, options);
    }
}
```

Appendix VI – app.js

```
//This app starts a server and listens on port 4000.
const express = require("express");
const app = express();
const port = 4000;

// Load .env file to read environment, motion, commands, status variables
require("dotenv").config();

// Used to assign the setting name to value
app.set("view engine", "ejs");

// Serve Static Files
app.use(express.static("public"));

//routes
const dashboardRouter = require("./routes/dashboard");

// Process and get topics
app.get("/mqttConnDetails", (req, res) => {
  res.send(
    JSON.stringify({
      mqttServer: process.env.MQTT_BROKER,
      mqttTopicEnv: process.env.MQTT_TOPIC_ENV,
      mqttTopicMotion: process.env.MQTT_TOPIC_MOTION,
      mqttBLEConnectTopic: process.env.BLE_Connect_TOPIC,
      mqttBLEDisconnectTopic: process.env.BLE_Disconnect_TOPIC,
      mqttBLEConnectionStatusTopic: process.env.BLE_Connection_Status_TOPIC,
      mqttBLEStatusTopic: process.env.BLE_MQTT_Status_TOPIC,
      mqttSensorCommandsTopic: process.env.MQTT_Sensor_Commands_TOPIC,
      mqttBatteryStatusCommandTopic: process.env.Battery_Status_Command_TOPIC,
      mqttBatteryStatusTopic: process.env.Battery_Status_TOPIC,
    })
  );
});

app.get("/", dashboardRouter);
app.listen(port, () => {
  console.log(`Listening on port ${port}`);
});
```

Appendix VII – dashboard.js

```
// Create modular, mountable route handlers
const express = require("express");
const router = express.Router();

// Render view and send it to the dashboard.
router.get("/", function (req, res) {
  res.render("pages/dashboard", {
    name: process.env.NAME, // Title of the left side
    dashboardTitle: process.env.DASHBOARD_TITLE, // Header title
  });
});

module.exports = router;
```

Appendix VIII – .env

```
// ** Titles on the dashboard **

NAME=Novia UAS
DASHBOARD_TITLE=Nicla Sense Dashboard

// ** Specify the topics and websocket mqtt address **

MQTT_Broker = ws://address:port/mqtt

MQTT_TOPIC_ENV=nicla/env
MQTT_TOPIC_MOTION= nicla/motion

MQTT_Sensor_Commands_TOPIC = nicla/commands

BLE_Connect_TOPIC = nicla/Connect
BLE_Disconnect_TOPIC = nicla/Disconnect

BLE_Connection_Status_TOPIC = nicla/BLEstatus
BLE_MQTT_Status_TOPIC = nicla/MQTTBLEstatus

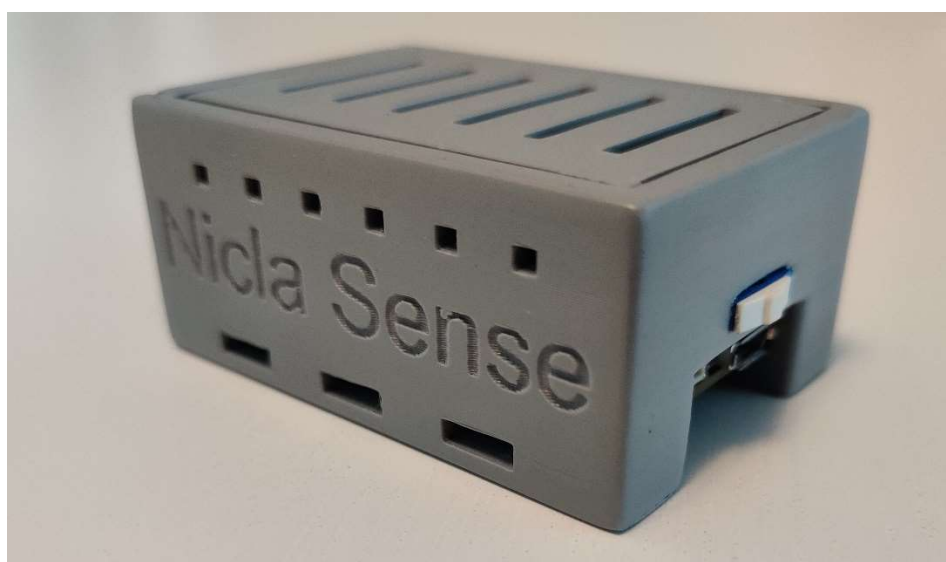
Battery_Status_Command_TOPIC = nicla/BatteryStatusCommand
Battery_Status_TOPIC = nicla/Batterystatus
```

Appendix IX – User manual (Swedish)



Användarmanual

Nicla Sense ME

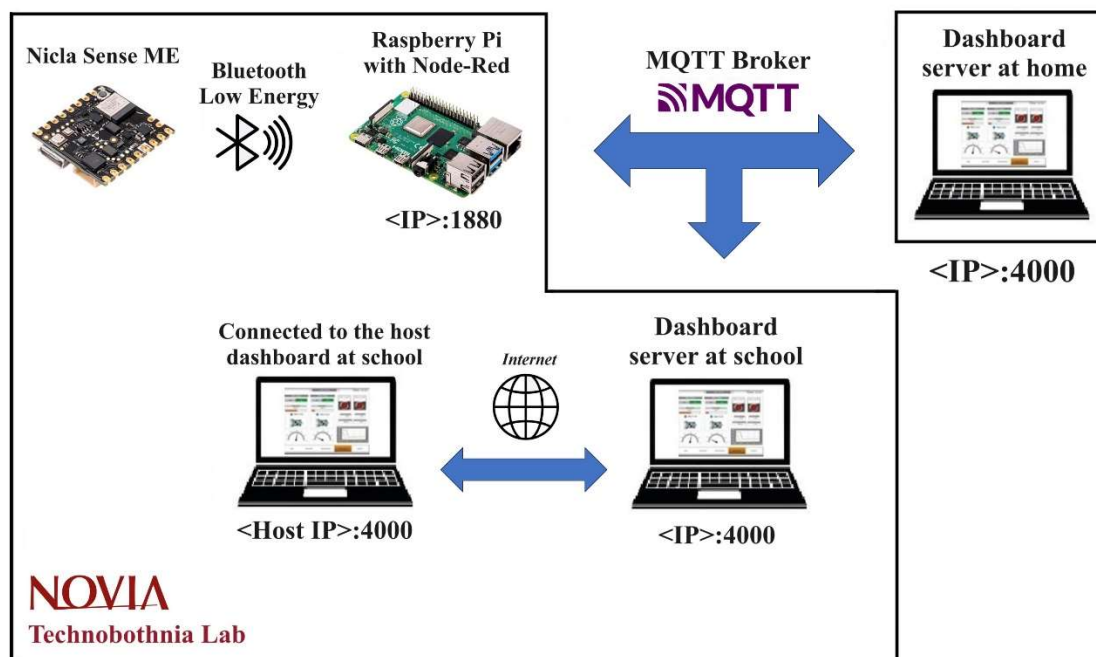


Språk: Svenska

Bilden nedan visar en överblick hur kommunikation förbindelsen fungerar, Nicla Sense kontrollern skickar data trådlöst via Bluetooth Low Energy till Raspberry Pi. Programmet Node-Red körs på Raspberry Pi som håller kommunikationen med Nicla Sense samt är uppkopplad till Novias MQTT Broker där datan skickas vidare till användargränssnittet på server datorn / datorerna. Node-Red körs på port nummer 1880 och för att komma till programmerings sidan öppnar man upp browsern och fyller i: (**<Raspberry Pi IP address>:1880**)

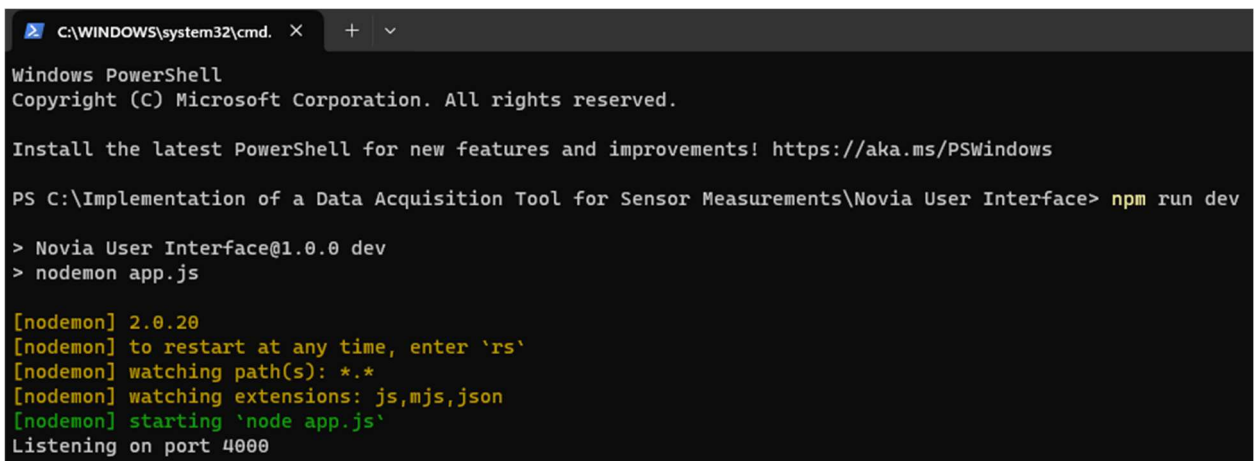
Då servern för användargränssnittet körs på till exempel Novias nätverk kan flera användare vara ansluten till samma användargränssnitt för att få tillgång till samma data. Detta görs genom att man öppnar upp browsern och fyller i ip adressen för användargränssnittets server dator samt port nummer 4000 (**<Hostname IP>:4000**).

På grund av säkerhetsskäl kan inte användare utanför nätverket ansluta till samma användargränssnitt. För att få tillgång till samma data så måste en ny server startas per nätverk. Det som bör noteras för att få tillgång till datan under insamlingen så måste någon av sensorerna vara ikryssad för att starta funktionen.



För att starta upp servern för användargränssnittet görs på följande sätt:

- Högerklicka på mappen (*Novia User Interface*) där alla filerna är samlade och tryck på öppna i Terminal.
- Kör kommandot ”*npm run dev*” som sedan startar upp servern (se bild nedan). Sök efter ip adressen genom ipconfig från terminalen, använd ip adressen samt port nummer 4000 för att ansluta till servern.
- För att stänga servern, tryck samtidigt (CTRL + C) sedan Y och Enter.



```
C:\WINDOWS\system32\cmd. X + v
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Implementation of a Data Acquisition Tool for Sensor Measurements\Novia User Interface> npm run dev

> Novia User Interface@1.0.0 dev
> nodemon app.js

[nodemon] 2.0.20
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node app.js`
Listening on port 4000
```

Överblick av Nicla Sense kontrollern med skyddsbox:

Bild 1: Blå markering hål för ventilation av luftflöde även övre lock. Röd markering, genomföring av buntband för att fästa mikrokontrollern.

Bild 2: Lock för att öppna upp skyddsboxen.

Bild 3: Orange markering led indikation. Lila markering hål för Nicla Sense inbydda omstart knapp.

Bild 4: Gul markering, brytare för av/på läge av Nicla Sense. Vit markering, ESLOV kontakt. Brun markering, MicroUSB kontakt.

1.



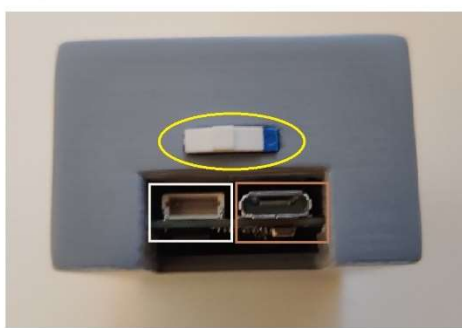
2.



3.



4.



Steg:

1. Öppna upp webbläsaren för datorn eller smarttelefonen och anslut till användargränssnittet: ip:4000, nedan visas en överblick av sidan.

NOVIA Novia UAS

Nici Sense Dashboard

Description:	Status:
Dashboard MQTT Connection:	Connected
Sensor MQTT Connection:	Unknown
Bluetooth Connection:	Unknown
Battery Status - Updated:	Unknown

Environment Sensors
 Sampling Time (ms):
 Temperature
 Humidity
 Pressure
 Co2
 Gas

Motion Sensors
 Sampling Time (ms):
 Accelerometer
 Gyroscope

Line Charts

Environment Sensors:

- Temperature
- Humidity
- Pressure
- Co2
- Gas

Motion Sensors:

- Accelerometer
- Gyroscope

Temperature, Humidity, Pressure, Co2, Gas, Accelerometer, Gyroscope

2. Kontrollera anslutningarna på sidan:

- Dashboard MQTT Connection: Connected.
- Sensor MQTT Connection: Connected.

Om anslutningarna visar ”Unknown”, vänta 30 sekunder för uppdatering av status. Om inte statusen ändras efter 1 minut se problem på sidan 5.

3. Slå på Nici Sense kontrollern från strömbrytaren, en röd led indikation skall börja lysa samt Bluetooth Connection Status ändras till ”Disconnected”.

4. Koppla upp Nicla Sense kontrollern genom att trycka på "Connect" knappen. "Connecting" visas en stund som sedan övergår till "Connected" och en grön led indikation skall visas på kontrollern. Ifall Bluetooth Connection Status återgår till "Disconnected" försök igen.
5. För att uppdatera batteri status tryck på "Battery Status" knappen (ej tillförlitlig för tillfället).
6. Överst på högra sidan finns en knapp som går att ställa in dag/natt läge.
7. För att följa upp och samla in sensor värden så kryssar man i rutan för de sensorerna som önskas. Det går endast att välja mellan "Environment Sensors" eller "Motion Sensors" och inte båda samtidigt. Då sensorerna är valda samt sampling tiden vald, tryck på "Start" knappen och sensorerna börja ta in värden.
8. Längst till vänster på sidan kan man se engångsvärden av de olika sensorerna och till höger kan man se olika grafer som ritar upp historiken av datan.
9. Det går att starta om insamlingen av datan genom att trycka på "Clear" knappen. Då man är nöjd med datan trycker man på "Stop" knappen och sedan trycker på "Download". Filen kommer att laddas ned till den lokala datorn under Hämtade filer som en csv. fil.
10. Öppna upp programmet Microsoft Excel, gå till Data → Hämta Data → Från fil → Från text/CSV och välj sedan den nedladdade csv filen. Följande inställningar används:
Filursprung: 1252: Västeuropeiska (Windows), **Avgränsare:** Komma, **Datatypesidentifiering:** Identifiera inte datatyper, enligt bild nedan och välj "Transformera data".

SensorData.csv

Filursprung: 1252: Västeuropeiska (Windows) | Avgränsare: Komma | Datatypsidentifiering: Identifiera inte datatyper

Column1	Column2	Column3	Column4	Column5	Column6	Column7
Sample	Timestamp	Temperature	Humidity	Pressure	Co2	Gas
25	10:07:35	20.4	34	1002.9	801	16077
29	10:07:39	20.3	34	1002.9	811	16188
13	10:07:23	20.3	34	1002.9	766	16263
18	10:07:28	20.4	34	1002.9	789	16089
4	10:07:14	20.3	34	1002.9	753	16389
14	10:07:24	20.3	34	1002.9	766	16250
5	10:07:15	20.3	34	1002.9	753	16263
28	10:07:38	20.3	34	1002.9	811	16139
2	10:07:12	20.3	34	1002.9	751	16415
6	10:07:16	20.3	34	1002.9	753	16402
26	10:07:36	20.4	34	1002.9	811	16089
7	10:07:17	20.3	34	1002.9	753	16288
19	10:07:29	20.4	34	1002.9	797	16176
3	10:07:13	20.3	34	1002.9	753	16326
10	10:07:20	20.3	34	1002.9	759	16339
23	10:07:33	20.3	34	1002.9	801	15980
24	10:07:34	20.4	34	1002.9	801	16126
27	10:07:37	20.3	34	1002.9	811	16151
17	10:07:27	20.3	34	1002.9	789	16188
15	10:07:25	20.3	34	1002.9	766	16213

Läs in | Transformera data | Avbryt

11. Måker alla kolumner och välj sedan "Använd första raden som rubriker". Efter det markera endast kolumn "Sample" och sedan välj "Identifiera datatyp" som finns under "Transformera". För att sedan sortera datan i rätt ordning, tryck på pilen vid kolumnen "Sample" och välj sortera stigande.
12. Då datan är sorterad välj "stäng och läs in" som sedan för in datan till Excel som sedan kan börja användas.

Rekommendationer och noteringar:

- Environment sensors samplings tid rekommenderas att inte sampla snabbare än 1000 ms per sampel (1 Hz).
- Motion sensors samplings tid rekommenderas att inte sampla snabbare än 250 ms per sampel (4 Hz).
- Regelbundet rekommenderas att ladda upp batteriet genom MicroUSB anslutningen och laddningen är begränsad till 100 mA. För att ladda batteriet krävs det att man har brytaren i på läge.
- Notera det tar en stund (ungefär 3 minuter) vid uppstart av Nicla Sense att reglera temperatur samt humidity sensorerna för att få noggrannare mätvärden.

- Co2 mätningar kan vara låsta vid 500 ppm, det krävs att Nicla Sense har varit igång 4 – 5 minuter för att börja läsa av riktiga värden.
- I ett sent skede av utvecklingen märktes att batteri status knappen inte fungerar som den ska, möjligtvis på grund av uppdateringar under utvecklingen och kan därför ge felaktig information. Det finns även en risk att man mister allt data för en sample sekvens om man under insamlingskedet trycker på batteri status knappen.

Problemsökning:

- *Nicla Sense indikation lämnat grön:* Slå av Nicla Sense brytaren och slå på igen efter en kort stund.
- *Användargränssnittet verkar ha fastnat:* Slå av Nicla Sense och uppdatera sidan, slå på kontrollern igen och vänta tills anslutnings statusarna har uppdaterats.
- *Disconnected status men går inte att koppla upp:* Stäng av Nicla Sense och vänta på Missing status, sätt igång Nicla Sense och vänta tills Disconnected blir synlig och därefter tryck på Connect knappen.
- *Plötslig fränkoppling under inspelning:* Tryck på knappen ”Stop” och prova anslut igen. Om inte återanslutningen lyckas, stäng av Nicla Sense tills status missing blir synlig och slå på igen.
- *Bluetooth statusförklaring:*
 - **Missing:** Tappat BLE kommunikationen, för långt avstånd från Raspberry Pi eller mikrokontrollern av.
 - **Disconnected:** Nicla Sense och Raspberry Pi hittat varandra men inte ännu ansluten.
 - **Connecting:** Försöker ansluta sig till Raspberry Pi.
 - **Connected:** Nicla Sense är ansluten till Raspberry Pi.
 - **Disconnecting:** Nicla Sense kopplar sig från Raspberry Pi.
 - **Error:** När ett oväntat fel inträffar.

LIST OF CODES

Code example 1. Function node (Motion Check).....	61
Code example 2. Function node (Environment Check).....	61
Code example 3. Function node (Battery lvl Check).	62
Code example 4. Function of BLE Connection Status.....	65
Code example 5. Function of BLE MQTT Status.....	65
Code example 6. The function of BLE Status Send.....	66
Code example 7. The function of BLE MQTT Status Send.	67
Code example 8. The background color of the symbol for temperature.....	72
Code example 9. Function of environmental sensor checkboxes.	75
Code example 10. sessionStorage setItem or clear.....	76
Code example 11. sessionStorage gets data and download function.	76

LIST OF TABLES

Table 1. Features of Nicla Sense ME BLE.....	7
Table 2. Recommended operating condition for Nicla Sense.....	7
Table 3. Power consumption test performed by Arduino.	8
Table 4. Features and power consumption of sensor BHI260AP.	11
Table 5. Features of BME688 sensor.....	12
Table 6. Description of one part of BSEC outputs.	12
Table 7. Features of sensor BMP390.....	13
Table 8. Key features of sensor BMM150.....	14
Table 9. Specifications of RPI4.....	15
Table 10. Common supported file formats for Rhino 7.	23
Table 11. History of Bluetooth versions.	26
Table 12. Differences between C and C++.....	36

LIST OF FIGURES

Figure 1. View of sensor data acquisition tool communication.....	1
Figure 2. Novia UAS logo (Product logos, u.d.).....	3
Figure 3. Technobothnia co-owned schools, VAMK, Vaasan yliopisto & Novia (Technobothnia, u.d.).....	4
Figure 4. Overview of the Internet of Things.....	4
Figure 5. Arduino company logo (About Arduino, n.d.).....	5
Figure 6. Arduino Nicla Sense ME with four Bosch Sensortec sensors (Arduino Nicla Sense ME, n.d.).....	6
Figure 7. Connect the battery to header pins (Connect a battery to Nicla Sense ME or Nicla Vision, 2022).....	8
Figure 8. Connect the battery to the connector (Connect a battery to Nicla Sense ME or Nicla Vision, 2022).....	9
Figure 9. Raspberry Pi 4 hardware. (Raspberry Pi 4 Tech Specs, u.d.).....	15
Figure 10. Explanation of Arduino IDE functions.....	17
Figure 11. Logo of Rhinoceros software (Rhinoceros, u.d.).....	20
Figure 12. Overview of the program Rhino 7.....	21
Figure 13. Important 3D model options in Solid Tools.....	22
Figure 14. Blocks for 3D drawing.....	22
Figure 15. UltiMaker Cura software.....	24
Figure 16. OSI Model layers and examples of protocols.....	25
Figure 17. Bluetooth LE specifications (Woolley, 2022).....	28
Figure 18. Example of Profile, Services, and Characteristics (Afaneh, Bluetooth GATT: How to Design Custom Services & Characteristics, u.d.).....	29
Figure 19. Connection event over BLE (Derhgawen, 2020).....	31
Figure 20. Illustration of publisher and subscriber arrangement.....	32
Figure 21. Quality of Service levels.....	34
Figure 22. The instructions of the source code are compiled into machine code.....	35
Figure 23. Node-Red editor.....	40
Figure 24. Node-Red nodes used in the project.....	41
Figure 25. Install driver at Board Manager for Nicla Sense.....	44
Figure 26. Install libraries at Manage Libraries for Nicla Sense.....	44
Figure 27. GATT transactions (Profile / Services / Characteristics).....	47
Figure 28. Link Layer Packet Structure (Michel, 2019).....	48
Figure 29. Raspberry Pi Imager.....	49
Figure 30. "ip r grep default" command.....	54
Figure 31. Commands in "dhcpd.conf" file.....	55
Figure 32. Overview of the Node-Red flow of the project.....	58
Figure 33. Settings of MQTT in node.....	59
Figure 34. Settings of Change node.....	59
Figure 35. Settings of Generic BLE in node.....	60
Figure 36. Settings of buffer - parser for the accelerometer.....	63
Figure 37. Settings of join node.....	63
Figure 38. Settings for JSON node.....	64
Figure 39. Change node for battery status.....	64
Figure 40. Settings of BLE Status node.....	65
Figure 41. Settings for filter and delay node.....	67
Figure 42. Overview of files used for graphical user interface.....	68
Figure 43. Before and after the use of the Boolean difference tool.....	78

Figure 44. Before and after recess for hatches.....	79
Figure 45. Nicla Sense box details.....	80
Figure 46. The final version of the Nicla Sense box with dimensions.....	81
Figure 47. UltiMaker Cura with settings of the print.....	82
Figure 48. Right: 3D printed box with supports. Left: complete protection box.....	83
Figure 49. BLE connection range test.....	87