

Kurs Big Data Analytics im SoSe 2022

Projektdokumentation

29.09.2022

Dozentin: Prof. Dr. Olga Willner

Bearbeitet von: Frank Kräßke
Enes Aydogan
Levent Arica

Zusammenfassung

In diesem Papier werden die Ergebnisse, Lernfortschritte und Schwierigkeiten der im Kurs übernommenen Aufgabe beschrieben. Aufgabe war es, zwei IoT Mikrocontroller per Bluetooth Low Energy miteinander zu verbinden und die Vorgehensweise zu dokumentieren.

Schlagworte

BigDataAnalytics HTW Berlin IoT Arduino Sense ESP 32

Abkürzungsverzeichnis

BLE Bluetooth Low Energy, ein neuer Bluetooth Standard. 3–5, 7–9, 11–16, 18, 25, 31

HTW Fachhochschule der Technik und Wirtschaft Berlin. 7

IDE Integrated Development Environment (Integrierte Entwicklungsumgebung): Eine integrierte Entwicklungsumgebung ermöglicht das Erstellen, Bearbeiten und Testen von Computerprogrammen. 9

IoT „Internet of things“, ein Sammelbegriff für Technologien, die den Zusammenschluss oder das Vernetzen von Alltagsgegenständen erreicht und somit u.a. deren entfernte Steuerung ermöglicht. 5

TUB Technische Universität Berlin. 5

Abbildungsverzeichnis

2.1	Arduino BLE 33 Nano Sense mit Beschreibung ausgewählter Sensoren (Eigene Aufnahme)	8
2.2	Hardware: ESP 32 Wroom 32E mit D0WD-V3 Chip (Eigene Aufnahme)	8
2.3	Arduino - IDE V2.0: Hier Installationsanleitung für Windows 10	10
2.4	Arduino - IDE V2.0: Ein Bord (Mikrocontroller) muss ausgewählt werden (blauer Rahmen).	11
2.5	Codevergleich für Peripherie- und Zentralgerät	14
2.6	Entwicklung und Kompatibilität verschiedener Bluetooth-Standards, siehe Townsend u. a. in [5], Kap. 1	15
2.7	GATT: „Generic Attribute“ definiert, wie ein BLE Dienst implementiert werden muss.	16
2.8	Beispiel Gyroskopsensor aus [3], S.174: Jeder Service & seine Charakteristiken müssen programmiert werden, hier in C++.	16
2.9	Fehler, Methode get_temperature() wird nicht erkannt, weil sie unterhalb der main()-Methode steht	17
2.10	Lösung: Methode muss vor loop(), oder hier main(), deklariert werden.	18

Inhaltsverzeichnis

1 Projektbeschreibung	5
1.1 Projektidee Radbahn Berlin	5
1.2 Projektaufgabe BLE	5
1.3 Projektpartner	5
1.4 Projektplan	6
2 Projektumsetzung	7
2.1 Erste Schritte / Planung des Vorgehens	7
2.2 Verwendete Hardware	7
2.2.1 Arduino Nano Sense 33 BLE (kurz: Arduino BLE)	7
2.2.2 Espressif ESP 32 WT32-Wroom 32E (kurz: ESP 32)	8
2.3 Verwendete Software	9
2.3.1 Arduino-IDE	9
2.3.2 Installation in Win10, Linux und Mac OS	9
2.4 Ansprechen der Controller	9
2.4.1 ESP 32 via Arduino-IDE	9
2.5 Literaturrecherche	11
2.5.1 Ein erster Lösungsansatz	12
2.6 Ein funktionierender Lösungsweg	12
2.6.1 Herleitung und Quelle	13
2.6.2 Reduktion des Originalcodes	13
2.7 Herausforderungen	13
2.7.1 BLE-Kenntnisse	14
2.7.2 C/C++ Programmierkenntnisse	17
2.7.3 Erkennen von Laufzeitfehlern	17
2.8 Ergebnis Fazit	18
A Anhang	19
A.1 Codevergleich Peripherie-Zentralgerät	19
A.2 Arduino-Code als BLE-Peripherie	25
A.3 ESP 32 Code für BLE Central	28
A.4 GitHub-Präsenz	31
B Quellen	32
Literaturquellen	32

Projektbeschreibung

1.1 Projektidee Radbahn Berlin

Die im Kurs „*Big Data Analytics*“ vergebenen Projektaufgaben standen im Zusammenhang zum Radbahn Berlin Projekt des „paper planes e.V.“

Zweck des Vereins sei es, so deren Internetpräsenz¹ „*gesellschaftliche und technologische Potenziale zu erforschen, die zu umwelt- und menschengerechteren und somit zu lebenswerteren Stadträumen führen.*“

Die Idee hinter der „Radbahn Berlin“ ist es, unter dem denkmalgeschützten Hochbahnviadukts der Berliner U-Bahn-Linie 1 einen Fahrstreifen anzulegen, der von Fahrrädern und ähnlichen Fahrzeugen genutzt werden kann.

1.2 Projektaufgabe BLE

Die im Kurs vergebenen Aufgaben wurden so ausgewählt, dass die Umsetzung der Aufgaben die Radbahn-Idee mit IoT-Projekten unterstützt.

So bestand die hier dokumentierte Projektaufgabe darin, zwei IoT Mikrocontroller per Bluetooth Low Energy (BLE) so miteinander zu verbinden, dass diese via BLE Daten austauschen können.

Dabei sollten die Vorgehensweise und Ergebnisse, aber auch erfolgte Lernfortschritte oder besondere Herausforderungen dokumentiert werden.

1.3 Projektpartner

Die konkrete Aufgabe erforderte keine Projektpartner. Es sei jedoch erwähnt, dass die Umsetzung der Aufgaben auch von der Technischen Universität Berlin (TUB) begleitet wurde und letztlich der Umsetzung der Radbahn-Idee des „paper planes e.V.“

¹ Vgl. <https://www.paper-planes.net/>

1.4 Projektplan

Da die Zeit für die Aufgabenerledigung vorgegeben, die Aufgabe klar² und das Zeitbudget zur Umsetzung nur eine Woche betrug, wurde kein Projektplan ausformuliert. Es war klar, dass der Sachstand der Umsetzung 24 Stunden vor der Abschlusspräsentation dokumentiert und dann vorgestellt würde - wie auch immer der Stand dann aussähe.

² „Verbinden Sie zwei Mikrocontroller per Bluetooth!“

Projektumsetzung

2.1 Erste Schritte / Planung des Vorgehens

Das Vorgehen bei der Lösungssuche wird in chronologischer Reihenfolge beschrieben. Dabei enthält die Dokumentation auch Lösungsansätze, die nicht zum Erfolg geführt haben.

Um die gestellte Aufgabe überhaupt lösen zu können, wurde das folgend beschriebene Vorgehen definiert:

- (1.) Verwendete Hardware beschreiben.
- (2.) Verwendete Software beschreiben.
- (3.) Ansprechen der Controller ermöglichen:
Wir verbinden beide Mikrocontroller mit einem PC und sprechen sie über die Arduino - IDE an.
- (4.) Recherche nach Quellen oder zu ähnlichen Projekten:
Wir studieren Quellen zu BLE, C/C++ Programmierung¹, um Problem und mögliche Lösungsansätze besser zu verstehen.
Wir suchen zeitgleich nach anderen Projektdokumentationen oder Tutorials, die dieselbe oder eine ähnlich Aufgabe umgesetzt haben, um unsere Lösung daran zu orientieren.
- (5.) Anwendung und Umsetzung der identifizierten Lösungswege.

2.2 Verwendete Hardware

Die Hardware wurde von der HTW für alle Projektgruppenmitglieder zur Verfügung gestellt. Es wurden die folgenden beiden Mikrocontroller verwendet.

2.2.1 Arduino Nano Sense 33 BLE (kurz: Arduino BLE)

Der Arduino Nano Sense 33 BLE ist anhand der auf dem Bord abgebildeten Typenbezeichnung gut zu identifizieren.

¹ Das ist die Codesprache, in der die Controller programmiert werden.

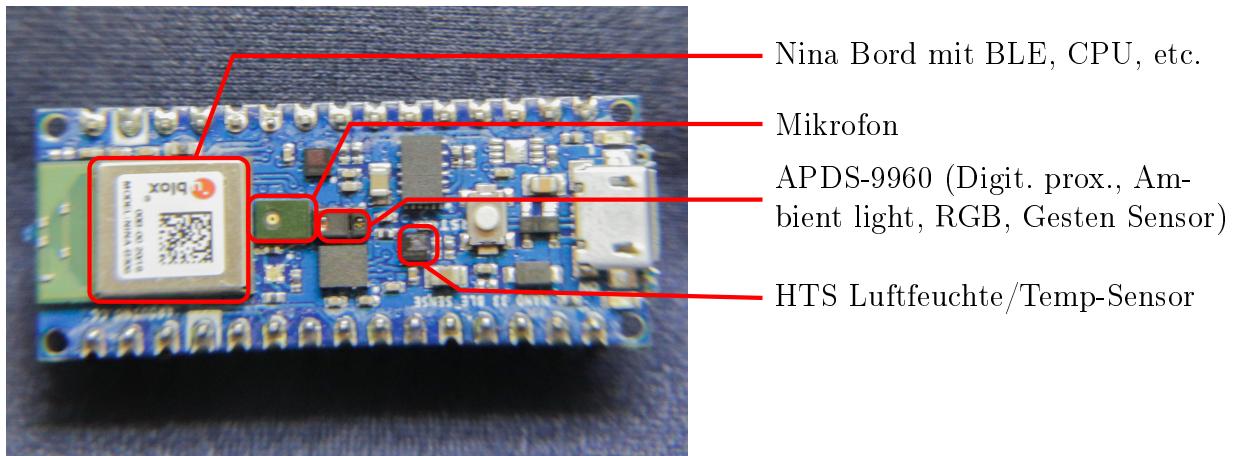


Abb. 2.1: Arduino BLE 33 Nano Sense mit Beschreibung ausgewählter Sensoren (Eigene Aufnahme)

2.2.2 Espressif ESP 32 WT32-Wroom 32E (kurz: ESP 32)

Laut Datenblatt² handelt es bei dem Bord um ein ESP 32-WROOM-32E (mit „on-board“ PCB Antenne) und bei dem Chip um einen ESP 32-D0WD-V3³. Laut ESP besitzt das Bord lediglich zwei sog. „on-chip“ Sensoren, nämlich einen Berührungs- und einen Magnetsensor.

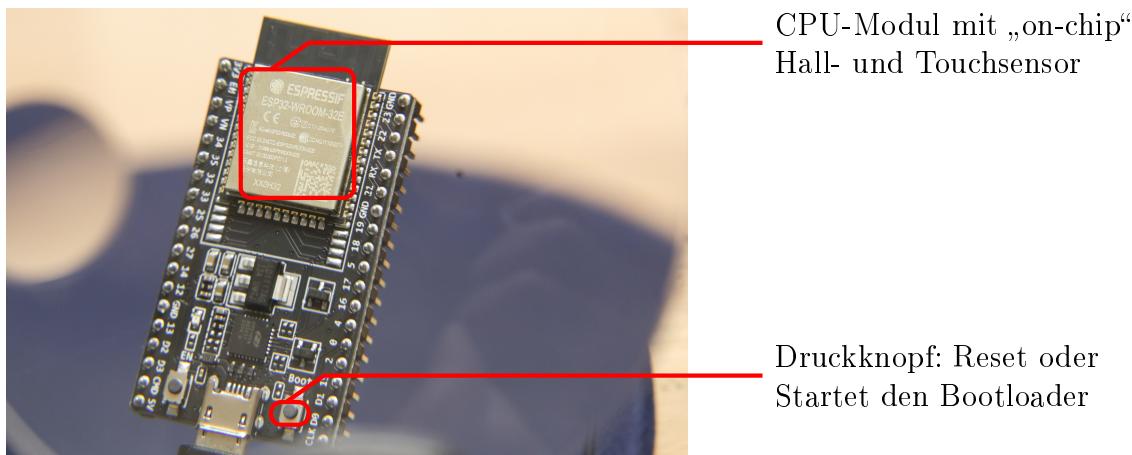


Abb. 2.2: Hardware: ESP 32 Wroom 32E mit D0WD-V3 Chip (Eigene Aufnahme)

² Vgl. auf <https://www.espressif.com/en/support/documents/technical-documents> das Dok „ESP 32-WROOM-32E & ESP 32-WROOM-32UE Datasheet“

³ Wird durch Upload-Ausgabe der Arduino-IDE bestätigt.

2.3 Verwendete Software

Zum Programmieren und Aufspielen des auszuführenden Codes wurde die Arduino Entwicklungsumgebung (IDE) genutzt. Mit ihr kann Code für beide Controller geschrieben und aufgespielt werden.

2.3.1 Arduino - IDE

Während in der Arduino - IDE eine eindeutige Softwarebibliothek für den Arduino BLE Controller vorhanden und installierbar ist⁴, ist die Identifizierung der korrekten Bibliothek für den ESP 32 schwieriger.

Da es sich zum einen nicht um einen Arduino-Controller handelt und zudem unzählige Modelle und Controllertypen existieren, ist für den genannten ESP 32-Controller kein passendes Bord auswählbar, auch nicht nach Installation der Espressif Bibliotheken. Wie genau der ESP 32 trotzdem mit der Arduino-IDE angesprochen werden kann, wird im Abschnitt 2.4 beschrieben.

2.3.2 Installation in Win10, Linux und Mac OS

Für die Installation der Arduino-IDE folgten wir den Anleitungen, so wie sie bspw. für Windows unter <https://docs.arduino.cc/software/ide-v1/tutorials/Windows> abrufbar sind. Die Installation für Mac OS oder Linux verlief problemlos. Daher wird vorgeschlagen, den auf arduino.cc veröffentlichten Anleitungen zu folgen.

2.4 Ansprechen der Controller

Der Code für beide Mikrocontroller wird in der Arduino-IDE entwickelt und von dort aus auf die Controller geladen. Beim ESP 32 ist das Ansprechen, also das Installieren der benötigten, korrekten Softwarebibliotheken etwas schwieriger als beim Arduino. Deswegen wird dieser Prozess im folgenden näher beschrieben.

2.4.1 ESP 32 via Arduino - IDE

Die folgende Anleitung beschreibt, wie der ESP 32-Controller mithilfe der Arduino-IDE programmiert werden kann:

- (1.) Schließe den ESP 32 per USB an einen PC.
- (2.) Installiere eine Softwarebibliothek, die zusammen mit dem ESP 32 harmoniert und funktioniert, z.B. die hier von Saumitra Jagdale⁵: <https://www>.

⁴ D. h. für genau diesen Typ ist ein Bord auswählbar. Vgl. Abb. 2.4

⁵ Artikel vom 23.04.2021

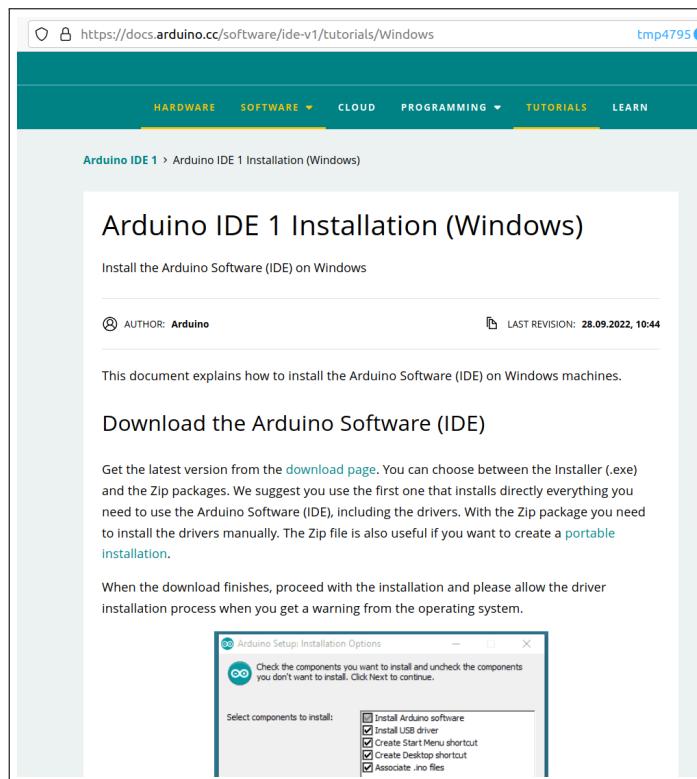


Abb. 2.3: Arduino - IDE V2.0: Hier Installationsanleitung für Windows 10

iottechtrends.com/getting-started-with-ESP32-wroom-devkitc/

Der wichtigste Schritt ist dabei, in der Arduino-IDE unter <Einstellungen> <Zusätzliche BordverwalterUrls> den folgenden Link einzufügen:

https://raw.githubusercontent.com/espressif/arduino-ESP32/gh-pages/package_ESP32_index.json

- (3.) Wähle anschließend im Menü <Bords> <Bord-Manager>. Gib im Suchfeld ESP oder ESP 32 ein.
- (4.) Installiere die Bibliothek ESP 32 von Espressif Systems.
- (5.) Wähle anschließend im Menü <Bord> das ESP 32-Wroom-DA-Modul⁶, und danach den Port, an dem der Controller angeschlossen ist.
- (6.) Der Controller sollte nun programmierbar sein, d.h. es sollten Codebeispiele (sog. Sketches) geladen werden können.

Zur Abgrenzung: Es existiert auch eine Anleitung, nach der der folgende, falsche oder nicht mit dem benutzten Bord harmonierenden Link hinzugefügt werden soll: https://d1.espressif.com/d1/package_ESP32_index.json. Mit dieser Bibliothek funktioniert der hier genutzte ESP 32 Mikrocontroller nicht!

⁶ Siehe Abb. 2.4, rechte Seite

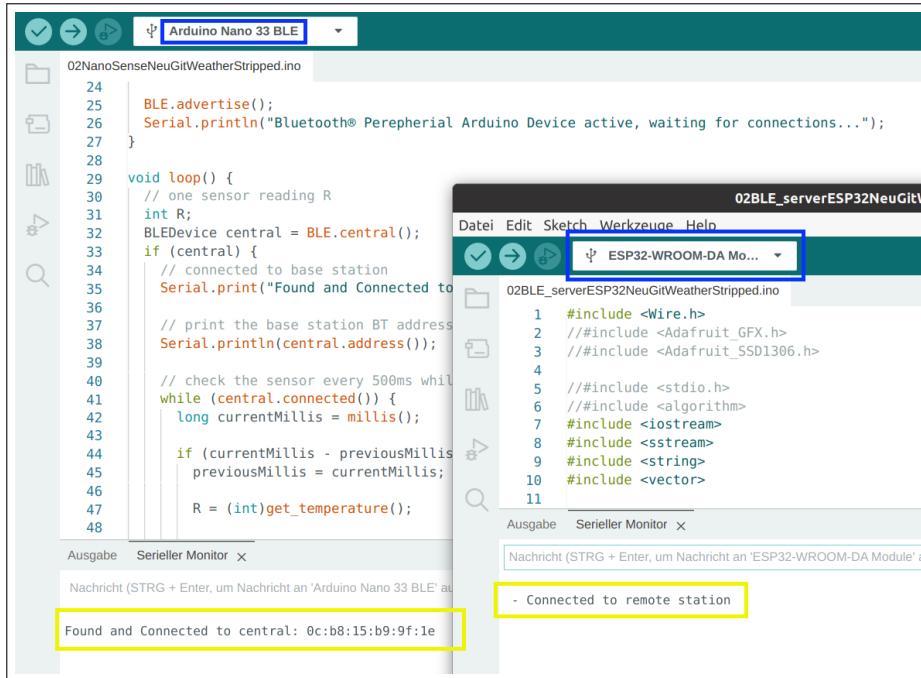


Abb. 2.4: Arduino - IDE V2.0: Ein Bord (Mikrocontroller) muss ausgewählt werden (blauer Rahmen).

Ergebniskontrolle

Wird auf dem so installierten ESP 32 bspw. das „*Wifi-Scan*“ Beispiel aus der IDE ausgeführt, so werden alle erreichbaren WLAN-Netze in der Umgebung angezeigt. Der ESP 32 funktioniert und kann für die zu erledigende Aufgabe programmiert werden.

2.5 Literaturrecherche

Grundsätzlich finden sich im Internet zur Schlagwortsuche „*ESP 32, Arduino Nano sense pairing BLE*“ kaum nützliche Hinweise. Als Treffer werden u. a. ausgegeben:

- Eine wenig hilfreiche Anfrage im Forum, ob dies möglich ist, <https://forum.arduino.cc/t/send-data-from-arduino-nano-33-ble-sense-to-ESP32-via-bluetooth/700275>
- Einen informativen Artikel zu ESP 32 BLE und zum Einrichten eines einfachen BLE-Scanners inkl. Codebeispiel,⁷
- Einen Arduino-Docs Artikel, wie man zwei Arduino über BLE verbindet,⁸

⁷ Vgl. bei Santos und Santos in [4]

⁸ Vgl. bei Bagur in [1]

Die Artikel informieren zwar zum Thema BLE, helfen jedoch bei der konkreten Problemlösung nicht weiter. Aber zumindest halfen sie bei der Entwicklung einer ersten Idee, wie die Verbindung zwischen dem Arduino-BLE und dem Espressif ESP 32 erreicht werden kann.

2.5.1 Ein erster Lösungsansatz

Die Ergebnisse der Literaturrecherche führten zu einer ersten Lösungsidee. So sollte aus der gefundenen ESP 32 Anleitung der Codeteil für den BLE-Server auf den ESP gespielt werden. Der ESP 32 wird also als BLE-Server oder Zentralgerät⁹ eingerichtet.

Wegen der Onboard-LEDs und wegen der zahlreichen Sensoren wird der Arduino BLE als Client oder Peripheriegerät programmiert. Dazu nehmen wir aus der Arduino-Anleitung den Teil zum Client und spielen ihn auf den Arduino.

Um miteinander via BLE interagieren zu können, müssen beide Codeteile aufeinander abgestimmt werden. Der Lösungsansatz, formuliert als Algorithmus, lautet also:

Vorgehen

- (1.) Installiere den Arduino als BLE-Peripherie- und den ESP als BLE-Central Device oder Server.
- (2.) Nutze für den Peripherie-Arduino den Code aus <https://docs.arduino.cc/tutorials/nano-33-ble-sense/ble-device-to-device>. an.
- (3.) Nutze für das BLE Central-Device den ESP 32 mit Code aus <https://randomnerdtutorials.com/ESP32-ble-server-client/#ESP32-BLE-Server>.
- (4.) Passe beide Codes so an, dass sie per BLE kommunizieren können.

Ergebnis

Mangels hinreichender C++ Kenntnisse und mangels eines tieferen Verständnisses der BLE Architektur und deren Implementierung als Code scheiterten die Versuche, beide grundsätzlich nicht aufeinander abgestimmt Codeteile passend zu machen.

2.6 Ein funktionierender Lösungsweg

Dem Gedanken „Zurück zum Reißbrett“, oder in unserem Fall „zurück zur Literaturrecherche“ folgend, haben wir nach diesem Fehlschlag eine weitere Suche gestartet, nur diesesmal explizit in Github.

⁹ Engl. Central Device

2.6.1 Herleitung und Quelle

Sucht man in Github nach „*Arduino Nano Sense ESP 32*“, so stößt man auf ein Projekt einer Studierenden der SRH Hochschule Heidelberg¹⁰, Anjali Pankan. Für ihr Projekt entwickelte sie eine Wetterstation, bei der Daten über BLE ausgetauscht werden. Als Controller nutzt sie -wie wir auch bei unserer Aufgabe- einen Arduino Nano Sense 33 BLE und einen ESP 32.

Da wir für unsere Aufgabe lediglich den Teil des Codes benötigten, der die BLE Übertragung ermöglicht, verfolgten wir im folgenden die Idee, den Open-Source Code von Anjali Pankan auf den Teil zu reduzieren, der für BLE notwendig ist, und diesen auf unsere Controller zu übertragen.

2.6.2 Reduktion des Originalcodes

Der Original-Code der Wetterstation ist umfangreich. Von den insgesamt 1.192 Zeilen, die der Code des Zentralgerätes umfasst, sind allein 300 Zeilen für die Definition der Piktogramme und 400 Zeilen zur Berechnung der Wettervorhersage bestimmt. Diese Codeteile werden für BLE nicht benötigt und wurden entfernt.

Als Ergebnis der Code-Reduktion wurden aus den 1.192 Zeilen des Originals rund 200 Zeilen für das Zentralgerät. Beim Client oder Peripheriegerät wurde der Code von 176 Originalzeilen auf für BLE benötigte 95 Codezeilen reduziert.

Wegen der Kürze der Bearbeitungszeit ist davon auszugehen, dass der Code weiter minimiert werden könnte. Es handelt sich bei dem hier vorgestellten Code also nicht um ein Minimalbeispiel.

Um beide Codeteile gut vergleichen zu können, wurden beide in einer Tabelle nebeneinander dargestellt und mit vereinzelten Kommentaren zu den BLE Komponenten versehen.

In Abb. 2.5 sind beide Codeteile auszugsweise nebeneinander dargestellt, um sie besser vergleichen zu können. Der gesamte Code ist im Anhang einsehbar¹¹.

Wird der Code auf beide Controller geladen, so erhebt der Arduino mit dem HTS221-Sensor die Temperatur und überträgt die Daten an die ESP 32 Zentralstation. Die gestellte Aufgabe ist somit gelöst.

2.7 Herausforderungen

Die Aufgabe bestand letztendlich darin, auf den Arduino und auf den ESP 32 einen Programmcode zu laden, der genau so in einem C-Dialekt programmiert ist, dass

¹⁰ <https://github.com/AnjaliPankan/Embedded-System-Weather-Station-using-Arduino-Nano-33-BLE-Sense-and-ESP32>

¹¹ Siehe Codevergleich im Anhang ??

Abb. 2.5: Codevergleich für Peripherie- und Zentralgerät

beide über BLE miteinander kommunizieren können.

Dazu sind folgende Fähigkeit notwendig:

- (1.) C/C++ Programmierkenntnisse
 - (2.) BLE - Kenntnisse
 - (3.) Möglichkeit zur Analyse von Laufzeitfehlern

Niemand in unserer Gruppe hatte C/C++ oder BLE-Kenntnisse. Wie man Laufzeitfehler mit der Arduino-IDE identifiziert und analysiert wissen wir bis heute nicht. Ich behaupte, dass dies nicht möglich ist, weil der Code in der IDE selbst nicht ausgeführt werden kann, sondern nur auf dem Controller selbst.

2.7.1 BLE-Kenntnisse

Um zwei Controller per BLE miteinander zu verknüpfen, bedarf es natürlich Kenntnisse darüber, was BLE ist und wie es grob funktioniert.

Hierzu existiert umfangreiche Literatur¹², die in der Kürze der zur Verfügung stehenden Zeit nur grob gesichtet wurde, woraus aber einige wichtige und notwendige Informationen gesammelt werden konnten.¹³

So ist z.B. die Entwicklung der Bluetooth-Standards ist in Abb. 2.6 dargestellt. Sie zeigt, dass der bekannte, originale oder klassische Standard zweimal weiterentwickelt wurde, und dass Bluetooth Classic mit Bluetooth Low Energy gar nicht mehr

¹² Siehe Literaturverzeichnis, hier: B

¹³ Vgl. bei Hoddie und Prader in [2], und Townsend u. a. in [5] sowie Kurniawan in [3].

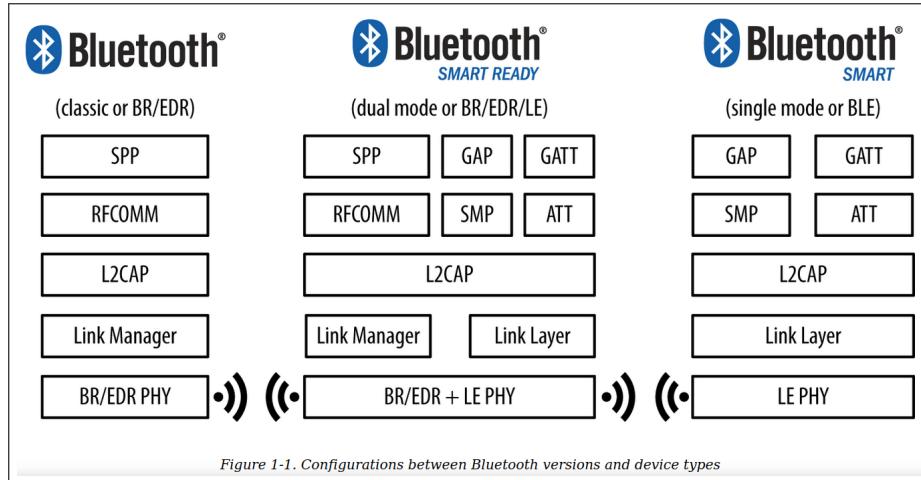


Abb. 2.6: Entwicklung und Kompatibilität verschiedener Bluetooth-Standards, siehe Townsend u. a. in [5], Kap. 1

kompatibel ist. Das bedeutet, dass ein für den klassischen Standard ausgelegtes Gerät mit einem BLE-Gerät nicht kompatibel ist, während ein „Smart-Ready“ oder Dual-Mode Gerät mit den beiden anderen Standards kommunikationsfähig ist.

Des Weiteren definiert das sog. GATT¹⁴ -wie in Abb. 2.7 dargestellt- den grundlegenden Aufbau des BLE-Datenformates.

Um einen BLE-Dienst implementieren zu können, müssen sog. Dienste und für die Dienste mind. eine Charakteristik definiert werden.

Beispiel

Will man die Werte zweier Sensoren übertragen -wie bspw. eines meteorologischen und eines Beschleunigungssensors, so empfiehlt es sich, dass zwei Dienste (Services) definiert werden, etwa:

- WeatherService, und
- ImuService (IMU - Inertial Measurement Unit: Bewegungsmesseinheit)

Sicherlich könnten auch sämtliche Werte in einem Service definiert und benannt werden - etwa „GenericService“. Diesem einen Dienst müssten dann aber sämtliche Charakteristiken, also die Sensorwerte, zugeordnet werden, was mit steigender Anzahl an Sensoren schnell unübersichtlich wird.

Dabei kann man sich einen Service als einen Ordner vorstellen, in dem die einzelnen Sensorwerte - die Charakteristiken - abgelegt werden.

Wie die Implementierung eines Services und seiner Charakteristiken auf einem Arduino Peripheriegerät aussieht, beschreibt Kurniawan in [3], dargestellt in Abb. 2.8. Dort wird ein IMU-Sensor implementiert. Da der Sensor die Beschleunigung in drei

¹⁴ Engl. „Generic Attribute“, etwa: Allgemeines Eigenschaftsmodell

CHAPTER 4 BLUETOOTH LOW ENERGY (BLE)

Profiles, Services, and Characteristics

GATT also defines the format of data, with a hierarchy of *profiles*, *services*, and *characteristics*. As illustrated in Figure 4-1, the top level of the hierarchy is a profile.

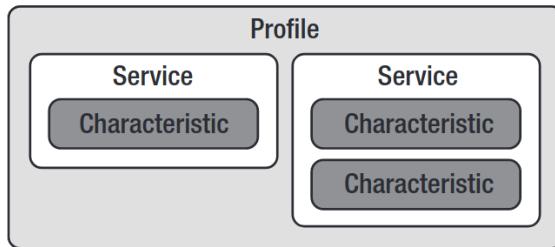


Figure 4-1. GATT profile hierarchy

Abb. 2.7: GATT: „Generic Attribute“ definiert, wie ein BLE Dienst implementiert werden muss.

Richtungen misst (in x -, y - und z -Richtung), müssen für den in der ersten Zeile definierten *sensorService* insgesamt drei Charakteristiken definiert werden (siehe gelbe Unterstreichungen in Abb. 2.8). Ansonsten könnten nicht alle Sensordaten vollständig übertragen werden.

```
BLEService sensorService("16150f38-e7a9-4fe1-ae08-48464baf25b2");
BLEStringCharacteristic xSensorLevel("ff99948c-18ff-4ed8-942e-
512b9b24b6da", BLERead | BLENotify,15);
BLEStringCharacteristic ySensorLevel("8084aa6b-6cae-461f-9540-
e1a5768de49d", BLERead | BLENotify,15);
BLEStringCharacteristic zSensorLevel("ab80cb77-fe74-40d8-9757-
96f8a54c16d9", BLERead | BLENotify,15);

// last sensor data
float oldXLevel = 0;
float oldYLevel = 0;
float oldZLevel = 0;
long previousMillis = 0;
```

Abb. 2.8: Beispiel Gyroskopsensor aus [3], S. 174: Jeder Service & seine Charakteristiken müssen programmiert werden, hier in C++.

```

1- *****
2- |          Online C++ Compiler.
3- |          Code, Compile, Run and Debug C++ program online.
4- |          Write your code in this editor and press "Run" button to compile and execute it.
5- *****
6
7 *****
8
9 #include <iostream>
10 using namespace std;
11
12
13
14
15 int main()
16 {
17     cout<<"Hello World";
18     int R = 0;
19     R = (int)get_temperature();
20     std::cout << "Temp = " << R << endl;
21     return 0;
22 }
23
24 // Get the temperature value.
25 float get_temperature()
26 {
27     float temperature = 7.65;
28
29     std::cout << "Temperature in getTemp() is "<<temperature<<endl;
30     return temperature;
31 }
```

Compilation failed due to following error(s).

```

main.cpp: In function 'int main()':
main.cpp:19:14: error: 'get_temperature' was not declared in this scope
  19 |     R = (int)get_temperature();
      |             ^~~~~~
```

Abb. 2.9: Fehler, Methode `get_temperature()` wird nicht erkannt, weil sie unterhalb der `main()`-Methode steht

2.7.2 C/C++ Programmierkenntnisse

Innerhalb ein bis zwei Wochen lassen sich keine tiefgreifende Kenntnisse in einer höheren Programmiersprache erwerben. Es ist jedoch hilfreich, wenn man andere Sprachen wie Java oder Python beherrscht. So war es uns zumindest möglich, einen in C/C++ verfassten und auf Github veröffentlichten Code so zu kürzen, dass er genau die gewünschten Operationen ausführt.

2.7.3 Erkennen von Laufzeitfehlern

Die Arduino-IDE ist unserer Auffassung nach zum Programmieren nur bedingt geeignet, weil in ihr keine Laufzeitfehler erkannt werden können. Das liegt daran, dass der Code in der IDE zwar beim Compilieren grammatisch/syntaktisch geprüft wird, jedoch nur auf dem Controller und nicht in der IDE ausgeführt werden kann.

So können Fehler zur Laufzeit nur daran erkannt werden, dass sich der Mikrocontroller „aufhängt“, wenn er z.B. in einer Schleife hängen bleibt. Dies kann bspw. passieren, wenn Methoden, die im Code nach der `loop()` Schleife definiert wurden, dem Interpreter nicht bekannt sind und deswegen nicht ausgeführt werden können.

Abb. 2.9 zeigt einen solchen Fehler in gekürzter Form in einem Online Compiler für C++. Als Ergebnis des gestarteten Programms erscheint eine Fehlermeldung (Rot). Der Controller „hängt“ oder würde hier hängenbleiben. Er müsste sogar neu gestartet werden, da er sonst nicht mehr ansprechbar wäre.

Die Lösung für dieses Problem ist in Abb. 2.10 dargestellt. Die Methode

Abb. 2.10: Lösung: Methode muss vor loop(), oder hier main(), deklariert werden.

`get_temperature()` muss vor der `main()`-Methode stehen, damit sie vom Compiler erkannt und ausgeführt werden kann. Der Controller würde sich somit nicht mehr „aufhängen“

2.8 Ergebnis Fazit

Die Aufgabe wurde erfolgreich erfüllt. Beide Controller tauschen Daten per BLE aus, hier Temperaturdaten des HTS221-Sensors des Arduino BLE-Peripheriegerätes.

Anhang

A.1 Codevergleich Peripherie-Zentralgerät

Das Dokument, das den Codevergleich enthält, beginnt aus Platzgründen auf der folgenden Seite.


```
// do nothing, as values are read explicitly
}

class WeatherClientCallback : public BLEClientCallbacks {
    void onConnect(BLEClient* pclient) {
        //do nothing
    }
    void onDisconnect(BLEClient* pclient) {
        connected = false;
        Serial.println("onDisconnect");
    }
};

//Connecting an in-range remote Device with the Server
- -> bool connectToServer() {
    Serial.print("Forming a connection to remote station");
    Serial.println(device->getAddress().toString().c_str());
    BLEClient* pClient = BLEDevice::createClient();
    Serial.println(" - Created client");

    pClient->setClientCallbacks(new WeatherClientCallback());
    // Connect to the remote BLE Device;
    pClient->connect(device);
    Serial.println(" - Connected to remote station");
    // Obtain a reference to the weather service in the remote BLE server.
    BLERemoteService* pRemoteService = pClient->getService(serviceUUID);
    if (pRemoteService == nullptr) {
        - -> Serial.print("Failed to find weather service UUID: ");
        Serial.println(serviceUUID.toString().c_str());
        pClient->disconnect();
        return false;
    }
}

Sucht nach einer Charakteristik des oben gefundenen 'weatherService' bzw. nach dessen UUID - ->
// Obtain a reference to the characteristic of weather service of the remote BLE server.
pRemoteCharacteristic = pRemoteService->getCharacteristic(charUUID);
if (pRemoteCharacteristic == nullptr) {
    Serial.print("Failed to find characteristic UUID: ");
    Serial.println(charUUID.toString().c_str());
    pClient->disconnect();
    return false;
}
Serial.println(" - Found a characteristic (a value)");


```

Eine entfernte Gerät befindet sich in Reichweite. Der Server versucht, das Gerät als Client zu verbinden. (Client bietet sich an, der Server versucht zu verbinden)

Prüft, ob der in der --> **If(pRemoteCharacteristic->canRead()** {
Charakteristik gespeicherte Wert
gelesen werden kann.
Wenn ja, wird der Wert
ausgegeben.

```
// Read the value of the characteristic.
if(pRemoteCharacteristic->canRead() {
    std::string value = pRemoteCharacteristic->readValue();
    Serial.print("The characteristic value was: ");
    Serial.println(value.c_str());
}

if(pRemoteCharacteristic->canNotify()
    pRemoteCharacteristic->registerForNotify(notifyCallback);

connected = true;
return true;
}

class WeatherAdvertisedDeviceCallbacks: public BLEAdvertisedDeviceCallbacks {
void onResult(BLEAdvertisedDevice advertisedDevice) {
    Serial.print("BLE Advertised Device found: ");
    Serial.println(advertisedDevice.toString().c_str());

    if (
        advertisedDevice.haveServiceUUID()&&advertisedDevice.isAdvertisingService(
            serviceUUID)
    ) {
        BLEDevice::getScan()->stop();
        device = new BLEAdvertisedDevice(advertisedDevice);
        doConnect = true;
        doScan = true;
    }
}
};

// Helper function to convert from string to integer vector.
std::vector<float> convertToFloatArray(std::string input) {
    std::replace(input.begin(), input.end(), ',', ' ');
    std::istringstream stringReader{ input };
    std::vector<float> result;

    float number;
    while (stringReader >> number) {
        result.push_back(number);
    }
    return result;
}
```

```

// The sensorData vector with index = 0 contains Temperature value.
int get_temperature(){
    return sensorData[0];
}

// Initialize remote station capabilities.
void setup() {
    Serial.begin(9600);
}

if (!BLE.begin()) {
    Serial.println("starting BLE failed!");
    while (1);
}

//initializing Temp Sensor
if (!HTS.begin()) {
    Serial.println("Failed to initialize HTS hum/temp sensor!");
    while (1);
}

//Setting up LED when connected to a central device.
pinMode(LED_BUILTIN, OUTPUT);
//BLE settings
BLE.setLocalName("WeatherMonitor");
BLE.setAdvertisedService(weatherService);
weatherService.addCharacteristic(weatherTemp);
BLE.addService(weatherService);
weatherTemp.writeValue(oldTemp);
//start advertising the Service
BLE.advertise();
Serial.println("Bluetooth® Peripheral Arduino Device active,
waiting for connections...");

}

void loop() {
    // one sensor reading R
    float R;
    // wait for a BLE central device
    BLEDevice central = BLE.central();

    if (central) {
        // connected to base station
        Serial.print("Found and Connected to central: ");
        // print the base station BT address:
        Serial.println(central.address());
    }
}

```

<-- Der oben definierte 'weatherService' und dessen Charakteristik werden auf dem Client angemeldet, mit einem alten Wert initialisiert. Der Client beginnt, den Service anzubieten ('advertise()').

```

void loop() {
    // Obtain a scanner and, do active scan and run for 5 seconds.
    BLEScan* pBLEScan = BLEDevice::getScan();
    pBLEScan->setAdvertisedDeviceCallbacks(new
        WeatherAdvertisedDeviceCallbacks());
    pBLEScan->setWindow(449);
    pBLEScan->setInterval(1349);
    pBLEScan->setActiveScan(true);

    pBLEScan->start(5, false);
}

void loop() {
    // Do the BLE connection and sensor data receive first.
    if (doConnect == true) {
        if (connectToServer()) {
            Serial.println("We are now connected to a remote BLE Device.");
        } else {
            Serial.println("We have failed to connect to a remote Device.");
        }
    }
}

<-- Bei Verbindung mit einem Zentralgerät beginnt der Client mit seiner Arbeit, hier:
1.) LED einschalten
2.) Temp-Sensor abfragen if (connected) {
3.) Temp-Wert updaten
}

```

```

und den Wert in der
Charakteristik
speichern.

// Turn on onboard LED's
digitalWrite(LED_R, LOW);
digitalWrite(LED_G, HIGH);
digitalWrite(LED_B, HIGH);
digitalWrite(LED_BUILTIN, HIGH);

// check the sensor every 500ms while connected 2 central:
while (central.connected()) {
    long currentMillis = millis();
    if (currentMillis - previousMillis >= 500) {
        previousMillis = currentMillis;
        Serial.print("Trying to get Temp Value... ");
        R = (float) get_temperature();
        Serial.print("Temp was found at ");
        Serial.print(R);
        Serial.println ("°C");
        updateTemp(R);
        Serial.println("");
    }
    digitalWrite(LED_R, HIGH);
    digitalWrite(LED_G, HIGH);
    digitalWrite(LED_B, HIGH);
    digitalWrite(LED_BUILTIN, LOW);
    Serial.print("Disconnected from central: ");
    Serial.println(central.address());
}

else {
    Serial.println("Disconnected from central!");
    delay(5000);
}

```

A.2 Arduino-Code als BLE-Peripherie

Original und unangepasst von <https://docs.arduino.cc/tutorials/nano-33-ble-sense/ble-device-to-device>. Für die Nutzung des ESP 32 Hall-Sensor muss der hier genannte Code noch angepasst werden.

```
/*
  BLE_Peripheral.ino

  This program uses the ArduinoBLE library to set-up an Arduino Nano
  33 BLE as a peripheral device and specifies a service and a characteristic.
  Depending of the value of the specified characteristic, an
  onboard LED gets on.

  The circuit:
  - Arduino Nano 33 BLE.

  This example code is in the public domain.
*/



#include <ArduinoBLE.h>

enum {
  GESTURE_NONE  = -1,
  GESTURE_UP    = 0,
  GESTURE_DOWN  = 1,
  GESTURE_LEFT  = 2,
  GESTURE_RIGHT = 3
};

const char* deviceServiceUuid =
"19b10000-e8f2-537e-4f6c-d104768a1214";
const char* deviceServiceCharacteristicUuid =
"19b10001-e8f2-537e-4f6c-d104768a1214";

int gesture = -1;

BLEService gestureService(deviceServiceUuid);
BLEByteCharacteristic gestureCharacteristic(deviceServiceCharacteristicUuid, BLERead


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

  pinMode(LED_R, OUTPUT);
  pinMode(LED_G, OUTPUT);
  pinMode(LED_B, OUTPUT);
  pinMode(LED_BUILTIN, OUTPUT);
```

```
digitalWrite(LED_R, HIGH);
digitalWrite(LED_G, HIGH);
digitalWrite(LED_B, HIGH);
digitalWrite(LED_BUILTIN, LOW);

if (!BLE.begin()) {
    Serial.println("- Starting Bluetooth® Low Energy module failed!");
    while (1);
}

BLE.setLocalName("Arduino Nano 33 BLE (Peripheral)");
BLE.setAdvertisedService(gestureService);
gestureService.addCharacteristic(gestureCharacteristic);
BLE.addService(gestureService);
gestureCharacteristic.writeValue(-1);
BLE.advertise();

Serial.println("Nano 33 BLE (Peripheral Device)");
Serial.println(" ");
}

void loop() {
    BLEDevice central = BLE.central();
    Serial.println("- Discovering central device...");
    delay(500);

    if (central) {
        Serial.println("* Connected to central device!");
        Serial.print("* Device MAC address: ");
        Serial.println(central.address());
        Serial.println(" ");

        while (central.connected()) {
            if (gestureCharacteristic.written()) {
                gesture = gestureCharacteristic.value();
                writeGesture(gesture);
            }
        }

        Serial.println("* Disconnected to central device!");
    }
}

void writeGesture(int gesture) {
    Serial.println("- Characteristic <gesture_type> has changed!");
```

```
switch (gesture) {
    case GESTURE_UP:
        Serial.println("* Actual value: UP (red LED on)");
        Serial.println(" ");
        digitalWrite(LED_R, LOW);
        digitalWrite(LED_G, HIGH);
        digitalWrite(LED_B, HIGH);
        digitalWrite(LED_BUILTIN, LOW);
        break;
    case GESTURE_DOWN:
        Serial.println("* Actual value: DOWN (green LED on)");
        Serial.println(" ");
        digitalWrite(LED_R, HIGH);
        digitalWrite(LED_G, LOW);
        digitalWrite(LED_B, HIGH);
        digitalWrite(LED_BUILTIN, LOW);
        break;
    case GESTURE_LEFT:
        Serial.println("* Actual value: LEFT (blue LED on)");
        Serial.println(" ");
        digitalWrite(LED_R, HIGH);
        digitalWrite(LED_G, HIGH);
        digitalWrite(LED_B, LOW);
        digitalWrite(LED_BUILTIN, LOW);
        break;
    case GESTURE_RIGHT:
        Serial.println("* Actual value: RIGHT (built-in LED on)");
        Serial.println(" ");
        digitalWrite(LED_R, HIGH);
        digitalWrite(LED_G, HIGH);
        digitalWrite(LED_B, HIGH);
        digitalWrite(LED_BUILTIN, HIGH);
        break;
    default:
        digitalWrite(LED_R, HIGH);
        digitalWrite(LED_G, HIGH);
        digitalWrite(LED_B, HIGH);
        digitalWrite(LED_BUILTIN, LOW);
        break;
}
}
```

A.3 ESP 32 Code für BLE Central

```
*****  
Rui Santos  
Complete instructions at https://RandomNerdTutorials.com/ESP32-ble-server-client/  
Permission is hereby granted, free of charge, to any person obtaining  
a copy of this software and associated documentation files.  
The above copyright notice and this permission notice shall be  
included in all copies or substantial portions of the Software.  
*****/  
  
#include <BLEDevice.h>  
#include <BLEServer.h>  
#include <BLEUtils.h>  
#include <BLE2902.h>  
#include <Wire.h>  
#include <Adafruit_Sensor.h>  
#include <Adafruit_BME280.h>  
  
//Default Temperature is in Celsius  
//Comment the next line for Temperature in Fahrenheit  
#define temperatureCelsius  
  
//BLE server name  
#define bleServerName "BME280_ESP32"  
  
Adafruit_BME280 bme; // I2C  
  
float temp;  
float tempF;  
float hum;  
  
// Timer variables  
unsigned long lastTime = 0;  
unsigned long timerDelay = 30000;  
  
bool deviceConnected = false;  
  
// See the following for generating UUIDs:  
// https://www.uuidgenerator.net/  
#define SERVICE_UUID "91bad492-b950-4226-aa2b-4ede9fa42f59"  
  
// Temperature Characteristic and Descriptor  
#ifdef temperatureCelsius  
BLECharacteristic bmeTemperatureCelsiusCharacteristics(  
    "cba1d466-344c-4be3-ab3f-189f80dd7518",  
    BLECharacteristic::PROPERTY_NOTIFY);
```

```
BLEDescriptor bmeTemperatureCelsiusDescriptor(  
    BLEUUID((uint16_t)0x2902));  
#else  
    BLECharacteristic bmeTemperatureFahrenheitCharacteristics(  
        "f78ebbf-fc8b7-4107-93de-889a6a06d408", BLECharacteristic::PROPERTY_NOTIFY);  
    BLEDescriptor bmeTemperatureFahrenheitDescriptor(  
        BLEUUID((uint16_t)0x2902));  
#endif  
  
// Humidity Characteristic and Descriptor  
BLECharacteristic bmeHumidityCharacteristics(  
    "ca73b3ba-39f6-4ab3-91ae-186dc9577d99", BLECharacteristic::PROPERTY_NOTIFY);  
BLEDescriptor bmeHumidityDescriptor(BLEUUID((uint16_t)0x2903));  
  
//Setup callbacks onConnect and onDisconnect  
class MyServerCallbacks: public BLEServerCallbacks {  
    void onConnect(BLEServer* pServer) {  
        deviceConnected = true;  
    };  
    void onDisconnect(BLEServer* pServer) {  
        deviceConnected = false;  
    }  
};  
  
void initBME(){  
    if (!bme.begin(0x76)) {  
        Serial.println("Could not find a valid BME280 sensor, check wiring!");  
        while (1);  
    }  
}  
  
void setup() {  
    // Start serial communication  
    Serial.begin(115200);  
  
    // Init BME Sensor  
    initBME();  
  
    // Create the BLE Device  
    BLEDevice::init(bleServerName);  
  
    // Create the BLE Server  
    BLEServer *pServer = BLEDevice::createServer();  
    pServer->setCallbacks(new MyServerCallbacks());  
  
    // Create the BLE Service  
    BLEService *bmeService = pServer->createService(SERVICE_UUID);
```

```
// Create BLE Characteristics and Create a BLE Descriptor
// Temperature
#ifndef temperatureCelsius
    bmeService->addCharacteristic(&bmeTemperatureCelsiusCharacteristics);
    bmeTemperatureCelsiusDescriptor.setValue("BME temperature Celsius");
    bmeTemperatureCelsiusCharacteristics.addDescriptor(
        &bmeTemperatureCelsiusDescriptor);
#else
    bmeService->addCharacteristic(
        &bmeTemperatureFahrenheitCharacteristics);
    bmeTemperatureFahrenheitDescriptor.setValue(
        "BME temperature Fahrenheit");
    bmeTemperatureFahrenheitCharacteristics.addDescriptor(
        &bmeTemperatureFahrenheitDescriptor);
#endif

// Humidity
bmeService->addCharacteristic(&bmeHumidityCharacteristics);
bmeHumidityDescriptor.setValue("BME humidity");
bmeHumidityCharacteristics.addDescriptor(new BLE2902());

// Start the service
bmeService->start();

// Start advertising
BLEAdvertising *pAdvertising = BLEDevice::getAdvertising();
pAdvertising->addServiceUUID(SERVICE_UUID);
pServer->getAdvertising()->start();
Serial.println("Waiting a client connection to notify...");

}

void loop() {
    if (deviceConnected) {
        if ((millis() - lastTime) > timerDelay) {
            // Read temperature as Celsius (the default)
            temp = bme.readTemperature();
            // Fahrenheit
            tempF = 1.8*temp +32;
            // Read humidity
            hum = bme.readHumidity();

            //Notify temperature reading from BME sensor
            #ifdef temperatureCelsius
                static char temperatureCTemp[6];
                dtostrf(temp, 6, 2, temperatureCTemp);
                //Set temperature Characteristic value and notify connected client
                bmeTemperatureCelsiusCharacteristics.setValue(temperatureCTemp);
                bmeTemperatureCelsiusCharacteristics.notify();
            #endif
        }
    }
}
```

```
    Serial.print("Temperature Celsius: ");
    Serial.print(temp);
    Serial.print(" °C");
#else
    static char temperatureFTemp[6];
    dtostrf(tempF, 6, 2, temperatureFTemp);
    //Set temperature Characteristic value and notify connected client
    bmeTemperatureFahrenheitCharacteristics.setValue(temperatureFTemp);
    bmeTemperatureFahrenheitCharacteristics.notify();
    Serial.print("Temperature Fahrenheit: ");
    Serial.print(tempF);
    Serial.print(" °F");
#endif

//Notify humidity reading from BME
static char humidityTemp[6];
dtostrf(hum, 6, 2, humidityTemp);
//Set humidity Characteristic value and notify connected client
bmeHumidityCharacteristics.setValue(humidityTemp);
bmeHumidityCharacteristics.notify();
Serial.print(" - Humidity: ");
Serial.print(hum);
Serial.println(" %");

lastTime = millis();
}
}
}
```

A.4 GitHub-Präsenz

Die beiden Programmcodes für das BLE-Zentral- und das Peripheriegerät sind in Github unter <https://github.com/leventarica/htw-nanosense> abrufbar, sofern die Lesenden entweder berechtigt oder das Repository für die Öffentlichkeit frei gegeben worden ist.

Quellen

Literaturquellen

- [1] José Bagur. *Connecting Nano 33 BLE Devices over Bluetooth*. 2021. URL: <https://docs.arduino.cc/tutorials/nano-33-ble-sense/ble-device-to-device>.
- [2] Peter Hoddie und Lizzie Prader. *IoT Development for ESP32 and ESP8266 with JavaScript*. 1. Ausg. Berkley, CA: Apress, 2020. ISBN: 978-1-4842-5070-9.
- [3] Agus Kurniawan. *Beginning Arduino Nano 33 IoT*. 1. Ausg. Berkley, CA: Apress, 2021. ISBN: 978-1-4842-6446-1.
- [4] Sara Santos und Rui Santos. *ESP32 BLE Server and Client*. 2021. URL: <https://randomnerdtutorials.com/esp32-ble-server-client/>.
- [5] Kevin Townsend u. a. *Getting Started with Bluetooth Low Energy*. 1. Ausg. Berkley, CA: O'Reilly Media, Inc., 2014. ISBN: 9781491949511. URL: <https://books.google.de/books?id=XjY4nwEACAAJ>.

Bildquellen

Alle dargestellten Bilder sind entweder Bildschirmfotos der heruntergeladenen Daten oder selbst erstellte Grafiken. Sonstige Bilder wurden am Ort mit Quellangaben versehen.