FAKULTÄT FÜR PHYSIK
Institut für Experimentelle Teilchenphysik

Konzeption eines Telemetriesenders zum Übertragen von Beschleunigungsdaten im Rahmen von PhyPiDAQ

Masterarbeit
von

Philipp Eckerle

am Institut für Experimentelle Teilchenphysik

Gutachter:	Prof. Dr. Günter Quast
Zweitgutachter:	Dr. Antje Bergmann

Bearbeitungszeit: 16.05.2022 – 09.11.2022

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

PhyPiDAQ (*Data Aquisition for Physics with Raspberry Pi*) ist ein Projekt zur transparenten, einfach verständlichen Datenerfassung mit einem Raspberry Pi. Mittels der Software-Anwendung von PhyPiDAQ können über den einfachen und günstigen Einplatinen-Computer Raspberry Pi Messdaten verschiedenster Sensoren ausgelesen, dargestellt und aufgezeichnet werden. Allgemeines Ziel ist, sowohl Schülern, Studenten als auch Lehrern im Physikunterricht einen einfachen und günstigen Zugang zu moderner Messtechnik und Datenerfassung zu geben – und zwar auf Grundlage allgemein verfügbarer und universell verwendbarer Hardware. Bisher wurden Sensoren entweder über Kabel oder über eine eigens für PhyPiDAQ entworfene Platine mit dem Raspberry Pi verbunden, was jedoch eine Vielzahl möglicher Messexperimente durch die räumliche Fixierung an die auswertende Einheit einschränkt. Deshalb soll in dieser Masterarbeit ein Telemetriesender entwickelt werden, der mit dem jeweiligen Sensor verbunden ist und dessen Daten drahtlos an den Raspberry Pi senden kann. Vor allem für Messungen an bewegten, rotierenden und beschleunigten Systemen, bei welchen physische Verbindungen wie Kabel hinderlich sind, wäre ein Telemetriesender für den Einsatz prädestiniert.

1.1. Bisherige Realisierungen im Schulkontext

1.1.1. Angebote von Lehrmittelfirmen

Bisher auf dem Markt verfügbare Realisierungen finden sich vorwiegend im Angebot der Lehrmittel-Firma Phywe; die dort vertriebenen Messsysteme der Reihe „Smart-Sense“ können – neben dem üblichen Kabelanschluss - optional Daten per Bluetooth an Auswertungsprogramme auf Laptops oder auch Smartphones übertragen [A1]. Gegenüber dem PhyPiDAQ-Projekt als nachteilig anzusehen ist hierbei, dass einzelne Messsysteme, jeweils im Kostenbereich von über hundert Euro, auf sehr eng eingegrenzte Messaufgaben spezialisiert sind. Als weiterer Nachteil zu erwähnen ist, dass Angebote wie von Phywe oft dem „Blackbox-Denken“ Vorschub leisten. Messinstrumente sind ergonomisch geformt und einfach per Knopfdruck bedienbar, jedoch bleiben die konkreten elektronisch verarbeitenden Elemente unter dem Plastikgehäuse verborgen. Dies führt zur Ausblendung der technischen Grundlagen der physikalischen Messung und gibt den fälschlichen Eindruck einer speziellen, für Messversuche entwickelten Technik, obwohl es gerade didaktisch angeraten wäre, die universelle Verwendung von Sensoren und Mikrocontrollern sowohl im Kontext einer physikalischen Messung als auch in allerlei technischen Anwendungen des Alltags deutlich zu machen.

1.1.2. Die App Phyphox

Eine weitere Möglichkeit der drahtlosen Übertragung und Verarbeitung von physikalischen Messdaten bietet die vom zweiten physikalischen Institut der RWTH Aachen entwickelte App Phyphox. Diese App nutzt die verschiedenen in heutigen Smartphones verbauten

Abbildung 1.1.: Drahtloser Drei-Achsen-Beschleunigungssensor von Phywe [B30]

Sensoren, um physikalische Messdaten aufzunehmen und auf diesem Wege einfache Messexperimente zu ermöglichen. Über die Funktion der „Fernsteuerung“ kann über den Webbrowser eines Rechners auf die Benutzeroberfläche der App zugegriffen werden, wenn sich beide Geräte im selben WLAN-Netz aufhalten. Somit können vom Rechner aus Messungen gestartet, gestoppt, Messdaten aufgenommen und heruntergeladen werden können. Das Smartphone dient so als Telemetriesender, welcher Daten drahtlos an den Rechner als verarbeitende Einheit überträgt [A2]. Zentraler Nachteil dieser Implementierung ist, dass es sich bei Smartphones um sehr persönliche Gegenstände handelt, auf denen sich ein beträchtlicher Teil des Privatlebens abspielt. Neben den Kosten eines Smartphones und der möglichen Gefahr einer Beschädigung bei Versuchsaufbau und Durchführung schränkt dies die Verwendung eines Mobiltelefons für Experimente im schulischen Kontext ein. Viele heutige Smartphone-Modelle haben zudem eine Größe und ein Gewicht, die für manche Versuchsaufbauten unpraktikabel sind. Oft dürfte die Nutzung von Smartphones auch mit dem Handyverbot an vielen Schulen kollidieren.

1.1.3. Bisherige Realisierungsversuche im Rahmen von PhyPiDAQ

In der Masterarbeit „Konzeption und Gestaltung eines digitalen Messwerterfassungssystems für den Physikunterricht in der Schule“ von Moritz Aupperle wurde bereits der Realisierungsversuch eines Telemetriesenders durchgeführt [A3]. Als Steuereinheit wurde ein Arduino-Nano mit einem ATmega328-Prozessor verwendet, welcher über den digitalen und kalibrierten Beschleunigungssensor MM8451 Daten ausliest und diese über das 2,4 GHz-Transceiver-Modul nRF24L01 an den Raspberry Pi überträgt. Um die Daten empfangen zu können, muss der Raspberry Pi ebenfalls mit einem entsprechenden Transceiver-Modul verbunden werden. Zur mobilen Stromversorgung des Telemetriesenders wurde ein Lithium-Polymer-Akkumulator verwendet, der zunächst über ein Lademodul mit einem Hochsetzsteller verbunden wird, welcher die Spannung des Akkus von 3,7 V auf die nötige Spannung hochstellt, um den Mikrocontroller über die Pinanschlüsse betreiben zu können. Über einen Schalter kann der Hochsetzsteller – und damit der gesamte Telemetriesender – an- und ausgeschaltet werden. Um mögliche Spannungsschwankungen auszugleichen und so für einen verlässlichen Betrieb des Transceivers zu sorgen, wird noch ein Elektrolyt-Kondensator parallel zur Spannungsversorgung des Funkmoduls geschaltet. Abbildung 1.2 veranschaulicht den Aufbau.

Die Schwäche dieser Realisierung liegt hauptsächlich in ihrer hohen Komplexität, die auch in der Masterarbeit von Aupperle eingestanden wird. Insgesamt neun technische Elemente – einschließlich des Transceiver-Moduls für den Raspberry Pi – werden für den Bau des

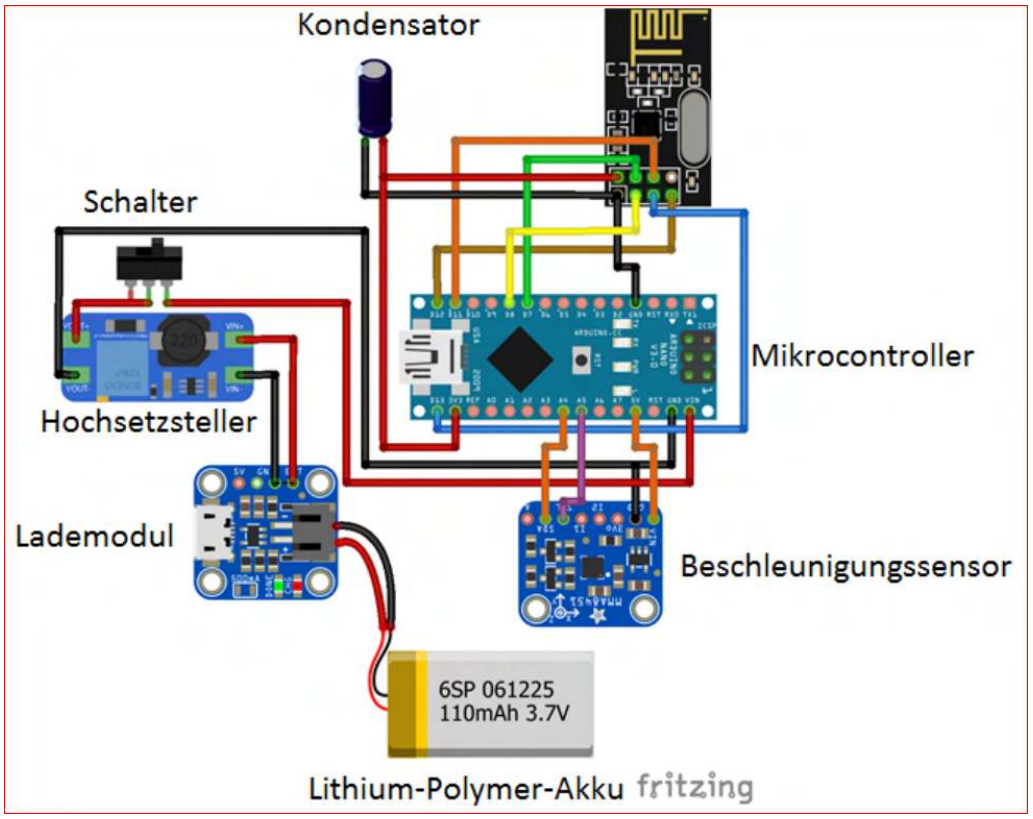

Abbildung 1.2.: Telemetriesender-Realisierung von Aupperle [B31]

Telemetriesenders benötigt. Davon müssen sieben Elemente auf korrekte Weise über ihre Pinanschlüsse mit einander verkabelt werden, wobei insgesamt 27 Leitungsenden an Pins angelötet werden müssen. Für Lehrkräfte stellt dies oftmals einen höheren handwerklichen und auch zeitlichen Aufwand dar. Für Schüler schlägt Aupperle eine Beschäftigung mit dem Telemetriesender nur im Rahmen größerer Projektarbeiten, auch im Verbund mit dem Fach Computertechnik an Technischen Gymnasien, vor. Trotz der vielen Bauelemente kann der gesamte Aufbau in einer gewöhnlichen Tic-Tac-Dose verpackt werden. Ein hierfür entsprechend klein gewählter Akkumulator dürfte durch die gleichzeitige Versorgung von Mikrocontroller, Beschleunigungssensor und Funkmodul jedoch stärker belastet werden und ein häufiges Nachladen nötig machen, wozu der Aufbau aber aus der Tic-Tac-Packung herausgenommen werden muss.

1.2. Zielsetzungen

Ausgehend von den eben behandelten, bisherigen Angeboten für Telemetriesender im Rahmen der Physikdidaktik und ihren Schwächen, soll eine möglichst optimale Telemetriesender-Lösung als Erweiterung für das PhyPiDAQ-System implementiert werden. Die Anforderungen an diese Lösung lassen sich dabei in eine praktische und eine technische Ebene unterscheiden.

Auf der praktischen Ebene soll die benötigte Hardware für den Telemetriesender möglichst einfach und günstig zu beziehen sein, und weiter einen minimalen Implementierungsaufwand benötigen. Ein aufwändiges Verkabeln und Verlöten verschiedener Elemente wie in der Telemetriesender-Lösung von Aupperle stellt gegenüber (erheblich teureren) Fertiglösungen wie von Phywe eine Anwendungshürde dar und soll vermieden werden. Allgemein soll sich der Telemetriesender hinsichtlich Größe und Form für klassische Experimente im Rahmen des Physikunterrichts eignen und einfach bedienbar sein.

Auf der technischen Ebene soll die Einheit aus Telemetriesender und Sensor sowohl den zu messenden physikalischen Wert als auch dessen zeitlichen Verlauf mit einer geeigneten Auflösung übertragen können. Die Funkverbindung zum Raspberry Pi soll möglichst einfach aufzubauen sein, und über möglichst große Distanzen Daten zuverlässig übertragen. Ferner ist eine möglichst stromsparende Lösung wünschenswert, die die Stromversorgung in Form eines Akkumulators oder mehrerer Batterien in einer Halterung nicht zu stark belastet.

Besonders hinsichtlich der technischen Anforderungen ist zu betonen, dass die einzelnen Anforderungen nicht voneinander unabhängig sind, sondern eine oft negative Abhängigkeit eines Faktors gegenüber dem anderen besteht. So führt eine größere Reichweite allgemein auch zu einem höheren Stromverbrauch. Ein leistungsfähigerer Telemetriesender auf der einen Seite erfordert auf der anderen Seite höhere Anschaffungskosten. Insgesamt soll bei der Konzeption des Telemetriesenders die Devise befolgt werden, mit minimalen Anschaffungskosten und minimalem Hardware-Aufwand eine für die anvisierten Zwecke optimale technische Lösung zu finden.

Die zentrale Hauptanwendung des Telemetriesenders soll das drahtlose Übertragen von Beschleunigungsdaten sein, die durch das Anbringen des Telemetriesenders an bewegten und rotierenden Objekten gewonnen werden. Diese Daten sollen von dem Raspberry Pi empfangen werden und mithilfe der PhyPiDAQ-Software in gewohnter Weise visualisiert und verarbeitet werden können. Darüber hinaus soll auch die Möglichkeit bestehen, andere Sensoren an den Telemetriesender anzuschließen.

2. Planung und Grundlagen

Die Planung zur Implementierung eines Telemetriesenders lässt sich in zwei Schritte unterteilen: der Wahl einer für die Zielsetzung geeigneten Hardware und der Wahl eines geeigneten drahtlosen Übertragungsverfahrens vom Telemetriesender hin zum Raspberry Pi.

2.1. Hardware

Als Hardware-Grundlage für Telemetriesender bieten sich Mikrocontroller-Entwicklungsboards mit integrierten Funk-Modulen an, die von verschiedenen Herstellern in jeweils unterschiedlichen Ausführungen angeboten werden. Gegenüber der Lösung von Aupperle, in welcher ein Funkmodul separat an einen Mikrocontroller angeschlossen wird, stellt dies bereits eine deutliche Verringerung des Hardware-Aufwands dar.

2.2. Frequenzbänder

Bei der Wahl eines drahtlosen Übertragungsverfahrens muss zunächst gewählt werden, welches Frequenzband verwendet werden soll und welche Vor- und Nachteile dieses mit sich bringt. Die Nutzung von Funkübertragung ist allgemein durch die Zuteilung sogenannter ISM-Bänder (*Industrial, Scientifical and Medical Band*) geregelt. Damit werden Frequenzbereiche bezeichnet, die durch Hochfrequenzgeräte in Industrie, Wissenschaft, Medizin oder auch häuslichen Bereichen lizenzfrei genutzt werden können. Besondere Relevanz haben das 433 MHz-ISM-Band (von 433,05 bis 434,79 MHz), das 902 MHz-ISM-Band (von 902 bis 928 MHz, nur in Nord- und Südamerika), sowie das 2,4 GHz-ISM-Band (von 2,4 bis 2,5 GHz). Parallel zur ISM-Band-Einteilung existiert eine Einteilung in SRD-Bänder (SRD = *Short Range Device*), die sich speziell auf Geräte mit kurzer Reichweite bezieht. Diese ist in weiten Teilen deckungsgleich mit der ISM-Band-Einteilung, enthält aber als praktischen Ersatz des nur in Nord- und Südamerika definierten 902 MHz-ISM-Bandes das 863 MHz-SRD-Band von 863 bis 870 MHz [A4].

Die Nutzung niederer Frequenzbänder wie des 433 MHz-ISM-Bandes bietet den Vorteil einer größeren Reichweite, da elektromagnetische Wellen größerer Wellenlänge weniger mit umgebender Materie wechselwirken und zum Beispiel Wände besser durchdringen. Allerdings reduziert die geringere Frequenz auch die übertragbare Datenmenge pro Zeit. Der hauptsächliche Nachteil besteht aber in einem zusätzlichen Hardwareaufwand, da am Raspberry Pi ein entsprechendes Empfangsmodul angeschlossen werden muss. Demgegenüber bietet das 2,4 GHz-ISM-Band mehrere Vorteile: Dieses wird sowohl für WLAN- als auch Bluetooth-Übertragung genutzt und genießt so die beste Hardware-Unterstützung. So können die integrierten WLAN- und Bluetooth-Schnittstellen des Raspberry Pi's (ab Model 3B) genutzt werden; ebenso werden auf Mikrocontrollern mit integrierten Funkmodulen meist WLAN- und Bluetooth-Schnittstellen verwendet. Daher soll im Rahmen dieser Arbeit der Fokus auf die Datenübertragung via WLAN und Bluetooth gelegt werden.

2.3. Übertragungsverfahren

Zwischen WLAN und Bluetooth bestehen dabei deutliche Unterschiede mit entsprechenden Vor- und Nachteilen. Bluetooth ist ein Punkt-zu-Punkt-Übertragungsverfahren zwischen zwei Geräten; die direkte Kommunikation erfordert einen komplexeren Verbindungsaufbau sowie eine aufwendigere Abstimmung der Geräte aufeinander. Demgegenüber bezeichnet das Akronym WLAN (*Wireless Local Area Network*) zunächst ein drahtloses lokales Netzwerk mit einem drahtlosen Zugangspunkt (*Wireless Access Point*) als zentrale verwaltende Einheit, oft in Form eines Routers. Geräte, die über WLAN mit einander kommunizieren wollen, müssen sich hierzu in einem gemeinsamen WLAN-Netzwerk befinden und sich mit einem Zugangspunkt verbinden, welcher die Nachricht eines Senders empfängt und diese an den Empfänger weiterleitet. Eine direkte Abstimmung beider Geräte aufeinander entfällt so und vereinfacht den Verbindungsaufbau, jedoch macht dies die Kommunikation zwischen zwei Geräten von der Existenz eines WLAN-Netzes abhängig.

Damit eine Kommunikation über WLAN stattfinden kann, muss weiter ein geeignetes Übertragungsprotokoll gewählt werden. Dieses legt den Aufbau einer Nachricht und den genauen organisatorischen Ablauf der Datenübertragung zwischen Sender und Empfänger fest. Die meiste Verwendung und die beste Unterstützung finden die Protokolle HTTP und MQTT. HTTP steht für *Hyper Text Transfer Protokoll* und wird allgemein zur Übertragung von Website-Daten genutzt, kann aber auch für IoT-Anwendungen verwendet werden. Allerdings ist HTTP nicht für die Nutzung durch Geräte mit geringen Ressourcen oder eine kontinuierliche Datenübertragung ausgelegt. Gegenüber MQTT ist der Anteil an Verwaltungsdaten, dem sogenannten Overhead, deutlich größer. Ebenso muss im Allgemeinen für jede Übertragung eines Datenpakets die Verbindung neu aufgebaut werden. Zwar gibt es hierbei die Möglichkeit, eine sogenannte Persistent Session einzurichten, um die Verbindung offenzuhalten, was jedoch von vielen einfachen HTTP-Implementierungen für Mikrocontroller nicht unterstützt wird. Ein weiterer Nachteil ergibt sich daraus, dass ein HTTP-Client zwar Daten senden, aber nicht ohne eigene Initiative Daten empfangen kann. Um Nachrichten empfangen zu können, muss der HTTP-Client in regelmäßigen Abständen Anfragen an den Server senden, was sowohl energie- als auch rechenintensiv ist [A5]. Insgesamt führen die genannten Nachteile zu einer deutlich geringeren Effizienz und Übertragungsrate von HTTP, weshalb in dieser Arbeit für die WLAN-Kommunikation das Protokoll MQTT verwendet wird, welches sich durch besondere Leichtgewichtigkeit und Einfachheit auszeichnet und darüber hinaus viele weitere Funktionalitäten bietet. Eine Ausführliche Behandlung der Übertragung über das MQTT-Protokoll und über Bluetooth findet sich in den Anhängen A und B.

3. Auswahl der Hardware

Für die Entwicklung von Telemetriesendern bieten sich Mikrocontroller-Boards mit integrierten Modulen zur drahtlosen Kommunikation an, die sich über eine USB-Schnittstelle programmieren lassen. Über I^2C- und SPI-Schnittstellen können beliebige Sensoren mit dem Board verbunden und von diesem angesteuert und ausgelesen werden, oder aber die benötigten Sensoren sind bereits im Board integriert, was den Hardwareaufwand weiter verringert und für Telemetrie-Anwendungen die effektivste Lösung darstellt. Die folgenden Abschnitte geben einen Überblick über WLAN- und Bluetoothfähige Mikrocontroller und ihre Anbieter, um auf dieser Grundlage schließlich die beste Wahl für die Implementierung eines Telemetriesenders zu begründen.

3.1. Angebot auf dem Markt

3.1.1. ESP-Entwicklungsboards

Zu einem namhaften Hersteller von WLAN- und Bluetooth-Modulen hat sich das Unternehmen Espressif Systems entwickelt. Unter den Bezeichnungen ESP8266, ESP32, ESP32-C und ESP32-S werden verschiedene Serien von Prozessoren mit integrierten WLAN- und Bluetooth-Modulen vertrieben. Auf Grundlage der ESP-Module entwickelt und vertreibt Espressif Systems auch eigene Entwicklungsboards (Abbildung 3.1), die mittels einer Mikro-USB-Schnittstelle programmiert werden können und eine Kommunikation mit dem ESP-Prozessor über diverse GPIO-Pins und Schnittstellen erlauben. ESP-basierte Entwicklungsboards sind häufig auch unter dem Namenszusatz NodeMCU bekannt. Ein speziell auf ESP-Entwicklungsboards spezialisierter Onlinehändler ist AZ-Delivery. ESP-Module werden auch von anderen Mikrocontroller-Herstellern wie Adafruit zur Bereitstellung von WLAN- und Bluetooth-Funktionalität in ihren Modellen verwendet.

3.1.2. Boards mit integrierten Funkmodulen von Adafruit

Der Hersteller Adafruit bietet unter der Serie „Adafruit Feather“ eine große Anzahl verschiedener Entwicklungsboards mit integrierten Funkmodulen an. Als vorrangiges Beispiel ist das Huzzah32-Entwicklungsboard zu nennen, welches mit einem ESP32-Modul ausgestattet ist und damit WLAN, Bluetooth und Bluetooth Low Energy[1] unterstützt [A6]. Boards mit ESP8266-Modulen verfügen nur über WLAN-Konnektivität. Als Alternative jenseits des 2,4 GHz-Bandes wird das Modell Feather-M0-RFM69HCW-Packet-Radio angeboten, welches wahlweise mit einem Funkmodul für das 433 MHz-Band oder das 868 MHz-Band ausgestattet ist [A7]. Von besonderer Bedeutung für die Anwendung als Telemetriesender ist das Modell Feather-nRF52840-Sense (Abbildung 3.2) [A8]. Dieses speziell für Messaufgaben konzipierte Board verfügt über eine Vielzahl fest auf dem Board verbauter Sensoren, darunter sind:

- ein 3-Achsen-Gyrometer und Accelerometer

[1] Auf niedrigen Energieverbrauch angepasste Variante von Bluetooth.

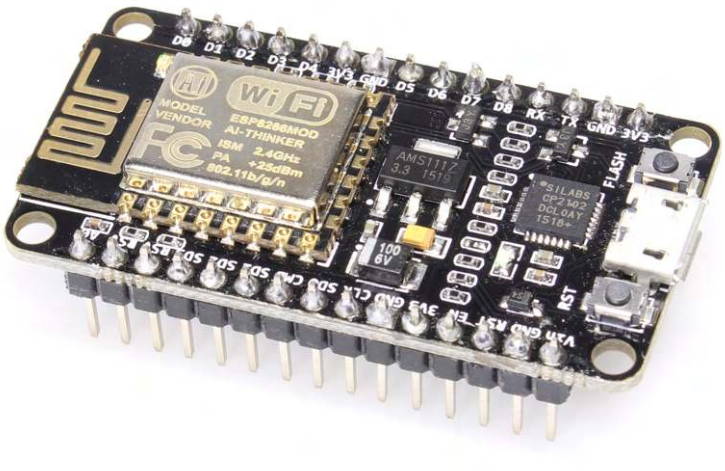

Abbildung 3.1.: ESP8266 NodeMCU-Board [B32]

- ein 3-Achsen-Magnetometer
- ein Licht-, Gesten- und Entfernungssensor
- ein Feuchtigkeitssensor
- ein Druck- und Temperatursensor
- ein Geräuschsensor (Mikrofon).

Als zentrale Verarbeitungseinheit wird der Nordic-nRF52840-Bluetooth-LE-Prozessor verwendet, der eine Bluetooth-LE-Schnittstelle zur drahtlosen Übertragung der Sensordaten beinhaltet. Alle Boards der Adafruit-Feather-Serie wurden mit dem Ziel einer möglichst guten Handlichkeit und Portabilität konzipiert und verfügen so über einen Anschluss für Lithium-Polymer-Akkumulatoren, die als Zubehör von Adafruit angeboten werden. Wird das Board über die Mikro-USB-Schnittstelle an einen Rechner angeschlossen, stellt die interne Laderegelung die Stromversorgung des Boards automatisch auf den USB-Anschluss um und lädt parallel den angeschlossenen Akkumulator auf. Wird die USB-Verbindung getrennt, wird die Stromversorgung automatisch auf den Akkumulator umgestellt.

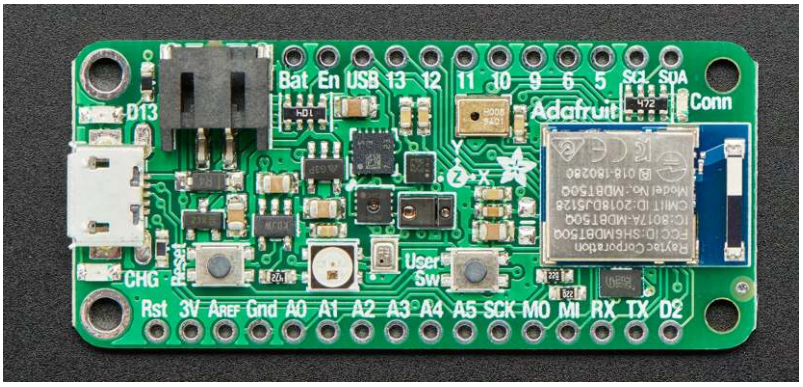

Abbildung 3.2.: Adafruit Feather nRF52840 Sense [B33]

3.1.3. Boards mit integrierten Funkmodulen von Arduino

In Konkurrenz zur Adafruit-Feather-Serie bietet der Hersteller Arduino unter seiner Arduino-Nano-Serie mehrere Boards mit ähnlicher Form und integrierter WLAN- oder Bluetooth-Konnektivität an, die zusätzlich über einen integrierten Beschleunigungssensor verfügen und sich daher grundsätzlich für die Nutzung als Telemetriesender zum Übertragen von Beschleunigungsdaten eignen. Im Gegensatz zu Adafruit setzen die im folgenden näher erläuterten Arduino-Nano-Boards auf Funkmodule des Herstellers u-blox.

Arduino-Nano-33-IoT

Das älteste Modell der Reihe, Arduino-Nano-33-IoT, ist mit einem NINA-W10-Modul ausgestattet und kann dadurch über WLAN, Bluetooth und Bluetooth Low Energy kommunizieren, wodurch es sich durch eine besondere Auswahl an Möglichkeiten zur Datenübertragung auszeichnet. Auf dem Board mitverbaut ist eine IMU (*Internal Measuring Unit*), die ein 3-Achsen-Accelerometer und ein 3-Achsen-Gyrometer beinhaltet [A9].

Arduino-Nano-33-BLE

Das Modell Arduino-Nano-33-BLE ist mit einem NINA-B306-Modul ausgestattet und verfügt dadurch nur über die Fähigkeit zur Kommunikation über Bluetooth Low Energy. Die IMU hingegen enthält zusätzlich zu einem 3-Achsen-Accelerometer und einem 3-Achsen-Gyrometer auch ein 3-Achsen-Magnetometer, was Anwendungsmöglichkeiten im Bereich der Navigation erweitert [A10].

Arduino-Nano-33-BLE-Sense

Das Modell Arduino-Nano-33-BLE-Sense stellt eine Erweiterung des obigen Modells dar, indem neben dem identischen Funkmodul und der IMU noch eine Reihe weiterer Sensoren auf dem Board verbaut sind; darunter ein Druck- und Temperatursensor, ein Feuchtigkeitssensor, ein Entfernungs-, Licht- und Gestensensor sowie ein Mikrofon. Hinsichtlich seiner Konnektivität und Sensorausstattung stellt das Arduino-Nano-33-BLE-Sense also ein Pendant zum Adafruit-Feather-nRF52840-Sense dar, allerdings ohne eine spezielle Vorrichtung für die Stromversorgung über einen Akkumulator [A11].

Arduino-Nano-RP2040-Connect

Das jüngste Modell der Serie ist das Arduino-Nano-RP2040-Connect, das sich durch die Verwendung des neuen, von der Raspberry-Pi-Foundation entwickelten und im Januar 2021 vorgestellten RP2040-Prozessors auszeichnet. Gegenüber vorangegangenen Arduino-Nano-Boards verfügt dieses Modell so über eine höhere Taktrate, einen deutlich größeren Flashspeicher und bietet besondere Unterstützung für die Programmierung in Micropython und Circuitpython. Somit überträgt es die Spezifikationen des mit dem RP-2040-Prozessor ausgestatteten Raspberry-Pi-Pico-Microcontrollers auf ein Arduino-Board und ergänzt diese mit zusätzlicher Hardware[2]. Mit auf dem Board verbaut ist ein Nina-W102-Modul, welches WLAN- und Bluetooth-Konnektivität bereitstellt. Wie alle zuvor genannten Boards verfügt auch dieses über eine IMU mit einem Integrierten 3-Achsen-Accelerometer und einem 3-Achsen-Gyrometer, allerdings ohne ein Magnetometer, welches häufig in Kombination dazu verwendet wird. Stattdessen verfügt die IMU über einen Kern für maschinelles Lernen, der dafür genutzt werden kann, Muster in Beschleunigungsdaten zu erkennen und dadurch spezielle Bewegungsabläufe wie Schritte zu detektieren und zu zählen. Ferner beinhaltet

[2] Wohl als Reaktion auf das Board Arduino-Nano-RP2040-Connect hat die Raspberry-Pi-Foundation im Jahr 2022 den Mikrocontroller Raspberry-Pi-Pico-W eingeführt, welcher den bisherigen Raspberry-Pi-Pico-Mikrocontroller um ein WLAN-Modul erweitert.

die IMU auch einen Temperatursensor. Als weiterer, separater Sensor ist ein Mikrofon auf dem Board verbaut, mit welchem Geräusche detektiert und in Abhängigkeit der Lautstärke zum Beispiel Leuchtdioden auf dem Board an- und ausgeschaltet werden können [A12]. Abbildung 3.3 veranschaulicht die verschiedenen Modelle.

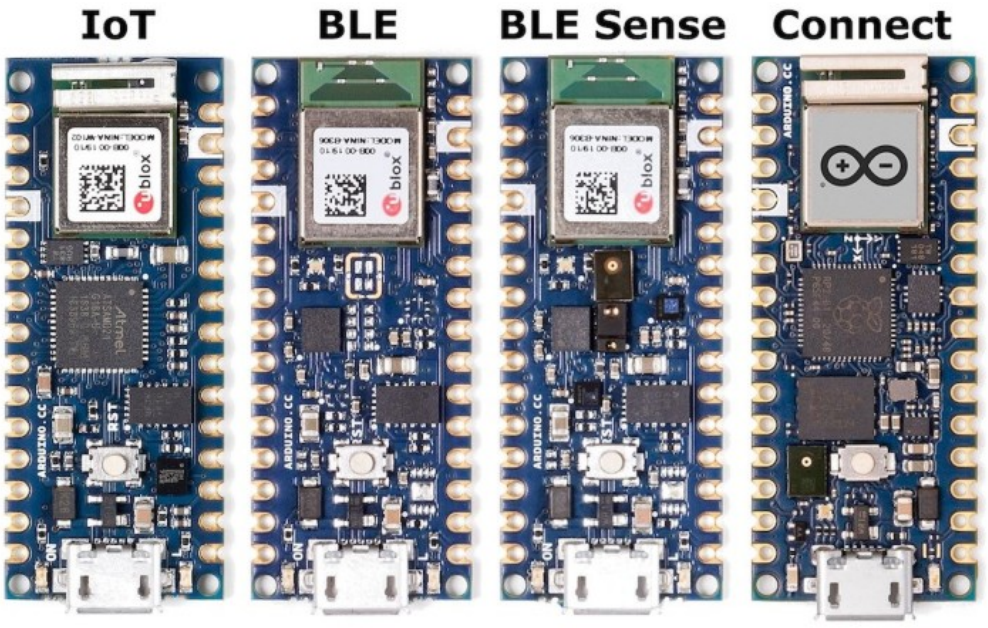

Abbildung 3.3.: Übersicht der Arduino-Nano-Boards mit integrierten Beschleunigungssensoren [B34]

3.2. Auswahl des Mikrocontroller-Boards

Für die Wahl als Telemetriesender stechen zwei der behandelten Modelle hervor: Das Feather-nRF52840-Sense von Adafruit sowie das Nano-33-BLE-Sense von Arduino. Beide Modelle verfügen über einen großen Satz verschiedener integrierter Sensoren, mit welchen sich viele physikalische Messungen durchführen lassen, allen voran Beschleunigungssensoren sowie Druck- und Temperatursensoren, für die sich besonders geeignete Telemetrieanwendungen ergeben. Als Nachteil kann betrachtet werden, dass beide Modelle lediglich über Bluetooth-LE-Konnektivität verfügen und die verlässliche Übertragung von Sensordaten so auf recht kurze Reichweiten beschränkt ist. Ein Grund für diese funktionelle Einsparung könnte darin liegen, dass der Betrieb von WLAN-Modulen energieintensiv ist und zu einer starken Erhitzung des Boards führt, die zu erhöhten Messabweichungen oder gar Störungen der Sensoren führen könnte. Der hohe Strombedarf dürfte ebenso einen eventuell angeschlossenen Akkumulator belasten, der für einen mobilen Einsatz der Sensoren als Spannungsversorgung nötig wird. Nicht zuletzt bewegen sich beide Boards bereits preislich auf einem höherem Niveau ab dreißig Euro; ein höherwertiges Funkmodul, das neben Bluetooth Low Energy auch eine Datenübertragung über klassisches Bluetooth und WLAN erlaubt, dürfte auch den Preis weiter anheben. Das Feather-nRF52840-Sense-Board verfügt gegenüber dem konkurrierenden Board von Arduino über den Vorteil eines eigenen Akkumulator-Anschlusses mit integrierter Ladevorrichtung, sowie eigens für Feather-Boards angebotener Akkumulatoren verschiedener Größe und Kapazität. Daher wäre allem voran das Feather-nRF52840-Sense-Board für die Verwendung als Telemetriesender prädestiniert.

Als problematisch erweist sich jedoch die Verfügbarkeit dieser speziellen Boards. So waren zum Zeitpunkt der Erstellung dieser Arbeit beide Modelle in den jeweiligen Onlineshops der

Hersteller nicht auf Lager. Verfügbare Angebote auf Online-Handelsplätzen wie Amazon oder Ebay fanden sich lediglich von kleineren Händlern, die aber zu deutlich höheren Preisen von knapp fünfzig Euro verkauft wurden (Stand Juni 2022). Eine weitere, gewisse Einschränkung ergibt sich auch dadurch, dass derartige Multi-Sensor-Entwicklungsboards zwar den Kauf und Anschluss externer Sensoren ersparen, jedoch gleichzeitig die Qualität der Messung mit fest verbauten Sensoren festlegen. Je nach Anspruch der geplanten Messung dürften Anwender eher dazu neigen, teurere und höherwertige Sensoren separat zu erwerben, um diese über übliche Schnittstellen an einen Mikrocontroller mit geeigneter Leistung anzuschließen. Darüber hinaus ist ein direktes Verbauen von Sensoren auf dem Board nicht in allen Fällen sinnvoll. So werden Licht- und Helligkeitssensoren häufig an Stellen montiert, wo ein ganzes Mikrocontroller-Board hinderlich wäre. Auf dem Board verbaute Temperatursensoren messen durch die Erhitzung des Funkmoduls vor allem die Temperatur des Boards selbst, nicht die Umgebungstemperatur.

Die Arduino-Nano-Modelle mit den Namen 33-IoT, 33-BLE sowie RP2040-Connect waren zum Zeitpunkt dieser Abschlussarbeit über unterschiedliche Händler gut verfügbar, für einen moderaten Preis von rund zwanzig Euro erhältlich und verfügen zudem über eine integrierte Messeinheit mit einem Accelerometer, sodass diese Modelle für die Anwendung als Telemetriesender zum Messen von Beschleunigungen und Bewegungen insgesamt die beste Wahl darstellen. Aufgrund des leistungsstarken RP2040-Prozessors und die damit einhergehende besondere Unterstützung für Micropython wurde das Board RP2040-Connect als Hardware-Grundlage ausgewählt. Im Anhang D werden die wesentlichen technischen Eigenschaften des Boards behandelt.

3.3. Spannungsversorgung

Für die Realisierung des Telemetriesenders wird eine mobile Spannungsversorgung benötigt, die durch Betätigung eines Schalters an- und ausschaltbar ist und mit dem ausgewählten Mikrocontroller-Board zu einer Einheit verbunden werden kann. Dabei werden zur Spannungsversorgung des Boards die Pins „VIN“ und „GND“ genutzt, über welche das Board alternativ zum Betrieb über den USB-Anschluss mit Spannung versorgt werden kann. Die Anschlusspins können eine Spannung von minimal 4 V bis maximal 20 V verarbeiten, was die wesentliche technische Anforderung für die Auswahl der Spannungsversorgung darstellt. Im Online-Handel findet sich eine Auswahl verschiedener Batteriehalterungen mit Abdeckung, Schalter und Adern für die Plus- und Minusleitung, die sich ideal als Spannungsversorgung für den Telemetriesender eignen. Dabei kann allgemein eine Auswahl getroffen werden zwischen Batteriehalterungen mit größeren Maßen und größerer Speicherkapazität, und kleineren Batteriehalterungen, die handlicher sind, jedoch ein häufigeres Wechseln der Batterien nötig machen. Die Abbildungen 3.4, 3.5 und 3.6 stellen eine Auswahl geeigneter Modelle dar. Im Anhang F werden alle Schritte und Hinweise zum Zusammenbau des Telemetriesenders dargelegt.

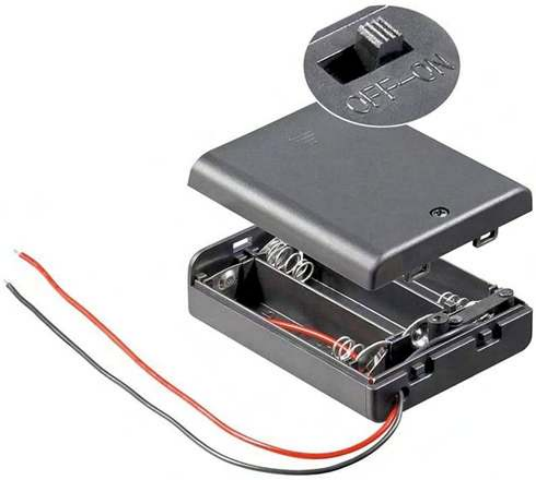

Abbildung 3.4.: 4,5 V-Batteriehalterung mit drei 1,5 V-AA-Batterien

Abbildung 3.5.: 9 V-Batteriehalterung für eine 9 V-Blockbatterie

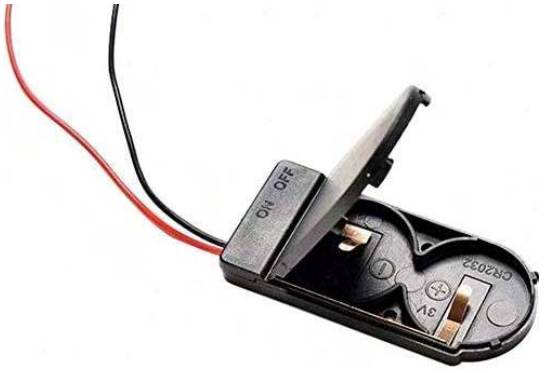

Abbildung 3.6.: 6 V-Batteriehalterung für zwei 3 V-Knopfzellen

4. Programmierung

Genauso wie die Arduino-Modelle Nano-33-IoT und Nano-33-BLE ist auch das Modell Nano-RP2040-Connect sowohl in C++ als auch in Micropython programmierbar. Beide Programmiersprachen bringen spezifische Vor- und Nachteile mit sich, die im folgenden im Hinblick auf ihre Verwendung im Schulkontext erörtert werden sollen.

4.1. C++

Bei C++ handelt es sich um eine hardwarenahe Programmiersprache, die besonders in der Programmierung von Mikrocontrollern, eingebetteten Systemen oder auch Betriebssystemen Anwendung findet. Um einen Quellcode in einen für Maschinen lesbaren Code umzuwandeln, wird ein Compiler verwendet. Dabei handelt es sich um ein in die Entwicklungsumgebung integriertes Programm, welches den Quellcode in mehreren Schritten in einen Binärcode umwandelt, der speziell für den jeweils ausführenden Prozessor ausgelegt wird. Aus dem Binärcode wird eine ausführbare Datei gebildet, die im Flash-Speicher abgelegt wird und beim Start des Systems vom Prozessor ausgeführt wird. Je nach Größe des Codes und Leistung des Rechners, auf welchem die Entwicklungsumgebung ausgeführt wird, kann das Kompilieren einige Minuten in Anspruch nehmen. Fehler im Code führen zum sofortigen Abbruch des Kompilierens; nach Korrektur des Fehlers muss der Kompilierprozess neu durchlaufen werden. Für Programmieranfänger stellt C++ aufgrund der komplexeren Syntax und der Vielzahl unterschiedlicher Datentypen, denen Variablen bei ihrer Definition zugewiesen werden müssen, eine anspruchsvollere Programmiersprache dar. Fehler im Quellcode werden häufig erst nach mehrmaligen Kompilierversuchen behoben, was zu einem erhöhten Zeitaufwand führt, der im Rahmen des Unterrichts häufig nicht erbracht werden kann. Für die Programmierung bietet Arduino eine eigene Entwicklungsumgebung mit der Programmiersprache Arduino an, bei der es sich im Wesentlichen um die Sprache C++ handelt, ergänzt um einen Satz spezieller Methoden und Funktionen.

4.2. Micropython und Circuitpython

Micropython baut auf der Programmiersprache Python auf und wurde speziell für die Mikrocontroller-Programmierung entwickelt. Im Gegensatz zu C++ ist Micropython – wie Python - eine interpretierende Programmiersprache. Der Quellcode wird hierbei nicht in eine fertige Datei umgewandelt, die vom Prozessor direkt ausgeführt wird. Stattdessen wandelt der Python-Compiler den Quellcode zunächst in einen sogenannten Bytecode um. Dieser stellt einen Satz von Anweisungen dar, die von einem Interpreter Schritt für Schritt während des Programmablaufs analysiert werden. Entsprechend den Anweisungen ruft der Interpreter Routinen aus seinen internen Bibliotheken auf, die wiederum die gewünschten Aktionen auf dem Prozessor ausführen. Da auf diese Weise der Code praktisch simultan zur Laufzeit in für den Prozessor ausführbare Befehle übersetzt wird, ist die Ausführung von Micropython-Code im Allgemeinen langsamer als ein in C++ geschriebener Code. Durch die Speicherung des Interpreters auf dem Mikrocontroller wird zudem Speicher belegt, der

nicht mehr für das auszuführende Programm zur Verfügung steht. Vorteile ergeben sich dadurch, dass Modifikationen im Code direkt ausprobiert und ihre Auswirkung beobachtet werden können, ohne dass jede Änderung ein neues Kompilieren erfordert. Dies spart Zeit und kommt insbesondere Anfängern entgegen. Python – und so auch Micropython – gilt allgemein als einfach zu erlernende Programmiersprache. Im Gegensatz zu C++ müssen Variablen keinem Datentyp zugewiesen werden; die korrekte Interpretation der Variablen wird stattdessen vom Interpreter durchgeführt. Eine weitere Vereinfachung gegenüber C++ ergibt sich auf syntaktischer Ebene: Codezeilen müssen nicht mit einem Semikolon abgeschlossen werden. Funktionskörper sowie Körper von Kontrollstrukturen werden nicht durch geschweifte Klammern definiert, sondern durch bloßes Einrücken kenntlich gemacht. Dies führt zu einem kürzeren, übersichtlicheren Code und reduziert die Fehleranfälligkeit. Aus den genannten Gründen eignet sich Python besonders für einen einfachen Einstieg in die Programmierung.

Das Unternehmen Adafruit, das eine breite Auswahl an Mikrocontrollern und Sensoren vertreibt, hat diesen didaktischen Aspekt aufgegriffen und den Micropython-Dialekt Circuitpython entwickelt, mit welchem alle von Adafruit unterstützten Boards programmiert werden können. Für jedes unterstützte Board wird eine eigene Ciruitpython-Version bereitgestellt; dabei wird Micropython durch eine spezielle Kompatibilitätsschicht modifiziert, sodass die von Adafruit bereitgestellten Circuitpython-Bibliotheken einheitlich für alle Boards verwendet werden können. Als Entwicklungsumgebung für Python, Micropython und Circuitpython bieten sich Thonny oder Mu an.

4.3. Auswahl der Programmiersprache

Ursprünglich war eine Programmierung in Micropython oder Circuitpython beabsichtigt, zum einen wegen der erwähnten didaktischen Vorteile, zum anderen aufgrund der erhofften Möglichkeit, in Python geschriebenen PhyPiDAQ-Code auf dem Mikrocontroller betreiben zu können. Letzteres erwies sich jedoch als nicht umsetzbar, da standardmäßig nur ein Teil der Python-Standardbibliothek in Micropython implementiert ist, um der begrenzten Leistung und Speicherkapazität von eingebetteten Systemen Rechnung zu tragen. Einzelne Module sind zudem für die Anwendung in eingebetteten Systemen angepasst. Dies führt insgesamt dazu, dass ein Teil der im Python-Code verwendeten Funktionen in Micropython nicht funktioniert.

Ein weiterer Aspekt betrifft die Ausführungsgeschwindigkeit des Codes. Bei der Datenübertragung über das MQTT-Protokoll arbeitete der C++-Code auf dem Arduino-Board deutlich schneller als der in Circuitpython geschriebene Code (eine ausführlichere Diskussion findet sich im Anhang A). Aufgrund der besseren Leistung, und da eine direkte Anwendung von PhyPiDAQ-Code auf dem Arduino-Board nicht möglich ist, wurde als Programmiersprache so C++ verwendet. Die Quellcodes der Sende- und Empfangsprogramme für die Übertragung über das MQTT-Protokoll und über Bluetooth finden sich in den Anhängen A und B.

5. Didaktische Anwendungen

5.1. Beschleunigung in Baden-Württembergischen Bildungsplänen

Ehe konkrete Anwendungen erläutert werden, soll am Beispiel ausgewählter Baden-Württembergischer Bildungspläne ein Überblick über den Themenkomplex „Beschleunigung“ und dessen Einordnung im Curriculum gegeben werden.

In Gymnasien wird die Beschleunigung erstmals in den Klassenstufen 7 und 8 qualitativ angesprochen; dabei wird vordergründig der Unterschied zwischen gleichförmiger Bewegung und beschleunigter Bewegung herausgearbeitet [A13]. Eine ausführliche Behandlung erfolgt in den Klassenstufen 9 und 10 im Rahmen der Kinematik [A14]. Dabei sollen Schüler folgende Kompetenzen erlangen:

- Beschleunigung als Änderungsrate der Geschwindigkeit erklären und berechnen
- Geradlinig gleichmäßig beschleunigte Bewegungen ($s(t) = \frac{1}{2}at^2,\ v(t) = at$) verbal und rechnerisch beschreiben
- Bewegungsabläufe experimentell aufzeichnen und unter anderem a-t-Diagramme interpretieren

Darauf aufbauend, werden im nächsten Teilgebiet der Mechanik, der Dynamik, Kräfte auf Grundlage von Masse und Beschleunigung behandelt. In diesem Zusammenhang wird auch die Zentripetalkraft und mit ihr die Zentripetalbeschleunigung thematisiert.

Im Bildungsplan für die Sekundarstufe I, worunter Hauptschulen, Werk-Realschulen und Realschulen zusammengefasst werden, findet nur auf dem erhöhten Anforderungsniveau[1] „E“ eine grundlegende Behandlung der Beschleunigung statt. Beschleunigte Bewegungen sollen hierbei verbal und mithilfe von Diagrammen beschrieben und klassifiziert werden [A15]. Im daran anschließenden Themengebiet der Dynamik wird Beschleunigung nur in Form des Ortsfaktors g thematisiert.

Für Abgänger der Sekundarstufe I findet eine tiefere Behandlung der Beschleunigungen erst auf weiterführenden Schulen wie dem Beruflichen Gymnasium statt. Im dortigen Bildungsplan wird im Themengebiet „Kinematik und Dynamik“ (Eingangsklasse) Beschleunigung in verschiedenen Zusammenhängen thematisiert. Dazu gehören:

- Beschleunigung in Bezug auf die Kraftdefinition und Impulsänderungen
- Beschleunigung aus der Ruhe heraus; Bremsbewegung (negative Beschleunigung) bis zum Stillstand
- Freier Fall, Fallbeschleunigung

Im Themengebiet „Teilchen in Feldern“(Jahrgangsstufe 1 und 2) werden schließlich auch Zentripetalbeschleunigung und Zentripetalkraft behandelt [A16].

[1] Der Bildungsplan der Sekundarstufe I ist in drei Niveaustufen unterteilt: im Grundlagen-Niveau „G“ wird Stoff auf Hauptschulniveau vermittelt. Das Mittel-Niveau „M“ entspricht dem Realschulniveau, das Experten-Niveau „E“ kann für leistungsstarke Schüler als Vorbereitung zum Besuch weiterführender Schulen unterrichtet werden.

5.2. Didaktische Erarbeitung des Funktionsprinzips des Accelerometers

5.2.1. Funktionsweise eines kapazitiven Accelerometers

Die Umwandlung eines physikalischen Werts in einen Spannungswert und die Umwandlung dieses Spannungswerts in eine brauchbare Zahlengröße ist Teil eines jeden physikalischen Messprozesses; daher wäre es didaktisch gewinnbringend, auch Verfahren zum Messen von Beschleunigungen zu thematisieren, statt Beschleunigungssensoren lediglich als Blackbox zu betrachten. In der Arbeit von Moritz Aupperle wurde das Funktionsprinzip eines Beschleunigungssensors bereits ausführlich behandelt und soll daher an dieser Stelle nur qualitativ zusammengefasst werden.

Das zugrundeliegende Messprinzip eines kapazitiven Beschleunigungssensors wird anhand eines einfachen Modells in Abbildung 5.1 veranschaulicht. In einem elektrisch nicht leitenden Rahmen sind die Platten zweier Kondensatoren C_1 und C_2 eingefasst. Die zwei Kondensatoren sind über eine seismische Masse, die federnd gelagert ist, mit einander verbunden und bilden zusammen einen sogenannten Differentialkondensator. Bei einer Beschleunigung in entsprechende Richtung verschiebt sich die seismische Masse so, dass sich der Plattenabstand des einen Kondensators verringert und sich der Plattenabstand des anderen Kondensators vergrößert. Die Beschleunigung wird also in eine Kapazitätsänderung gewandelt, die elektrisch genutzt werden kann.

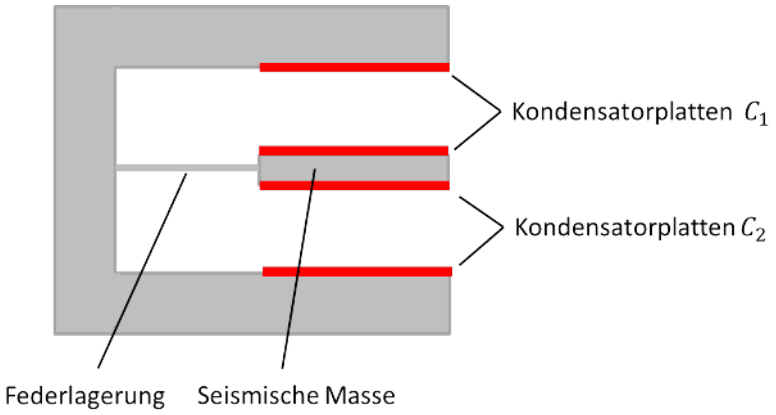

Abbildung 5.1.: Prinzipskizze eines kapazitiven Beschleunigungssensors

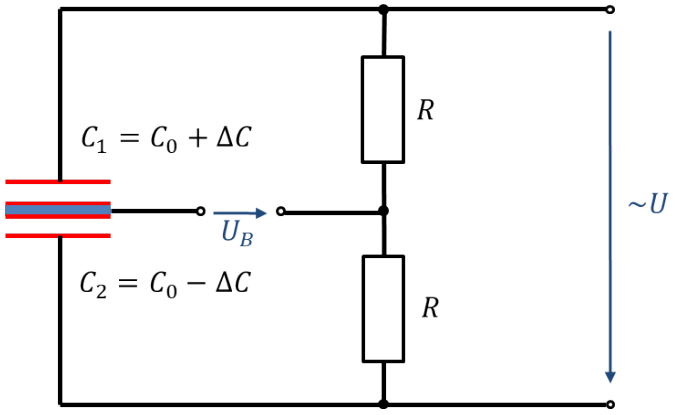

Abbildung 5.2.: Brückenschaltung zur Auswertung eines Differentialkondensators

Die Umwandlung der Kapazitätsänderung in ein Spannungssignal wird mit einer Brückenschaltung realisiert, wie in Abbildung 5.2 gezeigt. Die anliegende Wechselspannung U wird

durch die Reihenschaltung der Widerstände und die Reihenschaltung der Kondensatoren im unbeschleunigten Zustand je hälftig aufgeteilt, sodass die Brückenspannung U_B null ist. Ändern sich die Kapazitäten C_1 und C_2 beim Auftreten einer Beschleunigung wie beschrieben, ändert dies auch das Teilungsverhältnis gegenüber den gleichbleibenden Widerständen, sodass auf der Brücke eine Differenzspannung auftritt, die proportional zur Beschleunigung ist.

5.2.2. Nachbildung des Funktionsprinzips in einem Demonstrationsversuch

Eine experimentellen Realisierung des in Abbildung 5.1 veranschaulichten Modells wird ausführlich in [A17] behandelt. Dabei wird der Differentialkondensator nachgebildet, indem zwischen zwei üblichen Kondensatorplatten eine dritte Platte beweglich aufgehängt wird. Bei einem Abstand von etwa 2 cm zwischen zwei benachbarten Platten ergibt sich eine Kapazität im 10 pF-Bereich. Die drei Platten werden werden gemäß Abbildung 5.2 an eine Brückenschaltung angeschlossen, die mit 25 V Wechselspannung versorgt wird. Um einen Kurzschluss durch versehentliches Berühren zweier Platten zu vermeiden, werden die Ränder der beweglichen Platte mit Knetmasse versehen. Zum Messen der Brückenspannung sollte ein Messgerät mit hohem Innenwiderstand verwendet werden. Bei maximaler Auslenkung der beweglichen Platte lässt sich eine Brückenspannung von etwa 1 V messen.

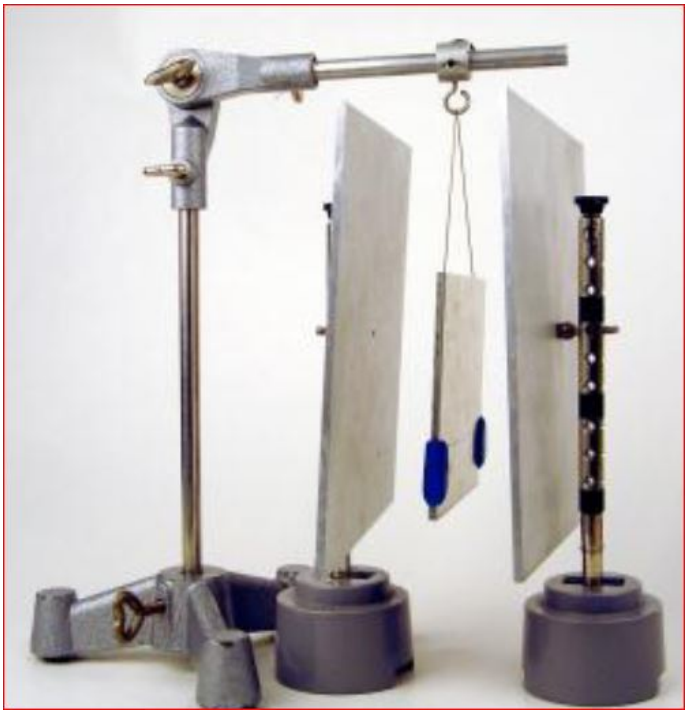

Abbildung 5.3.: Einfache Nachbildung eines Differentialkondensators [B35]

Der in Abbildung 5.3 gezeigte Aufbau ist zunächst nur dazu geeignet, die manuelle Positionsänderung der beweglichen Platte - die die seismische Masse des Beschleunigungssensors repräsentiert - in eine messbare Spannung zu übersetzen. Um diese Positionsänderung durch Einwirken einer Beschleunigung zu erzeugen, wird der Aufbau auf einen rollbaren Untersatz gestellt. Zur Nachstellung einer gefederten Lagerung wird die mittlere Platte durch zwei am unteren Ende angebrachte Spiralfedern auf Position gehalten. Abbildung 5.4 stellt den modifizierten Aufbau dar. Als problematisch erweist sich jedoch, dass der Aufbau stark zu Schwingungen neigt, was eine korrekte Übersetzung von Beschleunigungen in

Spannungswerten erschwert. Es kann lediglich versucht werden, das Auftreten von Schwingungen durch Vermeiden allzu ruckartiger Bewegungen zu minimieren. Ein Lösungsansatz könnte sein, eine Schwingungsdämpfung zu implementieren, indem das untere Ende der beweglichen Platte in eine Ölwanne eingetaucht wird. Die Stärke der Dämpfung kann dabei durch die Tiefe des Eintauchens justiert werden. Allgemein aber handelt es sich um einen anspruchsvolleren Demonstrationsversuch, der eine sorgfältige Abstimmung der Komponenten zueinander erfordert, um ein optimales Ergebnis zu liefern.

Eine grundsätzlichere Problematik in der Vermittlung der Funktionsweise besteht darin, dass die Elektrizitätslehre in den meisten Curricula nach der Mechanik behandelt wird, sodass das für Kondensatoren nötige Wissen noch nicht vorliegt, wenn die Beschleunigung im Rahmen der Mechanik erstmals behandelt wird. Unter Beibehaltung dieser curricularen Reihenfolge kann das Funktionsprinzip eines kapazitiven Beschleunigungssensors also nur im Nachhinein thematisiert werden. Auch dann aber stellen die zum Verständnis nötigen elektrischen Grundlagen - wie der Blindwiderstand eines Kondensators und das Funktionsprinzip einer Brückenschaltung - Themen dar, die im Physikunterricht allenfalls beiläufig behandelt werden, jedoch eingehend erarbeitet werden müssen, ehe die Funktionsweise eines Beschleunigungssensors selbst thematisiert werden kann.

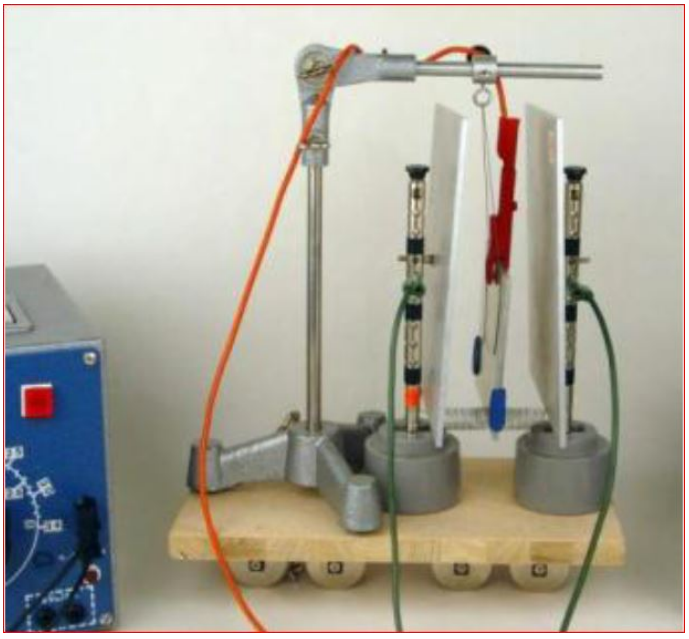

Abbildung 5.4.: Versuchsaufbau zur Nachbildung eines kapazitiven Beschleunigungssensors [B35]

5.3. Beispielanwendungen

5.3.1. Visualisierung der Beschleunigung auf einer Rennbahn

Der in dieser Arbeit entwickelte Telemetriesender erlaubt es, zum Beispiel die Zentrifugalbeschleunigung in den Kurven einer Rennbahn aufzuzeichnen und über die Benutzeroberfläche von PhyPiDAQ sichtbar zu machen. Um die Anwendung zu demonstrieren, wurde eine Modell-Autorennbahn aufgebaut, auf welcher Rennautos über Schleifkontakte mit Spannung versorgt und so gesteuert werden können. Abbildung 5.5 stellt den Aufbau dar. Der Telemetriesender wurde mithilfe eines gewöhnlichen Klebebandes auf einem Rennauto befestigt, wie Abbildung 5.6 veranschaulicht.

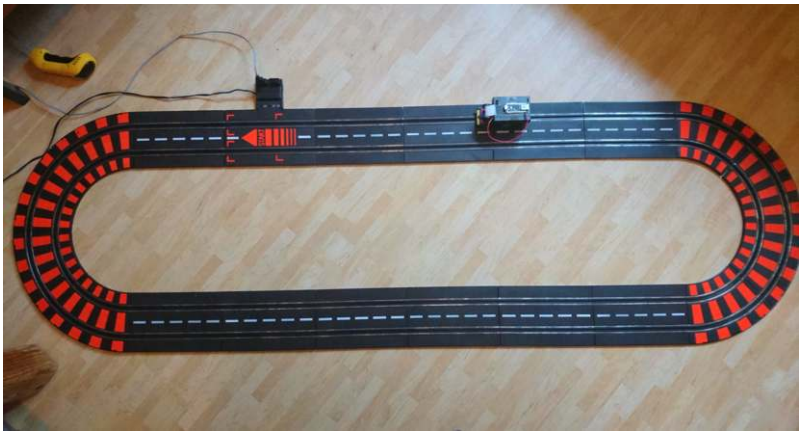

Abbildung 5.5.: Beispielhafter Aufbau einer Rennbahn mit Schleifkontakten

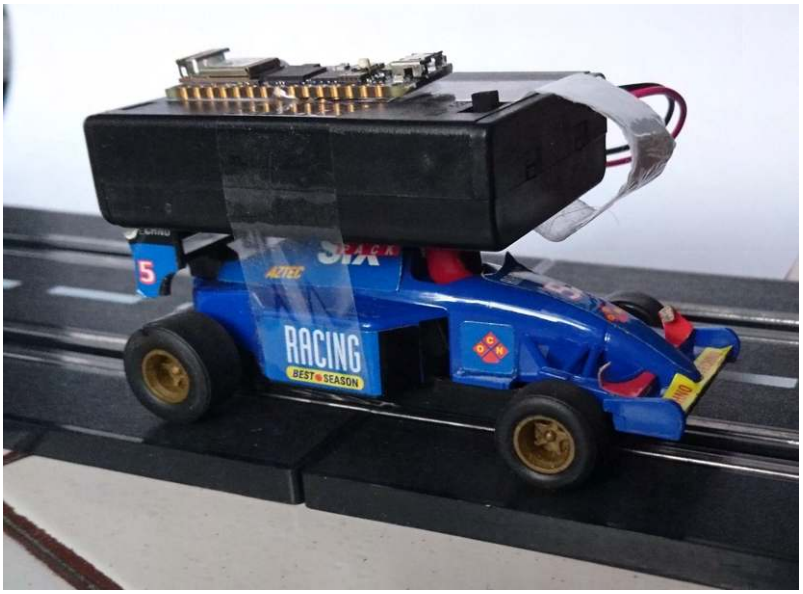

Abbildung 5.6.: Rennwagen mit Telemetriesender

Beim Steuern des Rennautos ist zu beachten, dass durch den aufgesetzten Telemetriesender der Schwerpunkt erhöht wird und der Wagen dadurch in Kurven zum Umfallen neigt. Vor allem vor dem Eintritt in eine Kurve sollte die Geschwindigkeit daher nicht zu hoch gewählt werden.

Der Verlauf der Zentrifugalbeschleunigung ist in Abbildung 5.7 dargestellt. Es ist zu beobachten, dass die Beschleunigungswerte eine hohe Streuung aufweisen. Ursächlich

hierfür sind Vibrationen des Motors und des Getriebes, ferner können auch durch die Schienenführung und die Schleifkontakte Erschütterungen entstehen. Erst durch die Bildung eines gleitenden Mittelwerts wird der Verlauf der Beschleunigung und deren Ausschlag beim Durchfahren einer Kurve sichtbar. Eine Durchführung im Unterrichtsversuch ist daher nur zu empfehlen, wenn der verwendete Aufbau möglichst vibrationsarm und der Verlauf der Beschleunigung klar zu erkennen ist. Andernfalls muss eine nachträgliche Bearbeitung der Messdaten in Form einer Glättung vorgenommen werden, was allerdings dem eigentlichen Ziel einer anschaulichen Darstellung der Beschleunigung in Kurven zuwiderläuft.

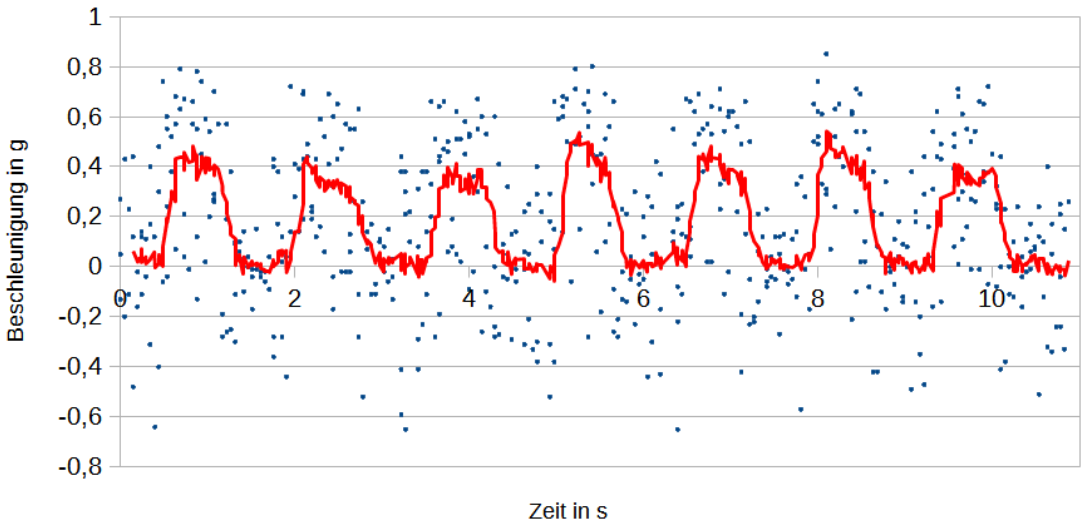

Abbildung 5.7.: Zentrifugalbeschleunigung auf der Rennbahn. Messpunkte in Blau bei einer Abtastrate von 50 Hz. Gleitender Mittelwert über 10 Messwerte in Rot.

5.3.2. Untersuchung der Zentrifugalbeschleunigung in Abhängigkeit von Winkelgeschwindigkeit und Radius

Der Telemetriesender kann verwendet werden, um den in der Schulphysik häufig genutzten Zusammenhang

$$a = \frac{v^2}{r} = \frac{(\frac{2\pi r}{T})^2}{r} = \omega^2 r$$

experimentell nachzuweisen. Vor allem der Quotient aus dem Quadrat der Geschwindigkeit und dem Kurvenradius wird sowohl im Themengebiet der Mechanik, als auch später bei der Bewegung von geladenen Teilchen in magnetischen Feldern zur Berechnung von Kräften und Beschleunigungen in Kreisbahnen verwendet. Durch Umformung kann der Quotient durch das Produkt aus der Winkelgeschwindigkeit zum Quadrat und dem Radius ersetzt werden. Die Winkelgeschwindigkeit ω wirkt in Schüleraugen abstrakter als die Bahngeschwindigkeit v, jedoch stellt sie die grundlegende Einheit zur Beschreibung von Drehbewegungen dar und ist einfacher zu messen, weshalb für die experimentelle Überprüfung der Abhängigkeiten der Zentrifugalbeschleunigung die Form $a = \omega^2 r$ geeigneter ist.

Die Durchführung der Messung erfolgt, indem die Zentrifugalbeschleunigung unter Variation des zu untersuchenden Parameters gemessen wird, während der jeweils andere Parameter konstant gehalten wird. Der benötigte Versuchsaufbau muss daher sowohl die Rotationsfrequenz als auch den Radius variieren können. Für den Messversuch ideal wäre ein Motor mit einstellbarer Drehzahl, an dessen Drehachse eine Scheibe oder ein Stab lotrecht angebracht

werden kann, um den Telemetriesender in variablem Abstand um die Drehachse rotieren zu lassen. Im Rahmen dieser Arbeit wurde eine drehbare Unterlage mit Kugellager verwendet, auf welcher behelfsmäßig eine Stativstange befestigt wurde. Abbildung 5.8 veranschaulicht den Aufbau. Der Telemetriesender wurde mit einer Stativklemme an die Stange montiert, wie in Abbildung 5.9 gezeigt. Um eine Unwucht bei der Drehung zu vermeiden, wurden am anderen Ende der Stange zwei weitere Stativklemmen als Gegengewichte montiert. Die Drehung der Stativstange wurde händisch angetrieben und die Drehfrequenz mit einer speziellen Apparatur gemessen, die im Praktikum „Klassische Physik Teil I“ des Physikstudiums am KIT verwendet wird. Eine Infrarot-Messsonde detektiert dabei die Drehfrequenz durch den periodischen Durchgang eines Reflexionsstreifens, der an einem Ende der Stativstange aufgeklebt wurde (Abbildung 5.10). Die gemessene Frequenz wird über ein Anzeigegerät ausgegeben. Eine alternative Möglichkeit zur Messung der Frequenz besteht in der Nutzung des Gyrometers, welches gemeinsam mit dem Accelerometer auf dem Board verbaut ist. Näheres findet sich in Anhang C.

Abbildung 5.8.: Gesamter Aufbau für den Messversuch

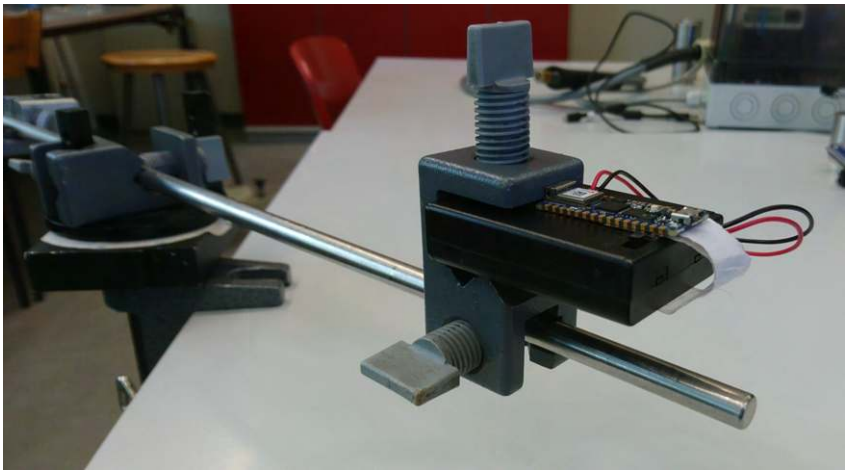

Abbildung 5.9.: Befestigung des Telemetriesenders mit einer Stativklemme

Eine Hauptquelle für Messunsicherheiten stellt das Ablesen der Rotationsfrequenz und der Beschleunigung dar, deren Messwerte aufgrund des händischen Antriebs größeren Schwankungen unterworfen sind. Die Auswertung der Messungen in den Abbildungen 5.11 und 5.12 betätigt die Modellfunktion.

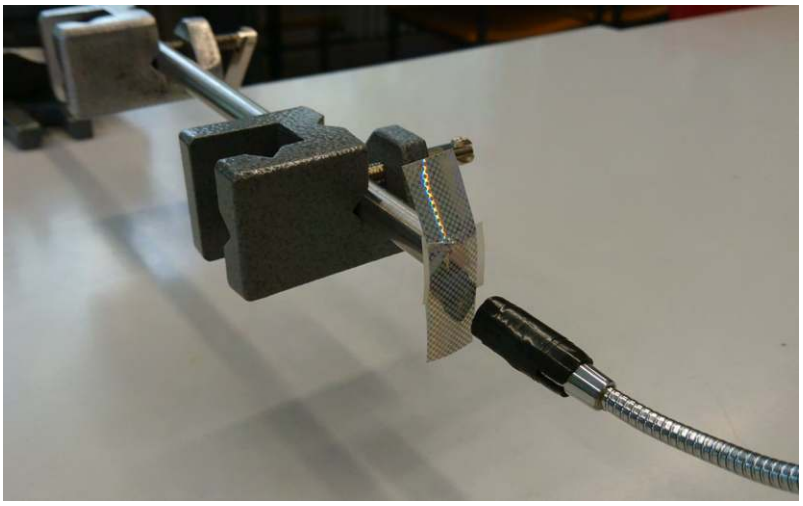

Abbildung 5.10.: Reflexionsstreifen an einem Ende der Stativstange mit Detektor

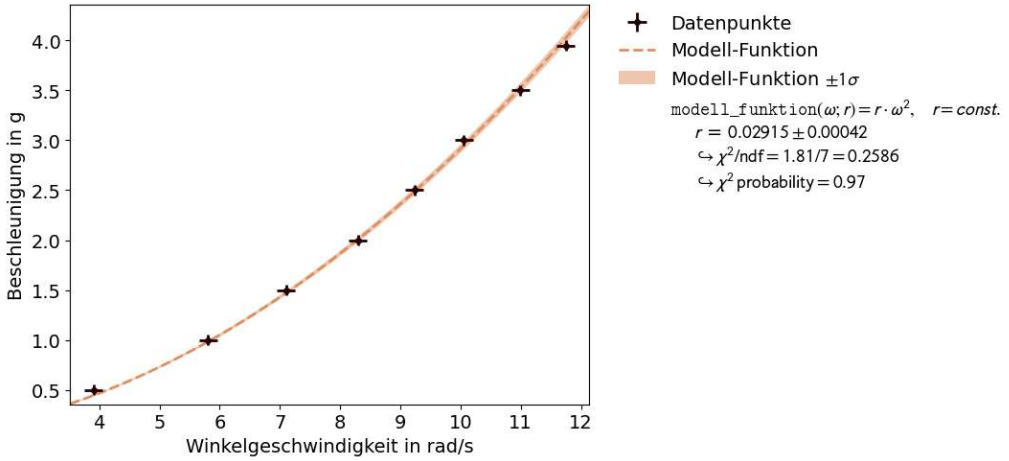

Abbildung 5.11.: Beschleunigung in g in Abhängigkeit der Winkelgeschwindigkeit ω

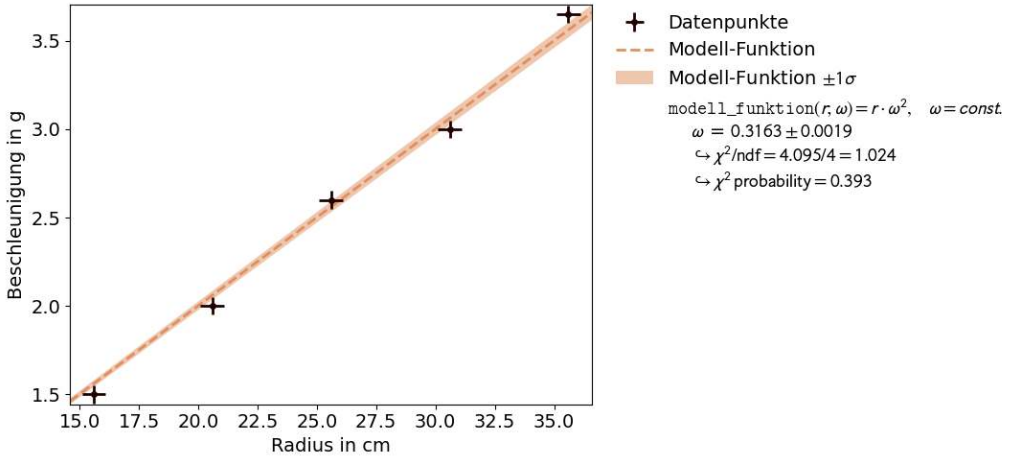

Abbildung 5.12.: Beschleunigung in g in Abhängigkeit des Radius r

5.3.3. Impuls und Beschleunigung auf einer Luftkissenbahn

Anders als beim anfangs behandelten Rennbahn-Versuch bietet eine Luftkissenbahn den Vorteil einer reibungslosen und damit erschütterungsfreien Fortbewegung. Für den hier erläuterten Versuch wurde eine Luftkissenbahn des Lehrmittelherstellers Phywe verwendet (Abbildung 5.13). Der Telemetriesender wird mithilfe von Klebeband an einen Schlitten befestigt, der über die Schiene gleiten kann (Abbildung 5.14). An den beiden Enden der Schiene ist ein Gummiband aufgespannt, welches den Schlitten beim Auftreffen abbremst und wieder in umgekehrte Richtung beschleunigt (Abbildung 5.15). Nach einmaligem Anschieben gleitet der Schlitten so mehrmals zwischen den Enden der Schiene hin und her. Die beim Umkehren der Bewegungsrichtung auftretende Beschleunigung kann über den Telemetriesender aufgezeichnet werden. Abbildung 5.16 stellt die aufgezeichnete Beschleunigung für das je zweimalige Auftreffen des Schlittens and den Enden der Schiene dar. Entsprechend der Bewegungsrichtung hat die Beschleunigung positives oder negatives Vorzeichen.

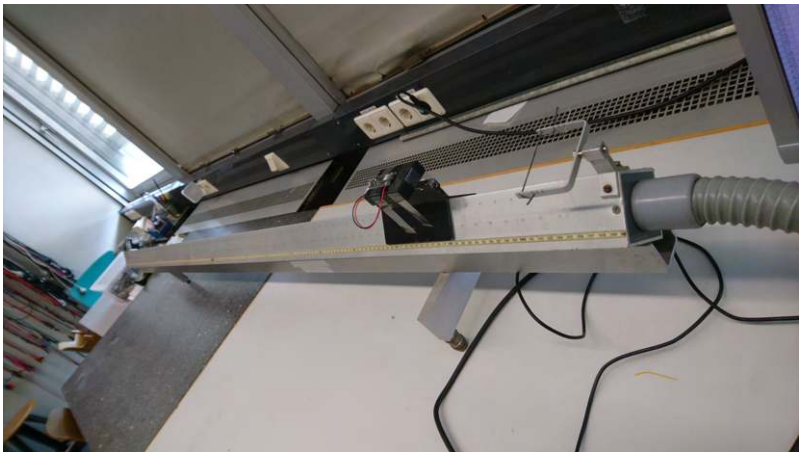

Abbildung 5.13.: Zentrifugalbeschleunigung auf der Rennbahn

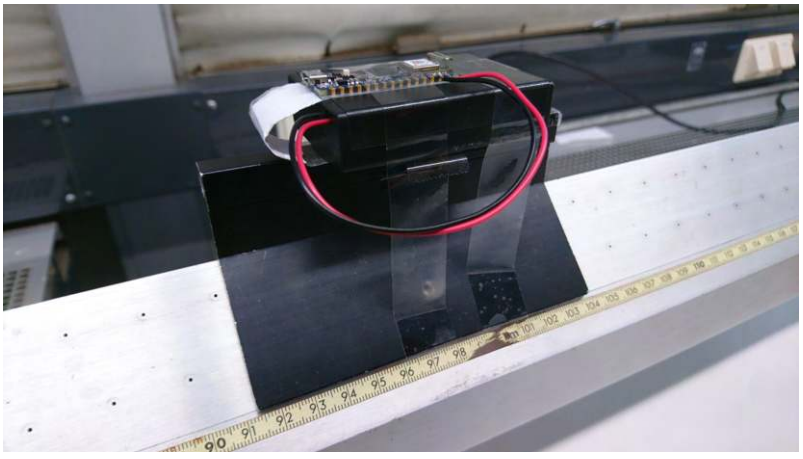

Abbildung 5.14.: Befestigung des Telemetriesenders an einen Schlitten

Modellierung der Beschleunigungspeaks

Mit jeder Änderung der Bewegungsrichtung führt der Schlitten auch eine Impulsänderung durch. Zusammen mit der gemessenen Dauer der Impulsänderung und dem Beschleunigungsverlauf kann daraus die theoretische Beschleunigungsspitze errechnet und so mit

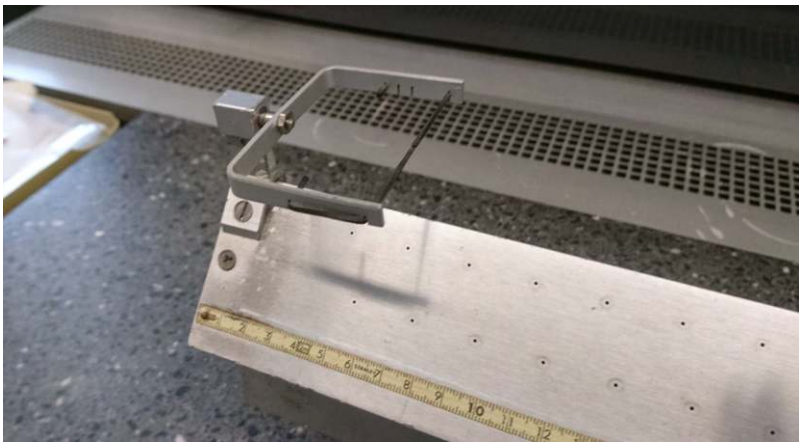

Abbildung 5.15.: Aufgespannter Gummifaden an den Enden der Schiene

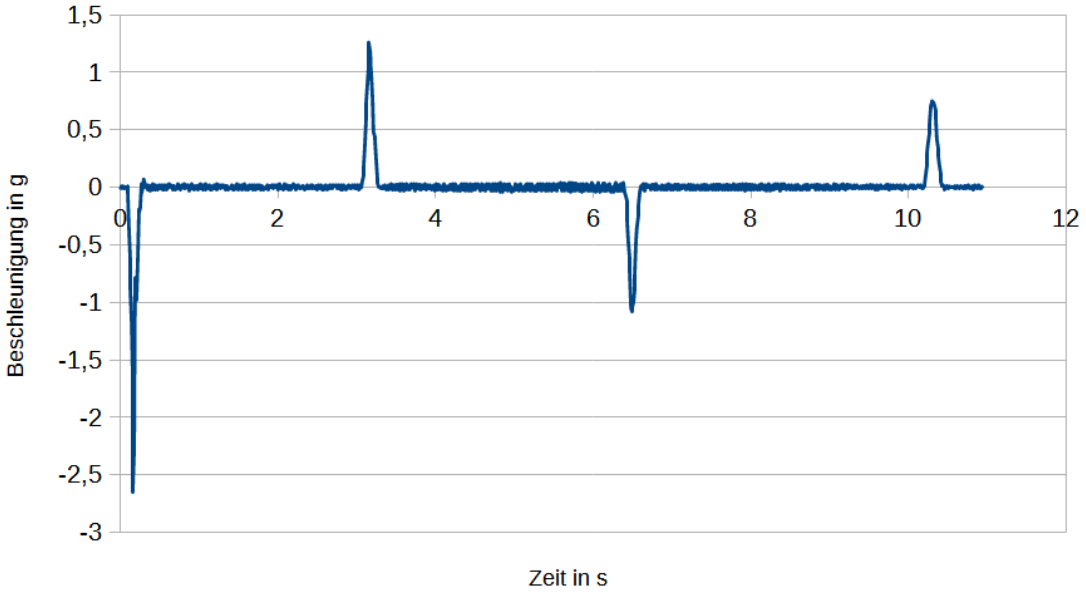

Abbildung 5.16.: Beschleunigung des Schlittens auf der Luftkissenbahn

der gemessenen verglichen werden kann. Eine Unsicherheit stellt die Modellierung des Beschleunigungsverlaufs dar. Da das aufgespannte Gummiband nicht proportional zur Strecke gedehnt wird, um die es durch den auftreffenden Schlitten eingedellt wird, kann kein einfacher Sinusbogen als Modellierung angenommen werden. Unabhängig davon führen auch Reibungseffekte während der Dehnung des Gummis zu einer Abweichung vom idealisierten Federverhalten. Der Erscheinung der Beschleunigungspeaks in Abbildung 5.16 entsprechend, sollen diese hier als gleichschenklige, spitz zulaufende Dreiecke modelliert werden. Mit der Masse des Schlittens m und den Zeitpunkten t_1 und t_2 an Anfang und Ende eines Peaks gilt für die Impulsänderung Δp und die Beschleunigungsspitze $\hat{a}$ der Zusammenhang:

$$\Delta p = \int_{-t}^{t} \vec{F}(t)\,\mathrm{d}t = m \int_{-t}^{t} \vec{a}(t)\,\mathrm{d}t = \frac{m}{2}(\hat{a}(t_2 - t_1)) \tag{5.1}$$

Zur Berechnung der Impulsänderung muss die Geschwindigkeit des Schlittens vor dem Auftreffen auf das abbremsende Gummiband bekannt sein. Mit dem Zeitpunkt t_0 beim

dem Verlassen des beschleunigenden Gummibandes, dem Zeitpunkt t_1 beim Auftreffen auf das abbremsende Gummiband und der Länge der Strecke l zwischen den Gummibändern folgt für die Impulsänderung:

$$\Delta p = 2mv = 2m\frac{l}{t_1 - t_0} \tag{5.2}$$

Durch Umstellen von 5.1 und Einsetzen von 5.2 folgt für die theoretisch erwartbare Beschleunigungsspitze:

$$\hat{a} = \frac{2\Delta p}{m(t_2 - t_1)} = \frac{4l}{(t_2 - t_1)(t_1 - t_0)} \tag{5.3}$$

Energieverlust bei Reflexion am Gummiband

Geht man davon aus, dass die Luftreibung gegenüber Reibungseffekten bei der Dehnung des Gummis gering ist, kann der Energieverlust während der Dehnung des Gummis durch Vergleich der Geschwindigkeiten vor und nach der Richtungsumkehr bestimmt werden. Mit der Zeitdauer T_0 für die Überwindung der Strecke vor der Reflexion an einem Gummiband und der Zeitdauer T_1 für die Überwindung der Strecke in umgekehrter Richtung danach folgt für den Verlust an kinetischer Energie:

$$E_V = E_{kin,0} - E_{kin,1} = \frac{1}{2}m(v_0^2 - v_1^2) = ml^2\left(\frac{1}{T_0^2} - \frac{1}{T_1^2}\right) \tag{5.4}$$

Für den relativen Verlust folgt somit:

$$\frac{E_V}{E_{kin,0}} = \frac{v_0^2 - v_1^2}{v_0^2} = 1 - \frac{T_0^2}{T_1^2} \tag{5.5}$$

6. Fazit und Ausblick

Ausgehend von den in Kapitel 1 aufgestellten Zielsetzungen kann die in dieser Arbeit vorgestellte Telemetriesender-Implementierung als erfolgreich betrachtet werden. Das Mikrocontroller-Board von Arduino kostet mitsamt Batteriehalterung und Batterien nur ein Bruchteil des von Phywe angebotenen Modells, sodass der PhyPiDAQ-Telemetriesender einen preislichen Vorteil gegenüber bisher auf dem Markt verfügbaren Telemetriesender-Lösungen bietet. Hinsichtlich der technischen Kenndaten ist der PhyPiDAQ-Telemetriesender mit dem Modell von Phywe vergleichbar. So leisten beide Sender eine Abtastrate von 100 Hz und eine Auflösung der Beschleunigung von 0,01 g. Lediglich der Messbereich des Phywe-Modells ist mit ±8 g doppelt so groß wie der Messbereich des PhyPiDAQ-Telemetriesenders mit ±4 g. Die Größe des PhyPiDAQ-Telemetriesenders hängt von der gewählten Batteriehalterung ab, ist aber in jedem Fall deutlich kleiner als ein modernes Smartphone, sodass sich auch hier ein Vorteil gegenüber der App Phyphox ergibt, mit welcher sich Smartphones als Telemetriesender nutzen lassen. Insgesamt stellt der PhyPiDAQ-Telemetriesender hinsichtlich Einfachheit und Benutzerfreundlichkeit eine deutliche Verbesserung gegenüber der Telemetriesender-Implementierung von Aupperle dar. Lediglich bei der WLAN-Übertragung über das MQTT-Protokoll wurden gelegentliche Unterbrechungen der Übertragung festgestellt, sodass hier noch technischer Klärungsbedarf nötig ist.

Der Telemetriesender wurde angehenden Lehrern sowie einer in Forschung und Lehre tätigen Person vorgeführt und Rückmeldungen bezüglich Funktionalität und Eignung für Experimentieraufgaben eingeholt. So wurde angemerkt, dass ein größerer Messbereich als ±4 g für stärkere Wurf- und Stoßvorgänge wünschenswert wäre. Hierzu müsste eine andere Bibliothek zur Ansteuerung der IMU gefunden werden, die eine Einstellung des Messbereichs ermöglicht, oder aber die schon verwendete modifiziert werden. Dabei muss aber beachtet werden, dass eine Vergrößerung des Messbereichs mit einer verringerten Auflösung einhergeht, die für kleinere Beschleunigungen nachteilig ist. Eine nützliche funktionale Erweiterung wäre, Messdaten (bei fehlender Verbindung zum Raspberry Pi) auf dem Sender zu speichern und nachträglich an das PhyPiDAQ-Programm zu übertragen. Für das PhyPiDAQ-Programm wurden mehrere Ergänzungs- und Verbesserungsvorschläge vorgebracht. So wurde eine einfachere Einstellung der Parameter gewünscht, zum Beispiel auch ein einfaches Wechseln zwischen zwischen der Darstellung der Beschleunigung in g und in m/s^2. Neben der bloßen grafischen Darstellung über der Zeit wäre auch eine direkte numerische Anzeige für die Beschleunigungen der drei Achsen nützlich. Durch Integration der Beschleunigung könnten in der PhyPiDAQ-Umgebung auch Geschwindigkeitsverläufe in verschiedenen Raumrichtungen dargestellt werden.

Als weiteres Anwendungsgebiet für Telemetriesender jenseits der Beschleunigungsmessung bietet sich das Messen von Druck und Temperatur in abgeschlossenen Behältern an. So könnte in speziellen Kolben, in welchen der Telemetriesender hineingelegt wird, die Temperatur- und Druckänderung in Abhängigkeit der Volumenänderung gemessen werden. Bei konstant gehaltenem Volumen hingegen könnte die proportionale Abhängigkeit von

Druck und Temperatur aufgezeichnet werden. Druck- und Temperatursensoren können dabei über SPI- oder I^2C-Pins an das Arduino-Board angeschlossen werden. Die Sende- und Empfangsprogramme müssten dafür an einzelnen Stellen angepasst werden.

Um die Abhängigkeit von einem einzelnen Mikrocontroller-Modell zu reduzieren, wäre es lohnenswert, auch die anderen in Kapitel 3 erwähnten Arduino-Nano-Modelle als Telemetriesender zu erproben. Die verwendeten Bibliotheken WiFiNINA, ArduinoBLE, sowie PubSubClient sind mit allen Arduino-Boards kompatibel, sodass die entsprechenden Codestellen auf allen Boards lauffähig sind. Lediglich die Bibliothek zur Ansteuerung der IMU muss an das jeweilige auf dem Arduino-Board verbaute Modell angepasst und die verwendeten Methoden gegebenenfalls abgeändert werden.

Anhang

A. Datenübertragung mit MQTT

Ursprünglich im Jahr 1999 als schlankes Übertragungsprotokoll für die Funkverbindung zu Öl-Pipelines via Satellit entwickelt, wird MQTT heute als Übertragungsprotokoll für das Internet der Dinge (*Internet of Things, IoT*) verwendet. Im Vergleich zu anderen Verfahren zur Datenübertragung gilt MQTT als einfach, leicht zu implementieren und benutzerfreundlich. Ein geringes Verhältnis von Verwaltungsdaten gegenüber Nutzdaten zeichnet MQTT gegenüber anderen netzwerkbasierten Protokollen wie HTTP aus und macht es zur ersten Wahl bei der Datenübertragung zwischen WLAN-fähigen Geräten. Die folgenden Ausführungen beruhen auf [A18].

A. Grundlegende Struktur von MQTT

Die Vorteile von MQTT werden dadurch erreicht, dass Sender und Empfänger nicht mehr direkt mit einander kommunizieren, sondern ein sogenannter Broker zwischengeschaltet wird, der Nachrichten vom Sender – dem Publisher – empfängt und diese an den Empfänger – dem Subscriber – weiterleitet. Dies führt zu einer Entkopplung auf drei Ebenen:

- Räumliche Entkopplung: Publisher und Subscriber müssen sich für den Datentransfer nicht kennen. Lediglich die IP-Adresse und der Port des Brokers müssen für den Publisher und den Subscriber bekannt sein.
- Zeitliche Entkopplung: vom Publisher gesendete Nachrichten müssen nicht unmittelbar an den Subscriber weitergeleitet werden. Ist der Subscriber gerade nicht mit dem Broker verbunden, kann der Broker Nachrichten zwischenspeichern und diese übertragen, sobald sich der Subscriber wieder mit dem Broker verbindet.
- Synchronisationsentkopplung: Publisher und Subscriber müssen sich beim Senden und Empfangen von Daten nicht an einen gemeinsamen Takt halten.

Ein MQTT-Broker dient als zentrale verwaltende Einheit, während Publisher und Subscriber die Peripherie darstellen und als Clients zusammengefasst werden. Theoretisch kann ein Broker eine unbegrenzte Anzahl an Clients verwalten. Dabei empfängt der Broker alle eingehenden Nachrichten von Publishern, bestimmt, welche Subscriber welche Nachrichten abonniert haben, und sendet die Nachrichten an diese Abonnenten. Die Publisher- und die Subscriber-Funktion schließen dabei einander nicht aus: jeder Client kann beide Funktionen erfüllen und als Subscriber Daten empfangen wie auch als Publisher Daten senden. Auch der Broker selbst kann als Publisher oder Subscriber verwendet werden, was in der Anwendung auch gängige Praxis ist.

Für die Verwaltung der Nachrichten besteht jede gesendete Nachricht im Wesentlichen aus zwei Teilen, einem Topic und dem Payload. Das Topic dient als Name der Nachricht, der diese eindeutig identifiziert, das Payload ist der eigentliche Inhalt. Will ein Subscriber so die Nachricht eines Publishers empfangen, muss er beim Broker das entsprechende Topic abonnieren.

B. Verbindungsaufbau

Um eine Verbindung aufzubauen, sendet der Client eine CONNECT-Botschaft an den Broker. Der Broker antwortet mit einer CONNACK-Botschaft, die unter anderem die Information beinhaltet, ob die Verbindung erfolgreich eingerichtet wurde, oder aus welchem Grund diese fehlgeschlagen ist. Ist die Verbindung eingerichtet, hält der Broker diese offen, bis der Client ein DISCONNECT-Befehl sendet oder die Verbindung abbricht.

Das Stehen der Verbindung wird durch den Keep-Alive-Mechanismus kontrolliert. Dabei sendet der Client beim Verbindungsaufbau ein Zeitintervall an den Broker, das die maximal erlaubte Dauer festlegt, während der keine Kommunikation zwischen Broker und Client stattfinden darf. Innerhalb dieses Intervalls sendet der Client eine PINGREQ-Botschaft an den Broker, womit dieser das Stehen der Verbindung zum Client überprüfen kann. Als Reaktion sendet der Broker eine PINGRESP-Botschaft an den Client zurück, womit dieser das Stehen der Verbindung zum Broker überprüft. Empfängt der Broker nach der anderthalbfachen Dauer des Keep-Alive-Intervalls keine Botschaft des Clients, trennt dieser die Verbindung. Durch das Ausbleiben der PINGRESP-Botschaft kann die getrennte Verbindung auch vom Client erkannt werden, der daraufhin gegebenenfalls wieder versuchen kann, diese herzustellen. PINGREQ-Botschaften werden nur gesendet, solange keine gewöhnlichen Publish-Nachrichten innerhalb des Keep-Alive-Zeitintervalls verschickt werden. Eine kontinuierliche Datenübertragung ersetzt den Keep-Alive-Mechanismus.

MQTT unterscheidet ferner zwischen zwei Arten der Verbindung zwischen Client und Broker: der Peristent Session und der Clean Session. Wird bei einer Clean Session die Verbindung unterbrochen, werden auch alle Daten des Clients beim Broker gelöscht. Bei der Wiederherstellung der Verbindung müssen somit auch alle abonnierten Topics neu abonniert werden. Bei der Persistent Session hingegen bleiben die Informationen der Sitzung auch nach Abbruch der Verbindung bestehen: bei Wiederherstellung der Verbindung müssen Topics nicht neu abonniert werden, außerdem können in der Zwischenzeit gesendete Nachrichten zwischengespeichert und bei Neuverbindung nachgereicht werden.

Für eine Persistent Session muss dem Client eine Client-ID zugewiesen werden, die diesen gegenüber dem Broker ausweist, und unter welcher die Status-Daten des Clients gespeichert werden.

C. Publish und Subscribe

Ein Publisher kann ohne weitere Bedingungen Nachrichten an den Broker senden, sobald eine Verbindung eingerichtet ist. Die Weiterverbreitung der Nachrichten an mögliche Subscriber obliegt dem Broker. Der Publisher erhält so keine Rückmeldung darüber, ob Subscriber an der Nachricht interessiert sind oder wie viele Subscriber die Nachricht erhalten haben. Zu sendende Nachrichten sind nicht auf ein bestimmtes Datenformat festgelegt; es ist möglich jede Art von Daten, von Bild- bis zu Textdaten, zu übertragen.

Damit der Subscriber Nachrichten vom Broker empfangen kann, muss dieser zuerst eine SUBSCRIBE-Nachricht an den Broker senden, um die entsprechende Nachricht zu abonnieren. Diese besteht aus einer speziellen Identifikationsnummer und einer Liste der zu abonnierenden Topics. Als Reaktion schickt der Broker eine SUBBACK-Nachricht an den Client zurück, die den Erfolg oder Misserfolg des Abonnements rückmeldet.

D. Quality of Service

MQTT bietet die Möglichkeit, die Zuverlässigkeit der Datenübertragung mit drei Quality-of-Service-Stufen (QoS) einzustellen. Sofern keine explizite Einstellung des QoS vorgenommen wird, wird jede Nachricht mit dem QoS-Level 0 übertragen: Dies heißt, eine Nachricht

wird maximal einmal übertragen; es gibt keine Kontrolle, die die korrekte Übertragung der Nachricht bestätigt. Beim QoS-Level 1 wird die Nachricht mindestens einmal übertragen; die Nachricht erreicht hier sicher den Empfänger, es ist aber möglich, dass Nachrichten doppelt empfangen werden. Soll der genau einmalige Empfang einer Nachricht sichergestellt werden, kann die Nachricht mit dem QoS-level 2 übertragen werden.

Jede Übertragung einer Nachricht vom Sender zum Empfänger ist in MQTT in zwei Abschnitte unterteilt: der Übertragung vom Publisher zum Broker und der Weitergabe der Nachricht vom Broker zum Subscriber. Dabei wird für jeden Abschnitt das QoS-Level getrennt festgelegt. So schickt der Publisher neben dem Topic und dem Payload auch die Information über das gewünschte QoS-Level an den Broker. Auf der Empfängerseite teilt der Subscriber dem Broker beim Abonnieren eines Topics mit, auf welchem QoS-Level die Nachricht empfangen werden soll.

Beim QoS-Level 1 speichert der Sender die gesendete Nachricht solange, bis dieser vom Empfänger eine PUBACK-Botschaft erhält, welche den erfolgreichen Empfang der Nachricht bestätigt. Trifft innerhalb eines festgelegten Zeitintervalls keine PUBACK-Botschaft ein, interpretiert der Sender dies als fehlgeschlagene Übertragung und sendet die Nachricht erneut. Eintreffende Nachrichten mit QoS-Level 1 werden vom Empfänger unmittelbar verarbeitet; ist der Empfänger ein Broker, wird die Nachricht sofort an Subscriber weitergesendet, ehe die PUBACK-Botschaft an den Publisher zurückgeschickt wird.

Das genau einmalige Übertragen einer Nachricht beim QoS-Level 2 wird durch insgesamt vier Kommunikationsschritte zwischen Sender und Empfänger sichergestellt. Erhält ein Empfänger eine Nachricht von einem Sender, wird ein PUBREC-Paket als Empfangsbestätigung an den Sender zurückgesendet. Nun kann der Sender das zwischengespeicherte Publish-Paket löschen. Daraufhin speichert der Sender das PUBREC-Paket und antwortet dem Empfänger mit einem PUBREL-Paket. Dieses signalisiert dem Empfänger, dass die gespeicherten Daten des Publish-Paketes gelöscht werden können. Zum Abschluss des Prozesses schickt der Empfänger ein PUBCOMP-Paket an den Sender zurück, womit der Prozess auf den Anfangszustand zurückgesetzt wird und nun das nächste Publish-Paket gesendet werden kann. Kommt auf jede gesendete Botschaft des Senders innerhalb eines festgelegten Zeitintervalls keine Rückmeldung, wird die Botschaft erneut gesendet. Vom Senden der Publish-Botschaft bis zum Senden des PUBCOMP-Paketes werden dabei die Identifikationsdaten des Publish-Pakets gespeichert; doppelt empfangene Publish-Nachrichten werden so erkannt und eine doppelte Verarbeitung verhindert.

Die mit den QoS-Levels einhergehenden Kontrollschritte haben den Nachteil, den Overhead zu erhöhen und die Datenübertragung zu verlangsamen. Die QoS-Level 1 und 2 werden daher nur dann eingesetzt, wenn jede gesendete Nachricht zwingend benötigt wird und keine hohe Datenübertragungsrate erforderlich ist.

E. Empfangsprogramm

Für die Datenübertragung mit dem MQTT-Protokoll wird der Raspberry Pi als Broker eingerichtet. Ein darauf ausgeführtes Python-Programm nimmt die Rolle eines Subscribers ein, indem es die Daten des Telemetriesenders verarbeitet und ausgibt. Zusätzlich können Steuerbefehle abgesendet werden. Für die Einrichtung als Broker wird der Open-Source-Broker Mosquitto auf dem Raspberry Pi installiert [A19].

```
1 import paho.mqtt.client as mqtt
2
3 mqtt_broker_ip  = "127.0.0.1"   #Standard "localhost"
4 subscribe_topic = "ms2"
```

```
5  publish_topic   = "Steuerung"
6
7  client = mqtt.Client()
8
9  # As soon as client is connected to broker, subscribe to Topic
10 def on_connect(client, userdata, flags, rc):
11     print("Connected!", str(rc))
12     client.subscribe(subscribe_topic)
13
14 def on_message(client, userdata, msg):
15     str=msg.payload.decode("utf-8")
16     list = str.split(" ")
17     print(float(list[0]), float(list[1]), float(list[2]))
18
19 client.on_connect = on_connect
20 client.on_message = on_message
21
22 # Connect to the broker
23 client.connect(mqtt_broker_ip, 1883)
24
25 # Run continously
26 client.loop_start()
27
28 while True:
29     if input() == "s":
30         print("Messung starten")
31         client.publish(publish_topic, "start")
32     if input() == "e":
33         client.publish(publish_topic, "stopp")
34         print("Messung stoppen")
35     else:
36         pass
```

Zeile 1:
Für den Aufruf der notwendigen Funktionen wird die paho-mqtt-Bibliothek auf dem Raspberry Pi installiert [A20], das Modul `paho.mqtt.client` in das Programm importiert und diesem für nachfolgende Aufrufe der Namensraum `mqtt` zugewiesen.

Zeilen 3 - 5:
Um sich mit dem Broker über das WLAN-Netz zu verbinden, muss der Client über dessen IP-Adresse verfügen. Da sich in diesem Falle Broker und Client auf demselben Gerät befinden, kann stattdessen die Localhost-Domaine „127.0.0.1“ verwendet werden, die standardmäßig den lokalen Rechner adressiert, auf welchem das jeweilige Programm ausgeführt wird.

Zur Übersichtlichkeit werden die Namen des Subscribe-Topics und des Publish-Topics am Kopf des Programmes definiert. Unter dem Topic `ms2` werden die Beschleunigungsdaten des Telemetriesenders empfangen, während unter dem Topic `Steuerung` Befehle an den Telemetriesender gesendet werden können, die das Senden der Daten starten oder stoppen.

Zeile 7:
Um nachfolgende Methoden aufzurufen, wird ein `client`-Objekt erzeugt.

Zeilen 10 - 12:
Paho-MQTT nutzt sogenannte Callback-Funktionen, die automatisch beim Eintreffen eines

spezifischen Ereignisses aufgerufen werden, und vom Anwender selbst definiert werden müssen. Die Funktion `on_connect` wird aufgerufen, sobald eine Verbindung zum Broker hergestellt ist. Dabei wird über die Konsole die Nachricht „Connected!“ ausgegeben und sofort das Subscribe-Topic zum Empfang der Beschleunigungsdaten abonniert.

Zeilen 14 - 17:
Die Callback-Funktion `on_message` wird beim Empfang jeder Nachricht aufgerufen und verarbeitet die ankommenden Daten, die über die Variable `msg` abgerufen werden können. Die empfangene Nachricht liegt zunächst in Form eines Byte-Strings vor und wird in einen String umgewandelt. Die Beschleunigungswerte für die drei zu einander orthogonalen Richtungen x, y, und z sind durch ein Leerzeichen getrennt. Dieses wird von der `split`-Funktion genutzt, um den String in drei Teile zu teilen und eine Liste mit den drei Beschleunigungen als Elemente zu erzeugen. Über den `print`-Befehl werden die Beschleunigungen auf der Konsole ausgegeben, wobei die einzelnen Elemente der Liste als String vorliegen und über den `float`-Befehl in eine Dezimalzahl umgewandelt werden.

Zeilen 19 - 20:
Die eben behandelten Callback-Funktionen können beliebig benannt werden, lediglich ihre Parameter sind vorgegeben. Damit die definierten Funktionen vom Client erkannt und von diesem aufgerufen werden können, müssen die Funktionen dem Client auf entsprechende Weise zugeordnet werden.

Zeile 23:
Über die `connect`-Methode wird die Verbindung zum Broker aufgebaut; als Parameter werden die IP-Adresse des Brokers und der Port 1883 des TCP-Protokolls verwendet, auf dessen Grundlage MQTT operiert.

Zeile 26:
Mit der `loop_start`-Funktion wird der kontinuierliche Ablauf des Programms gestartet.

Zeilen 28 - 36:
Als beispielhafte Realisierung für eine Steuerung des Telemetriesenders werden in einer `while`-Schleife kontinuierlich Tastatureingaben abgefragt. Wird in die Konsole der Buchstabe „s“ eingetippt, wird über die `publish`-Methode das Publish-Topic mit der Nachricht „start“ an den Telemetriesender gesendet, der bei Empfang der Nachricht das Senden der Beschleunigungsdaten startet. Bei Eintippen des Buchstabens „e“ wird die Nachricht „stopp“ gesendet, womit der Telemetriesender das Senden der Daten stoppt. Damit die Schleife bei Ausbleiben einer Eingabe durchläuft, ist eine `else`-Bedingung mit dem Schlüsselwort `pass` notwendig.

F. Sendeprogramm in C++

Das Arduino-Board nimmt die Rolle eines Publishers ein, der die Beschleunigungsdaten an den Raspberry Pi als Broker sendet. Im Hinblick auf den hohen Energieverbrauch des WLAN-Moduls wurde eine Steuerung implementiert, die es erlaubt, das Senden von Daten an- und auszuschalten. So sollen nur auf Befehl des Empfangsprogramms hin Daten gesendet und damit unnötiger Stromverbrauch vermieden werden. Zur Benutzerfreundlichkeit und zur Sichtbarmachung von Fehlfunktionen werden auf dem Board eine blaue LED angeschaltet, wenn die Verbindung zum Broker unterbrochen ist, und eine grüne LED angeschaltet, wenn keine Verbindung zum WLAN-Netz vorliegt.

```
1 #include <WiFiNINA.h>
2 #include <PubSubClient.h>
3 #include <Arduino_LSM6DSOX>
```

```
4
5  // Adapt this for your own WiFi-Network
6  const char* ssid        = "name_of_your_WiFi_network";
7  const char* password    = "password_of_your_WiFi_network";
8  const char* BrokerIP = "xxx.xxx.x.xxx";
9
10 // set time between two samples in milliseconds
11 unsigned long Abtastrate = 10;
12
13 // set different ClientID if you use more than one sender
14 const char* ClientID = "Arduino1";
15
16 const char* Publish_Topic = "ms2";
17 const char* Subscribe_Topic = "Steuerung";
18
19 float x, y, z;
20 bool flag;
21 String message = "stopp";
22 unsigned long lastMsg = 0;
23
24 WiFiClient RP2040Client;
25 PubSubClient client(RP2040Client);
26
27 void setup_wifi() {
28   pinMode(LEDG, OUTPUT);
29
30   while (WiFi.status() != WL_CONNECTED) {
31     digitalWrite(LEDG, HIGH);
32     WiFi.begin(ssid, password);
33     digitalWrite(LEDG, LOW);
34     delay(100);
35   }
36 }
37
38 void callback(char* topic, byte* payload, unsigned int length) {
39   Serial.println("Message arrived on topic: " + String(topic));
40   Serial.print("payload:  ");
41   for (int i = 0; i < length; i++) {
42     message += (char)payload[i];
43   }
44   Serial.println(message);
45
46   if (topic = "Steuerung"){
47     if (message == "start"){flag = true;}
48     if (message == "stopp"){flag = false;}
49     message = "";
50   }
51 }
52
53 void reconnect() {
54   pinMode(LEDB, OUTPUT);
55
```

```
  while (!client.connected()) {
    digitalWrite(LEDB, HIGH);
    if (client.connect(ClientID,"NULL","NULL",{},{},{},{},0)) {
      digitalWrite(LEDB, LOW);
    } else {
      Serial.print("Connection to broker failed, rc=");
      Serial.println(client.state());
    }
  }
}

void setup() {
  Serial.begin(115200);
  delay(1000);

  // connect to WiFi
  setup_wifi();

  // set broker and callback
  client.setServer(BrokerIP, 1883);
  client.setCallback(callback);

  // connect to broker and subscribe
  reconnect();
  client.subscribe(Subscribe_Topic, 1);

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

void loop() {
  if (!client.connected()) {
    reconnect();
  }
  client.loop();

  if(flag == true){
    unsigned long now = millis();
    if (now - lastMsg > Abtastrate) {
      lastMsg = now;

      IMU.readAcceleration(x, y, z);

      client.publish(Publish_Topic, (String(x) + " " +
                                     String(y) + " " +
                                     String(z)).c_str());
    }
  }
}
```

Zeilen 1 - 3:
Für das Sendeprogramm auf dem Arduino-Board werden zunächst die nötigen Bibliotheken eingebunden. `WiFiNINA` wird für die Verwendung des WLAN-Moduls benötigt. Mit der `PubSubClient`-Bibliothek können sämtliche MQTT-bezogenen Funktionen eingerichtet werden. Um den Accelerometer zu verwenden und die Beschleunigungsdaten auszulesen, wird schließlich die Bibliothek `Arduino_LSM6DSOX` eingebunden.

Zeilen 6 - 8:
Damit das Arduino-Board sich mit dem WLAN-Netz und dadurch mit dem Broker auf dem Raspberry Pi verbinden kann, müssen die Zugangsdaten des Netzes und die IP-Adresse des Raspberry Pi eingegeben werden. Diese werden Variablen zugewiesen, um an der entsprechenden Stelle im Quellcode aufgerufen zu werden. Alle Daten werden dabei in Anführungszeichen, also als Strings definiert. Die IP-Adresse des Raspberry Pi kann durch Eingabe des Befehls `hostname -I` im Terminal bezogen werden.

Zeile 11:
Die Abtastrate, also die zeitliche Dichte des Ablesens der Beschleunigungswerte, kann eingestellt werden, indem die Dauer zwischen zwei Abtastungen in Millisekunden der Variable `Abtastrate` zugewiesen wird. Zehn Millisekunden entsprechen einer Rate von 100 Hz; dies ist die maximal mögliche Übertragungsrate.

Zeile 14:
Damit abonnierte Topics vom Broker nach einer Unterbrechung der Verbindung nicht neu abonniert werden müssen, soll eine Persistent Session eingerichtet werden. Dazu muss dem Client, also dem Telemetriesender, eine Client-ID zugewiesen werden, welche diesen gegenüber dem Broker eindeutig ausweist.

Zeilen 16 - 17:
Analog zum Empfangsprogramm werden auch hier die Topic-Namen `ms2` und `Steuerung` definiert, unter welchen die Beschleunigungsdaten versendet und umgekehrt Steuerbefehle empfangen werden.

Zeilen 19 - 22:
Am Kopf des Programmcodes werden schließlich alle nötigen Variablen für den Programmablauf definiert. Die Variable `message` dient zum Speichern der vom Empfangsprogramm gesendeten Steuerbefehle. Damit erst bei Empfang einer „start“-Nachricht Daten gesendet werden, ist der Inhalt „stopp“ vordefiniert.

Zeilen 24 - 25:
Als nächstes wird eine teilinizialisiertes `client`-Objekt erzeugt. Hierfür wird der Network Client `WiFiClient` gewählt und diesem ein frei gewählter Name wie „RP2040Client“ zugewiesen. Dieser Name wird als Parameter in die Klammer des Client-Constructors geschrieben.

Zeilen 27 - 36:
Die `setup_wifi`-Funktion führt die Verbindung zum WLAN-Netz durch, indem eine `while`-Schleife den Verbindungsstatus abfragt. Solange keine Verbindung zu einem WLAN-Netz vorhanden ist, wird als optische Rückmeldung die grüne LED angeschaltet und die Methode `WiFi.begin` periodisch aufgerufen, welche den Verbindungsprozess durchführt. Ist dieser erfolgreich, wird die Schleife verlassen.

Zeilen 38 - 51:
In der `callback`-Funktion werden ankommende Nachrichten abgerufen und verarbeitet. Dabei können über die Parameter `topic`, `payload` und `length` sowohl das Topic der Nachricht, der Inhalt der Nachricht als auch die Länge der Nachricht in Bytes bezogen werden. Für das Auslesen der Nachricht wird eine `for`-Schleife verwendet, die mithilfe

der `length`-Variable die `payload`-Ladung Byte für Byte ausliest, in den Datentyp `char` umformatiert und damit die anfangs definiert Variable `message` auffüllt. In Abhängigkeit der empfangenen Nachricht können nun beliebige Programmschritte veranlasst werden. Für die Steuerung des Telemetriesenders wird so mit zwei `if`-Bedingungen der Inhalt der bezogenen Nachricht kontrolliert. Wurde „start“ gesendet, wird eine Flagge (*flag*) auf `true` gesetzt. Wurde „stopp“ gesendet, wird die Flagge auf `false` gesetzt. Die `flag`-Variable ist im ganzen Programm sichtbar und wird später verwendet, um Senden von Nachrichten in der `loop`-Funktion auf einfache Weise an- und auszuschalten. Um zu verhindern, dass die `message`-Variable mit zwei aufeinanderfolgenden Nachrichten befüllt wird, wird diese am Ende des Funktionsablaufs wieder geleert.

Zeilen 53 - 65:
Mit der Funktion `reconnect` wird die Verbindung zum Broker durchgeführt. Dabei prüft eine `while`-Schleife den Verbindungsstatus ab und schaltet bei fehlender Verbindung als optische Rückmeldung die blaue LED ein. Mit der `connect`-Methode wird die Verbindung zum Broker eingerichtet, wobei dem letzten Parameter der boolsche Wert null zugewiesen wird, um eine Persistent Session einzurichten. Bei erfolgreicher Verbindung wird die blaue LED ausgeschaltet und die Schleife verlassen; bei Fehlschlag wird über die `print`-Befehle eine Fehlermeldung ausgegeben, die im Seriellen Monitor der Arduino-IDE augelesen werden kann, und die Schleife neu durchlaufen.

Zeilen 67 - 87:
In der `setup`-Funktion wird die Initialisierung des Programms vorgenommen. Um Meldungen im Seriellen Monitor ausgeben zu können, wird der serielle Port des Boards mit `Serial.begin` initialisiert. Zur Verbindung mit dem WLAN-Netz wird die zuvor definierte `setup_wifi`-Funktion ausgeführt. Mit der Methode `setCallback` wird die zuvor definierte `callback`-Funktion dem `client`-Objekt zugewiesen und kann dadurch erkannt und ausgeführt werden. Im nächsten Schritt wird die Verbindung zum Broker hergestellt und zum Erhalt von Steuerbefehlen das Topic „Steuerung“ mit dem QoS-Level „1“ abonniert. Zuletzt wird mit dem Initialisieren der IMU das Accelerometer aktiviert; bei Fehlschlag wird eine Fehlermeldung ausgegeben.

Zeilen 89 - 107:
Die `loop`-Funktion beinhaltet den kontinuierlich auszuführenden Teil des Programms. Zu Beginn jeden Durchlaufs wird die Verbindung zum Broker kontrolliert und bei Abbruch der Verbindung diese neu hergestellt. Die `loop`-Methode stellt eine Verknüpfung dar, die den Inhalt der `callback`-Funktion importiert und so dafür sorgt, dass die `flag`-Variable kontinuierlich aktualisiert wird. Das Auslesen und Senden der Beschleunigungswerte findet innerhalb von zwei verschachtelten `if`-Bedingungen statt. Die äußere `if`-Bedingung schaltet das Senden der Beschleunigungswerte in Abhängigkeit der `flag`-Variable an oder aus. Die innere `if`-Bedingung steuert die Abtastrate, indem die `millis`-Methode den in der Variable `Abtastrate` hinterlegten Wert abzählt. Mit `readAcceleration` werden die Beschleunigungswerte für die drei Richtungen x, y, und z in der Einheit g ($g = 9{,}81\,\mathrm{m/s^2}$) ausgelesen und mit der `publish`-Methode als String versendet. Dabei wird zwischen jedem Wert ein Leerzeichen eingefügt, um im Empfangsprogramm die Werte durch dieses wieder von einander zu trennen.

G. Sendeprogramm in Circuitpython

Das Arduino-Board kann auch mit Micropython und Circuitpython programmiert werden. Um die Möglichkeiten zu vergleichen, wurde auch ein Sendeprogramm mit Circuitpython implementiert. Dazu muss eine uf2-Datei auf das Board geladen werden, die unter anderem einen Python-Interpreter installiert [A21]. Die nötigen Bibliotheken können unter https://circuitpython.org/libraries heruntergeladen werden.

```
import board
import busio
from digitalio import DigitalInOut
from adafruit_esp32spi import adafruit_esp32spi
from adafruit_esp32spi import adafruit_esp32spi_wifimanager
import adafruit_esp32spi.adafruit_esp32spi_socket as socket
import adafruit_minimqtt.adafruit_minimqtt as MQTT
from adafruit_lsm6ds import Rate
from adafruit_lsm6ds.lsm6dsox import LSM6DSOX

# Adapt this for your own WiFi-Network
secrets = {
    "ssid"        : "name_of_your_WiFi_network",
    "password"    : "password_of_your_WiFi_network",
    "broker"      : "xxx.xxx.x.xxx",
    "broker_port" : 1883,
}

flag            = False
publish_topic   = "ms2"
subscribe_topic = "Steuerung"

esp32_cs = DigitalInOut(board.CS1)
esp32_ready = DigitalInOut(board.ESP_BUSY)
esp32_reset = DigitalInOut(board.ESP_RESET)
spi = busio.SPI(board.SCK1, board.MOSI1, board.MISO1)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready
                                       , esp32_reset)

i2c = board.I2C()  # uses board.SCL and board.SDA
sensor = LSM6DSOX(i2c)
sensor.accelerometer_data_rate = Rate.RATE_104_HZ

def on_connect(client, userdata, flags, rc):
    print("Connected to MQTT Broker!")
    print("Subscribing to %s" % subscribe_topic)
    client.subscribe(subscribe_topic)

def on_message(client, topic, message):
    print("New message on topic {0}: {1}".format(topic, message))

    global flag

    if message == "start":
        flag = True
    if message == "stopp":
        flag = False

print("Connecting to WiFi...")
wifi = adafruit_esp32spi_wifimanager.ESPSPI_WiFiManager(esp,
                                    secrets)
```

```
50 wifi.connect()
51 print("Connected!")
52
53 MQTT.set_socket(socket, esp)
54 client = MQTT.MQTT(broker=secrets["broker"], port=secrets["
                                    broker_port"])
55
56 client.on_connect = on_connect
57 client.on_message = on_message
58
59 print("Attempting to connect to %s" % client.broker)
60 client.connect()
61
62 while True:
63     try:
64         client.loop(timeout=0.01)
65     except (ValueError, RuntimeError) as e:
66         print("Failed to get data, retrying\n", e)
67         wifi.reset()
68         client.reconnect()
69         continue
70
71     if flag == True:
72         client.publish(publish_topic,
73                        "%.3f %.3f %.3f " % (sensor.acceleration))
```

Zeilen 1 - 9:
Zunächst werden im Sendeprogramm die notwendigen Module eingebunden. Das Modul `board` enthält eine Sammlung von Konstanten, mit welchen die Pins des Boards angesprochen werden können. Das Modul `busio` enthält Klassen zur Steuerung von Schnittstellen wie I^2C, SPI und UART. Vom Modul `digitalio` wird die Funktion `DigitalInOut` genutzt, um die entsprechenden Pins als digitale Ein- und Ausgänge zu definieren. Um das Funk-Modul anzusprechen (welches auf einem ESP32-Modul basiert), wird das Modul `adafruit_esp21spi` verwendet. Weiter wird für die MQTT-Funktionalität das Paket `adafruit_minimqtt` verwendet und diesem für eine kürzere Darstellung der Namensraum `MQTT` zugewiesen. Um auf den Beschleunigungssensor zuzugreifen und die Auslesefrquenz festzulegen, werden zuletzt die Module `LSM6DSOX` und `Rate` importiert.

Zeilen 12 - 17:
Um zentrale Einstellungen gebündelt vorzunehmen, wird ein Dictionary mit dem Namen `secrets` definiert, im welchem der Name des WLAN-Netzes, dessen Passwort, sowie die IP-Adresse des Brokers und der Broker-Port festgelegt werden können.

Zeilen 19 - 21:
Danach werden eine Reihe weiterer Variablen definiert. So wird eine Flagge (`flag`) als boolsche Variable genutzt, um das Senden der Beschleunigungsdaten an- und auszuschalten. Ebenso werden die variablen `publish_topic` und `subscribe_topic` definiert und über diese die Topics für die zu sendenden und zu empfangenden Nachrichten festgelegt.

Zeilen 23 - 27:
Der RP2040-Prozessor ist über eine interne SPI-Verbindung mit dem Funk-Modul Nina-W102 verbunden. Durch die Initialisierung der SPI-Schnittstellen des Moduls und des Prozessors wird eine Verbindung hergestellt und schließlich das Objekt `esp` erzeugt.

Zeilen 29 - 31:
Der RP2040-Prozessor ist über eine interne I^2C-Schnittstelle mit der IMU verbunden. Analog zum Funk-Modul wird durch Initialisieren der I^2C-Schnittstelle die Verbindung zur IMU hergestellt. Die internen Pins für die I^2C-Schnittstelle benötigen dabei keinen expliziten Aufruf. Mit dem so erzeugten Objekt `sensor` kann später die Methode zum Ablesen der Beschleunigungsdaten aufgerufen werden. Ferner wird die Auslesefrequenz auf 104 Hz festgelegt.

33 - 36:
Analog zur Verwendung im Empfangsprogramm veranlasst die `on_connect`-Callback-Funktion ein Abonnement des Subscribe-Topics, sobald eine Verbindung zum Broker aufgebaut wurde. Die `print`-Befehle im Funktionskörper dienen lediglich der Rückmeldung über den Progammablauf bei geöffneter Entwicklungsumgebung und erleichtern eine mögliche Fehleranalyse.

38 - 46:
Analog zum Empfangsprogramm werden mit der Funktion `on_message` in Abhängigkeit der empfangenen Nachricht spezifische Programmschritte ausgeführt. Wird die Nachricht „start“ empfangen, wird die Flagge auf `True` gesetzt, wodurch – wie weiter unten erläutert – das Senden der Beschleunigungsdaten angeschaltet wird. Umgekehrt wird bei Empfang der Nachricht „stopp“ die Flagge auf `False` gesetzt, womit das Senden der Daten gestoppt wird.

Zeilen 48 - 51:
Um eine Verbindung zum WLAN-Netz aufzubauen, muss zunächst das Objekt `wifi` erzeugt werden, wofür das Objekt `esp` sowie die im Dictionary `secrets` festgehaltenen Daten als Parameter dienen. Mit dem `wifi`-Objekt kann daraufhin die `connect`-Funktion aufgerufen werden. Bei erfolgreichem Verbindungsaufbau zum Router wird mit einem `print`-Befehl Rückmeldung über die erfolgreiche Verbindung gegeben.

Zeilen 53 - 54:
Um eine MQTT-Verbindung zum Broker einzurichten, wird zunächst ein Socket gesetzt und ein `client`-Objekt erzeugt, mit welchem alle weiteren MQTT-bezogenen Methoden aufgerufen werden können.

Zeilen 56 - 60:
Damit die definierten Callback-Funktionen vom Client erkannt und verwendet werden können, müssen die Funktionen dem Client in gezeigter Weise zugeordnet werden. Über die `connect`-Methode wird schließlich die Verbindung hergestellt.

Zeilen 62 - 73:
Anstelle der `loop`-Funktion im Arduino-Programm wird im Circuitpython-Programm eine `while`-Dauerschleife verwendet, in welcher sonst weitgehend identische Programmschritte wie im Arduino-Code ablaufen. In einer `try/except`-Struktur wird die Verbindung zum Broker überprüft. Die `loop`-Methode erfüllt die Aufgabe, in festgelegten Zeitintervallen eingehende Nachrichten von abonnierten Topics zu registrieren und den Inhalt der `on_message`-Funktion in die `while`-Schleife einzubetten. Das `timeout`-Zeitintervall wurde hierbei auf 0,01 s festgelegt. Bei einer Störung des WLAN-Netzes oder einer unterbrochenen Verbindung zum Broker gibt die `loop`-Methode entsprechende Fehlermeldungen zurück, wodurch die unter `except` aufgeführten Methoden aufgerufen werden, die ein Reset des WLANs sowie eine Neuverbindung zum Broker durchführen.

Innerhalb der `if`-Bedingung werden mit dem Befehl `publish` die Beschleunigungsdaten des Sensors gesendet. In Abhängigkeit davon, ob die Variable `flag` den Wert `True` oder `False` hat, kann somit das Senden und Auslesen der Daten an- und ausgeschaltet werden. Über `sensor.acceleration` wird hierbei ein Tupel aus den Beschleunigungsdaten der

drei Richtungen x, y und z in m/s^2 ausgegeben, die als float-Zahlen in einen String eingefügt werden.

H. Vergleich beider Sendeprogramme

Die leichte Erlernbarkeit von Python, das Entfallen oft zeitraubenden Kompilierens sowie die große hard- und softwareseitige Unterstützung der Firma Adafruit machen das Programmieren mit Ciruitpython zu einer guten Wahl, wenn das Board auch durch Schüler im Rahmen von Unterrichtsprojekten programmiert werden soll. Die Geschwindigkeit des ausgeführten CircuitPython-Codes erwies sich jedoch als deutlich langsamer als bei der Implementierung in Arduino. Während im Arduino-Code die Beschleunigungsdaten mit einer Rate von 104 Hz übertragen wurden, wie sie auch durch die Voreinstellungen der verwendeten Funktionen festgelegt war, konnte beim CircuitPython-Code lediglich eine Übertragungsrate von etwa 12 Hz gemessen werden. Als Ursache wäre denkbar, dass gewisse Voreinstellungen in der verwendeten minimqtt-Bibliothek die Ausführungsgeschwindigkeit des Codes beschränken; in der Dokumentation des Moduls konnten aber keine solche Einstellungen gefunden werden. Durch seine Funktionsweise als interpretierende Sprache ist Python jedoch grundsätzlich langsamer als hardwarenahe Sprachen wie C++, weshalb in der Mikrocontroller-Programmierung nach wie vor C++ als verwendete Sprache dominiert. Aus genannten Gründen wurde der Mikrocontroller daher mit Arduino programmiert.

B. Datenübertragung mit Bluetooth Low Energy

Das Bluetooth-Übertragungsverfahren ist in zwei Teil-Verfahren zu unterscheiden, die zwar grundlegende technische Eigenschaften miteinander teilen, sich aber in der Organisation der Datenübertragung und dem Verbindungsaufbau unterscheiden. Diese Teil-Verfahren werden als Bluetooth und Bluetooth Low Energy bezeichnet, wobei zur besseren Unterscheidung das ursprüngliche Bluetooth auch als Bluetooth Classic bezeichnet wird. Zur Darlegung der Unterschiede wird zunächst ein Überblick über die Funktionsweise und Struktur von Bluetooth Classic gegeben, ehe das für die Programmierung relevante Bluetooth Low Energy beschrieben wird.

A. Bluetooth (Classic)

Bluetooth ist ein Industriestandard für die drahtlose Datenübertragung zwischen Geräten über kurze Reichweiten, der in den 1990er Jahren durch die Bluetooth Special Interest Group (Bluetooth SIG) – eine Interessengemeinschaft gegründet von den Firmen Ericson, Nokia, IBM, Toshiba und Intel – entwickelt wurde. Heute (Stand 2021) gehören der Bluetooth SIG mehr als 35000 Unternehmen an. Hersteller von Bluetooth-fähigen Geräten müssen die von der Bluetooth SIG gesetzten Standards einhalten, um ihre Geräte als Bluetooth-fähig vermarkten zu können.

Bluetooth wird typischerweise für Punkt-zu-Punkt-Datenübertragungen über eine Reichweite von weniger als 10 Metern verwendet. Häufige Anwendungen im Alltag sind zum Beispiel die kabellose Verbindung zwischen einem Smartphone und Kopfhörern oder die drahtlose Anbindung von Tastaturen und Mäusen an Rechnern. Genauso wie die WLAN-Übertragung operiert auch die Bluetooth-Übertragung im 2,4 GHz-ISM-Band. Um gegenseitige Störungen durch die Überlagerung mit fremden Signalen im gleichen Frequenzbereich zu vermeiden, verwendet Bluetooth das Frequenzsprungverfahren (*Frequency Hopping Spread Spectrum, FHSS*). Dabei wird das 2,4 GHz-Band in 79 Kanäle mit einer Bandbreite von 1 MHz unterteilt, beginnend mit der niedrigsten Frequenz bei 2,402 GHz bis zu der höchsten Frequenz bei 2,480 GHz. Jedes Datenpaket wird nach einem pseudo-zufälligem Muster über einen der 79 Kanäle gesendet, wobei je nach Länge der Datenpakete der verwendete Kanal bis zu 1600 Mal pro Sekunde gewechselt werden kann. Wird die Übertragung eines Datenpaketes durch Interferenz mit einem anderen Signal gestört, wird das Signal auf einem anderen Kanal erneut gesendet. Auf diesem Wege wird Frequenzbereichen, die zum Beispiel schon durch WLAN-Netze genutzt werden, ausgewichen [A22].

Bluetooth nutzt eine Master-Slave-Architektur. Dabei nimmt ein Bluetooth-Gerät, welches die Verbindung zu einem anderen Gerät initiiert, automatisch die Funktion eines Masters an, während das die Verbindung annehmend Gerät zum Slave wird. Der Verbindungsprozess wird in mehreren Schritten durchgeführt. Dabei sendet das Master-Gerät zunächst eine Inquiry-Nachricht an alle Bluetooth-Geräte aus, die sich in Reichweite befinden. Diese

auch als Zugangsknoten bezeichneten Geräte antworten mit ihrer MAC-Adresse, über welche Bluetooth-fähige Geräte sich gegenseitig identifizieren können. Das Master-Gerät sucht sich eine Adresse aus, und synchronisiert sich mit dem Zugangsknoten mittels einer Technik, die Paging genannt wird. Auf dieser Grundlage wird eine Verbindung zum Slave-Gerät aufgebaut, mit welcher das Master-Gerät die angebotenen Dienste des Slave-Geräts detektiert. Schließlich wird der endgültige Kommunikationskanal zum Slave-Gerät aufgebaut. Von Seiten des Slave-Geräts kann ein zusätzlicher Sicherheitsmechanismus – das sogenannte Pairing – durchgeführt werden, welches den Zugriff auf autorisierte Master-Geräte beschränkt [A23].

Ein Master-Gerät kann abwechselnd mit bis zu sieben Slave-Geräten in einem sogenannten Piconetz kommunizieren. Der Master stellt als zeitliches Gerüst für die Kommunikation einen Takt bereit, welcher die Zeit in Zeitschlitze – sogenannte Slots - von 625 µs Länge einteilt. Ein Datenpaket kann dabei eine Länge von einem, drei, oder fünf Slots beanspruchen. Werden Pakete von einem Slot Länge versendet, ergibt sich - da nach jedem Paket ein anderer Frequenzkanal verwendet wird - eine Kanalwechselfrequenz von 1/0,625 µs = 1600 Hz. Um eine bidirektionale Kommunikation zu ermöglichen, geht nach jeder übertragenen Nachricht des Masters an den Slave das Senderecht an das Slave-Gerät über, worauf dieses über maximal fünf Slots Daten an das Master-Gerät senden kann. Sind keine zu übertragenden Nutzdaten für das Master-Gerät vorhanden, wird ein leeres Paket als Bestätigung für die empfangene Nachricht gesendet. Danach geht das Senderecht wieder auf das Master-Gerät über, welches nun das nächste Paket an dieses oder auch ein anderes verbundenes Slave-Gerät senden kann. Ab Bluetooth-Version 1.2 antwortet das Slave-Gerät dabei auf demselben Frequenz-Kanal, auf welchem es das Paket des Master-Geräts empfangen hat.

Welcher Frequenzkanal für die Übertragung eines Datenpakets gewählt wird, wird mithilfe eines Algorithmus aus der Taktnummer und der MAC-Adresse des Master-Geräts ermittelt. Durch die Verbindung mit einem Master-Gerät ist sowohl dessen MAC-Adresse als auch die Zahl der vergangenen Takte dem Slave-Gerät bekannt; dadurch kann das Frequenzsprungverfahren auf beiden Geräten synchron durchgeführt werden [A24].

B. Bluetooth Low Energy

Bluetooth Low Energy (BLE), manchmal auch als Bluetooth Smart bezeichnet, ist eine leichtgewichtige Abwandlung des klassischen Bluetooth und wurde als Teil der Bluetoothversion 4.0 eingeführt. Anders als es der Name suggeriert, hat BLE einen völlig anderen Ursprung und wurde von Nokia als internes Projekt namens Wibree gestartet, bevor es von der Bluetooth SIG übernommen wurde. Dem Namen gemäß wurde BLE für eine besonders stromsparende Datenübertragung entwickelt, wie sie bei batteriebetriebenen Geräten erforderlich ist [A25]. BLE operiert wie das klassische Bluetooth im 2,4 GHz-ISM-Band und benutzt zur Umgehung von belegten Frequenzen das Frequenzsprungverfahren. Dennoch bestehen zwischen den beiden Funkverfahren erhebliche Unterschiede, sodass BLE und klassisches Bluetooth nicht zueinander kompatibel sind, sondern auf Dual-Mode-Chipsätzen oft getrennte integrierte Schaltkreise zur Umsetzung implementiert werden, die softwareseitig getrennt angesprochen werden und lediglich eine gemeinsame Antenne teilen.

Ein bedeutender Unterschied besteht in der Implementierung des Frequenzsprungverfahrens: Während klassisches Bluetooth das 2,4 GHz-Band in 79 Kanäle unterteilt, wird bei BLE die Unterteilung auf 40 Kanäle reduziert, die dafür jedoch den doppelten Abstand von 2 MHz zueinander aufweisen. Diese Wahl wurde getroffen, um das technische Design kleiner BLE-fähiger Geräte zu vereinfachen [A22]. Ein weiterer Unterschied zu klassischem Bluetooth besteht im Prozess des Verbindungsaufbaus. So wird bei BLE wird das Advertising als Verbindungsverfahren eingeführt, das im folgenden Abschnitt näher erläutert wird.

B.1. Verbindungsaufbau

Der Advertising-Prozess wird durch das Generic Access Profile (GAP) kontrolliert. GAP definiert vier funktionale Rollen, die von Geräten eingenommen werden können, wovon zwei Rollen für den Aufbau einer festen Datenverbindung Verwendung finden: die Rolle der Zentrale (*Central*) und der Peripherie (*Peripheral*). Das Peripheriegerät hat die Aufgabe, regelmäßig sogenannte Advertisements abzusenden, um seine Existenz gegenüber anderen Geräten sichtbar zu machen. Das Zentralgerät scannt die Umgebung nach diesen Advertisements ab und initiiert daraufhin die Verbindung. Auf technischer Ebene werden dabei drei der vierzig Frequenzkanäle dazu genutzt, Advertisements zu senden und zu empfangen, wobei diese in möglichst großem Frequenzabstand zu einander angeordnet sind. So befindet sich jeweils ein Kanal am oberen und unteren Ende des verwendeten Frequenzspektrums und ein Kanal in der Mitte. Diese Verteilung minimiert die Wahrscheinlichkeit, dass alle drei Kanäle durch Störfrequenzen blockiert sind. Das Peripheriegerät sendet auf mindestens einem Kanal in regelmäßigen Zeitintervallen Advertisements ab, während das Zentralgerät die drei Kanäle abwechselnd nach Advertisements abhört. Die Prozedur ist erfolgreich, sobald ein gesendetes Paket in eine Empfangsphase fällt [A26]. Das Warten des Zentralgeräts nach Advertisements wird auch als passives Scanning bezeichnet. Um mehr Daten zu übermitteln, hat das Zentralgerät die Option, nach Eingang eines Advertisements zusätzlich eine sogenannte Scan Response Payload vom Peripheriegerät anzufordern, was als aktives Scanning bezeichnet wird. Ein Zentralgerät kann eine stetige Verbindung einrichten, indem es auf ein empfangenes Advertising-Paket mit einer Verbindungsaufforderung antwortet, die verschiedene Daten zur Organisation der Verbindung beinhaltet. Das Peripheriegerät kann die Verbindungssaufforderung annehmen; beide Geräte wechseln so in einen verbundenen Zustand. Während beim Advertising lediglich Pakete von bis zu 31 Bytes Länge gesendet werden können, können im Verbindungszustand wesentlich größere Datenmengen in beide Richtungen übertragen werden. Sobald ein Verbindungszustand eingerichtet ist, findet eine Kommunikation nur noch zwischen Zentralgerät und Peripheriegerät statt; außenstehende Geräte können sich nicht mit Zentrale oder Peripherie verbinden oder deren gesendete Daten empfangen.

Ein Gerät hat die Möglichkeit, Daten an mehrere Empfänger gleichzeitig zu senden, indem es die Broadcaster-Rolle einnimmt. Statt das Advertising zur Implementierung einer gesicherten Verbindung zu verwenden, werden Advertising-Pakete selbst zur Datenübertragung verwendet. Die Empfänger nehmen die sogenannte Observer-Rolle ein und lesen gesendete Nachrichten aus. Ein Datentransport findet dabei ausschließlich vom Broadcaster zum Observer statt; ein Observer gibt dem Broadcaster keine Rückmeldung über den erfolgreichen Empfang der Daten, sodass keine verlässliche Übertragung gewährleistet werden kann. Um die Verlässlichkeit zu erhöhen, können Daten doppelt, aber auf verschiedenen Frequenzbändern übertragen werden. Eine Datenübertragung über das Broadcaster-Observer-Prinzip sollte im Allgemeinen nur erfolgen, wenn Nachrichten an viele Geräte gesendet werden sollen, nur kleine Datenmengen benötigt werden und der Verlust einzelner Daten-Pakete in Kauf genommen werden kann [A27].

B.2. Datenübertragung

Wurde durch Advertising eine Verbindung zwischen zwei Geräten hergestellt, wird der Advertising-Prozess gestoppt und es können keine weiteren Advertising-Pakete gesendet werden. Andere Geräte können das Peripheriegerät nicht mehr detektieren und sich mit ihm verbinden. Für die bidirektionale Kommunikation zwischen Zentralgerät und Peripheriegerät im verbundenen Zustand wird das Generic Attribute Profile (GATT) verwendet, welches auf dem Attribute Protocoll (ATT) aufbaut. Dabei wird die Datenübertragung in sogenannte Characteristics und Services geordnet. Characteristics dienen als Behälter,

in welchen Nutzdaten zwischen verbundenen BLE-Geräten transportiert werden können. Ein Sendevorgang wird initiiert, sobald das Sendegerät ein Characteristic mit Nutzdaten beschreibt. Das Empfangsgerät erhält die Daten, indem es das jeweilige Characteristic ausliest. Characteristics, die in einem logischen Zusammenhang stehen, werden in Services zusammengefasst. Die Beziehung zwischen Services und Characteristics lässt sich anhand eines Multifunktionssensors veranschaulichen, der Luftdruck, Luftfeuchtigkeit und Temperatur zugleich misst. Der Service definiert hier den Sensor allgemein, die Characteristics die von diesem gemessenen Werte für Luftdruck, Luftfeuchtigkeit und Temperatur.

Services und Characteristics identifizieren sich durch einen UUID-Code (UUID = *Universial Unique IDentifier*), der im Allgemeinen 128-Bit lang ist und sich aus 16 Bytes in Hexadezimaldarstellung zusammensetzt. UUID's werden auch in Protokollen und Anwendungen jenseits von Bluetooth verwendet; ihr Format und ihre Nutzung wird von der International Telecommunication Union (ITU) spezifiziert. Die Länge von 128 Bit sorgt für eine hohe Wahrscheinlichkeit, dass die UUID einzigartig ist, jedoch nimmt diese einen erheblichen Teil der übertragenen Daten in Anspruch und erhöht den Overhead. Daher wurden spezielle UUID-Formate mit der Länge von 16 Bit und 32 Bit eingeführt; diese können nur für UUID's genutzt werden, die von der Bluetooth SIG als Standard-UUID's für festgelegte Anwendungen gelistet werden.

Für die Organisation der Kommunikation zwischen zwei Geräten definiert GATT zwei Rollen: Der GATT-Client dient als Master, der alle Arten der Datenübertragung startet und Anfragen (*resquests*) an einen GATT-Server sendet. Der GATT-Server nimmt die Rolle des Slaves an, empfängt Anfragen des GATT-Clients und sendet Antworten zurück. Ein Client kann Verbindungen zu mehreren Servern halten, wohingegen ein Server selbst kann nur mit einem einzigen Client verbunden sein. Die GATT-Rollen sind hierbei unabhängig von den zuvor erwähnten GAP-Rollen: Geräte, die beim Verbindungsprozess als Zentrale oder Peripherie fungiert haben, können sowohl Client als auch Server sein, oder auch beide Funktionen zugleich erfüllen, je nachdem ob das Gerät gerade Nutzdaten empfangen oder selbst welche senden soll.

Innerhalb von GATT gibt es mehrere Moden der Datenübertragung. Beim sogenannten Polling bietet der Server dem Client ein Verbindungsintervall an, innerhalb welchem sich der Client für jedes Empfangen eines Datenpackets neu verbindet und das Senden der Daten anfordert. Vor allem bei der Kommunikation mit mehreren Servern ist dies zweckmäßig, da sich so der Client abwechselnd mit verschiedenen Servern verbinden und deren Daten geordnet empfangen kann. Das Polling gibt dem Client die Kontrolle über die Datenübertragung, jedoch verlangsamt es die Übertragungsgeschwindigkeit und erhöht sowohl Stromverbrauch als auch Overhead. Mit der Nutzung sogenannter server-initiierten Updates kann dem Server die weitgehende Kontrolle über die Datenübertragung gegeben werden; dabei handelt es sich um Datenpakete, die von Servern asynchron - also ohne Aufforderung des Clients - an diesen verschickt werden können. Dabei sind zwei Arten zu unterscheiden: Notifications können vom Server in beliebiger Anzahl und zu beliebiger Zeit versendet werden, der Client sendet jedoch keine Bestätigung für den Empfang eines Datenpakets zurück. Im Gegensatz dazu erfordern Indications für jedes empfangene Paket eine Empfangsbestätigung, die der Client an den Server zurückmeldet. Neue Indications – auch von anderen Characteristics – werden erst verschickt, wenn die Empfangsbestätigung des vorherigen Indication-Pakets beim Server angekommen ist. Trotz des selbstständigen Sendens von Notifications und Indications durch den Server verbleibt die Initiative für das Einrichten von server-initiierten Updates beim Client. Um Notifications und Indications zu ermöglichen, schickt der Client zu Beginn eine entsprechende Anforderung zum Server. Wurden in den Characteristics Flags gesetzt, die zum Senden von Notifications und Indications befähigen, antwortet der Server mit einem „Write"-Paket und führt sofort eine Sendung aus, sobald ein Characteristic mit einem neuen Wert beschrieben wird [A28].

C. Empfangsprogramm

Um das Empfangsprogramm auf dem Raspberry Pi auszuführen, muss der Bluetooth-Manager Blueman heruntergeladen werden. Wie bei der Datenübertragung über WLAN, muss auch bei der Bluetooth-Übertragung vor Ausführung des Programms die Bluetooth-Schnittstelle über den entsprechenden Button auf dem Desktop oben rechts angeschaltet sein. In diesem Empfangsprogramm ist keine Funktion zur Steuerung des Telemetriesenders implementiert, da aufgrund des geringeren Energieverbrauchs auf eine Kontrolle der Datenübertragung vom Empfangsprogramm aus verzichtet wird.

```
1  from bluepy import btle
2  import struct
3
4  # Adapt the MAC-Address for your own Sending Device!
5  MAC                         = "84:cc:a8:79:15:0a"
6  ACCEL_SERVICE_UUID          = "2ACA"
7  ACCEL_CHARACTERISTIC_UUID   = "2719"
8
9  P = btle.Peripheral(MAC)
10 service = P.getServiceByUUID(ACCEL_SERVICE_UUID)
11 ch = service.getCharacteristics(ACCEL_CHARACTERISTIC_UUID)[0]
12
13 class MyDelegate(btle.DefaultDelegate):
14     def __init__(self, handle):
15         btle.DefaultDelegate.__init__(self)
16
17     def handleNotification(self, cHandle, data):
18         data = struct.unpack('fff', data)
19         data = (round(data[0],2),
20                 round(data[1],2),
21                 round(data[2],2))
22         print(data)
23
24 P.setDelegate(MyDelegate(ch.valHandle+1));
25
26 # Enable notifications
27 P.writeCharacteristic(ch.valHandle+1, b"\x01\x00");
28
29 while True:
30     if p.waitForNotifications(1):
31         # handleNotification() was called
32         continue
```

Zeilen 1 - 2:
Das Empfangsprogramm benötigt zwei Module. Das Modul `bluepy` stellt eine API für Linux-Systeme (wie dem Betriebssystem des Raspberry Pi) bereit, mit welchem Programme für die Kommunikation über Bluetooth Low Energy geschrieben werden können [A29]. Zum Aufruf der nötigen Methoden und Objekte wird das Submodul `btle` importiert. Weiter wird das Modul `struct` importiert, welches für die Bearbeitung der empfangenen Daten verwendet wird.

Zeilen 5 - 7:
Für die Kommunikation mit dem Server werden zunächst die MAC-Adresse des Arduino-

Boards sowie sowie die UUID-Codes festgelegt, die Server und Chracteristic identifizieren. Die MAC-Adresse ist die Hardware-Adresse von Ethernet-, WLAN- und Bluetooth-Schnittstellen und dient als Identifikator, mit welchem verschiedene Geräte in einem Netz einander adressieren können. Sie wird herstellerseitig fest auf Ethernet-, WLAN- oder Bluetooth-Netzwerkkarten einprogrammiert, weshalb sie auch als physikalische Adresse bezeichnet wird. UUID's werden als Kette von Hexadezimalzahlen angegeben und als Strings den entsprechenden Variablen zugewiesen. Um den Anteil an Verwaltungsdaten möglichst gering zu halten, wurden für die Service-UUID mit dem Namen `ACCEL_SERVICE_UUID` und die Characteristic-UUID mit dem Namen `ACCEL_CHARACTERISTIC_UUID` 16-Bit-UUID's gewählt, die von der Bluetooth SIG für häufige Anwendungen definiert wurden. Die UUID „2ACA“ unterliegt dabei keiner näheren Spezifikation, während die UUID „2719“ speziell für Beschleunigungen definiert ist. Prinzipiell können für den privaten Gebrauch alle vordefinierten 16-Bit-UUID's verwendet werden. Dabei muss jedoch berücksichtigt werden, dass durch andere aktive Bluetooth-Geräte, die in Reichweite sind und identische vordefinierte UUID's zur Kommunikation verwenden, Übertragungsstörungen bei beiden Geräten auftreten können.

Zeilen 9 - 11:
Zum Aufbau einer Verbindung wird zunächst ein Objekt der `Peripheral`-Klasse erstellt, welches die Grundlage für den Aufruf aller weiteren Methoden darstellt und die MAC-Adresse des Peripheriegerätes als Eingabewert verwendet. Mit der Methode `getServiceByUUID` und der zuvor definierten Service-UUID als Parameter wird das Objekt `service` erzeugt, mit welchem wiederum die Methode `getCharacteristic` aufgerufen wird, um das zu verwendende Characteristic auszuwählen und das Objekt `ch` zu erzeugen.

Zeilen 13 - 22:
Für das kontinuierliche und effiziente Übertragen von Sensordaten ist Polling unzweckmäßig. Stattdessen wird hier das Senden der Daten in Form von Notifications realisiert, was die effizienteste Möglichkeit zur Datenübertragung darstellt, dies allerdings auf Kosten der Übertragungssicherheit erreicht. Zwar bietet Bluepy auch die Möglichkeit zum Empfang von Indications, die eine sichere Übertragung durch eine Empfangsbestätigung sicherstellen, jedoch führt dies unweigerlich zu einer deutlichen Verlangsamung der Datenübertragung. Da der Verlust einzelner Datenpakete bei ausreichend hoher Frequenz in Kauf genommen werden kann, bleiben Notifications die beste Wahl. Zur Nutzung von Notifications muss zunächst eine Klasse erzeugt werden, innerhalb derer die Methode `handleNotification` definiert wird. `handleNotification` wird immer dann aufgerufen, sobald ein Notification-Paket empfangen wurde. Über die Variable `data` können die empfangenen Nutzdaten abgerufen und innerhalb der Funktion weiterverarbeitet werden. In diesem Fall wurden die Sensordaten als Tuple aus drei float-Werten verschickt, die zunächst jeweils als Folge aus vier Bytes in Hexadezimaldarstellung vorliegen. Mithilfe der `unpack`-Methode können die empfangenen Bytes in ein Tupel umgewandelt werden. Die float-Werte innerhalb des Tupels werden anschließend auf zwei Nachkommastellen gerundet und über den `print`-Befehl ausgegeben.

Zeile 24:
Schließlich wird ein Objekt der selbstdefinierten `MyDelegate`-Klasse erzeugt und dieses mit der Methode `setDelegate` referenziert, wodurch die darin enthaltenen Methoden in den Programmablauf eingebunden werden können.

Zeile 27:
Mit der Methode `writeCharacteristic` wird der Befehl zum Senden von Notifications in Form von zwei Bytes in Hexadezimaldastellung an das Peripheriegerät gesandt.

Zeilen 29 - 32:
Zum kontinuierlichen Empfang von Notifications wird eine `while`-Dauerschleife gebildet, in welcher innerhalb einer `if`-Struktur die Methode `waitForNotifications` aufgerufen

wird. Diese gibt den Wert `False` zurück, bis entweder ein Notification-Paket empfangen wird oder der als Parameter gesetzte Zeitwert in Sekunden überschritten wird. Bei Empfang eines Notification-Pakets wird die Methode `handleNotification` innerhalb der Schleife aufgerufen und deren Inhalt ausgeführt.

D. Sendeprogramm

Während bei der Datenübertragung mit dem MQTT-Protokoll die Programmierung des WLAN-Moduls in den Sprachen C++ und Circuitpython realisiert wurde, kann das Bluetooth-Modul nur in C++ programmiert werden. Der Grund hierfür ist, dass die auf dem Bluetooth-Modul herstellerseitig einprogrammierte Firmware von dem Circuitphyton-Modul für BLE nicht unterstützt wird. Den Erfahrungen aus der Programmierung für MQTT folgend, wäre aber auch hier zu erwarten, dass die Ausführungsgeschwindigkeit des Circuitpython-Codes hinter dem C++-Code zurück bleibt. Um eine Verbindung zum Empfangsprogramm auf dem Raspberry Pi aufzubauen, muss das Sendeprogramm (und damit der Telemetriesender) ein paar Sekunden vor dem Empfangsprogramm gestartet werden. Eine fehlende oder abgebrochene Verbindung aufgrund von zu großer Entfernung oder sonstigen Fehlfunktionen wird durch eine blinkende LED auf dem Board sichtbar gemacht.

```
1  #include <ArduinoBLE.h>
2  #include <Arduino_LSM6DSOX.h>
3
4  // set time between two samples in milliseconds
5  #define UPDATE_INTERVALL 15
6
7  #define ACCEL_SERVICE_UUID          "2ACA"
8  #define ACCEL_CHARACTERISTIC_UUID   "2713"
9
10 #define NUMBER_OF_SENSORS 3
11 union multi_sensor_data
12 {
13   struct __attribute__((packed))
14   {
15     float values[NUMBER_OF_SENSORS];
16   };
17   uint8_t bytes[NUMBER_OF_SENSORS * sizeof(float)];
18 };
19 multi_sensor_data IMU_Data;
20
21 BLEService sensorDataService(ACCEL_SERVICE_UUID);
22 BLECharacteristic sensorDataCharacteristic(
     ACCEL_CHARACTERISTIC_UUID, BLENotify, sizeof IMU_Data.bytes);
23
24 void setup() {
25   Serial.begin(115200);
26   delay(700);
27
28   if (!BLE.begin()) {
29     Serial.println("starting BLE failed!");
30     while (1);
31   }
```

```
32
33    if (!IMU.begin()){
34      Serial.println("Failed to initialize IMU!");
35      while (1);
36    }
37
38    pinMode(LED_BUILTIN, OUTPUT);
39
40    BLE.setAdvertisedService(sensorDataService);
41    sensorDataService.addCharacteristic(sensorDataCharacteristic);
42    BLE.addService(sensorDataService);
43
44    BLE.setConnectable(true);
45
46    if(BLE.advertise()){
47      Serial.println("Bluetooth Device Active, Waiting for
          Connections...");
48    }
49  }
50
51  void loop() {
52    static float acc_x, acc_y, acc_z;
53    static long previousMillis = 0;
54
55    BLEDevice central = BLE.central();
56
57    if(central) {
58      Serial.print("Connected to Central");
59      while(central.connected()) {
60        unsigned long currentMillis = millis();
61        if (currentMillis - previousMillis >= UPDATE_INTERVALL) {
62          previousMillis = currentMillis;
63
64          IMU.readAcceleration(acc_x, acc_y, acc_z);
65          IMU_Data.values[0] = acc_x;
66          IMU_Data.values[1] = acc_y;
67          IMU_Data.values[2] = acc_z;
68          sensorDataCharacteristic.writeValue(IMU_Data.bytes,
              sizeof  IMU_Data.bytes);
69        }
70      }
71    }
72    Serial.println("Disconnected from Central");
73    Serial.print("MAC-Address of Device is: ");
74    Serial.println(BLE.address());
75    Serial.println();
76
77    digitalWrite(LED_BUILTIN, HIGH);
78    delay(100);
79    digitalWrite(LED_BUILTIN, LOW);
80    delay(100);
81    digitalWrite(LED_BUILTIN, HIGH);
```

```
82    delay(100);
83    digitalWrite(LED_BUILTIN, LOW);
84    delay(300);
85  }
```

Zeilen 1 - 2:
Für die Programmierung von BLE-Anwendungen und das Auslesen des Beschleunigungssensors werden zuerst die Bibliotheken `ArduinoBLE` und `Arduino_LSM6DSOX` eingebunden.

Zeile 5:
Zum Einstellen der Abtastrate kann am Kopf des Quellcodes die Zeit zwischen dem Versenden zweier Datenpakete in Millisekunden eingestellt werden. Die maximale Abtastrate wurde bei etwa 80 Hz gemessen, dies entspricht einem minimalen Update-Intervall von 13 Millisekunden.

Zeilen 7 - 8:
Analog zum Empfangsprogramm werden hier die gleichen UUID's für Service und Characteristic definiert und durch Namen substituiert, die anstelle der UUID's an den entsprechenden Code-Stellen verwendet werden.

Zeilen 10 - 19:
Damit Daten übertragen werden können, müssen diese in Form eines Byte-Arrays vorliegen, was – verglichen mit dem zuvor behandelten MQTT-Sendeprogramm – eine komplexere Aufbereitung der Daten notwendig macht. Hierfür wird mithilfe von `union` ein spezieller Datentyp mit dem Namen `multi_sensor_data` erzeugt. `union` zeichnet sich dadurch aus, dass Variablen verschiedenen Datentyps in einen neuen Datentyp zusammengefasst werden können und sich dabei denselben Speicherplatz teilen. Auf diese Weise können komplexere Datenkonvertierungen vorgenommen werden. Innerhalb von `multi_sensor_data` werden zwei Variablen definiert: ein float-Array namens `values` und ein byte-Array namens `bytes`. Das float-Array wird dabei mit dem `__attribute__((packed))` Befehl modifiziert. Dies trägt der Tatsache Rechnung, dass je nach Größe des Datenpakets und Speicherorganisation der benötigte Speicherplatz des Arrays um zusätzliche Bytes ergänzt wird, um ein effizientes Auslesen des Speichers zu ermöglichen. Der Befehl unterbindet dieses sogenannte „data structur padding" und sorgt so dafür, dass das `values`-Array und das `bytes`-Array auf eine gleiche Byte-Länge gebracht werden können. Die Größe des `values`-Arrays wird über den Bezeichner `NUMBER_OF_SENSORS` auf drei float-Variablen festgelegt. Die Größe des `bytes`-Arrays wird auf die Größe des `values`-Arrays festgelegt, indem mithilfe des `sizeof`-Operators die benötigte Byte-Anzahl für eine float-Variable mit drei multipliziert wird. Anschließend wird mit `IMU_Data` eine Variable des eben definierten Datentyps `multi_sensor_data` erzeugt.

Zeilen 21 - 22:
Den anfangs definierten UUID's für die Identifikation als Service oder Characteristic werden die Bezeichnungen `sensorDataService` und `sensorDataCharacteristic` zugewiesen. Die Definition des Characteristics beinhaltet zudem die Zuweisung von Eigenschaften, die festlegen, welche Art der Datenübertragung unter dem Characteristic stattfindet. Für das Senden von Notifications wird die Bitmaske `BLENotify` gesetzt (und für das Senden von Indications entsprechend `BLEIndicate`). Weiter wird mit dem `sizeof`-Operator und dem Aufruf des zuvor definierten `bytes`-Arrays die maximale Länge des zu versendenden Datenpakets in Bytes angegeben.

Zeilen 24 - 38:
In der `setup`-Funktion werden alle nötigen Schritte zur Initialisierung und zum Starten des Advertisings ausgeführt. So werden der Serielle Port, das Bluetooth-Modul sowie die

IMU initialisiert. Im Falle einer fehlgeschlagenen Initialisierung des Bluetooth-Moduls oder der IMU wird eine Fehlermeldung ausgegeben. Das Pin, über welches die eingebaute LED angesteuert wird, wird als Ausgang konfiguriert, um später durch Blinken der LED Rückmeldung über die Verbindung zum Raspberry Pi zu geben.

Zeilen 40 - 49:
Zum Starten des Advertisings müssen mehrere Schritte in der richtigen Reihenfolge durchlaufen werden. So wird mit `setAdvertisedService` der Service gesetzt und diesem per `addCharacteristic` das zuvor definierte Characteristic zugewiesen. `addService` fügt den eben gesetzten Service zu den internen Services hinzu, die BLE für den Übertragungsprozess benötigt. Mit `setConnectable` wird das Board im Anschluss an den Advertising-Prozess zum Aufbau und Halten einer Verbindung zum Raspberry Pi befähigt. Mit `advertise()` wird das Advertising gestartet, was bei Anschluss an einen Rechner durch eine Ausgabe am Seriellen Monitor bestätigt wird.

Zeilen 51 - 71:
Alle Schritte zum Senden der Beschleunigungsdaten spielen sich in der `loop`-Funktion ab. Mit der Methode `central()` wird die Verbindung zum Zentralgerät überprüft, indem einer als `central` benannten Variablen der Wert `true` oder `false` zugewiesen wird. Diese dient zur Steuerung des darauf folgenden `if`-Blocks. Liegt so eine Verbindung vor, wird für die Sichtbarmachung des Programmablaufs die Meldung „Connected to Central“ ausgegeben. Solange die Verbindung fortbesteht, wird innerhalb einer `while`-Schleife das zuvor festgelegte Zeitintervall abgezählt, dann die Beschleunigung für die x-, y- und z-Achse ausgelesen und das `values`-Array mit diesen Werten beschrieben. Das Senden der Daten wird ausgeführt, indem das Characteristic mit der Methode `writeValue` beschrieben wird. Der durch das `values`-Array mit drei float-Zahlen belegte Speicher wird nun als `bytes`-Array ausgelesen und als Parameter in die Klammer eingegeben. Als zweiter Parameter wird die Größe des Arrays benötigt.

Zeilen 72 - 85:
Bricht die Verbindung zum Zentralgerät ab, endet die `while`-Schleife. Solange keine erfolgreiche (Neu-)Verbindung erfolgt, werden die Meldung „Disconnected from Central“ sowie die MAC-Adresse des Boards periodisch zum Seriellen Monitor ausgegeben. Zur optischen Sichtbarmachung der fehlenden Verbindung führt die orangene LED ein Blinkmuster aus. Bei der erstmaligen Implementierung der Bluetooth-Übertragung wird die hier ausgegebene MAC-Adresse im Quellcode des Empfangsprogramms eingegeben, damit eine Verbindung zwischen Raspberry Pi und dem Arduino-Board hergestellt werden kann.

C. Programm zur Auswertung der Zentrifugalbeschleunigung

Im Kapitel „Didaktische Anwendungen“ wurde der Telemetriesender genutzt, um den Zusammenhang $a = \omega^2 r$ zu bestätigen. Dabei wurde lediglich die Zentrifugalbeschleunigung durch den Telemetriesender gemessen und übertragen, während die Winkelgeschwindigkeit getrennt durch eine separate Apparatur gemessen wurde. Die IMU des verwendeten Arduino-Boards verfügt neben einem Accelerometer auch über ein Gyrometer, welches Drehbewegungen misst und sich somit dazu eignet, die Winkelgeschwindigkeit zu detektieren. Dadurch können sowohl Beschleunigung als auch Winkelgeschwindigkeit durch das Arduino-Board gemessen und an ein PhyPiDAQ-Auswertungsprogramm übertragen werden.

Der folgende Code-Abschnitt stellt die `loop`-Funktion eines Arduino-Programmes zur Messung und Übertragung von Beschleunigung und Winkelgeschwindigkeit in einem rotierenden System dar. Die Übertragung erfolgt in diesem Beispiel durch das MQTT-Protokoll. Auf eine An-und-Ausschaltsteuerung sowie eine Steuerung der Abtastrate wird hier verzichtet. Die Beschleunigungen in Richtung der kartesischen Koordinaten x, y, und z werden in den Variablen `ax`, `ay`, und `az` gespeichert. Drehbewegungen um die kartesischen Koordinaten werden in Grad pro Sekunde ausgelesen und in den Variablen `gx`, `gy`, und `gz` gespeichert. Für eine Darstellung als Winkelgeschwindigkeit müssen die Werte umgerechnet und den Variablen neu zugewiesen werden. Die in der `publish`-Methode eingestellten Dreh- und Beschleunigungsachsen sind in Abbildung .1 veranschaulicht. Bei einer anderen Anordnung des Arduino-Boards im rotierenden Aufbau müssen die Dreh- und Beschleunigungsachse entsprechend geändert werden.

```
1  void loop() {
2    if (!client.connected()) {
3      reconnect();
4    }
5
6    static float ax, ay, az;
7    static float gx, gz, gz;
8
9    if (IMU.accelerationAvailable()) {
10     IMU.readAcceleration(ax, ay, az);
11   }
12   if (IMU.gyroscopeAvailable()) {
13     IMU.readGyroscope(gx, gy, gz);
14
15     gx = (2*3.141*gx)/360;
16     gy = (2*3.141*gy)/360;
17     gz = (2*3.141*gz)/360;
```

```
18    }
19
20    client.publish(Publish_Topic, (String(ax) + " " +
21                                   String(gz)).c_str());
22  }
```

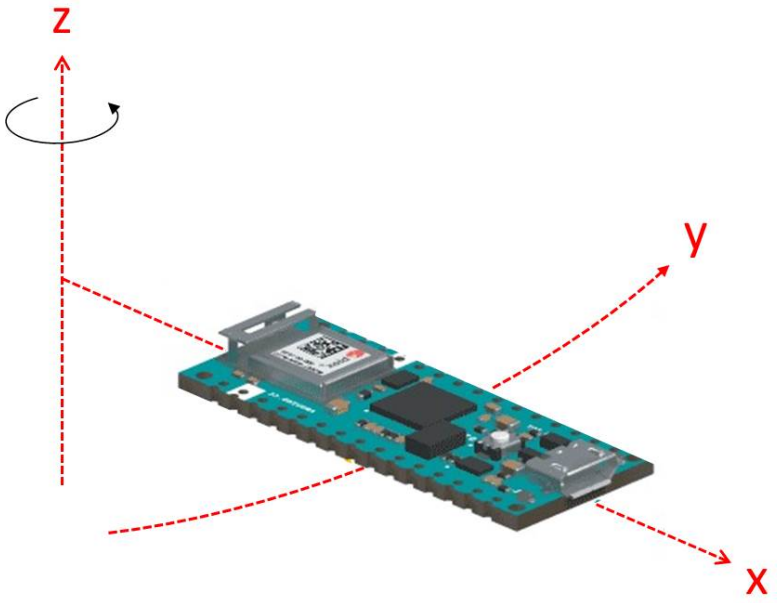

Abbildung .1.: Dreh- und Beschleunigungsachse bei einem Experiment zur Zentrifugalbeschleunigung. Die Darstellung des Arduino-Boards wurde [B36] entnommen.

D. Technische Charakterisierung des Arduino-Nano-RP2040-Connect

A. Pinbelegung

Wie alle Arduino-Nano-Boards verfügt auch das RP2040-Connect über insgesamt 30 Pins, die unterschiedliche Funktionen bereitstellen und so eine Vielzahl unterschiedlicher Anwendungen unterstützen. Abbildung A.1 bietet eine Übersicht über die Anordnung der Pins und ihre Funktionen.

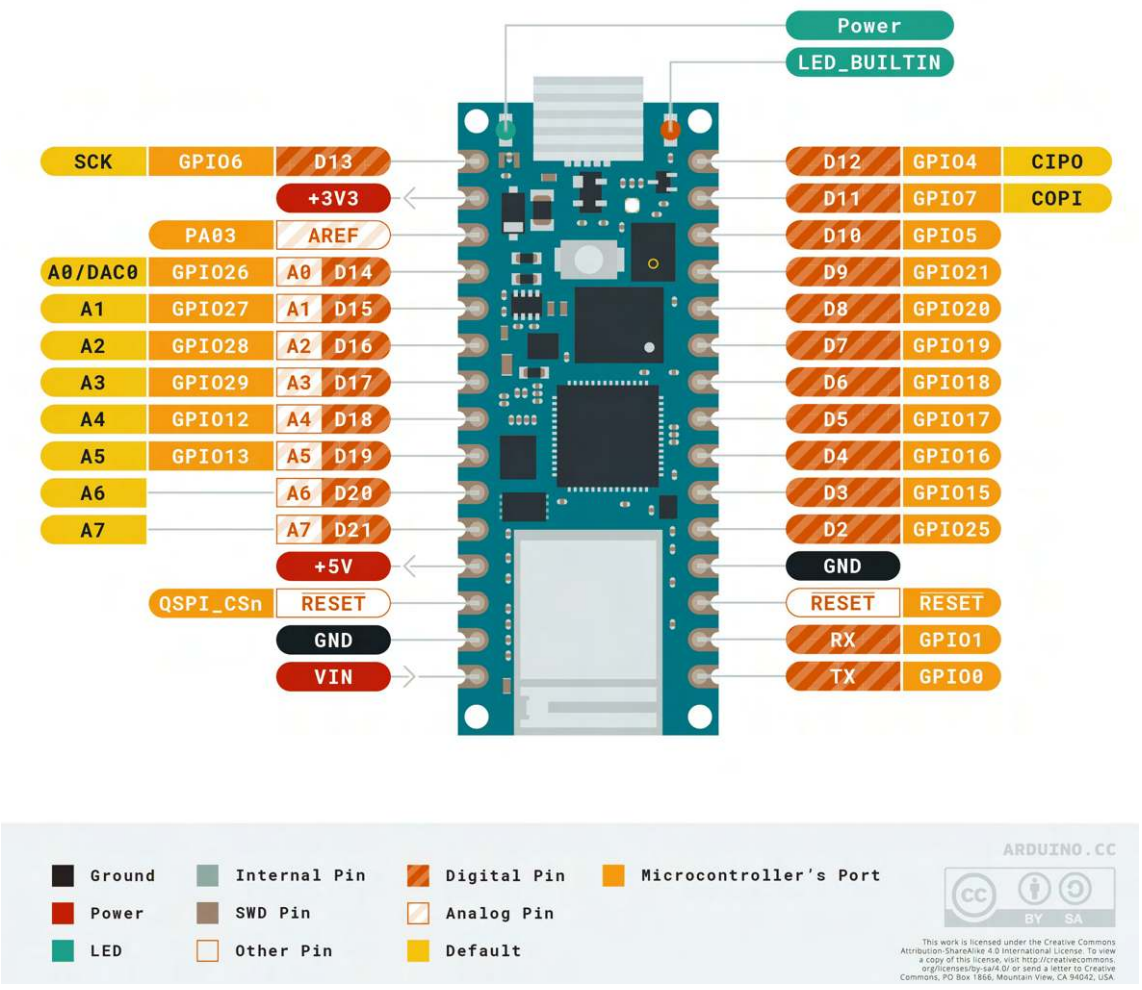

Abbildung A.1.: Pinbelegung des Boards Arduino-Nano-RP2040-Connect [B37]

Digitale Pins

Digitale Pins arbeiten auf zwei Spannungsebenen: 0 V (*Low*) und 3,3 V (*High*). Diese können in der Programmierung als digitaler Ausgang oder Eingang konfiguriert werden, sodass über die digitalen Pins Low- oder High-Signale ausgegeben oder registriert werden können. Als Anwendung können so zum Beispiel Dioden angesteuert werden, oder aber es können als Reaktion auf ein eingehendes High-Signal diverse Prozess-Schritte auf dem Board veranlasst werden.

Analoge Pins

Analoge Pins arbeiten ausschließlich als Eingänge. Dabei können sie im Gegensatz zu digitalen Pins, die nur die Zustände High und Low kennen, in einem Bereich von 0 V bis 3,3 V Spannungen unterschiedlicher Höhe erkennen. Dies geschieht, indem die an einem Analog-Pin angelegte Spannung zu einem auf dem Board integrierten Analog-Digital-Wandler weitergeleitet wird. Dieser liest die physische Spannung aus und weist dieser einen Zahlenwert zu, der wiederum in einen konkreten Wert in der Einheit Volt umgerechnet werden kann. Dies gibt die Möglichkeit, kontinuierliche Spannungspegel auszumessen und in Abhängigkeit eines konkreten Spannungswerts Prozess-Schritte zu veranlassen. Auf dem Board können die digitalen Pins D14 bis D21 auch als Analog-Pins (A0 bis A7) konfiguriert werden.

Anschlüsse zur Spannungsversorgung

Neben dem Mikro-USB-Anschluss, über welchen das Board auch programmiert wird, kann das Board auch über das VIN-Pin (*Voltage IN*) mit Betriebsspannung versorgt werden. Dabei kann eine Spannung im Bereich von 4 V bis 20 V angelegt werden; diese wird über einen internen Tiefsetzsteller auf das 3,3 V-Logik-Level abgeregelt. Als Anschluss für die Masseleitung bietet sich das benachbarte GND-Pin an. Für die Spannungsversorgung von Peripheriegeräten wie Sensoren kann das 3,3 V-Pin verwendet werden. Über das VUSB-Pin (*Voltage USB*) kann die USB-Spannung von 5 V ausgegeben werden. Da aber das Board - sowie die meisten Peripheriegeräte - mit einer 3,3 V-Logik arbeitet, würde ein leichtsinniges Verbinden anderer Pins mit dem 5 V-Anschluss zur Beschädigung der Hardware führen, weshalb das VUSB-Pin physisch von der USB-Spannung abgetrennt ist. Um das Pin verwenden zu können, müssen die entsprechenden Kontaktflächen auf der Rückseite des Boards über eine Lötschicht miteinander verbunden werden.

Schnittstellen

Für den Datenaustausch mit anderen Geräten verfügt das Board über jeweils einen I^2C-, SPI- und UART-Anschluss. Beim SPI-Bus werden anstelle der üblichen Bezeichnungen MOSI und MISO für Ein- und Ausgangspins die Bezeichnungen COPI (*Controller Out, Peripheral In*) und CIPO (*Controller In, Peripheral Out*) verwendet. Dabei gelten folgende Pinzuweisungen:

I^2C:	SDA $\longrightarrow$ D18
	SCL $\longrightarrow$ D19
SPI:	CIPO $\longrightarrow$ D12
	COPI $\longrightarrow$ D11
	SCK $\longrightarrow$ D13
	CS $\longrightarrow$ Jedes GPIO-Pin außer D20/D21
UART:	Rx $\longrightarrow$ D0
	Tx $\longrightarrow$ D1

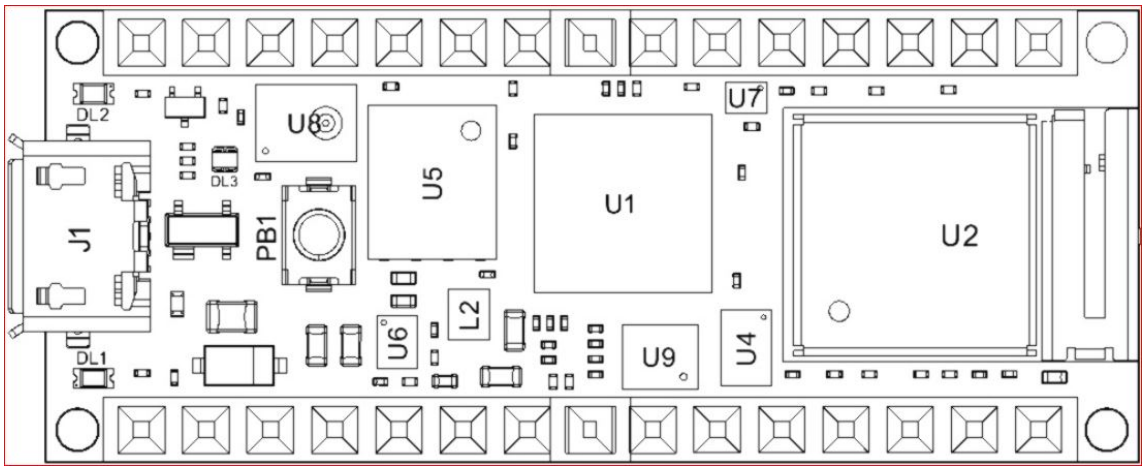

Abbildung B.2.: Board-Topologie des Arduino-Nano-RP2040-Connect [B38]

Pulsweitenmodulation

Als digitaler Ausgang konfiguriert, können digitale Pins durch hochfrequentes Wechseln von High- und Low-Pegeln und Variation der Pulsdauer des High-Pegels eine beliebige Ausgangsspannung zwischen 0 V und 3,3 V erzeugen. Als Anwendung lässt sich beispielsweise die Helligkeit einer Leuchtdiode steuern, oder aber das Drehmoment eines Gleichstrommotors. Die meisten Analog- und Digital-Pins können für die Pulsweitenmodulation genutzt werden; Ausnahmen sind die Pins D20 und D21, die nur als digitaler Input verwendet werden können, und die Pins D18 und D19, die die Als I^2C-Bus reserviert sind.

B. Board-Topologie

Abbildung B.2 stellt die Position der einzelnen technischen Elemente auf dem Board dar. Im Folgenden werden die Eigenschaften und Funktionen der wesentlichen technischen Komponenten erläutert.

U1: Als Prozessor wird der Raspberry-Pi-RP2040 verwendet. Dieser Beinhaltet als Rechenzentrum zwei Arm-Cortex-M0+-Kerne mit einem Takt von 133 MHz. Der Prozessor verfügt über einen 264 KB-SRAM-Speicher, der als Cache zum Zwischenspeichern von Rechenschritten genutzt wird. Ebenso ist ein 12 Bit-Digital-Analog-Wandler im Chip integriert. Für den Betrieb des Prozessors wird die Betriebsspannung von 3,3 V intern auf 1,1 V abgeregelt. Der RP2040 kontrolliert die anderen Peripherie-Module des Boards; auch werden alle digitalen Pins sowie die Analog-Pins A0 bis A3 durch den Prozessor direkt angesteuert.

U2: WLAN- und Bluetooth-Konnektivität wird durch das Nina-W102-Modul bereitgestellt, welches als Co-Prozessor zum RP2040 agiert. Die Analog-Pins A4 bis A7 werden durch das Nina-W102-Modul kontrolliert, ebenso ist die RGB-LED (Siehe DL3) mit dem Modul verbunden und wird über dieses angesteuert.

U4: Der ATECC608A-Cryptographic-IC ist ein kryptografischer Co-Prozessor, um in speziellen Smart-Home-Anwendungen für die Ver- und Entschlüsselung von Daten zu sorgen.

U5: Bei dem AT25SF128A-MHB-T handelt es sich um einen externen 16 MB-Flash-Speicher, der von dem RP2040-Prozessor über eine sogenannte QSPI-Schnittstelle angesprochen wird. Gegenüber dem vorher erwähnten SRAM-Speicher bleiben bei einem Flash-Speicher gespeicherte Daten auch ohne Spannungsversorgung erhalten.

Programme, die auf das Board geladen werden, werden im Flashspeicher gespeichert, und bei jedem Start des Boards vom Prozessor abgerufen.

U6: Bei dem MP2322GQH-Step-Down-Buck-Regulator handelt es sich um einen Tiefsetzsteller, der die am USB-Anschluss oder dem VIN-Pin anliegende Spannung auf die interne Betriebsspannung von 3,3 V abregelt, mit welcher bis auf den Prozessor alle auf dem Board verbauten Module betrieben werden.

U7: Für Übertragungsprotokolle wie I^2C wird ein eigener Takt mittels eines MEMS-Oszillators bereitgestellt. (*MEMS: Micro Electro-Mechanical System*)

U8: Das MP34DT06J-Mikrofon kann zur omnidirektionalen Aufnahme von Geräuschen verwendet werden und beruht auf einem kapazitiven Sensor-Element mit einem hohen (64 dB) Signal-zu-Rausch-Verhältnis.

U9: Die 6-Achsen-IMU (*IMU: Internal Measuring Unit*) beinhaltet ein 3-Achsen-Accelerometer zum Messen von Beschleunigungen in den kartesischen Koordinaten x, y und z sowie ein 3-Achsen-Gyrometer zum Messen von Drehbewegungen in drei kartesischen Raum-Achsen. Abbildung B.3 veranschaulicht die Orientierung der Achsen relativ zum Board.

DL1: Grüne Power-On-LED, leuchtet durchgehend während des Betriebs

DL2: Orangene Build-In-LED, für das Erzeugen von optischen Signalen programmierbar

DL3: RGB-LED: Modul aus drei Leuchtdioden in den Farben Rot, Grün und Blau, die einzeln programmiert werden können. Die RGB-LED ist intern mit dem Nina-W102-Modul verbunden und kann daher nur angesteuert werden, wenn das Modul im Programmcode aktiviert wird. Während der Nutzung von Bluetooth kann die RGB-LED nicht angesteuert werden.

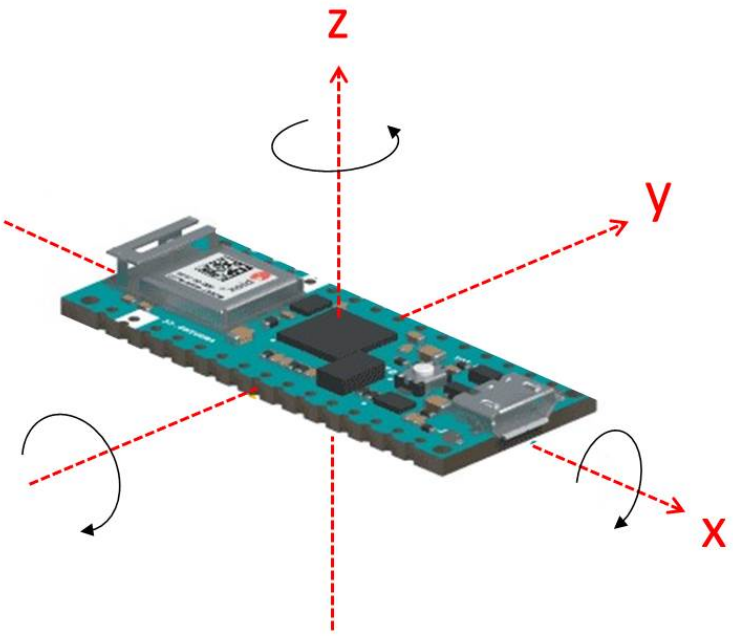

Abbildung B.3.: Orientierung der kartesischen Koordinaten relativ zum Board

E. Messtechnische Charakterisierung

A. Datenblatt

Zur Charakterisierung des Beschleunigungssensors innerhalb der IMU werden im zugehörigen Datenblatt verschiedene Werte angegeben. „Linear acceleration sensitivity“ beschreibt die Diskretisierung der Beschleunigungswerte in Abhängigkeit des eingestellten Messbereichs. Die Spanne der von dem Beschleunigungssensor ausgegebenen analogen Spannungswerte wird dabei durch einen internen 16 Bit-Analog-Digital-Wandler in $2^{16} = 65536$ diskrete Werte unterteilt, die hier in Beschleunigungswerte in der Einheit $g = 9{,}81\,\mathrm{m/s^2}$ umgerechnet werden. Für den in der verwendeten Arduino-Bibliothek eingestellten Messbereich von $\pm 4\,\mathrm{g}$ ergibt sich damit eine Diskretisierung von

$$\frac{4-(-4)}{2^{16}} = 1{,}22 \cdot 10^{-4}\,\frac{\mathrm{g}}{\mathrm{LSB}} = 0{,}122\,\frac{\mathrm{mg}}{\mathrm{LSB}} \tag{A.1}$$

Das Akronym LSB steht hierbei für *Last Significant Bit* und besagt, dass eine Änderung am niedrigstwertigen Bit der Binärzahl, in welcher die Spannungswerte des Accelerometers diskretisiert werden, einer Änderung des Beschleunigungswerts um $1{,}22 \cdot 10^{-4}\,\mathrm{g}$ entspricht. Dies stellt die theoretisch erreichbare Auflösung des Sensors dar. Real wird diese Auflösung nicht erreicht, da Beschleunigungen dieser Größenordnung von Messrauschen überlagert werden. Das Datenblatt gibt unter „RMS noise“ einen Wert zur Quantifizierung des Rauschens an; darunter wird der Quadratische Mittelwert des Rauschens verstanden. Dieser wird für den eingestellten Messbereich von $\pm 4\,\mathrm{g}$ mit $2{,}0\,\mathrm{mg}$ angegeben und ist damit deutlich größer die Diskretisierung der Messwerte in Vielfache von $0{,}122\,\mathrm{mg}$.

B. Messgenauigkeit

In der verwendeten Arduino-Bibliothek wird die in g angegebene Beschleunigung auf zwei Nachkommastellen gerundet, sodass der Rauschbereich abgeschnitten wird und alle Beschleunigungswerte als Vielfache von $0{,}01\,\mathrm{g}$ angegeben werden. Um von dieser Grundlage ausgehend die Messgenauigkeit zu bestimmen, wurde mit dem Telemetriesender eine Messkurve der Beschleunigung einer Federschwingung aufgenommen und diese mit dem erwartbaren theoretischen Verlauf verglichen. Da sich der Streckenverlauf $s(t)$ einer Federschwingung bekanntlich als Sinusschwingung darstellen lässt, ergibt sich durch zweimaliges Ableiten auch für den theoretischen Verlauf der Beschleunigung eine Sinusschwingung:

$$s(t) = \hat{s}\sin(\omega t) \tag{B.1}$$

$$v(t) = \dot{s}(t) = \hat{s}\omega\cos(\omega t) \tag{B.2}$$

$$a(t) = \ddot{s}(t) = -\hat{s}\omega^2\sin(\omega t) = -\hat{a}\sin(\omega t) \tag{B.3}$$

Um die Messkurve der Beschleunigungen korrekt zu modellieren, wird der Sinus-Funktion eine Offset-Beschleunigung a_0 für die Berücksichtigung der Erdbeschleunigung sowie eine Phase ϕ hinzugefügt. Die Dämpfung der Schwingung infolge der Luftreibung wird

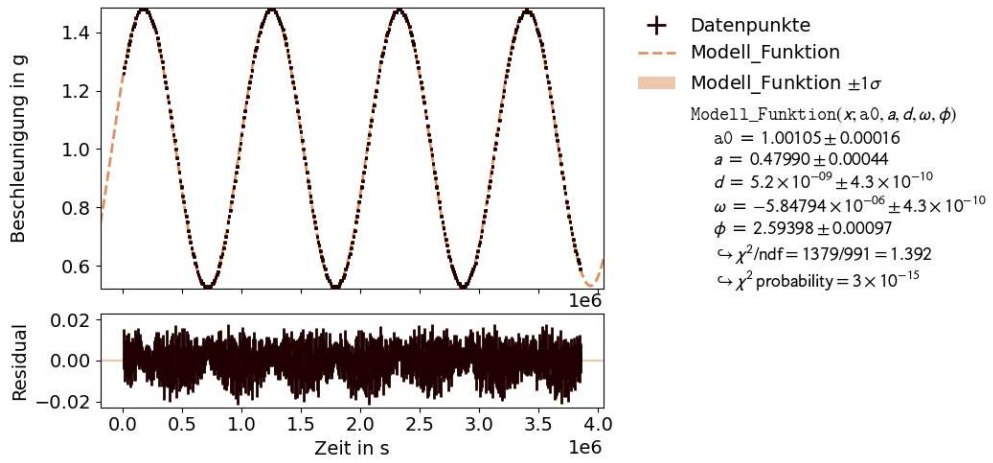

Abbildung B.1.: Modellierung der Messdaten mit Residuen

durch Multiplikation mit einer fallenden Exponentialfunktion berücksichtigt. Für die Modellierungsfunktion ergibt sich damit:

$$a(t) = a_0 + a\exp(-dt)\sin(\omega t + \phi) \tag{B.4}$$

Die Anpassung an die Messdaten wurde mit den KIT-eigenen Pythonmodulen PhyPraKit und kafe2 durchgeführt. Das Ergebnis ist in Abbildung B.1 veranschaulicht. Die Zeitachse wurde auf eine µs genau aufgezeichnet; entsprechend wurde der Zeit-Fehlerbalken mit ±0,5 µs angegeben. Der Auflösung der Beschleunigung auf 0,01 g entsprechend, wurde der Fehlerbalken für die Beschleunigung auf ±0,005 g festgelegt. Die Residuen quantifizieren die Abweichung der Messpunkte von der modellierten Funktion. Die Verteilung erscheint über der Zeitachse homogen, was darauf hinweist, dass die Messdaten mit einer passenden Funktion modelliert wurden. Es lässt sich insgesamt eine durchschnittliche Abweichung von etwa ±0,01 g erkennen. Weiter lässt sich beobachten, dass die Residuen bei einer schnellen zeitlichen Änderung der Beschleunigung größer ausfallen und bei minimaler zeitlicher Änderung - also zum Zeitpunkt positiver oder negativer Maxima - am kleinsten sind. Dieser Effekt entsteht dadurch, dass bei schnellerer Änderung der Beschleunigung der Messfehler im Moment der Abtastung größer wird.

F. Anleitung zum Aufsetzen des Telemetriesenders

Die Anleitung setzt einen fertig aufgesetzten Raspberry Pi voraus. Eine Anleitung für die Installation des Betriebssystems „Raspberry Pi OS Full“ sowie dessen Bedienung findet sich in der PhyPiDAQ-Dokumentation. Diese Anleitung beschränkt sich auf die reine Nutzung der Sende- und Empfangsprogramme zur Übertragung von Beschleunigungsdaten; eine Verwendung im Rahmen von PhyPiDAQ ist noch nicht implementiert. Eine Anleitung zur Nutzung des Telemetriesenders im Rahmen von PhyPiDAQ ist geplant.

A. Benötigtes Material

Für den Telemetriesender:

- Einen Mikrocontroller des Typs Arduino-Nano-RP2040-Connect
- Eine Batteriehalterung mit Deckel und Schalter
- Batterien entsprechend der gewählten Batteriehalterung

Hilfsmittel

- Einen Lötkolben mit Lötzinn
- Einen Rechner mit Internetzugang
- Ein USB-Kabel
- Klebeband oder sonstige Mittel zur Befestigung

B. Einrichten der Arduino IDE

Für die Programmierung des Mikrocontrollers wird die Arduino-Entwicklungsumgebung verwendet, die auf der Softwareseite von Arduino heruntergeladen werden. Zur Auswahl stehen die ältere Version „Arduino IDE 1.8.x“ sowie die neuere Version „Arduino IDE 2.0.x“ (Stand 2022). Die folgende Anleitung bezieht sich auf die ältere Version, kann aber auch für die neuere Version verwendet werden, wobei bei der neuen Version ein häufigeres Auftreten von Fehlfunktionen möglich ist. Nach erfolgreicher Installation wird die Arduino IDE geöffnet. Für die Programmierung des Boards muss zuerst der entsprechende Core heruntergeladen werden. Dazu wird unter „Werkzeuge “ > „Board “ der Boardverwalter aufgerufen (Abbildung B.1).

Durch Klicken öffnet sich ein Fenster, auf welchem die Cores für verschiedene Arduino-Modelle und Serien aufgelistet sind und durch Klicken auf „Installieren“ heruntergeladen werden können (Abbildung B.2). Für das hier verwendete Board muss das Paket „Arduino Mbed OS Nano Boards“ ausgewählt werden.

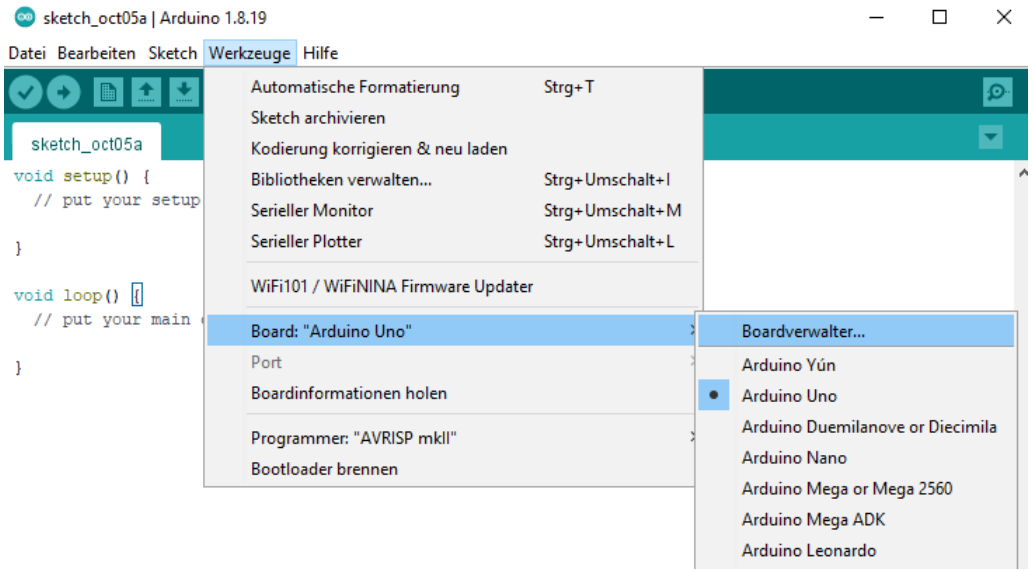

Abbildung B.1.: Aufrufen des Boardverwalters

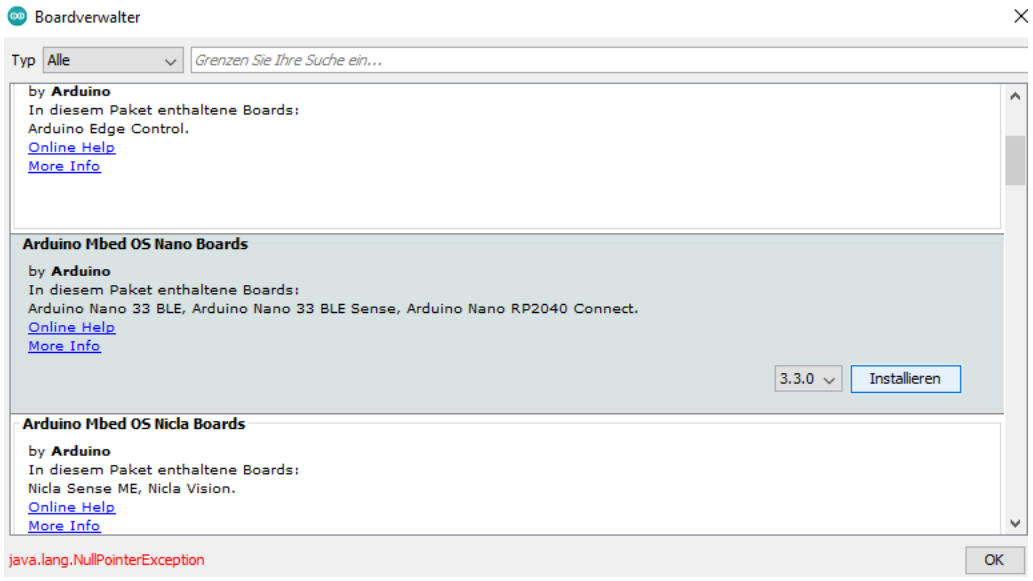

Abbildung B.2.: Auswählen und Installieren des Cores

Unter „Werkzeuge“ > „Board“ findet sich nun in der Liste unterhalb der Boardverwaltung der Name des heruntergeladenen Pakets. Unter diesem kann nun das verwendete Board „Arduino Nano RP2040 Connect“ ausgewählt werden (Abbildung B.3).

Die Sendeprogramme für WLAN und Bluetooth verwenden spezifische Bibliotheken, die eigens heruntergeladen werden müssen. Diese sind:

- `Arduino_LSM6DSOX`
- `PubSubClient`
- `WiFiNINA`
- `ArduinoBLE`

Über den Pfad „Werkzeuge“ > „Bibliotheken verwalten“ können beliebige Bibliotheken gesucht und heruntergeladen werden (Abbildung B.4). Durch Klicken öffnet sich ein neues

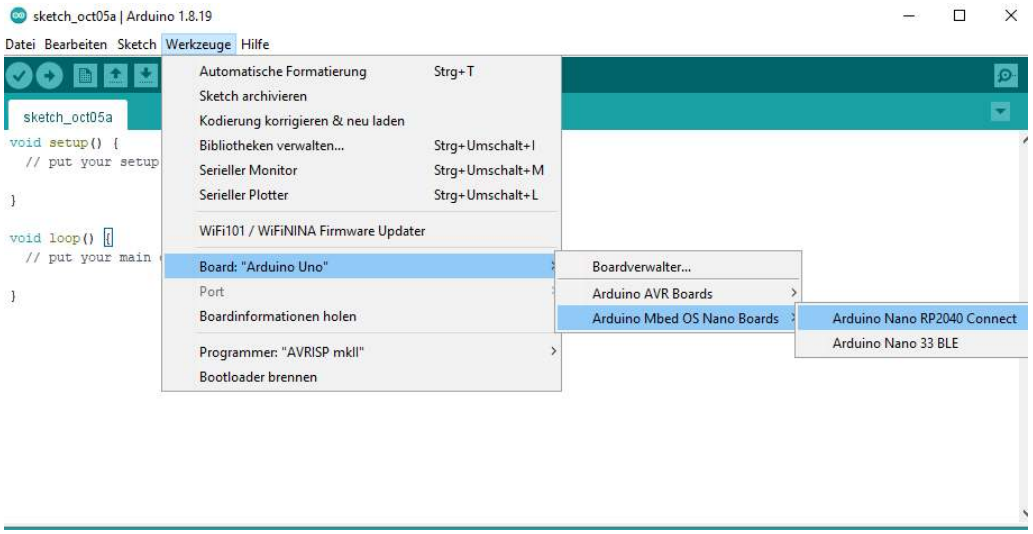

Abbildung B.3.: Auswählen des Board-Modells

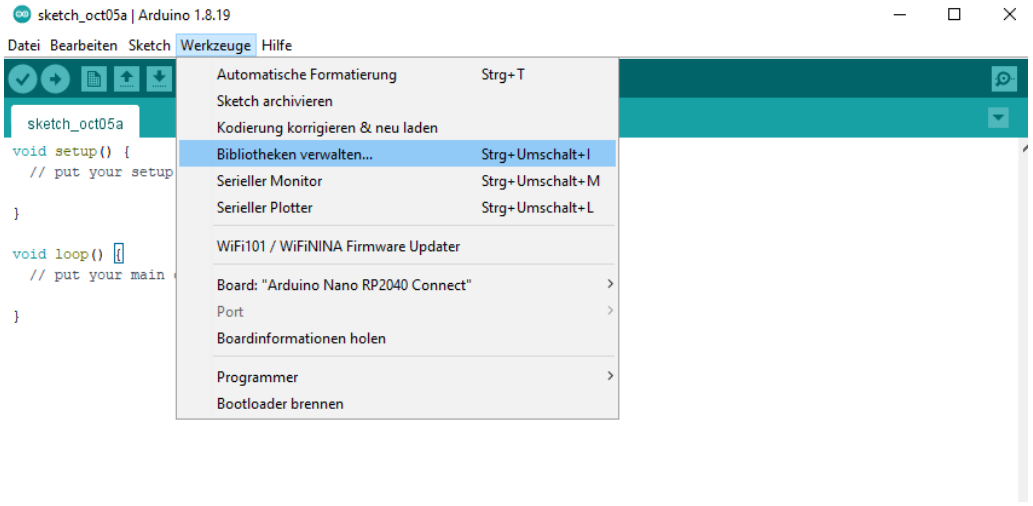

Abbildung B.4.: Verwalten von Bibliotheken

Fenster, in welchem die gewünschte Bibliothek durch Eingabe in die Suchzeile automatisch gesucht und aus der gefundenen Auswahl heruntergeladen werden kann.

C. Herunterladen der Sende- und Empfangsprogramme

Die Sende- und Empfangsprogramme können über ein eigens eingerichtetes Github-Repository heruntergeladen werden. Die Quellcodes für die Datenübertragung mit WLAN und Bluetooth LE finden sich in den Ordnern „WLAN-Übertragung (MQTT)“ und „BLE-Übertragung“. Im Ordner „Messung Beschleunigung und Winkelgeschwindigkeit“ findet sich ferner ein Empfangs- und ein Sendeprogramm für die gleichzeitige Messung von Beschleunigung und Winkelgeschwindigkeit in einem rotierenden System.

Die Empfangsprogramme können direkt auf den Raspberry Pi heruntergeladen werden. Die Sendeprogramme sind als Ordner angelegt; diese werden auf den Rechner mit der installierten Arduino IDE heruntergeladen und in den Ordner „Arduino“ verschoben, der bei der Installation automatisch erstellt wird und über welchen die IDE alle Arduino-Programme verwaltet. Um das Sendeprogramm aufzurufen, wird der Order geöffnet und

darin die gleichnamige „ino“-Datei (auch Sketch genannt) mit einem Doppelklick geöffnet. Alternativ kann auch zuerst die Arduino IDE geöffnet und durch Klicken auf den nach oben zeigenden Pfeil auf der Kopfleiste der Sketch ausgewählt und aufgerufen werden. Zum Laden des Sendeprogramms auf das Arduino-Board muss dieses über ein USB-Kabel mit einem Rechner verbunden werden. Dabei ist zu beachten, dass das USB-Kabel zum Datentransfer geeignet sein muss, da manche USB-Kabel lediglich als Ladekabel fungieren und nur über Adern zur Stromversorgung verfügen. Das mit dem Board mitgelieferte weiße USB-Kabel ist hierfür geeignet. Ist das Board auf diese Weise mit dem Rechner verbunden, wird in den meisten Fällen automatisch der Port erkannt, über welchen der Rechner mit dem Board kommuniziert. Andernfalls muss dieser manuell über den Pfad „Werkzeuge“ > „Port“ eingestellt werden. Über den nach rechts zeigenden Pfeil in der Kopfleiste wird das Hochladen gestartet, was je nach Leistung des Rechners einige Minuten dauern kann. Für die WLAN-Datenübertragung mit dem MQTT-Protokoll müssen im Sendeprogramm vor dem Hochladen auf das Board Einstellungen vorgenommen werden, die weiter unten im Abschnitt „Anwendungshinweise“ näher erläutert werden.

D. Zusammenbau des Telemetriesenders

Das Arduino-Board kann über die Pins „GND“ und „VIN“ mit einer Spannung von 4 V bis 20 V versorgt werden (siehe Anhang D). Als portable Spannungsversorgung eignen sich Batteriehalterungen mit einer Abdeckung, einem Schalter, und zwei Adern als Plus- und Minusleitung. Die Abbildungen D.5, D.6 und D.7 geben einen Überblick über verschiedene geeignete Modelle.

Die rote Leitung wird mit dem Pin „VIN“ verlötet, die schwarze Leitung mit dem Pin „GND“. Für ein sauberes Anlöten der Leitungen sollte darauf geachtet werden, mit der Lötspitze sowohl den Draht als auch die Pin-Fläche zu erhitzen, damit das Lötzinn eine beidseitige Verbindung herstellt und sich gut verteilt. Das Lötzinn wird in Form eines Lötdrahts vorsichtig an der zu erhitzenden Stelle geschmolzen. Ein Verbinden der beiden benachbarten Pins durch überschüssiges Lötzinn ist dabei zu vermeiden; dies führt zu einem Kurzschluss. Abbildung D.8 veranschaulicht die verlöteten Drahtenden.

Schließlich muss das Board auf der Batteriehalterung fixiert werden. Eine einfache Lösung kann mit Klebeband erzielt werden (Abbildung D.9).

E. Anwendungshinweise

E.1. Datenübertragung über WLAN

Für die Datenübertragung über das WLAN-Netz wird das MQTT-Protokoll verwendet. Dazu muss auf dem Raspberry Pi der MQTT-Broker Mosquitto installiert werden. Eine Anleitung dazu findet sich zum Beispiel hier. Für die Ausführung des Empfangsprogramms muss das Python-Modul paho-mqtt mit dem Befehl

```
pip3 install paho-mqtt
```

im Terminal heruntergeladen werden. Bevor das Sendeprogramm auf das Arduino-Board hochgeladen wird, müssen SSID und Passwort des WLAN in der Datei „Einstellungen.h“ eingegeben werden. Die BrokerID kann durch Eingabe des Befehls "hostname -Iïm Terminal des Raspberry Pi bezogen werden.

Im Empfangsprogramm ist eine Steuerung implementiert, mit der das Senden der Daten an- und ausgeschaltet werden kann. Wird so das Empfangsprogramm auf der Thonny IDE ausgeführt, wird das Senden der Daten durch durch Eingabe des Buchstabens „s“ in der Kommandozeile und Drücken von Enter gestartet. Die Eingabe des Buchstabens „e“

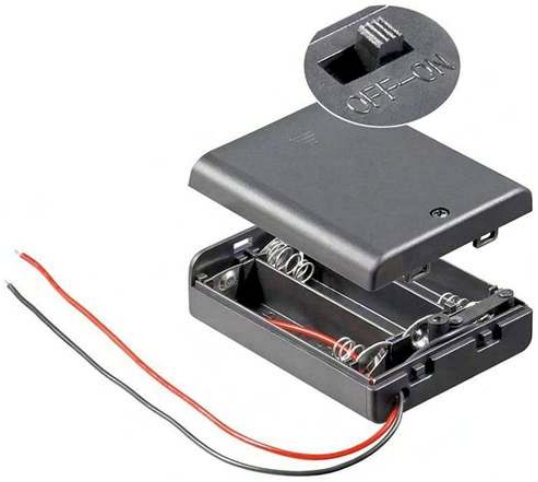

Abbildung D.5.:
4,5 V-Batteriehalterung mit drei 1,5 V-AA-Batterien

Abbildung D.6.:
9 V-Batteriehalterung für eine 9 V-Blockbatterie

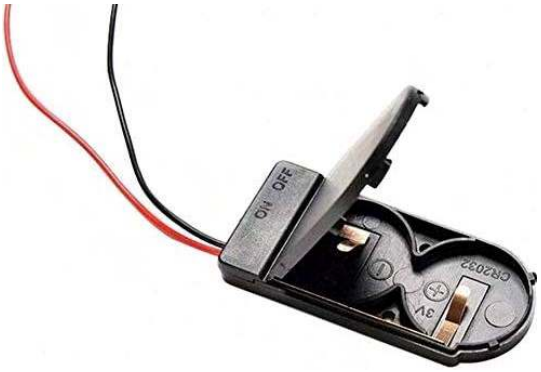

Abbildung D.7.:
6 V-Batteriehalterung für zwei 3 V-Knopfzellen

Abbildung D.8.: Anlöten der Adern an die Pin-Flächen zur Spannungsversorgung

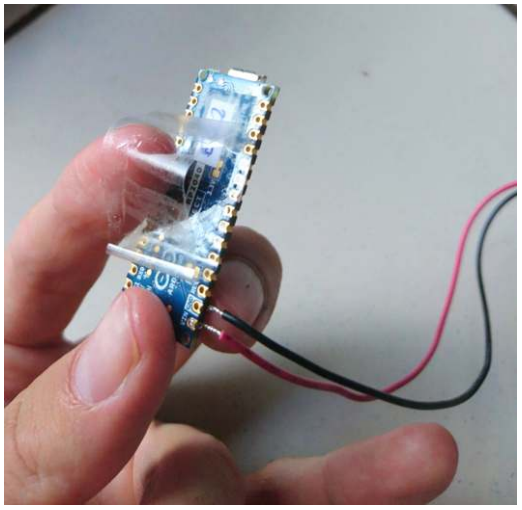

Abbildung D.9.: Klebeband zur flexiblen Befestigung des Boards an der Batteriehalterung

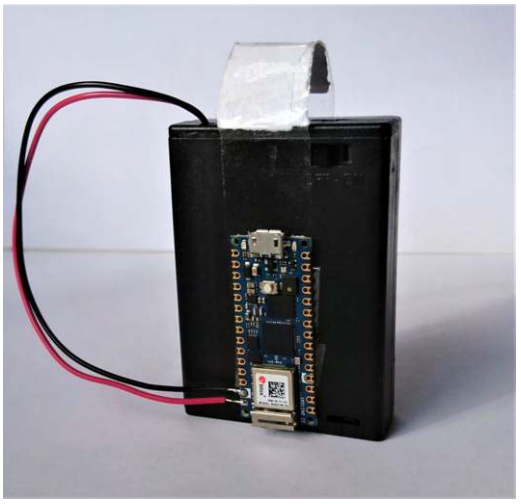

Abbildung D.10.: Beispiel eines fertigen Telemetriesenders mit einer 4,5 V-Batteriehalterung

stoppt das Senden der Daten. Mit den auf dem Board verbauten Farb-LED's wird der Verbindungszustand signalisiert und Rückmeldung über mögliche Fehlfunktionen gegeben. Ein Leuchten der grünen LED signalisiert, dass keine Verbindung zum WLAN-Netz vorliegt. Leuchtet die blaue LED, liegt keine Verbindung zum Raspberry Pi vor.

Eine Datenübertragung über WLAN bietet den Vorteil einer größeren Reichweite im Vergleich zu Bluetooth. Als Nachteile sind ein höherer Stromverbrauch des Telemetriesenders und die Abhängigkeit vom Vorhandensein eines WLAN-Netzes zu nennen. Schwierigkeiten können sich dadurch ergeben, dass WLAN-Netzwerke in der Schule häufig passwortgeschützt sind und so ein Zugang nur mit Angabe der SSID und des zugehörigen WLAN-Schlüssels nicht möglich ist. Als Lösung kann ein Pocket-Router verwendet werden, der an einer Steckdose angesteckt werden kann und so ein eigenes WLAN-Netz (ohne Verbindung zum Internet) aufspannt, dessen Zugangsdaten beliebig angepasst werden können. Als Beispiel sei das Modell TL-WR710N vom Hersteller TP-Link genannt (Abbildung E.11).

Abbildung E.11.: Portabler Pocket Router am Beispiel des Modells TL-WR710N von TP-Link

E.2. Datenübertragung über Bluetooth

Während die Datenübertragung bei WLAN über einen Router als Vermittler stattfindet, stellt Bluetooth eine Punkt-zu-Punkt-Übertragung vom Telemetriesender zum Raspberry Pi dar. Damit entfällt die Notwendigkeit eines WLAN-Netzes und die damit verbundene Eingabe der WLAN-Zugangsdaten im Sendeprogramm.

Für die Ausführung des Empfangsprogramms muss das Python-Modul BluePy heruntergeladen werden. Dazu wird folgender Befehl im Terminal eingegeben:

```
sudo pip3 install bluepy
```

Zum Ein- und Ausschalten von Bluetooth Low Energy muss der Blueman-Bluetooth-Manager verwendet werden. Dieser wird durch Eingabe des folgenden Befehls im Terminal installiert:

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

Um eine Verbindung zwischen Telemetriesender und Raspberry Pi herzustellen, muss im Empfangsprogramm lediglich die MAC-Adresse des verwendeten Arduino-Boards angegeben

werden. Diese wird in der Arduino IDE periodisch im Seriellen Monitor ausgegeben, wenn das Arduino-Board über den USB-Anschluss mit dem Rechner verbunden ist und keine Bluetooth-Verbindung vorliegt. Der Serielle Monitor wird durch Klicken auf das Lupen-Symbol oben rechts auf der Benutzeroberfläche geöffnet. Für eine korrekte Darstellung der ausgegebenen Daten muss darauf geachtet werden, dass die Baudrate (unten rechts) auf 115200 eingestellt ist. Damit der Verbindungsaufbau gelingt, muss der Telemetriesender zwei bis drei Sekunden vor dem Empfangsprogramm gestartet werden. Auch darf nicht vergessen werden, die Bluetooth-Funktion des Raspberry Pi's durch Klicken auf das Desktop-Symbol oben rechts zu aktivieren.

Der einzige wesentliche Nachteil gegenüber der WLAN-Übertragung liegt in einer geringeren Reichweite von etwa 5 Metern. Bricht die Verbindung bei zu großer Entfernung des Telemetriesenders zum Empfänger (dem Raspberry Pi) ab, wird dies durch ein Blinkmuster einer LED des Boards signalisiert. Die genaue Reichweite und die Qualität der Übertragung hängen insgesamt von der Räumlichkeit und von der Belastung des genutzten 2,4 GHz-Bandes durch umgebende WLAN-Netze ab. WLAN-Strahlung durch Handys und Router wirkt tendenziell störend auf die Bluetooth-Übertragung.

Literaturverzeichnis

Textquellen

[A1] PHYWE. *Cobra SMARTsense*. Webseite. URL: https://www.phywe.de/sensoren-software/cobra-smartsense/, besucht am 01.10.22.

[A2] RWTH Aachen. *Phyphox Fernsteuerung*. Webseite. URL: https://phyphox.org/de/fernsteuerung/, abgerufen am 02.10.22.

[A3] Moritz Aupperle. *Konzeption und Gestaltung eines digitalen Messwerterfassungssystems für den Physikunterricht in der Schule*. Master's thesis, Karlsruher Institut für Technologie (KIT), 2018. S. 69 - 71.

[A4] Frank Sichla. *Short-Range Radio auf den Punkt gebracht*. *hf-praxis*, 2016.

[A5] Limor Fried. *All the Internet of Things - Episode Two: Protocols*. Webseite. URL: https://learn.adafruit.com/alltheiot-protocols/http, besucht am 02.10.22.

[A6] Adafruit. *Adafruit HUZZAH32 – ESP32 Feather Board*. Webseite. URL: https://www.adafruit.com/product/3405, besucht am 02.10.22.

[A7] Adafruit. *Adafruit Feather 32u4 with RFM69HCW Packet Radio - 433MHz - RadioFruit*. Webseite. URL: https://www.adafruit.com/product/3077, besucht am 02.10.22.

[A8] Kattni Rembor. *Adafruit Feather nRF52840 Sense*. Webseite. URL: https://learn.adafruit.com/adafruit-feather-sense/overview, besucht am 02.10.22.

[A9] Arduino. *Nano 33 IoT*. Webseite. URL: https://docs.arduino.cc/hardware/nano-33-iot, besucht am 03.10.22.

[A10] Arduino. *Nano 33 BLE*. Webseite. URL: https://docs.arduino.cc/hardware/nano-33-ble, besucht am 03.10.22.

[A11] Arduino. *Nano 33 BLE Sense*. Webseite. URL: https://docs.arduino.cc/hardware/nano-33-ble-sense, besucht am 03.10.22.

[A12] Arduino. *Arduino Nano RP2040 Connect*. PDF. URL: https://content.arduino.cc/assets/ABX00053-datasheet.pdf, besucht am 03.10.22.

[A13] Ministerium für Kultus, Jugend und Sport Baden-Württemberg. *3.2.6 Mechanik: Kinematik*. Webseite. URL: http://www.bildungsplaene-bw.de/,Lde/LS/BP2016BW/ALLG/GYM/PH/IK/7-8/06, besucht am 03.10.22.

[A14] Ministerium für Kultus, Jugend und Sport Baden-Württemberg. *3.3.5.1 Kinematik*. Webseite. URL: http://www.bildungsplaene-bw.de/,Lde/LS/BP2016BW/ALLG/GYM/PH/IK/9-10/05/01, besucht am 03.10.22.

[A15] Ministerium für Kultus, Jugend und Sport Baden-Württemberg. *3.2.6 Mechanik: Kinematik*. Webseite. URL: https://www.bildungsplaene-bw.de/,Lde/LS/BP2016BW/ALLG/SEK1/PH/IK/7-8-9/06, besucht am 03.10.22.

[A16] Ministerium für Kultus, Jugend und Sport Baden-Württemberg. *Physik*. Webseite. URL: http://www.bildungsplaene-bw.de/Ph_OS, besucht am 03.10.22.

[A17] Bianca Watzka et. al. *Beschleunigungssensoren*. PDF. URL: http://www.thomas-wilhelm.net/veroeffentlichung/Beschleunigungssensoren_PdN.pdf, besucht am 09.10.22.

[A18] HiveMQ Team. *MQTT essentials, The Ultimate Guide to MQTT for Beginners and Experts*. Webseite, ebook. URL: https://www.hivemq.com/mqtt-essentials/, besucht am 03.10.22.

[A19] Eclipse Foundation. *Eclipse Mosquitto. An open source MQTT broker*. Webseite. URL: https://mosquitto.org, besucht am 03.10.22.

[A20] Roger Light. *paho-mqtt 1.6.1*. Webseite. URL: https://pypi.org/project/paho-mqtt/, besucht am 03.10.22.

[A21] Liz Clark, Kattni Rembor. *CircuitPython on the Arduino Nano RP2040 Connect*. Webseite. URL: https://learn.adafruit.com/circuitpython-on-the-arduino-nano-rp2040-connect/install-circuitpython, besucht am 03.10.22.

[A22] Himanshu Bhalla, Oren Haggai. Unraveling Bluetooth LE Audio, chapter 2. Bluetooth Overview. Apress, Berkeley, CA., 2021. DOI: 10.1007/978-1-4842-6658-8_2, ISBN: 978-1-4842-6657-1.

[A23] Silke Grasreiner. *Funktionsweise von Bluetooth*. Webseite. URL: https://de.ccm.net/contents/605-funktionsweise-von-bluetooth, besucht am 03.10.22.

[A24] Martin Sauter. *Grundkurs Mobile Kommunikationssysteme*, chapter 6. Bluetooth. Springer Vieweg Wiesbaden, 2015. DOI: 10.1007/978-3-658-08342-7_6, ISBM: 978-3-658-08342-7.

[A25] Kevin Townsend. *Introduction to Bluetooth Low Energy*. Webseite. URL: https://learn.adafruit.com/introduction-to-bluetooth-low-energy, besucht am 03.10.22.

[A26] o. A. *BLE Advertising Primer*. URL: https://www.argenox.com/library/bluetooth-low-energy/ble-advertising-primer/, besucht am 03.10.22.

[A27] Martin Woolley. *The Bluetooth® Low Energy Primer*. PDF, Juni 2022. URL: https://www.bluetooth.com/wp-content/uploads/2022/05/Bluetooth_LE_Primer_Paper.pdf, besucht am 03.10.22.

[A28] Kevin Townsend et. al. Getting Started with Bluetooth Low Energy, chapter 4. GATT (Services and Characteristics). O'Reilly Media, Inc., 2014. URL: https://www.oreilly.com/library/view/getting-started-with/9781491900550/ch04.html, besucht am 03.10.22.

[A29] Ian Harvey. *bluepy - a Bluetooth LE interface for Python*. Webseite. URL: https://ianharvey.github.io/bluepy-doc/, besucht am 03.10.22.

Bildquellen

[B30] PHYWE. *Cobra SMARTsense Acceleration (3-axis) - Sensor zur Messung der Beschleunigung in 3 Achsen ± 8 g (Bluetooth + USB)*. URL: https://www.phywe.de/geraete-zubehoer/messgeraete/kraft-beschleunigung-bewegung/cobra-smartsense-acceleration-3-axis-sensor-zur-messung-der-beschleunigung-in-3-achsen-8-g-bluetooth-usb_2092_3023/, besucht am 03.10.22.

[B31] Moritz Aupperle. *Konzeption und Gestaltung eines digitalen Messwerterfassungssystems für den Physikunterricht in der Schule*. Master's thesis, Karlsruher Institut für Technologie (KIT), 2018. S. 69 - 71.

[B32] Make Magazin DE. *Evaluierungsplattfom für NodeMCU*, 2016. URL: https://upload.wikimedia.org/wikipedia/commons/e/e5/Nodemcu_amica_bot_02.png, besucht am 02.10.22. Lizenz: CC BY-SA 4.0.

[B33] Kattni Rembor. *sensors_Feather_Sense_top.jpg*. URL: https://learn.adafruit.com/assets/89096, besucht am 02.10.22. Lizenz: CC BY-SA 3.0.

[B34] Clemens Valens. *Review: Arduino Nano RP2040 Connect*. Webseite. URL: https://www.elektormagazine.de/articles/review-arduino-nano-rp2040-connect, besucht am 03.10.22.

[B35] Bianca Watzka et. al. *Beschleunigungssensoren*. PDF. URL: http://www.thomas-wilhelm.net/veroeffentlichung/Beschleunigungssensoren_PdN.pdf, besucht am 09.10.22.

[B36] Karl Söderby. *Accessing IMU Data on Nano RP2040 Connect*, Oktober 2022. URL: https://docs.arduino.cc/tutorials/nano-rp2040-connect/rp2040-imu-basics, besucht am 08.11.22.

[B37] Arduino. *Arduino Nano RP2040 Connect*. Webseite. URL: https://docs.arduino.cc/static/8b9e4e17c1e1afa836057c5ba87c27c9/2f891/pinout.png, besucht am 03.10.22.

[B38] Arduino. *Arduino Nano RP2040 Connect*. PDF. URL: https://content.arduino.cc/assets/ABX00053-datasheet.pdf, besucht am 03.10.22.