AudiTim Dokumentation

Bearbeitungszeitraum	10 Wochen
Gruppe, Kurs	5, INF2023AI
Gruppenmitglieder	Aaron Reiber, Jan-David Oberländer, Luca Müller, Moritz Flaig

Inhaltsverzeichnis

1 Einleitung

Im Rahmen des Software-Engineering-Projekts wird in einem Team ein System zur akustischen Raumüberwachung entwickelt. Ziel des Projekts ist die kontinuierliche Erfassung und spätere Analyse der Lautstärkeverteilung in einem typischen Vorlesungsraum über den Zeitraum mehrerer Tage hinweg.

1.1 Projektüberblick

Die Umsetzung erfolgt nach agilen Prinzipien unter Anwendung von Scrum, wobei die jeweiligen Sprints von Freitag bis Freitag festgelegt wurden.

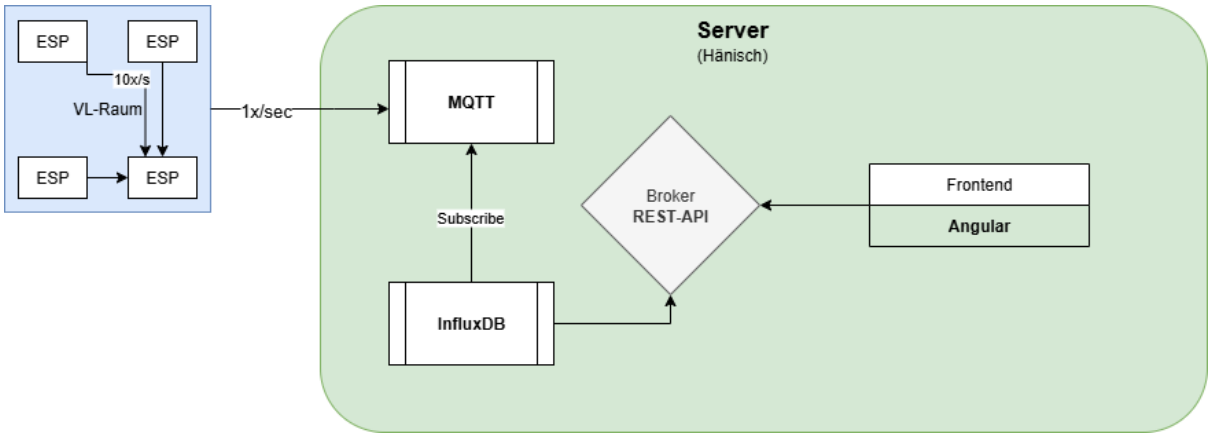
Die technische Architektur sieht den Einsatz von vier Mikrocontroller-Einheiten (ESP32) vor, die jeweils mit einem Mikrofon oder einem Dezibel-Messsensor ausgestattet werden. Die Geräte werden in den vier Ecken des Vorlesungsraums positioniert, um eine möglichst flächendeckende Abdeckung des Raumes zu gewährleisten. Über ein Kommunikationsprotokoll tauschen sich die ESPs untereinander aus und senden die gesammelten Messdaten gemeinsam an ein zentrales Backend.

1.2 Ziele und Motivation

Ziel ist es, ein skalierbares und robustes System zu entwickeln, das Lautstärkepegel über einen längeren Zeitraum erfassen kann. Die gesammelten Daten sollen im Backend gespeichert und im Nachgang analysiert werden. Hierbei sind verschiedene Darstellungsformen wie etwa Heatmaps oder zeitbasierte Diagramme denkbar, um Muster oder Auffälligkeiten im akustischen Verhalten des Raumes sichtbar zu machen.

1.3 MVP und Qualitätsmerkmale

Im Rahmen des MVPs soll ein verteiltes System zur akustischen Raumüberwachung auf Basis von vier Mikrocontrollern der ESP-Serie entwickelt werden. Jeder ESP wird über ein extern angeschlossenes Mikrofon verfügen, mit dem kontinuierlich Lautstärkedaten erfasst werden. Drei der Module sollen ihre Messwerte mithilfe des ESP-NOW-Protokolls an eine zentrale Einheit übermitteln, die zusätzlich eigene Daten erfasst. Die aggregierten Informationen aller vier Einheiten werden in festen Intervallen über das MQTT-Protokoll an einen Server gesendet und dort zeitsynchron in einer InfluxDB gespeichert. Über eine REST-API sollen die Daten einem Angular-basierten Web-Frontend zur Visualisierung der akustischen Messwerte zur Verfügung gestellt werden.



Aufgrundlage des MVPs sollen die folgenden Qualitätsmerkmale sichergestellt werden:

- **Echtheit & Qualität der Daten:** Nur qualitativ "gute"Daten z.B. durch optimierte Mikrofongehäuse ermöglichen realistische Messwerte und damit eine sinnvolle Analyse der akustischen Verhältnisse im Raum.
- **Accountability der Daten:** Es muss nachvollziehbar sein, welche Sensoren zu welchem Zeitpunkt welche Daten erfasst haben.

2 SFMEA-Analyse

Systemfunktion	Fehlermöglichkeit	Fehlerursache	Fehlerfolge	B	A	E	RPZ	Maßnahmen
Dezibel erfassen	Falsche oder keine Messwerte	Mikrofon defekt, Sensor ungeeignet	Ungenaue oder keine Daten	8	6	3	144	Robuste Mikros, Kalibrierung, Ersatzsensor bereit
ESP kommuniziert nicht mit anderen	Keine oder fehlerhafte Datenübertragung	WLAN-Probleme, falsches Protokoll	Daten gehen verloren, keine Heatmap	7	5	4	140	Netzwerk prüfen, Fallback, Fehler-Logs
Daten werden nicht ans Backend gesendet	ESP schickt keine Daten	Stromausfall, Code-crash, Timeout	Datenlücken, unvollständige Analyse	7	5	3	105	Watchdog, Logging, häufigere Syncs
GitHub-Zugang fehlerhaft	Kein Zugriff oder Merge-Konflikte	Rechte falsch, keine Git-Strategie	Team kann nicht arbeiten	6	4	2	48	Git-Workflow, Rechte regeln

Systemfunktion	Fehlermöglichkeit	Fehlerursache	Fehlerfolge	B	A	E	RPZ	Maßnahmen
Backend speichert keine Daten	Fehlerhafte Speicherung / Absturz	Server voll, Codefehler	Datenverlust, Ausfall	9	4	4	144	Monitoring, Logging, Backups, Testlauf
Daten werden falsch ausgewertet	Heatmap falsch / Werte inkonsistent	Algorithmusfehler, unvollständige Daten	Falsche Rückschlüsse	6	4	3	72	Testdaten prüfen, Algo validieren, Visualisierung testen
ESPs messen nicht synchron	Unterschiedliche Messzeitpunkte	Kein Sync, NTP fehlt	Daten nicht vergleichbar	7	5	4	140	NTP-Sync, Zeitstempel ergänzen

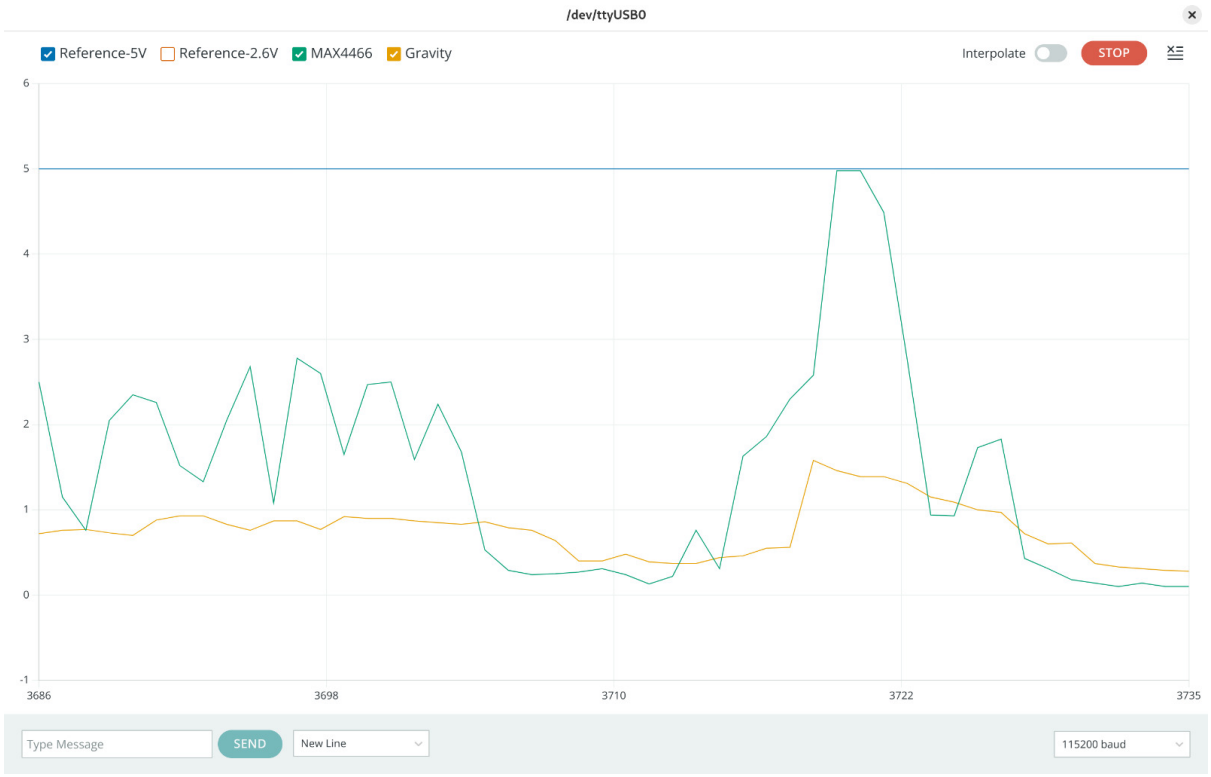
3 Auswahl und Bewertung der Hard- und Softwarekomponenten

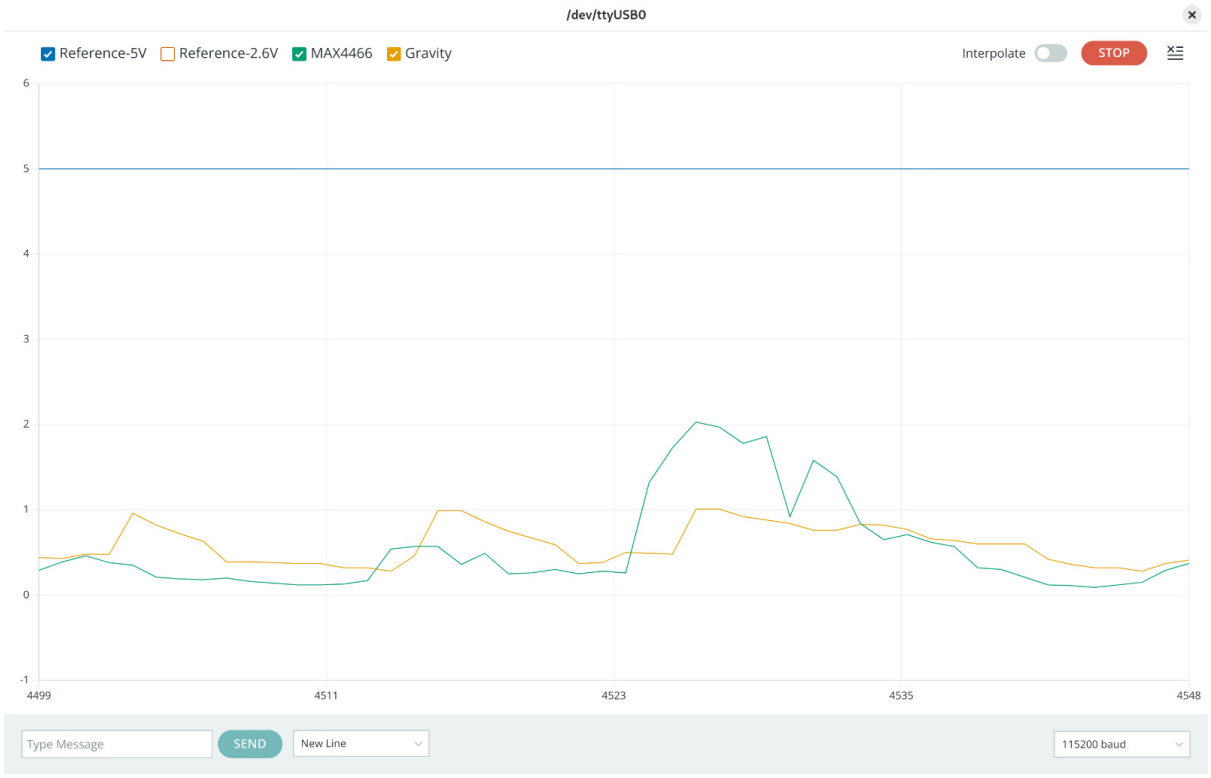
3.1 Hardwarekomponenten

3.1.1 Mikrofonmodule

Zur Erfassung des Lautstärkepegels wurden drei unterschiedliche Mikrofonmodule in Betracht gezogen. Der erste Sensor ist ein kostengünstiges Modul "GY-MAX4466"(ca. 4 €), das ursprünglich als Klatschsensor konzipiert wurde. Der zweite Sensor ist das *Sound Level Meter V2.0* von DFRobot (ca. 40 €), das über zusätzliche Hardware zur Signalverarbeitung verfügt und den Schalldruckpegel bereits gefiltert und bereinigt im dBA-Format ausgibt. Der dritte in Betracht gezogene Sensor ist ein handelsübliches DB-Messgerät (ca. 20 €), das jedoch nicht für die Integration in das Projekt geeignet ist, da es keine digitale Schnittstelle bietet und somit nicht direkt mit dem ESP32 kommunizieren kann. Dieser Sensor wird daher zur Überprüfung der Ergebnisse verwendet, ist aber nicht für die direkte Integration in das Projekt vorgesehen.

Da die absolute Einheit (dBA) für die Auswertung im Rahmen eines 3D-Diagramms nicht zwingend erforderlich ist, wurde der Fokus auf die Vergleichbarkeit der analogen Ausgangssignale gelegt. Beide Sensoren wurden gleichzeitig an einem ESP32 betrieben und ihre Ausgangssignale mithilfe des *Serial Plotter* in Echtzeit visualisiert. Dabei zeigte sich, dass die Kurvenverläufe bei normalen Sprachgeräuschen nahezu identisch sind. Lediglich bei impulsartigen Geräuschen mit sehr geringer Distanz (z. B. Klatschen oder Klopfen) erzeugt das günstigere Mikrofon deutlich stärkere Peaks, während diese durch den DFRobot-Sensor wirksam herausgefiltert werden.


/dev/ttyUSB0
Reference-5V
Reference-2.6V
MAX4466
Gravity
Interpolate
STOP
6
5
4
3
2
1
0
-1
3686
3698
3710
3722
3735
Type Message
SEND
New Line
115200 baud


/dev/ttyUSB0
Reference-5V
Reference-2.6V
MAX4466
Gravity
Interpolate
STOP
6
5
4
3
2
1
0
-1
4499
4511
4523
4535
4548
Type Message
SEND
New Line
115200 baud

Hier sind die Sensorwerte des günstigeren Klatschsensors (grün), sowie die des Teureren Dba sensors (gelb) dargestellt. Die In Blau dargestellte 5v Referenzlinie stellt den Maximalwert da, welcher von den Sensoren zurückgegeben werden kann.

Da die in den Vergleichsdiagrammen dargestellten Abweichungen bei größerem Abstand deutlich abnehmen und im Anwendungsfall als vernachlässigbar eingestuft werden können, wurde entschieden, den günstigeren Sensor zu verwenden. Um mögliche Reststörungen dennoch weiter zu reduzieren, ist geplant, eine mechanische Dämpfung oder Abschirmung am Sensor zu testen.

Insgesamt konnte festgestellt werden, dass der günstigere Sensor trotz fehlender dBA-Ausgabe für die angestrebte Visualisierung ausreichend präzise Ergebnisse liefert und somit für den weiteren Projektverlauf verwendet wird.

3.2 Softwarekomponenten

3.2.1 Entwicklungsumgebungen

Zu Beginn des Projekts wurden zwei gängige Entwicklungsumgebungen für die Programmierung von Mikrocontrollern evaluiert: die klassische *Arduino IDE*, welche auch von den betreuenden Dozierenden empfohlen wurde, sowie die moderne, zunehmend verbreitete *PlatformIO*-Erweiterung für Visual Studio Code.

Im Projektteam wird plattformübergreifend mit Linux, Windows 10 und Windows 11 gearbeitet. Beide Entwicklungsumgebungen sind grundsätzlich mit allen eingesetzten Betriebssystemen kompatibel. Zur Prüfung der technischen Umsetzbarkeit wurden zwei separate Repositories eingerichtet, in denen jeweils einfache Testprojekte zur Ansteuerung eines ESP32 implementiert wurden.

Beide IDEs konnten den Mikrocontroller erfolgreich ansprechen. PlatformIO bietet durch seine deklarative Verwaltung von Bibliotheken und Abhängigkeiten Vorteile in Bezug auf Reproduzierbarkeit und Build-Konsistenz. In der Praxis kam es jedoch auf einem System im Team zu anhaltenden technischen Problemen mit PlatformIO, die nicht zufriedenstellend gelöst werden konnten.

Um eine einheitliche und zuverlässige Entwicklungsgrundlage für alle Teammitglieder sicherzustellen, fiel die Entscheidung letztlich zugunsten der *Arduino IDE*. Diese ermöglicht eine barrierefreie Mitarbeit aller Beteiligten, auch wenn sie im Funktionsumfang nicht ganz an PlatformIO heranreicht.

3.2.2 Kommunikationsmethode

ESP-NOW wurde ausgewählt, da es im Vergleich zu anderen Protokollen einen deutlich geringeren Overhead besitzt und dadurch eine schnellere und effizientere Datenübertragung ermöglicht. Die Struktur ist weniger komplex, was die Implementierung vereinfacht und die Fehleranfälligkeit reduziert. Dadurch wird das System insgesamt robuster und zuverlässiger: ESP-Now ist ein Kommunikationsprotokoll von Espressif, das speziell für die ESP32- und ESP8266-Chips entwickelt wurde. Es verwendet die 2,4 Ghz Frequenz, welche eigentlich für WiFi Kommunikation verwendet wird. Im Gegensatz zu anderen Protokollen sind die Nachrichten auf Schicht 2 des ISO-OSI-Modells. Keine IP-Adressen sondern nur MAC-Adressen. Die ESPs müssen sich dafür im gleichen Kanal befinden. Die 2,4 Ghz Frequenz hat grundsätzlich nur 3 Nutzbare Kanäle. Der zu verwendene Kanal muss von dem ÄI401"Wlan Netz übernommen werden.Es ist mit relativ wenig Code möglich eine große Menge an Daten in sehr Kurzen Zeitabschnitten zu senden. Theoretisch ist eine Datenübertragung von 1 bis 2 Mbps möglich. Die Nachrichtenlänge ist auf 250 Byte pro Paket beschränkt. Da unsere Nachrichten nur wenige Bit lang sind, haben wir hier keine Probleme. Das Protokoll hat keine festgelegten Sender/Empfänger Rollen. Jeder ESP kann senden und empfangen. In unserem Beispiel senden aber 3 ESPs und ein Edge-Device empfängt alle Daten. Es wird 10 mal pro Sekunde der Lautstärkepegel erfasst und gesendet.

3.2.3 ESP32

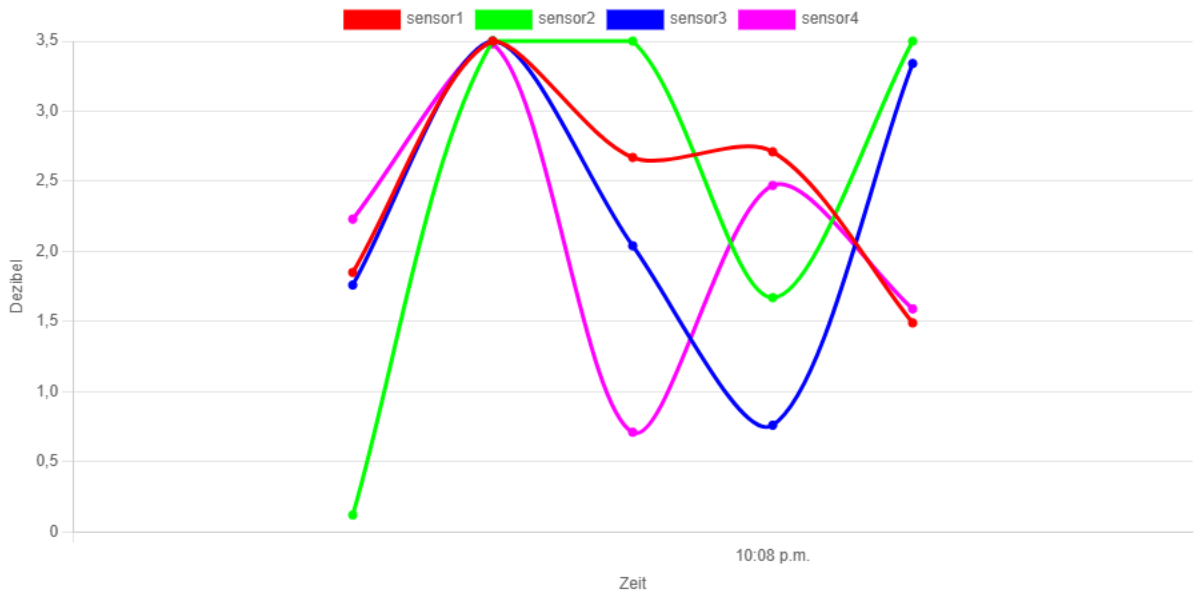
Die ESP-Module bieten im Vergleich zu klassischen Arduinos eine integrierte WLAN-Funktionalität und unterstützen direkt das ESP-NOW-Protokoll. Dadurch entfällt zusätzlicher Hardware- oder Softwareaufwand für die Netzwerkkommunikation. In Kombination mit ESP-NOW ergibt sich somit eine kompakte, performante und zuverlässige Lösung.

3.2.4 Frontend

Für die Umsetzung des Frontends wurde Angular gewählt, da im Team bereits umfangreiche Erfahrung mit diesem Framework vorhanden war. Angular bietet eine klare Struktur durch komponentenbasiertes Design, erleichtert die Wiederverwendbarkeit von UI-Elementen und unterstützt eine saubere Trennung von Logik und Darstellung. Dadurch konnte effizient und wartbar entwickelt werden.

Das Frontend läuft in einem eigenen Docker-Container und kommuniziert, über ein internes Netzwerk, mit dem separat laufenden Backend-Container. So bleibt die Architektur modular, skalierbar und einfach wartbar.

Die Darstellung der Daten wird über Angular-Komponenten realisiert, die Visualisierung in einfachen Liniendiagrammen wird mit der ng2-charts-Bibliothek ermöglicht.



3.2.5 Backend

Das Backend wurde mit Node.js und dem minimalistischen Framework Express.js umgesetzt. Express ist das am weitesten verbreitete Node.js-Framework für Webserver und APIs. Es bietet eine ausgereifte Middleware-Architektur, die sehr flexibel und modular ist. Im Vergleich zu ähnlich minimalistischen Frameworks wie Koa, das von den gleichen Entwicklern stammt, bietet Express mehr out-of-the-box Funktionalitäten

und eine größere Community mit umfangreichen Plugins und Support. (Community, 2025; Appventurez, 2025)

Koa hingegen ist moderner und basiert stärker auf ES6 Features wie async/await, was den Code oft sauberer macht. Allerdings ist Koa weniger „batteries included“ und erfordert mehr Initialaufwand und zusätzliche Libraries, um Funktionen bereitzustellen, die Express standardmäßig mitbringt. Für dieses Projekt, welches einen schnellen Einstieg, umfangreiche Dokumentation und stabile Middleware braucht, ist Express daher die pragmatischere Wahl. Zudem sind Gruppenmitglieder bereits mit Express vertraut, was die Einarbeitungszeit verkürzt und die Produktivität steigert.

Die Anbindung an eine InfluxDB erfolgt über die offizielle Client-Bibliothek. Die Entscheidung für InfluxDB basiert auf der Empfehlung der "Verteilten SystemeDozenten sowie der nahtlosen Integration in das Node.js-Backend, was die Implementierung vereinfacht. Neben dem Abrufen historischer Sensorwerte werden auch Funktionen zum Schreiben von Dummy-Daten bereitgestellt. Diese werden, solange die Verbindung zu den wirklichen Sensorwerten noch nicht hergestellt ist, zum Testen verwendet.

Das Backend ist ebenfalls in einem eigenen Docker-Container gekapselt. Die Kommunikation mit dem Frontend erfolgt über definierte HTTP-Endpunkte im Container-Netzwerk. Diese Trennung verbessert die Wartbarkeit und erlaubt eine flexible Skalierung beider Komponenten unabhängig voneinander.

4 Befestigung der Hardware

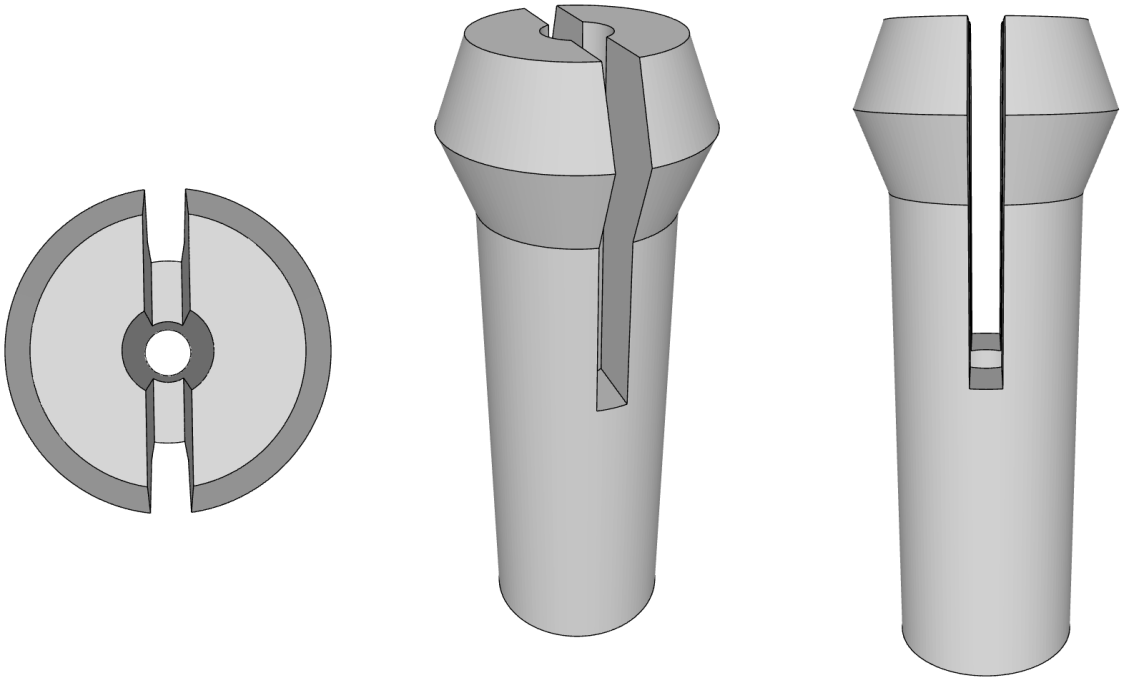
Um die Hardware des AudiTim-Systems sicher und stabil zu befestigen, sind verschiedene Montagemöglichkeiten verfügbar. Ein Großes Problem bei vorgefertigten halterungen ist jedoch, dass diese oft an die and geschraubt oder anderweitig befestigt werden müssen. Daher fiel für unser Projekt die wahl auf eine selbst designte Halterung, welche in einem 3D-Drucker hergestellt werden kann. Dies Garantiert dass keine schäden an der Hardware oder am Raum in welchem die Hardware montiert wird entstehen.

4.1 Platzungskonzept

Aus akustischer Sicht liefern Messungen die zuverlässigsten Ergebnisse, wenn die Sensoren auf Kopfhöhe im Sitzen (1,2-1,5 m Höhe) angebracht sind und mindestens 1 Meter Abstand zu Wänden und der Schallquelle haben. Dieses Vorgehen minimiert frühe Reflexionen, stehende Wellen und lokale Pegelabweichungen - Faktoren, die die Messgenauigkeit maßgeblich beeinträchtigen (Oltheten, 2019) . Leider ist diese ideale Positionierung in unserem Fall praktisch nicht umsetzbar. So birgt Montage am Boden, Tischen oder gesonderte SSTänderërhöhte Gefahr von Beschädigungen, Verschiebungen und Manipulation. An der Decke jedoch beifinden sich bereits vorhandene Löcher; diese können genutzt werden, um die Sensoren und den Mikrocontroller sicher zu befestigen. Auch den einen Meter Abstand zu Wänden und Schallquellen kann so eingehalten werden. Dieser Kompromiss ermöglicht eine schützende, wartungsarme und vergleichbare Positionierung der Sensoren. So bleibt der Messaufbau reproduzierbar, sicher und funktional, auch wenn die akustisch ideale Lösung nicht vollständig umsetzbar ist.

4.2 Deckenhalterung

Um die Hardware an der Decke zu befestigen, wurde eine spezielle Halterung entworfen, die eine sichere Montage ermöglicht. Diese Halterung nutzt die in den Decken vorhandenen "Löcher"für die Montage und wurde als eine Art Pin konzipiert.

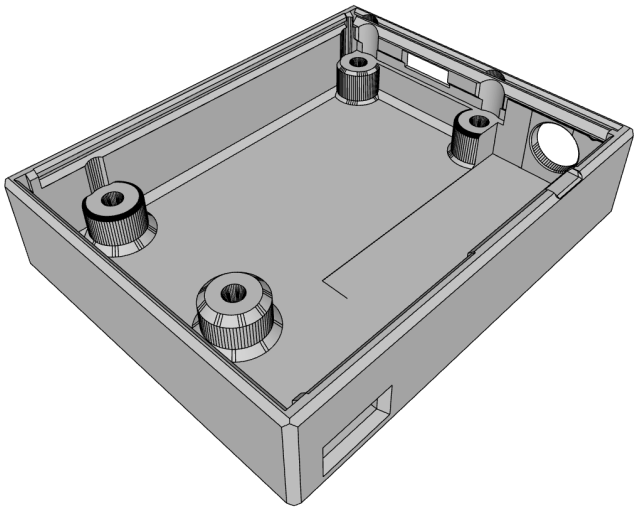


Der in den Bildern Dargestellte Pin kann durch den Spalt in der Mitte in die öffnung der Decke gesteckt werden. Durch die Konische form der Spitze des Pins wird garantiert dass dieser auch bei der Demontage des Projekts keine schäden hinterlässt. Das runde loch welches sich durch den Pin zieht dient dazu, den Pin mit einer Schraube an der Hardware zu befestigen.

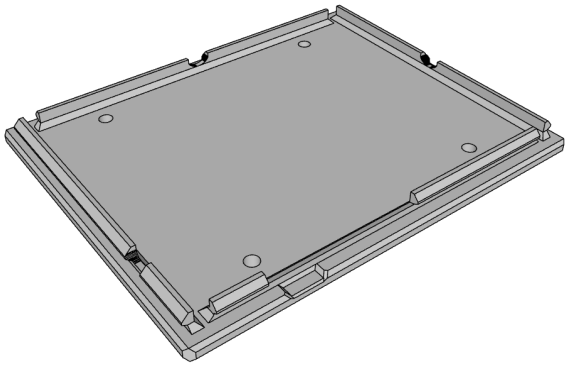
4.3 ESP32-Halterung

Der Esp benötigt eine Box welche sicher an der Decke mit den Pins befestigt werden kann. Bei dem Design der Box wurde sich an einigen Vorlagen orientiert um eine

optimale Passform zu gewährleisten. Die Box ist so gestaltet, dass sie den ESP32 sicher hält und gleichzeitig genug Platz für die Verkabelung bietet.

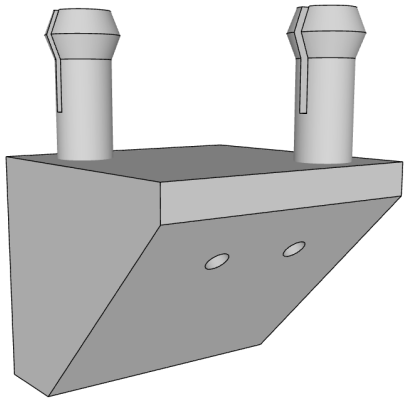


Die angesprochenen löcher der Pins kommen in der Decke der Box zum Einsatz, um die Box sicher an der Decke zu befestigen. Diese werden durch die löcher in der Decke mit schrauben von innen befestigt.



4.4 Sensor-Halterung

Die Sensorhalterung wurde so entworfen dass sie platz für einen möglichen Dämpfer bietet, welcher um das Mikrofon befestigt werden kann. Auch diese Halterung wird mit den Pins an der Decke befestigt, jedoch wird sie nicht geschraubt sondern direkt mit den Pins gedruckt. Der Sensor wird dann einfach mit zwei schrauben an der Halterung befestigt.



Literaturverzeichnis

Appventurez (2025), 'Node.js frameworks comparison: Express.js, koa.js, and more'. Accessed: 2025-08-01.
URL: *https://www.appventurez.com/blog/node-js-framework*

Community, B. (2025), 'Koa.js vs express.js: Which node.js framework to choose?'. Accessed: 2025-08-01.
URL: *https://betterstack.com/community/guides/scaling-nodejs/koajs-vs-expressjs*

Oltheten, W. (2019), 'The 7 steps of ideal mic placement'. Accessed: 2025-07-31.
URL: *https://sonicscoop.com/7-steps-for-ideal-mic-placement/*