Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering

InIT Institut für angewandte
Informationstechnologie

Bachelorarbeit Informatik

Real-Stereo Extended: Automatische Lautstärkenangleichung durch Personenlokalisierung

Autoren	Marc Berchtold Cyril Wanner
Hauptbetreuung	Jürgen Spielberger

Datum	11.06.2021

Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering

Erklärung betreffend das selbstständige Verfassen einer Bachelorarbeit an der School of Engineering

Mit der Abgabe dieser Bachelorarbeit versichert der/die Studierende, dass er/sie die Arbeit selbständig und ohne fremde Hilfe verfasst hat. (Bei Gruppenarbeiten gelten die Leistungen der übrigen Gruppenmitglieder nicht als fremde Hilfe.)
Der/die unterzeichnende Studierende erklärt, dass alle zitierten Quellen (auch Internetseiten) im Text oder Anhang korrekt nachgewiesen sind, d.h. dass die Bachelorarbeit keine Plagiate enthält, also keine Teile, die teilweise oder vollständig aus einem fremden Text oder einer fremden Arbeit unter Vorgabe der eigenen Urheberschaft bzw. ohne Quellenangabe übernommen worden sind.

Bei Verfehlungen aller Art treten die Paragraphen 39 und 40 (Unredlichkeit und Verfahren bei Unredlichkeit) der ZHAW Prüfungsordnung sowie die Bestimmungen der Disziplinarmassnahmen der Hochschulordnung in Kraft.

Ort, Datum:	**Name Studierende:**
Suhr, 11.06.2021	Marc Berchtold
Bachenbülach, 11.06.2021	Cyril Wanner

Zusammenfassung

In der vorgängig durchgeführten Projektarbeit [1], welche den Projektnamen Real-Stereo trug, wurde ein System erarbeitet, welches die Lautstärke von mehreren Lautsprechern kontinuierlich an die Position des Hörers im Raum anpasst. Diese Bachelorarbeit befasst sich mit der Weiterführung und Ausarbeitung der verwendeten Methoden für die Personenlokalisierung, das Balancing- und Kalibrationssystem und die Benutzeroberfläche.

Die Erkennungsgenauigkeit und -geschwindigkeit soll verbessert werden und alle Komponenten, welche für die Verwendung der Real-Stereo Software benötigt werden, sollen möglichst kabellos für den Endbenutzer als eigenständiges System installierbar sein. Um dieses Ziel zu erreichen, werden Sonos™ Lautsprecher und Raspberry Pis mit eingebauten Kameras eingesetzt, welche die Verwendung von kabellosen Netzwerken unterstützen. Auch wird die benötigte Funktionalität erarbeitet, damit der Endbenutzer das System einfach erweitern kann und auch in mehreren Räumen gleichzeitig einsetzen kann. Eine Webapp wird entwickelt, welche vom Endbenutzer auf einem Smartphone aufgerufen werden kann und womit alle wichtigen Funktionen und Einstellungen des Systems kontrolliert und kalibriert werden. Unter anderem kann damit für jede der eingesetzten Kameras der Personenerkennungs-Algorithmus ausgewählt werden, welcher die beste Leistung für den Raum bietet. Auch die Strategie, wie mit mehreren erkannten Personen im Raum umgegangen werden soll, kann über die Webapp vom Endbenutzer jederzeit angepasst werden.

Das Resultat dieser Bachelorarbeit ist ein marktreifes, modulares System, welches mittels mehreren Kamera-Einheiten die Position des Hörers kontinuierlich ermittelt. Anhand dieser Positionsdaten wird die Lautstärke der am System angeschlossenen Lautsprecher angepasst, damit der Hörer an jeder Position im Raum die gleiche Lautstärke wahrnimmt. Die durchgeführten Tests bestätigen die Effektivität des Systems und das Erreichen der gesetzten Ziele.

Abstract

In the previously conducted project Real-Stereo [1], a system was developed which continuously adjusts the volume of individual loudspeakers to the position of the listener in the room. This bachelor thesis deals with the continuation and extension of the used methods and techniques for person localization, the balancing and calibration system, and the user interface.

A special focus has been put on improving detection accuracy and speed. Additionally, the entire system should be as wireless as possible for the end-user to install. To achieve this goal, Sonos™ speakers and Raspberry Pis with built-in cameras, which support the use of wireless networks, are to be used. New functionality is added to enable the end-user to easily expand and use the system in multiple rooms at the same time. A web app is to be developed which the end-user can use on his or her smartphone and with which all important functions and settings of the system can be controlled and calibrated. Among other things, it can be used to select the person detection algorithm for each of the cameras used to maximise the person detection performance for the room. The strategy for dealing with multiple detected people in one room can also be adjusted by the end-user at any time using the web app.

The result of this bachelor thesis is a market-ready, modular system that uses multiple camera units to continuously determine the position of the listener. Based on this position data, the volume of the loudspeakers connected to the system is adjusted so that the listener perceives the same volume at every position in the room. The performed tests confirm the effectiveness of the system and the achievement of the goals.

Vorwort

Unsere Motivation für diese Bachelorarbeit waren die Resultate der vorhergehenden Projektarbeit Real-Stereo. Die Resultate und der entwickelte Prototyp bestätigten die Machbarkeit der grundliegenden Idee und zeigten uns, in welchen Bereichen noch weitere Arbeit nötig ist. Zusätzlich gaben sie uns Anhaltspunkte, wie die Resultate weiter verbessert werden könnten.

Zusammen mit unserem Betreuer Prof. Jürgen Spielberger definierten wir die Ziele und Fragestellungen, welche wir mit dieser Bachelorarbeit erreichen und beantworten möchten. Durch seine wertvollen Feedbacks und Inputs aus der Sicht eines Endbenutzers im gesamten Verlauf der Arbeit ermöglichte er uns viele schwierige Entscheidungen, welche im Verlauf der Arbeit getroffen werden mussten, richtig zu treffen.

Ebenfalls möchten wir uns bei der ZHAW und dem Institut für angewandte Informationstechnologie bedanken, da sie uns durch die grosszügige Bereitstellung von mehreren Sonos™ Lautsprechern und Raspberry Pis die effiziente Entwicklung dieser Bachelorarbeit ermöglichten.

Inhaltsverzeichnis

1 Einleitung

1.1 Ausgangslage

Die Ausgangslage für diese Bachelorarbeit bildet die vorhergehende Projektarbeit Real-Stereo [1]. Die darin erarbeiteten Technologien in den Bereichen der Personenerkennung und -lokalisierung, sowie Stereo-Sound, werden in dieser Arbeit wiederaufgenommen und erneut untersucht, um Verbesserungen vorzunehmen.

Die Real-Stereo Software wurde in der vorhergehenden Arbeit als Desktop Software implementiert. In dieser Arbeit soll sie neu konzipiert und auf Embedded Geräte ausgelegt werden. Die Personenerkennung wird dabei auf mehrere Geräte verteilt, um sowohl ein modulares und erweiterbares System zu erstellen als auch aufwändige Berechnungen verteilen zu können.

1.2 Zielsetzung

Ziel dieser Bachelorarbeit ist die Entwicklung eines praxisreifen Hardware- und Software-Systems, welches eine Person innerhalb eines Raumes kontinuierlich und in Echtzeit lokalisiert. Mit den daraus resultierenden Positionsdaten werden die konfigurierten Sonos™ Lautsprecher balanciert, damit der Zuhörer unabhängig von seiner aktuellen Position alle Lautsprecher gleich laut wahrnimmt.

Damit der Benutzer das System selbstständig erweitern kann, soll das System modular aufgebaut und die einzelnen Teile möglichst kostengünstig erwerbbar sein. Mittels kabelloser Netzwerktechnologien soll die Installation durch den Benutzer ohne grossen Aufwand durchführbar sein und in mehreren Räumen gleichzeitig funktionieren.

Die offizielle Aufgabenstellung von Prof. Jürgen Spielberger ist im Anhang ersichtlich.

2 Theoretische Grundlagen

In diesem Kapitel werden die wesentlichen Grundlagen erläutert, welche für das bessere Verständnis der folgenden Arbeit vorausgesetzt werden. Die folgenden Themen wurden aufgrund ihrer Wichtigkeit für die Arbeit ausgewählt und bilden den Kern der verwendeten Technologien und Algorithmen.

2.1 Computer Vision

Computer Vision beschreibt die Disziplin, mit Programmen und Algorithmen ein Bild zu verarbeiten und analysieren, um darin beispielsweise Strukturen zu finden. In dem spezifischen Anwendungsfall dieser Arbeit werden in einer Reihe von Bildern Personen gesucht. Es gibt eine Vielzahl an Möglichkeiten, um dies zu erreichen. In den Kapiteln unter dem Abschnitt 3.4 wird genauer auf verschiedene Algorithmen eingegangen.

Während die Algorithmen die Aufgabe der Personenerkennung auf verschiedene Weise durchführen, bleibt das Format der Resultate gleich. Für jede erkannte Person wird eine Bounding Box zurückgegeben. Dies sind Koordinaten und Grössen eines Rechteckes, welches die erkannte Person vollständig umschliesst. Je nach verwendetem Algorithmus kann zusätzlich ein Konfidenzwert zwischen 0 und 1 zurückgegeben werden. Dieser Wert gibt an, wie sicher sich der Algorithmus beim zurückgegebenen Resultat ist. Damit ist es möglich, uneindeutige Resultate zu ignorieren.

Für die Umsetzung wird auf die Open-Source Library *OpenCV* gesetzt. Diese stellt nicht nur Funktionen für die Bearbeitung, Manipulation und Erstellung von Bildern zur Verfügung, sondern implementiert auch eine Vielzahl an Algorithmen aus dem Bereich der Computer Vision. Einige der Algorithmen sind spezifisch auf Personen- und Objekterkennung ausgelegt, andere können allgemeine Strukturen oder Konturen in Bildern erkennen. Beide dieser Arten werden in dieser Arbeit verwendet.

2.2 Neuronale Netze

Ein Teilgebiet der Künstlichen Intelligenz sind Neuronale Netze. Diese sind bezüglich des Aufbaues an biologische Neuronen angelehnt. Es werden Input-Informationen durch verschiedene Schichten von Neuronen verarbeitet, wobei sich jedes Neuron auf eine einzelne Aufgabe fokussiert. Diese sind miteinander vernetzt und treffen am Schluss eine gemeinsame Entscheidung.

Eine Anwendungsmöglichkeit von neuronalen Netzen befindet sich in der Bildverarbeitung. So basiert beispielsweise der in dieser Arbeit verwendete Personenerkennungs-Algorithmus *YOLOv3* [2] darauf. Dessen Neuronen untersuchen verschiedene Features in einem Bild und können, durch lokale Eigenschaften und anliegende Eigenschaften, Objekte und Personen erkennen.

2.3 Raspberry Pi

Der Raspberry Pi ist ein kompakter Einplatinecomputer, weit verbreitet, und eignet sich für den Gebrauch in einem Embedded-Umfeld. Aufgrund seiner Beliebtheit gibt es etliche Literatur, Projekte, Softwarelibraries und Hardware-Module. Das Modell wird stetig weiterentwickelt und ist aktuell in der 4. Generation. Dieses wird ebenfalls für Real-Stereo verwendet und besitzt wahlweise 2 bis 8 GB RAM, einen Quad-Core-Prozessor und diverse Anschlüsse für Peripheriegeräte und Erweiterungen.

2.4 Interpolation

Für die Real-Stereo Software ist es wichtig, die wahrgenommene Lautstärke im ganzen Raum zu kennen. Während der Kalibration werden jedoch nur Lautstärken von einzelnen Punkten im Raum gemessen. Für die restlichen Bereiche des Raumes wird die Lautstärke anhand der umliegenden bekannten Punkte interpoliert.

Weil es sich um einen zweidimensionalen Raum handelt und jede Koordinate aus einer x- und y-Position besteht, muss die verwendete Interpolationsart multivariat sein und somit mehr als nur eine Unbekannte unterstützen. Spezifisch wurde für die Interpolation *Shepard's Methode für verstreute Daten* verwendet, welche für eine örtliche Interpolation entwickelt wurde und durch einen Parameter an verschiedene Anwendungsfälle angepasst werden kann.

2.5 Schnelle Fourier-Transformation

Die schnelle Fourier-Transformation ist ein Algorithmus, um die Frequenz-Dimension eines Signals zu berechnen, welches über einen gewissen Zeitraum stattfindet. Im Kontext dieser Bachelorarbeit wird die schnelle Fourier-Transformation verwendet, um die gemessene Lautstärke für alle Frequenzbereiche einzeln zu bestimmen. Die gemessenen Lautstärken werden daraufhin verwendet, um eine wahrgenommene Lautstärke zu bestimmen, welche für die Interpolation verwendet wird.

3 Methoden

Im Folgenden werden die Grundüberlegungen und die Details der Realisierung aller Komponenten beschrieben, aus welchen die Real-Stereo Software besteht.

3.1 Embedded Plattform

Als embedded Plattform für Real-Stereo wurde der Raspberry Pi 4 Model B [3] gewählt. Die Raspberry Pi Plattform eignet sich für den Einsatz aufgrund der weiten Verbreitung, und damit breiter Unterstützung von Libraries. Auch der günstige Preis, die direkte Unterstützung von Kamera-Modulen und Wireless LAN sind für das Real-Stereo System benötigte Eigenschaften. Als Alternative wurde die Nvidia Jetson [4] Plattform in Betracht gezogen. Dies aufgrund der ausgezeichneten Leistung im KI-Bereich, was als grosser Leistungssprung bei der Personenerkennung resultieren könnte. Die Raspberry Pi Plattform wurde schlussendlich ausgewählt, da die Nvidia Jetson Plattformen deutlich mehr Kosten verursacht und mit mehr Konfigurationszeit vom Ziel der Arbeit abgelenkt hätte.

Das entwickelte System benötigt mindestens zwei Raspberry Pis mit angeschlossenem Kamera-Modul mit Weitwinkelobjektiv, kann aber jederzeit durch weitere Raspberry Pis erweitert werden. Dabei agiert ein Raspberry Pi im Real-Stereo System entweder als *Coordinator Node* oder als *Tracking Node*.

Durch die Verteilung der Personenerkennung auf mehrere Tracking Nodes, und somit auf mehrere Raspberry Pis, kann eine höhere Erkennungsgeschwindigkeit mit gleichzeitig verbesserter Genauigkeit erreicht werden.

3.1.1 Coordinator Node

Ein einzelner Raspberry Pi übernimmt die Koordinationsrolle für das gesamte Real-Stereo System und wird somit zur Coordinator Node. Die Coordinator Node ist für die folgenden Aufgaben zuständig:

1. Bereitstellung der Webapp durch einen Python Webserver, welcher mittels der populären AIOHTTP [5] Programmbibliothek implementiert wurde. Diese basiert auf dem *asyncio* Modul von Python und erlaubt es, Requests asynchron zu beantworten.
2. Bereitstellung der entwickelten API, welche von der Webapp verwendet wird, um das System zu steuern und zu kontrollieren.
3. Bereitstellung der Testton-MP3-Datei, welche von den Sonos™ Lautsprechern bei der Raumkalibrierung und im Testmodus wiedergegeben wird.
4. Weiterleitung der Kamera-Livestreams der anderen Tracking Nodes, falls der Benutzer die Kamera-Vorschau verwendet.

5. Automatische Entdeckung von neuen Tracking Nodes, welche zum System hinzugefügt werden.
6. Steuerung aller Tracking Nodes im System. Dazu gehört das Starten und Stoppen der Personenerkennung, sowie die Entgegennahme der Positionsdaten, welche von den Tracking Nodes berechnet wurden.
7. Automatische Entdeckung von Sonos™ Lautsprechern im lokalen Netzwerk.
8. Steuerung der ausgewählten Sonos™ Lautsprechern. Dazu gehört das Verwalten der Gruppierung der Lautsprecher sowie die Steuerung der Laustärke einzelner Sonos™ Lautsprecher und ihr Wiedergabestatus.

Die Coordinator Node agiert gleichzeitig auch als Tracking Node, wodurch kein separater Raspberry Pi für die Koordination und Steuerung des Systems eingesetzt werden muss.

3.1.2 Tracking Node

Der Endbenutzer kann jederzeit zusätzliche Raspberry Pis zum Real-Stereo System hinzufügen. Diese melden sich mittels des in Abschnitt 3.8 beschriebenen Netzwerk Protokolls bei der Coordinator Node des Systems und werden als Tracking Node aufgenommen. Alle Tracking Nodes werden von der Coordinator Node gesteuert und melden die gesammelten Daten und berechneten Resultate, wie zum Beispiel Positionsdaten, an die Coordinator Node.

3.2 Kameras

Die vorangegangene Projektarbeit [1] zeigte auf, dass die darin verwendeten USB-Webcams für die Personenerkennung in einem Raum nicht geeignet waren. Aufgrund eines kleinen Blickwinkels konnten grosse Bereiche der Räume nicht erfasst werden. Um dieses Problem zu lösen, sollen Weitwinkel-Kameras verwendet werden.

3.2.1 Kameratypen

Es stehen mehrere Möglichkeiten zur Auswahl, wie die Kameras mit dem Raspberry Pi verbunden werden.

Bei Kameras, welche per USB angeschlossen werden, gibt es eine grosse Auswahl. Diese sind jeweils in sich abgeschlossene Geräte und besitzen ein eigenes Gehäuse. Folglich brauchen sie mehr Platz und können nicht einfach in kompakte Embedded Geräte integriert werden.

Abhilfe schaffen Kamera-Module, welche nur aus dem Kamera-Chip und der Linse bestehen und sich somit in das Raspberry Pi Board integrieren lassen. Die Verbindung läuft anstatt über USB über das Camera Serial Interface, von welchem jeder Raspberry Pi genau eines besitzt. Der Vorteil davon ist, dass das Kamera-Modul über diese

Schnittstelle direkt mit der GPU verbunden ist. In diesem Fall wird die CPU für das Verarbeiten der Kameradaten nicht benötigt, wie es bei USB-Kameras der Fall ist. Das Resultat ist eine bessere Performance, weil die CPU nicht von der Kamera blockiert wird und für andere Berechnungen verwendet werden kann.

Alternativ standen IP-Kameras zur Diskussion, welche das Kamera-Bild per HTTP- oder RTMP-Protokoll im lokalen Netzwerk zur Verfügung stellen. Die Raspberry Pis hätten somit an einem beliebigen Ort im Netzwerk installiert werden können. Jedoch sind die IP-Kameras nicht markant kleiner als ein Raspberry Pi und benötigen gleich viele Anschlüsse wie ein Raspberry Pi. Die Installation ist somit nicht einfacher und zudem befinden sich IP-Kameras in einem höheren Preissegment. Aus diesen Gründen wurde gegen IP-Kameras und für Kamera-Module entschieden.

Zum Zeitpunkt der Bestellung war nur ein Weitwinkel-Kamera-Modul in der Schweiz verfügbar, welches mit der aktuellen Generation des Raspberry Pi kompatibel ist. Das «RPi Camera (I), Fisheye Lens» Modul von Waveshare erfüllt mit einem Blickwinkel von 170° [6] die Anforderungen und ermöglicht eine beinahe vollständige Abdeckung der Räume.

3.2.2 Kamera Kalibration

Aufgrund des grossen Blickwinkels von 170° wird das Bild vom Kamera-Modul verzerrt wiedergegeben. Damit die Personenerkennungs-Algorithmen trotzdem zuverlässig funktionieren können, ist eine Entzerrung des Kamerabildes erforderlich.

Um Bilder entzerren zu können, müssen einige Parameter bekannt sein. Darunter gehören die Verzerrungskoeffizienten, Rotations- und Verschiebungsvektoren. Diese spezifizieren, um welche Verzerrung es sich handelt. Bei radialen Verzerrungen, die bei Weitwinkelobjektiven auftreten, werden gerade Linien, und somit das ganze Bild, gekrümmt. Tangentiale Verzerrungen sind das Resultat von Perspektiven, wenn die Kamera nicht parallel auf ein Objekt gerichtet ist. Häufig ist die Verzerrung eine Kombination einer radialen und tangentialen Verzerrung.

Zusätzlich spielen auch die intrinsischen und extrinsischen Parameter der Kamera eine Rolle. Diese sind in der folgenden Kamera-Matrix der Gleichung 1 angegeben, wobei f für die Brennweiten und c für die optischen Zentren stehen.

$$cameraMatrix = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \quad (1)$$

Die Bestimmung der Parameter von Hand ist umständlich und variiert pro Kamera, weshalb dieser Prozess automatisiert wurde. Weil verzerrte Bilder im Bereich der Computer Vision keine Seltenheit sind, bietet die für diesen Teil der Applikation verwendete Library *OpenCV* bereits einige Hilfsfunktionen. Dafür wird ein Bild der

Kamera gebraucht, in welchem ein verzerrtes Objekt sichtbar ist. Es muss ausserdem bekannt sein, wie dieses Objekt in einem nicht verzerrten Zustand aussehen würde. Für diese Anwendung eignet sich deshalb ein Schachbrettmuster, weil es eine Vielzahl an geraden Linien hat und der Aufbau bekannt ist. Ausserdem sind, aufgrund des quadratischen Musters, innerhalb des Schachbrettes weitere Punkte bekannt, was zu einer genaueren Berechnung führt.

OpenCV stellt die Funktion *cv.findChessboardCorners()* bereit, um in einem gegebenen Bild ein Schachbrett der angegebenen Anzahl Quadrate zu finden. In der folgenden Abbildung 1 wird das Ergebnis dieser Funktion visuell aufgezeigt.

Abbildung 1 Verzerrtes Kamerabild mit gefundenen Punkten im Schachbrettmuster.

Sind die Punkte bekannt, kann das Ergebnis mit *cv.cornerSubPix()* verbessert werden. Diese Hilfsfunktion bildet die zuvor ungefähr gefundenen Punkte pixelgenau auf die tatsächlichen Ecken der Quadrate ab. Der Rückgabewert dieser Funktion sind somit die Ecken der einzelnen Quadrate im zweidimensionalen Raum der Kamera des Schachbrettmuster-Objektes. Damit die benötigten Parameter berechnet werden können, werden zusätzlich die Koordinaten dieses Objektes im realen dreidimensionalen Raum benötigt. Der Ursprung der Koordinaten und die Skalierung kann selbst gewählt werden. Der Einfachheit halber wird der Ursprung unten links auf das Schachbrett gelegt und einem Quadrat die Länge 1 gegeben. Ein 8x8 Schachbrett reicht demnach von $(0, 0, 0)$ bis zu $(8, 8, 0)$.

Um das Ergebnis weiter zu verbessern, kann der Vorgang mehrere Male wiederholt werden, wobei das Schachbrettmuster aus verschiedenen Winkeln und an verschiedenen Positionen in die Kamera gezeigt werden soll.

All diese gesammelten Informationen werden nun der *cv.calibrateCamera()* Funktion übergeben, welche die Kamera-Matrix, Verzerrungskoeffizienten, Rotations- und Verschiebungsvektoren berechnet. Diese Parameter werden nun für die Kamera abgespeichert, damit sie nicht erneut berechnet werden müssen.

Um ein Bild nun zu entzerren, kann dieses, zusammen mit allen zuvor gespeicherten Parametern, der Funktion *cv.undistort()* übergeben werden. Der Vergleich in Abbildung 2 zeigt das Resultat der Entzerrung. Weil in den Ecken Bildpunkte fehlen, werden Teile der Ränder beim entzerrten Bild nicht dargestellt.

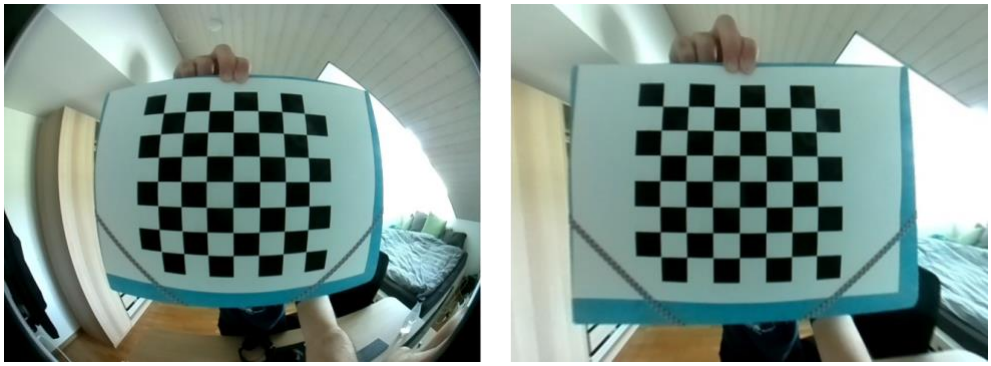


Abbildung 2 Vergleich des verzerrten Bildes (links) und dem Resultat der Entzerrung (rechts).

3.3 Ad Hoc Netzwerk

Möchte ein Benutzer ein Real-Stereo Gerät in Betrieb nehmen, stellt sich das Problem, dass sich dieses zuerst zu einem Netzwerk verbinden muss, damit es sowohl Zugriff auf die Sonos™ Lautsprecher bekommt, als auch mit anderen Real-Stereo Geräten kommunizieren kann. Zusätzlich kann das neue Gerät auch nicht über das geplante Web-Interface angesprochen werden, da es noch nicht Teil des Netzwerkes ist und somit keine erreichbare IP-Adresse besitzt.

Weil es umständlich ist, den Raspberry Pi nur für den Setup an einen Bildschirm und einer Tastatur anschliessen zu müssen und dieser zusätzlich kein grafisches User Interface besitzt, wird eine WLAN-Konfiguration implementiert, welche bereits in einigen anderen Smart Home Geräten ähnlich umgesetzt ist. Für diesen Vorgang agiert der Raspberry Pi als Access Point und startet ein eigenes Ad Hoc Netzwerk. Für die Konfiguration kann sich der Benutzer in dieses neue öffentliche WLAN verbinden und den Raspberry Pi unter der statischen IP-Adresse 10.1.1.1 erreichen. Der Raspberry Pi bietet eine Webseite an, auf welcher der Benutzer die Anmeldeinformationen des eigentlichen WLANs einrichten kann. Wurden die Verbindungsinformationen des richtigen Netzwerkes gespeichert, wird das separate Ad Hoc Netzwerk wieder deaktiviert und der Raspberry Pi verbindet mit dem eigentlichen WLAN.

Der Vorgang der WLAN-Konfiguration kann durch zwei unterschiedliche Ereignisse automatisch gestartet werden. Entweder, wenn ein Real-Stereo Gerät zum ersten Mal gestartet wird, oder wenn nach dem Start eines Gerätes nach 30 Sekunden keine Verbindung zu einem konfigurierten WLAN hergestellt werden konnte. Letzteres wird im Aktivitätsdiagramm der Abbildung 3 beschrieben und erlaubt das erneute

Konfigurieren nach einer Änderung am WLAN-Passwort oder der SSID oder bei einer Installation in einer neuen Umgebung, in dem sich noch kein bekanntes Netzwerk befindet.

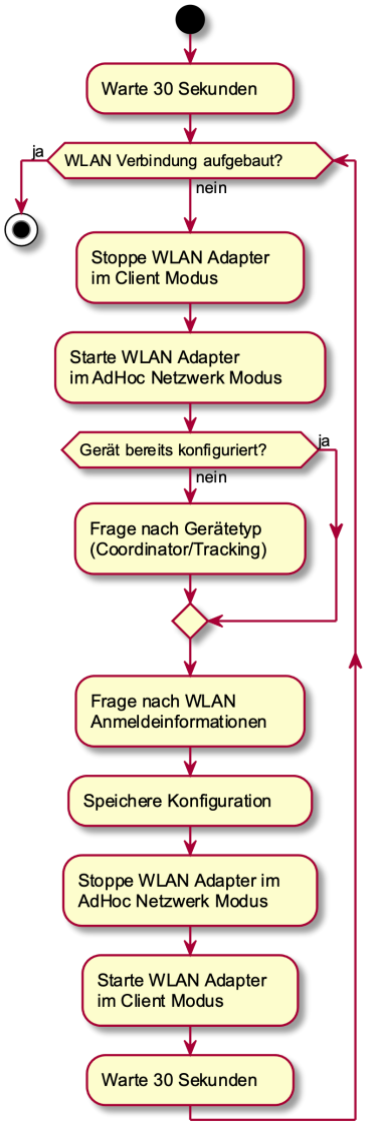


Abbildung 3 Ablauf der WLAN-Konfiguration über das Ad Hoc Netzwerk.

Um sich bei einem Netzwerk anzumelden, benutzt das Betriebssystem von Raspberry Pi *wpa_supplicant*. Dieses Modul kommuniziert mit dem WLAN-Treiber und implementiert die Authentifizierung und Key-Negotiation über *Wi-Fi Protected Access* (WPA) [7]. Die Anmeldeinformationen werden dafür aus der */etc/wpa_supplicant/wpa_supplicant.conf* Datei ausgelesen. Ein Beispiel dieser Konfiguration wird in Abbildung 4 gezeigt.

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
country=CH

network={
  ssid="MyHomeNetwork"
  psk=bf1334f57165cc60977e090216bee4eb74fe4fcbc5acb6b31c52ad8fb46830fa
}
```

Abbildung 4 Beispiel einer wpa_supplicant.conf Datei.

Die Informationen eines einzelnen Netzwerkes werden in einem *network*-Block gespeichert. Es ist möglich, mehrere dieser Blöcke in einer Datei zu definieren, was es erlaubt, die Verbindungsinformationen von mehreren WLANs zu speichern. Der Raspberry Pi beziehungsweise das wpa_supplicant Modul verbindet in diesem Fall zum nächstgelegenen Netzwerk. Wie im Beispiel ersichtlich, wird das Passwort nicht im Klartext gespeichert, sondern in einer 32-Byte verschlüsselten hexadezimalen Abbildung [8]. Um ein Passwort zu verschlüsseln, wird das Kommandozeilentool *wpa_passphrase* zur Verfügung gestellt und vom Python Code per *system*-Aufruf ausgeführt. Besitzt ein WLAN kein Passwort und ist somit öffentlich, wird die *psk*-Zeile durch *key_mgmt=None* ersetzt.

3.4 Personenerkennung

Eine weitere Verbesserungsmöglichkeit, welche aus der Projektarbeit [1] hervorgegangen war, ist die Personenerkennung. Das darin verwendete Verfahren, *Histogram of oriented gradients*, erzielte zwar in den Räumen, die während der Entwicklung verwendet wurden, gute Resultate. Jedoch funktionierte die Erkennung in anderen Testräumen nicht optimal und mit den verfügbaren Parametern konnte keine Einstellung gefunden werden, die für alle möglichen Räume optimale Resultate liefert.

Folglich wurden Alternativen evaluiert und drei verschiedene Algorithmen implementiert. Jeder Algorithmus ist dabei für andere Verhältnisse ausgelegt und der Benutzer hat pro Kamera die Wahl, welcher verwendet werden soll. Somit soll es in jedem Raum und unterschiedlichen Verhältnissen möglich sein, die Personen zuverlässig erkennen zu können.

3.4.1 YOLOv3 Real-Time Object Detection

You only look once (YOLOv3) ist ein Objekterkennungsalgorithmus und basiert auf einem neuronalen Netz, welches als Ganzes auf das vollständige Bild angewendet wird [2]. Durch dieses Verfahren ist der Algorithmus performanter als andere, welche mehrere neuronale Netzwerke verwenden. Auch Algorithmen, welche das zu

untersuchende Bild aufteilen und das neuronale Netz mehrfach auf einzelne Teile anwenden, erreichen eine schlechtere Performanz als der YOLOv3 Algorithmus. Ausserdem wird der Kontext des gesamten Bildes für die Beurteilung berücksichtigt und es werden nicht nur Personen, sondern 80 verschiedene Objekte klassifiziert.

3.4.2 Motion Detection

Dieses Verfahren wurde eigens mit den Hilfsfunktionen von OpenCV entwickelt und basiert auf der Differenz von zwei aufeinanderfolgenden Bildern. Dadurch sollen Personen durch ihre natürlichen Bewegungen erkannt werden. Kann keine Bewegung erkannt werden, wird angenommen, dass sich die Person nicht bewegt hat und sich somit noch bei der letzten bekannten Position befindet. Es wird jedoch nicht geprüft, ob es sich bei der Bewegung tatsächlich um eine Person handelt, weshalb es auch möglich ist, dass beispielsweise Haustiere oder Autos durch Fenster erkannt werden. Der Vorteil besteht darin, dass es in Situationen funktioniert, in denen Objekterkennungsalgorithmen Personen nicht zuverlässig erkennen können. Solche Situationen können sehr dunkle Räume sein oder Räume mit Objekten, welche die Personen fast vollständig verdecken. Ausserdem ist dies das performanteste Verfahren, weil es auf vergleichsweise einfachen Bildoperationen basiert.

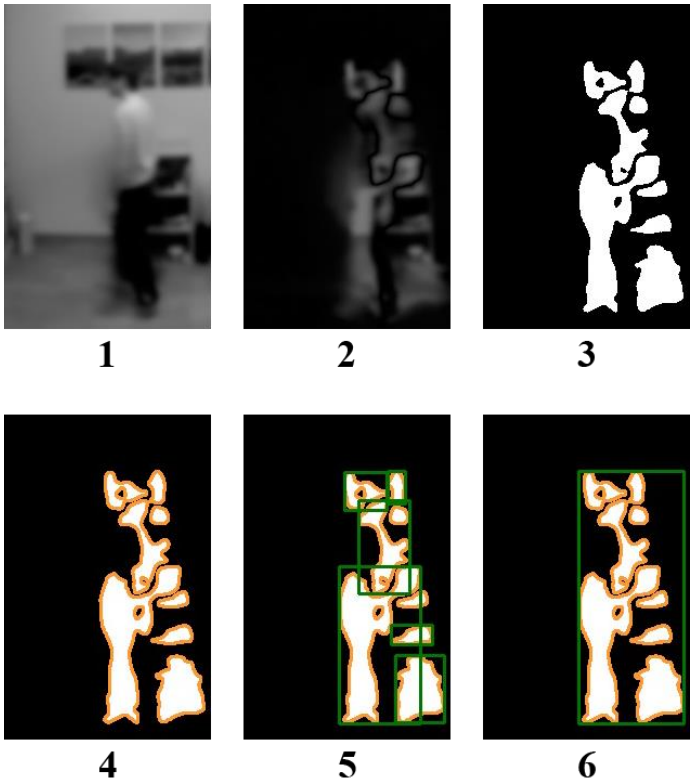


Abbildung 5 Einzelne Schritte des Motion Detection Algorithmus anhand eines Beispiels.

In Abbildung 5 werden die einzelnen Schritte des Verfahrens aufgezeigt. Als Erstes wird das Bild in ein Graustufenbild umgewandelt und darauf ein Unschärfefilter angewendet, damit Rauschen im Bild und andere kleine Pixelunterschiede die weiteren Schritte nicht beeinträchtigen. Das Resultat wird mit dem des vorherigen Frames subtrahiert und auf die Betragsfunktion angewendet. Hat sich ein Pixel nicht verändert, besitzt er den Wert 0 und ist somit schwarz. Je grösser die Veränderung ist, desto weisser ist er. Der dritte Schritt besteht darin, ein Thresholding anzuwenden, um ein Binärbild zu erhalten. Grauwerte unter dem definierten Threshold werden schwarz, die restlichen weiss. Damit werden Pixel ignoriert, welche eine zu kleine Differenz zum vorherigen Bild haben. In dem Binärbild werden in Schritt Vier die Konturen von allen weissen Flächen gesucht und von der OpenCV Funktion *cv2.findContours()* als Pfade zurückgegeben. Der fünfte Schritt besteht daraus, für jeden Pfad, der eine Mindestlänge erreicht und somit eine genug grosse Änderung darstellt, ein vollumfassendes Rechteck zu generieren. Dies wurde mit der Funktion *cv2.boundingRect()* erreicht. Abschliessend werden überlappende und nahe gelegene Rechtecke zu einem grossen zusammengefasst, welches die gesuchte Person darstellt.

3.4.3 Histogram of oriented gradients

Als Letztes verfügbares Verfahren wurde *Histogram of oriented gradients*, welches bereits bei der Projektarbeit [1] verwendet wurde, implementiert. In diesem Algorithmus wird das Bild in mehrere kleine Quadrate aufgeteilt. Innerhalb dieser Regionen werden Gradienten, sprich Kanten und Ecken, extrahiert. Anhand dieser werden nun lokale Eigenschaften von Personenumrissen gesucht, wofür ein durch maschinelles Lernen trainiertes Modell benutzt wird.

3.5 Lokalisierung

Pro Tracking Node kann eines der oben genannten Verfahren ausgewählt werden. Alle der drei Verfahren geben ein Rechteck, welches die erkannte Person umschliesst, zurück. Hat das System diese Information von zwei Kameras im gleichen Raum, ist es nun möglich, die Person innerhalb des Raumes zu lokalisieren. Für das bestmögliche Ergebnis sind die zwei Raspberry Pis in einem 90° Winkel zueinander aufgestellt, wie in Abbildung 6 dargestellt. Dies ist jedoch nicht immer möglich und solange sich der Winkel zwischen 45° und 135° befindet, sind die Perspektiven der zwei Kameras genug unterschiedlich, um eine Person präzise lokalisieren zu können.

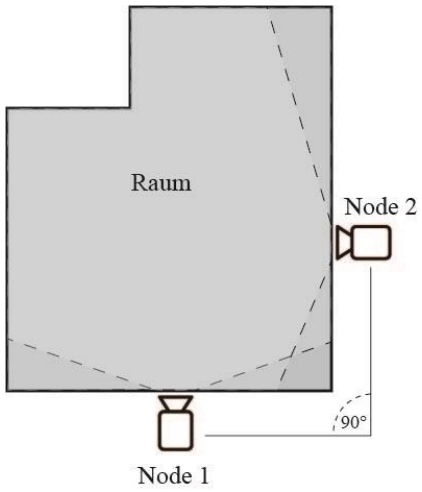


Abbildung 6 Anordnung von zwei Nodes in einem Raum. Quelle: Angepasst von [1].

Jede Node ist für die Berechnung einer Koordinate zuständig. Zusammen können diese somit die X- und Y-Koordinate einer Person im zweidimensionalen Abbild des Raumes berechnen.

$$coordinate = rect_x + \frac{rect_{width}}{2} \tag{2}$$

Wie aus der obigen Gleichung 2 zur Berechnung einer Koordinate anhand der erkannten Person (*rect*) zu erkennen ist, wird nur die Position auf der horizontalen Achse von jedem Bild berücksichtigt. Weil der Raum zweidimensional abgebildet ist, kann die vertikale Achse ignoriert werden.

3.5.1 Personentracking

Personen werden über mehrere Frames hinweg getracked. Es wird demnach analysiert, ob im letzten Frame an ungefähr der gleichen Position bereits eine Person erkannt wurde. Ist dies der Fall, wird auch die Person im aktuellen Bild bestätigt. Kann die Person nicht bestätigt werden, wird sie für die Berechnung der Koordinate nicht berücksichtigt. Erst wenn eine unbestätigte Person über drei Frames hinweg erkannt wird, wird diese Person als bestätigt angesehen und für die Berechnungen berücksichtigt.

Dieses Vorgehen hat den Vorteil, dass False-Positives minimiert werden. Je nach verwendetem Verfahren können wegen Rauschen, Lichtreflexionen oder anderen Umständen, fälschlicherweise Personen an Orten erkannt werden, an denen es keine gibt. Diese werden oft nur in einem einzelnen Frame erkannt. Dadurch, dass geprüft wird, ob es ungefähr am gleichen Ort in den vergangenen Frames ebenfalls bereits eine Person gab, kann dieses Phänomen reduziert werden.

Um auch Bewegungen zu erlauben, muss die Person nicht exakt am gleichen Ort stehen, sondern kann sich um eine definierte Anzahl Pixel bewegen. Diese wurde mit 50 Pixel in beide Richtungen, bei einem Bild der Grösse von 640 Pixel, grosszügig gewählt, damit die Funktionalität nicht eingeschränkt wird. Zusätzlich wird die aktuelle Position nicht gleich mit allen vorherigen Frames verglichen, sondern rekursiv durchgeführt. Wurde im vorherigen Frame eine Person gefunden, wird dann diese vorherige Position wiederum mit dem Frame davor geprüft.

Als Nebeneffekt können die vorherigen Positionen auch verwendet werden, wenn in einem Frame keine Person erkannt werden konnte. Trifft dies ein, wird angenommen, dass sich die Person nicht bewegt hat, und es wird die letzte bekannte Position zurückgegeben. Gleiches wird auch angewendet, wenn der Benutzer den Raum verlässt. Die zuletzt bekannte Position ist demnach an der Türe oder einem Bildrand. Bis die Person den Raum wieder betritt, wird die letzte bekannte Position verwendet.

3.5.2 Mehrere Personen

Die vorangegangenen Kapitel nehmen an, dass sich zu jedem Zeitpunkt höchstens eine Person im Raum befindet. Dies ist im Alltag jedoch kein realistisches Szenario. Aus diesem Grund wurden verschiedene Methoden gesucht, wie sich das System in diesem Fall verhalten muss. Wie bei der Implementation der Personenerkennungs-Algorithmen, hat auch hier der Benutzer die Wahl, welche Methode verwendet wird. Diese sind jeweils für unterschiedliche Raumverhältnisse ausgelegt. Bei all den verfügbaren Methoden werden die Positionen von mehreren Personen auf eine einzige Koordinate heruntergebrochen. Das führt dazu, dass bei den weiteren Schritten keine weiteren Unterscheidungen gemacht werden müssen, wenn sich mehr als eine Person im Raum befindet.

Die **erste Möglichkeit**, welche auch als Standard definiert wurde, ist die *Average* Methode. Sie ist gut geeignet für Räume, in denen sich häufig mehrere Personen aufhalten, wie beispielsweise einem Wohnzimmer.

$$groupedCoordinate = \frac{\sum_{i=1}^{N} coordinate_i}{N} \tag{3}$$

Die Berechnung ist in obiger Gleichung 3 ersichtlich. Dabei steht N für die Anzahl erkannten Personen und $coordinate_i$ für die Koordinate der Person i. Das Resultat ist der Durchschnitt aller Koordinaten und somit der Personen im Raum.

Die **zweite Möglichkeit** basiert auf der zuletzt bekannten Position einer Person. Es wird jeweils die Koordinate zurückgegeben, welche sich am nächsten an der zuletzt zurückgegebenen Position befindet, wie aus der folgenden Gleichung 4 zu entnehmen ist. Alle anderen erkannten Personen werden demnach ignoriert. In der Gleichung steht C für die Menge aller Koordinaten der neu erkannten Personen und $prev$ für die zuletzt

bekannte Koordinate. Sollten mehrere Koordinaten exakt die gleiche Entfernung zur vorherigen haben, wird nur die erste verwendet. Diese zweite Methode eignet sich gut für Räume, in denen sich meistens nur eine Person befindet, wie etwa einem Arbeitszimmer. Kommt zwischenzeitlich kurz eine zweite Person hinzu, verändert sich das Hörgefühl für die Hauptperson nicht.

$$groupedCoordinate = \{c | c \forall x \in C : |prev - c| \leq |prev - x|\} \quad (4)$$

Abschliessend muss zu diesem Thema gesagt werden, dass das Real-Stereo System hauptsächlich auf eine Person ausgelegt ist. Sind mehrere Personen im Raum, ist es nicht möglich, alle Lautsprecher so zu balancieren, dass die Lautstärke für alle Personen gleich laut erscheint. Trotzdem wird mit diesen zusätzlichen Methoden sichergestellt, dass die Applikation auch in diesem Fall funktioniert und den Umständen entsprechend die bestmöglichen Resultate erzielt. Der Benutzer hat dabei die Möglichkeit, diese mit der Wahl der geeigneten Methode zu beeinflussen.

3.6 Raum Kalibrierung

Um aus der aktuellen Position des Zuhörers die korrekten Lautstärken berechnen zu können, muss jeder Raum, in dem das System eingesetzt werden soll, kalibriert werden. Ein Testgeräusch wird separat auf jedem Lautsprecher im Raum wiedergegeben und die wahrgenommenen Lautstärken dieses Testgeräusches an mehreren Positionen im Raum gemessen. Dadurch kann berechnet werden, wie jeder Lautsprecher eingestellt werden muss, damit die Lautstärke an verschiedenen Punkten im Raum gleich laut wahrgenommen wird.

Diese Kalibration wird anschliessend für den Raum abgespeichert und kann jederzeit durch den Benutzer überschrieben werden. Eine erneute Konfiguration ist zum Beispiel notwendig, wenn die Position oder die Anzahl der Lautsprecher verändert wurde.

3.6.1 Testgeräusch

Bei der vorhergehenden Projektarbeit Real-Stereo [1] wurde ein 2000hz Sinuston als Testgeräusch verwendet. Der Sinuston sollte in der Theorie einfach, und mithilfe von Frequenzfiltern, genau gemessen werden können. Dieses Testgeräusch stellte sich in der Praxis jedoch als schlechte Wahl heraus. Die daraus entstandenen Messwerte waren unstabil. Höchstwahrscheinlich, weil sich der Sinuston durch Reflektionen an Oberflächen im Raum teilweise selbst überlagerte. Damit wurde die gemessene Lautstärke, und somit die Messwerte, verfälscht.

Als Verbesserung für diese Arbeit wurden verschiedene Formen von zufälligem Rauschen auf Eignung analysiert, mit der Überlegung, dass das breite Frequenzspektrum von Rauschen nicht dieselben Probleme wie die vorherige Lösung besitzt.

Das als Erstes untersuchte sogenannte *graues Rauschen* besitzt eine Frequenzkurve, welche eine uniforme Lautstärke für das menschliche Gehör zu produzieren versucht. Dabei werden sehr tiefe und sehr hohe Frequenzen lauter wiedergegeben, um der niedrigeren Sensitivität des menschlichen Gehörs in diesen Bereichen entgegenzuwirken [9].

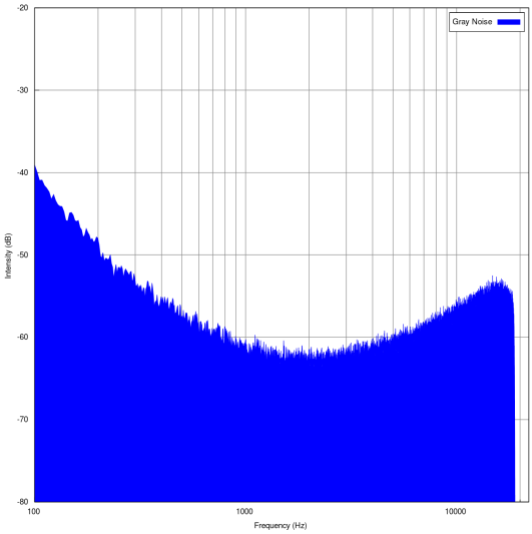


Abbildung 7 Frequenzkurve graues Rauschen.

Weisses Rauschen besitzt im Kontrast zu grauem Rauschen eine flache Frequenzkurve, welche dadurch auf dem gesamten Spektrum, und somit für jede Frequenz, gleich viel Lautstärke produziert [10]. Die Unterschiede der zwei Rauscharten ist deutlich in den Frequenzkurven in Abbildung 7 und Abbildung 8 zu sehen.

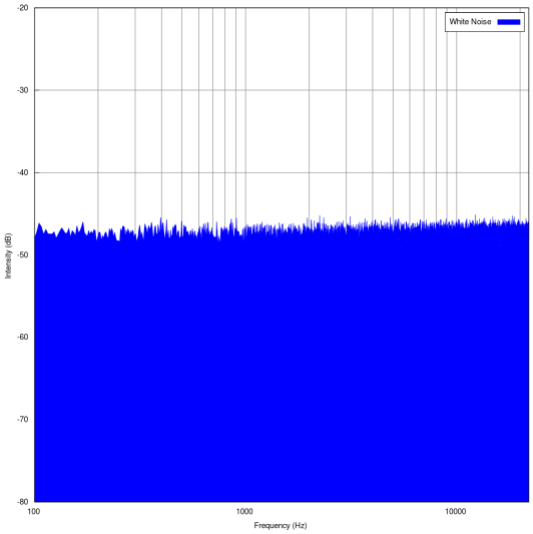


Abbildung 8 Frequenzkurve weisses Rauschen.

Da Lautsprecher aber niemals eine perfekte Wiedergabe in allen Frequenzbereichen leisten können, wurde weisses Rauschen für das Testgeräusch gewählt und die Anpassungen für das menschliche Gehör stattdessen bei der Berechnung der Lautstärke miteinbezogen.

3.6.2 Messung wahrgenommener Lautstärke Testgeräusch

Die Messung des Testgeräusches soll mittels der Webapp während der Raumkalibration durchgeführt werden. Dazu wird die *AudioContext WEB API* [11] verwendet, um über den Browser auf das Mikrofon des vom Benutzer eingesetzten Gerätes zuzugreifen. Mit der API kann eine Audio-Pipeline aufgebaut werden und das Audiosignal des Mikrofons an weitere Komponenten weitergegeben werden. Für die Messung der wahrgenommenen Lautstärke wird nur eine *AnalyserNode* [12] in die Audio-Pipeline eingefügt. Die AnalyserNode führt eine schnelle Fourier-Transformation durch und erlaubt somit den Zugriff auf die Frequenz-Dimension des vom Mikrofon aufgenommenen Audiosignals.

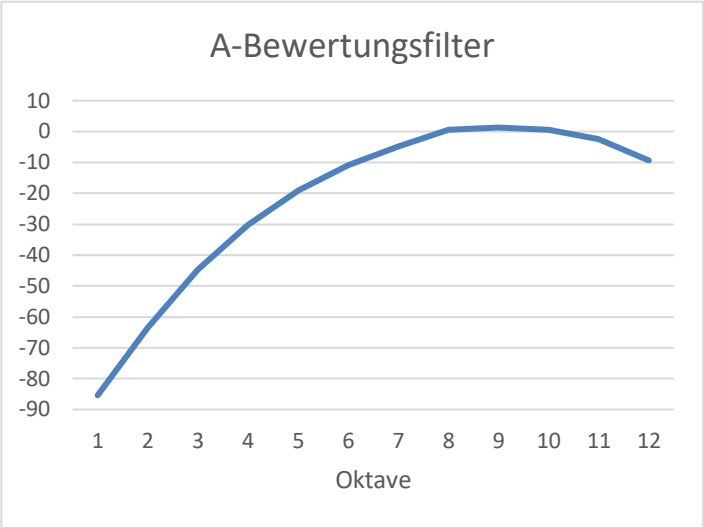
Bei der Messung muss beachtet werden, dass das menschliche Gehör verschiedene Frequenzbereiche bei gleichbleibender Schallleistung unterschiedlich laut wahrnimmt. Aus diesem Grund werden die Schallleistungen der verschiedenen Frequenzen pro Oktave zusammenaddiert. Die Frequenzen, welche zu einer Oktave gehören, steigen exponentiell mit der Nummer der Oktave an [13]. Um die von einem Menschen wahrgenommene Lautstärke, auch Lautheit genannt, einer Oktave zu berechnen, werden diese mit einem A-Bewertungsfilter [14] und der folgenden Gleichung 5 bewertet:

$$L = E_{Oktave} * 10^{\frac{ABewertungsfilter_{Oktave}}{10}} \quad (5)$$

Da diese Berechnungen auf dem Smartphone des Benutzers und 20-mal pro Sekunde durchgeführt werden, wurde die Oktavenanzahl aus Performancengründen auf 12 beschränkt und der in Tabelle 1 beschriebene A-Bewertungsfilter mit 12 Einträgen verwendet.

Tabelle 1 A-Bewertungsfilter.

Oktave	Gewicht
1	-85.4
2	-63.6
3	-44.8
4	-30.3
5	-19.1
6	-10.8
7	-4.8
8	0.6
9	1.3
10	0.6
11	-2.5
12	-9.3



Um die durchschnittliche Lautheit des Testgeräusches über einen Messzeitraum zu bestimmen, wird der Median aller berechneter Lautheitswerten gewählt. Dadurch wird verhindert, dass Ausreisser das Resultat verfälschen.

3.7 Balancing

Durch die Kalibration ist die aufgenommene Lautstärke an einzelnen Punkten im Raum bekannt. Um eine freie Bewegung und ein stufenloses Balancing im ganzen Raum zu ermöglichen, muss die wahrgenommene Lautstärke zwischen den Kalibrationspunkten berechnet werden. Aufgrund der guten Resultate in diesem Bereich der Projektarbeit [1], kann der damals verwendete Algorithmus, Shepard's Methode für verstreute Daten [15], mit den damals bestimmten Parametern übernommen werden.

Es wird jedoch nicht mehr der ganze Raum im Voraus interpoliert und im RAM gespeichert. Weil der Raum nun auf ein Koordinatensystem der Grösse 640x640 anstatt wie in der Projektarbeit auf 100x100 dargestellt wird, würde mehr Speicher in einem bereits begrenzten Umfeld gebraucht werden. Zusätzlich hat sich herausgestellt, dass der Algorithmus sehr performant ist und die Interpolation an einer bestimmten Koordinate in unter einer Millisekunde durchgeführt werden kann.

3.7.1 Sonos Schnittstelle

Die offizielle Sonos™ Control API [16] erlaubt es die Audio Wiedergabe und die Gruppen-Zuteilung zu steuern. Für diese Arbeit besonders interessant ist die Fähigkeit, die Lautstärke einzelner Wiedergabegeräte zu steuern. Die Schnittstelle läuft auf den Servern von Sonos™ und erfordert zur Verwendung einen gültigen Sonos™ Account. Die Schnittstelle leitet die gewünschten Aktionen an die mit dem Account verbundenen Wiedergabegeräte weiter.

Damit die Steuerung durchgeführt werden kann, muss dies der Besitzer der Wiedergabegeräte autorisieren. Dazu muss die Applikation, welche die Steuerung durchführen möchte, den Benutzer zur offiziellen Website von Sonos™ weiterleiten. Dieser muss sich dort mit seinem Account einloggen und die Steuerung seines Sonos™ Systems durch die Real-Stereo Applikation autorisieren. Anschliessend wird der Benutzer mit dem Autorisierungscode zurück zur Real-Stereo Webapp geführt. Mit diesem Autorisierungscode können nun Steuerungsbefehle versendet werden.

Sonos™ setzt jedoch voraus, dass die Webapp HTTPS unterstützt und öffentlich über das Internet erreichbar ist, damit dieser Autorisierungsfluss erfolgreich durchgeführt werden kann. Dies stellt für die geplanten Zwecke in dieser Arbeit ein Problem dar, da das Real-Stereo System nicht vom Internet erreichbar sein soll. Dies hat mehrere Gründe, nicht zuletzt um den Datenschutz und somit die Privatsphäre des Benutzers gewährleisten zu können.

Aus diesem Grund wurde für die Steuerung die Python Softwarebibliothek SoCo [17] eingesetzt. Diese verwendet die undokumentierte und inoffizielle, auf UPnP basierende, API von Sonos™ Lautsprechern. Damit wird keine zusätzliche Authentifizierung benötigt, wenn sich die Real-Stereo Geräte und Sonos™ Lautsprecher im gleichen WLAN befinden. Auch werden die Steuerungsbefehle direkt an die Lautsprecher geschickt, ohne den Umweg über die Server von Sonos™.

Im Real-Stereo System wurde der Zugriff auf die SoCo Bibliothek mit einem Adapter-Pattern umgesetzt, damit später auch Unterstützung für andere Lautsprechertypen hinzugefügt werden könnte.

3.8 Netzwerk Protokoll

Ein zentraler Teil der Applikation ist die Kommunikation der einzelnen Nodes innerhalb des Netzwerkes. Hierfür wurden Protocol Buffer [18] auf dem UDP- und TCP-Protokoll verwendet. Diese bieten einen einfachen und performanten Weg, um strukturierte Daten sprachunabhängig mit möglichst wenig Overhead zu serialisieren und am anderen Ende der Kommunikation wieder zu deserialisieren. Jegliche Kommunikation geht dabei immer über den von Real-Stereo bestimmten Port 5605.

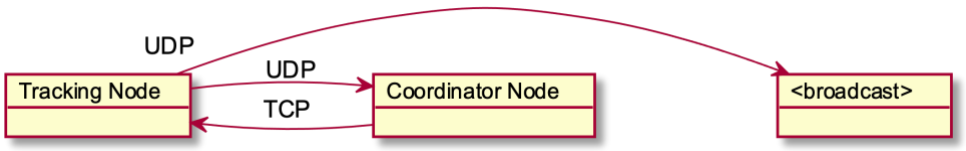


Abbildung 9 Kommunikationsprotokolle zwischen Nodes.

Wie in der Abbildung 9 aufgezeigt, basiert die ausgehende Kommunikation einer Tracking Node auf UDP und die der Coordinator Node auf TCP. Eine Tracking Node

sendet nach jedem Frame die Koordinate einer Person an die Coordinator Node. Dies kann, abhängig vom verwendeten Personenerkennungsalgorithmus, mehrmals pro Sekunde ausgeführt werden. Deshalb wird für weniger Netzwerkauslastung auf den Overhead eines TCP-Paketes verzichtet und auf UDP gesetzt. Ebenfalls ist ein möglicher Paketverlust von UDP in diesem Anwendungsfall vernachlässigbar, da das nächste Paket mit der nächsten Koordinate innerhalb einer Sekunde zu erwarten ist und somit die TCP-Retransmission redundant wäre. Für die Service Discovery, welche in der folgenden Abbildung 10 erläutert wird, wurde aufgrund des verwendeten Broadcasting ebenfalls auf UDP gesetzt.

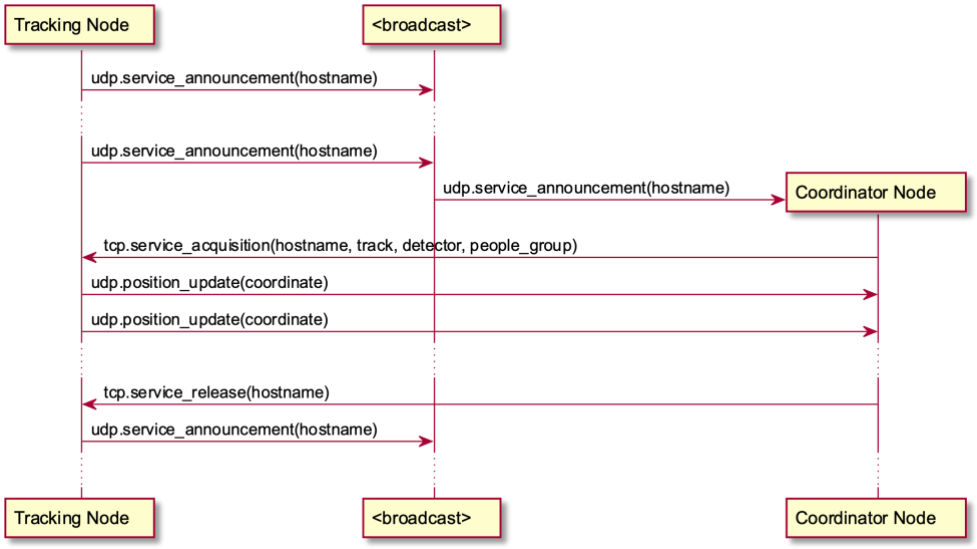


Abbildung 10 Ablauf des Service Discovery.

Solange eine Tracking Node nicht einer Coordinator Node, welche gerade online ist, zugewiesen ist, sendet sie das *ServiceAnnouncement* Paket an die Broadcast Domain des Netzwerkes. Dieses wird von der Coordinator Node aufgefasst, sobald diese verfügbar ist. Die Tracking Node wird darauf per TCP benachrichtigt, an welche Coordinator Node sie ab jetzt die Positions-Updates senden soll. Diese sendet sie jedoch nur, wenn das Balancing auch aktiviert ist. Der Balancing-Status wird entweder direkt in der *ServiceAcquisition* Nachricht mit dem *track* Parameter oder einer separaten *ServiceUpdate* Nachricht mitgeteilt.

Wird die Tracking Node über das Web Interface aus dem Netzwerk gelöscht, wird das *ServiceRelease* Packet geschickt. In diesem Fall stellt die Node den Service ein und sendet erneut die *ServiceAnnouncement* Nachricht an die Broadcast Domain, bis eine neue Coordinator Node den Service beansprucht. Dasselbe passiert ebenfalls, falls die Tracking Node über eine längere Zeit keine Nachricht mehr erhalten hat und somit annimmt, dass die Coordinator Node nicht mehr erreichbar ist. Um dies zu verhindern,

während noch alle Teilnehmer verfügbar sind, schickt die Coordinator Node an alle verbundenen Tracking Nodes eine *Ping* Nachricht in einem festgelegten Intervall.

3.8.1 Nachrichtenwrapper

Im Folgenden werden alle Nachrichten beschrieben, die für das Protokoll im Voraus definiert worden sind. Dafür wird das Protocol Buffer Format verwendet.

```
message Wrapper {
  uint32 app = 1;
  uint32 version = 2;
  oneof message {
    ServiceAnnouncement serviceAnnouncement = 3;
    ServiceAcquisition serviceAcquisition = 4;
    ServiceRelease serviceRelease = 5;
    PositionUpdate positionUpdate = 6;
    ServiceUpdate serviceUpdate = 7;
    Ping ping = 8;
    CameraCalibrationRequest cameraCalibrationRequest = 9;
    CameraCalibrationResponse cameraCalibrationResponse = 10;
  }
}
```

Abbildung 11 Wrapper Nachricht.

Alle Nachrichten werden in der Wrapper Nachricht, welche in Abbildung 11 ersichtlich ist, eingebettet. Jede Nachricht startet immer mit dem *app* Feld. Dies ist ein fest definierter Integer und erlaubt es zu unterscheiden, ob es sich bei der Empfangenen Nachricht um eine von Real-Stereo handelt oder von einem anderen Programm verschickt wurde. Dies könnte theoretisch auftreten, wenn der gleiche Port verwendet wird.

Zusätzlich wird die Version von Real-Stereo mitgeschickt. Damit lässt sich erkennen, welche Protokoll-Version verwendet wird und falls diese zu unterschiedlich sind, kann die Kommunikation verweigert und ein Update der Software gefordert werden. Weil Protocol Buffers richtig angewendet aber sowohl aufwärts- als auch abwärtskompatibel sind, sollte das nur sehr selten vorkommen.

Im *message* Feld wird jeweils die eigentliche Nachricht geschickt. Die Möglichkeiten dafür werden in den folgenden Unterkapitel weiter erläutert.

3.8.2 Service Nachrichten

Die Service Nachrichten werden, bis auf das *ServiceAnnouncement* aus Abbildung 12, von der Coordinator Node zu einer oder mehreren Tracking Nodes geschickt und informieren diese über den Status des Netzwerkes. Da sich die IP-Adresse einer Node je nach Netzwerk und DHCP-Konfiguration ändern kann, wird der Hostname als

eindeutige Identifikation verwendet und bei jeder Service Nachricht mitgeschickt. Der Hostname wird beim Aufsetzen eines neuen Real-Stereo Gerätes automatisch gesetzt und danach nicht mehr geändert.

```
message ServiceAnnouncement {
  string hostname = 1;
}
```

Abbildung 12 Aufbau der ServiceAnnouncement Nachricht.

Eine Tracking Node zeigt mit der *ServiceAnnouncement* Nachricht ihre Bereitschaft im Netzwerk an. Bekommt sie keine Antwort, sendet sie diese Nachricht alle 15 Sekunden erneut an die Broadcast Domain.

```
message ServiceAcquisition {
  bool track = 1;
  string hostname = 2;
  string detector = 3;
  string people_group = 4;
}
```

Abbildung 13 Aufbau der ServiceAcquisition Nachricht.

Auf eine *ServiceAnnouncement* Nachricht folgt im Falle, dass eine Coordinator Node erreichbar ist, immer eine *ServiceAcquisition* Nachricht, welche in Abbildung 13 ersichtlich ist. Dabei wird der Tracking Node der Hostname der Coordinator Node und der Status des Netzwerkes mitgeteilt. Die empfangende Node weiss somit, ob sie gleich die Personenerkennung starten soll und mit welchen Algorithmen.

```
message ServiceRelease {
  string hostname = 1;
}
```

Abbildung 14 Aufbau der ServiceRelease Nachricht.

Ist eine Tracking Node im Netzwerk nicht mehr erwünscht oder wird das System abgeschaltet, erhalten alle zuvor erschlossenen Nodes die *ServiceRelease* Nachricht der Abbildung 14. Diese stoppen damit die Personenerkennung, falls diese aktiv war, und gehen wieder in den Status über, in dem sie die *ServiceAnnouncement* Nachrichten an die Broadcast Domain schicken. Sie zeigen damit die erneute Verfügbarkeit für Coordinator Nodes.

```
message ServiceUpdate {
  bool track = 1;
  string detector = 2;
  string people_group = 3;
}
```

Abbildung 15 Aufbau der ServiceUpdate Nachricht.

Eine Coordinator Node kann mit der *ServiceUpdate* Nachricht aus Abbildung 15 jederzeit alle Teilnehmer im Netzwerk über einen neuen Status informieren. Dies geschieht, wenn das Balancing aktiviert oder deaktiviert wird, über das User Interface einen anderen Personenerkennungs-Algorithmus eingestellt wird, oder die Methode geändert wird, wie mehrere Personen im Bild verarbeitet werden.

3.8.3 Positionsupdate

Falls das Balancing aktiviert ist, führt jede Tracking Node die Personenerkennung durch und sendet, sobald es eine Änderung der Position gibt, die *PositionUpdate* Nachricht wie in Abbildung 16 dargestellt.

```
message PositionUpdate {
  uint32 coordinate = 1;
}
```

Abbildung 16 Aufbau der PositionUpdate Nachricht.

Diese Nachricht besitzt nur ein einziges Feld, in welchem die Koordinate der erkannten Person gesendet wird. Die Coordinator Node kann diese zusammen mit der *PositionUpdate* Nachricht der zweiten Tracking Node des gleichen Raumes in eine Koordinate des zweidimensionalen Raumes abbilden und somit die Person lokalisieren.

3.8.4 Kamera-Kalibrierungs Nachrichten

Für die in Kapitel 3.2.2 erläuterte Kamera Kalibration ist ebenfalls eine Kommunikation zwischen den Tracking Nodes und der Coordinator Node erforderlich.

```
message CameraCalibrationRequest {
  bool start = 1;
  bool finish = 2;
  bool repeat = 3;
}
```

Abbildung 17 Aufbau der CameraCalibrationRequest Nachricht.

Die *CameraCalibrationRequest* Nachricht der Abbildung 17 wird von der Coordinator Node an eine Tracking Node geschickt, welche kalibriert werden soll. Die Nachricht kann auf jeden möglichen Status abgebildet werden. So kann etwa mitgeteilt werden, ob die Kalibration gestartet oder beendet wird, ob ein einzelner Schritt wiederholt wird, oder ob die ganze Kalibration neu gestartet wird, falls *finish* und *repeat* beide auf *True* gesetzt werden.

```
message CameraCalibrationResponse {
  uint32 count = 1;
  string image = 2;
}
```

Abbildung 18 Aufbau der CameraCalibrationResponse Nachricht.

Die Antwortsnachricht auf einen Request wird in Abbildung 18 aufgezeigt. Sie wird ausgelöst, sobald ein Schachbrettmuster erkannt wurde und somit weitere Parameter für die Kalibration berechnet werden konnten. Es wird die Information mitgeschickt, wie viele Bilder bereits für die Kalibration verwendet werden und der Pfad des letzten Bildes, welches im User Interface zur Bestätigung angezeigt wird.

3.8.5 Ping Nachricht

Wie bereits zu Beginn von Kapitel 3.8 erläutert, erhalten alle Tracking Nodes Ping Nachrichten, welche bestätigen, dass die Coordinator Node immer noch erreichbar ist. Wie in Abbildung 19 zu sehen ist, werden dabei keine weiteren Informationen mitgesendet.

```
message Ping {}
```

Abbildung 19 Aufbau der Ping Nachricht.

Die Coordinator Node hat ebenfalls Interesse am Status der Tracking Nodes. In diese Richtung wurde jedoch keine Ping Nachricht eingeführt, da die *PositionUpdate* Nachrichten, falls das Balancing aktiviert ist, bereits genug häufig gesendet werden. Für den Fall, dass das Balancing deaktiviert ist oder über längere Zeit keine Person erkannt werden konnte, wird die letzte *PositionUpdate* Nachricht wiederholt gesendet und agiert in diesem Fall als Ping.

3.9 User Interface

Da das Real-Stereo System vom Benutzer mittels eines Smartphones bedient und konfiguriert wird, soll eine Webapp entwickelt werden, welche diese Funktionalitäten einfach und übersichtlich zur Verfügung stellt. Die Webapp wird mittels *React* [19] erstellt.

3.9.1 Kommunikationsprotokoll

Für die Kommunikation der Webapp mit der Coordinator Node des Systems wird das Echtzeitprotokoll *Websocket* mithilfe der Programmbibliothek *Socket.io* [20] eingesetzt. Dies ermöglicht der Coordinator Node die Änderungen an der Konfiguration und den Status des Systems ohne spürbare Latenz an alle Geräte zu senden, welche die Webapp aktuell verwenden. Anders als bei einer API, welche über HTTP umgesetzt wird, werden Events ausgelöst und auf Events gehört, welche zu jeder Zeit ausgelöst werden können. Für die Organisation der Schnittstelle wird diese in mehrere *Namespaces* unterteilt. Im Folgenden werden alle implementierten *Namespaces*, ihre Verwendungsweise und die dabei verwendeten Datentypen beschrieben. Die Datentypen sind in der *TypeScript* [21] Sprache definiert.

Rooms

Der *rooms* Namespace wird von der Webapp verwendet, um alle vom Benutzer konfigurierten Räume abzufragen, zu verändern und zu löschen. Ebenfalls können in diesem Namespace neue Räume erstellt werden. Diese Aktionen werden mit den Events durchgeführt, welche in der Abbildung 21 dargestellt sind. Die Definition der verwendeten Datentypen sind in Abbildung 20 abgebildet.

```
type Room = {
  id: number;
  name: string;
  nodes: Omit<Node, 'room'>[];
  people_group?: string;
}

type UpdateRoom = Omit<Room, 'nodes'>
type CreateRoom = Omit<UpdateRoom, 'id'>
```

Abbildung 20 Datentypen im Namespace Rooms.

```
get: () => Room[]
create: (data: CreateRoom) => Acknowledgment
update: (data: UpdateRoom) => Acknowledgment
delete: (id: number) => Acknowledgment
```

Abbildung 21 Events im Namespace Rooms.

Das *nodes* Feld beinhaltet bei einem Raum alle Tracking Nodes, welche diesem Raum zugewiesen sind. Das *people_group* Feld beinhaltet die ausgewählte Strategie, welche bei mehreren Menschen im Raum für das Tracking verwendet werden soll, siehe Abschnitt 3.5.2.

Nodes

In diesem Namespace können von der Webapp alle neu entdeckten und bereits konfigurierten Tracking Nodes verwaltet werden. Dazu gehört das Konfigurieren von neuen und das Löschen von bereits konfigurierten Tracking Nodes.

```
type Node = {
  id: number;
  name: string;
  online: boolean;
  ip: string;
  hostname: string;
  room?: Omit<Room, 'nodes'>;
  detector?: string;
}

type UpdateNode = {
  id: number;
  name: string;
  // only the `id` attribute of the room is needed
  // more can still be submitted but will be ignored
  room: {
    id: number;
  };
  detector?: string;
}
```

Abbildung 22 Datentypen im Namespace Nodes.

```
get: () => Node[]
update: (data: UpdateNode) => Acknowledgment
delete: (id: number) => Acknowledgment
```

Abbildung 23 Events im Namespace Nodes.

Das *detector* Feld beinhaltet den Personenerkennungsalgorithmus, welcher von der Tracking Node während des Balancings und der Raumkalibration verwendet werden soll.

Speakers

Der *speakers* Namespace wird für die gleichen Aufgaben wie der *nodes* Namespace verwendet, jedoch für die Verwaltung der Sonos™ Lautsprecher anstatt der Tracking Nodes.

```
type Speaker = {
  id: string;
  name: string;
  room: Omit<Room, 'nodes'>;
}

type UpdateSpeaker = {
  id: string;
  name: string;
  // only the `id` attribute of the room is needed
  // more can still be submitted but will be ignored
  room: Partial<Room> & {
    id: number;
  };
}
```

Abbildung 24 Datentypen im Namespace Speakers.

```
get: () => Speaker[]
update: (data: UpdateSpeaker) => Acknowledgment
delete: (id: number) => Acknowledgment
```

Abbildung 25 Events im Namespace Speakers.

Balances

Über diesen Namespace kann der momentane Status des Balancings aller Lautsprecher, und somit das Lautstärkelevel dieser Lautsprecher durch die Webapp abgefragt werden.

```
type Balance = {
  volume: number;
  speaker: Speaker;
}
```

Abbildung 26 Datentypen im Namespace Balances.

```
get: () => Balance[]
```

Abbildung 27 Events im Namespace Balances.

Settings

Der momentane Operationsmodus des Systems, Balancing oder Testmodus, und der zu verwendende Netzwerkmodus der Tracking oder Coordinator Node wird mit den Events im Namespace *settings* gesteuert. Ebenfalls wird der Namespace bei aktivem Testmodus verwendet, um die aktuellen Koordinaten des Personentracking an die Webapp zu senden.

```
type Settings = {
  configured: boolean;
  balance: boolean;
  testMode: boolean;
  network: 'client' | 'adhoc';
}

type UpdateSettings = {
  balance?: boolean;
  testMode?: boolean;
  nodeType?: 'master' | 'tracking';
  network?: CreateNetwork;
}

type SettingsTestModeResult = {
  room: Room;
  positionX: number;
  positionY: number;
}[]
```

Abbildung 28 Datentypen im Namespace Settings.

```
get: () => Settings
testModeResult: () => SettingsTestModeResult
update: (data: Settings) => Acknowledgment
```

Abbildung 29 Events im Namespace Settings.

Networks

Der *networks* Namespace dient dazu, eine WLAN- Konfiguration abzuspeichern. Diese wird anschliessend von der Tracking oder Coordinator Node verwendet um sich mit dem konfigurierten WLAN Netzwerk zu verbinden.

```
type CreateNetwork = {
  ssid: string;
  psk?: string;
}
```

Abbildung 30 Datentypen im Namespace Networks.

```
create: (data: CreateNetwork) => Acknowledgment
```

Abbildung 31 Events im Namespace Networks.

Das *ssid* Feld wird verwendet, um den Namen des WLAN-Netzwerks anzugeben, während das optionale *psk* Feld das Passwort für die Verbindung enthalten soll.

Camera-Calibration

In diesem Namespace wird der Kamera-Kalibrationsprozess, wie in Abschnitt 3.2.2 beschrieben, koordiniert. Als Antwort wird der Webapp der Link für das, von der Tracking Node aufgenommene, Testbild zurückgegeben, damit es für den Benutzer angezeigt werden kann.

```
type CameraCalibrationRequest = {
  node: {
    id: number;
  };
  start?: boolean;
  finish?: boolean;
  repeat?: boolean;
}

type CameraCalibrationResponse = {
  node: {
    id: number;
  };
  count: number;
  image: string;
}
```

Abbildung 32 Datentypen im Namespace Camera-Calibration.

```
get: () => CameraCalibrationResponse
update: (data: CameraCalibrationRequest) => Acknowledgment
```

Abbildung 33 Events im Namespace Camera-Calibration.

Room-Calibration

Ähnlich wie der Namespace *camera-calibration* wird der Namespace *room-calibration* verwendet, um die Raumkalibration, wie in Abschnitt 3.6 beschrieben, zu koordinieren. Auch werden über diesen Namespace die aufgenommenen und berechneten Lautstärken der Webapp an den verschiedenen Kalibrationspunkten an die Coordinator Node übertragen.

```
type RoomCalibrationRequest = {
  room: {
    id: number;
  };
  start?: boolean;
  startVolume?: number;
  finish?: boolean;
  repeatPoint?: boolean;
  confirmPoint?: boolean;
  nextPoint?: boolean;
  nextSpeaker?: boolean;
}

type RoomCalibrationPoint = {
  coordinateX: number;
  coordinateY: number;
  measuredVolume: number;
  speaker_id: string;
}

type RoomCalibrationResponse = {
  room: {
    id: number;
  };
  calibrating: boolean;
  positionX: number;
  positionY: number;
  positionFreeze: boolean;
  currentSpeakerIndex: number,
  currentPoints: RoomCalibrationPoint[],
  previousPoints: RoomCalibrationPoint[],
}

type RoomCalibrationResult = {
  room: {
    id: number;
  };
  volume: number;
}
```

Abbildung 34 Datentypen im Namespace Room-Calibration.

```
get: () => RoomCalibrationResponse
update: (data: RoomCalibrationRequest) => Acknowledgment
result: (data: RoomCalibrationResult) => Acknowledgment
```

Abbildung 35 Events im Namespace Room-Calibration.

3.9.2 Verschlüsselte Verbindung

Wie in Abschnitt 3.6 ausgeführt, wird die Aufnahme und Analyse des Testgeräusches durch die Webapp durchgeführt. Moderne Browser setzen voraus, dass die Webapp

mittels HTTPS aufgerufen wird, um Zugriff auf das Mikrofon des Gerätes zu erlauben [22]. Dadurch müssen alle anderen Zugriffe auf die Websocket API und die statischen Ressourcen, wie Kalibrationsbilder der einzelnen Tracking Nodes, ebenfalls über HTTPS bereitgestellt und aufgerufen werden.

Um dieses Problem zu lösen, wird eine Klasse implementiert, welche beim ersten Start einer Tracking oder Coordinator Node ein SSL Zertifikat generiert. Dieses Zertifikat wird danach für die HTTPS Verbindung verwendet und erneut generiert, falls es nicht mehr gültig ist oder sich die IP-Adresse oder der generierte Hostname der Node geändert hat. Dadurch kann sichergestellt werden, dass jederzeit ein gültiges und korrektes Zertifikat zur Verfügung steht. Da es sich hierbei um ein selbstsigniertes Zertifikat handelt, muss der Benutzer beim ersten Aufruf mit einem neuen Zertifikat eine Ausnahme für dieses Zertifikat im verwendeten Betriebssystem oder Browser hinzufügen, um es als vertrauenswürdig zu markieren.

3.9.3 Hautpmenü

Vor der Umsetzung des User Interfaces wurden Wireframes der wichtigsten Seiten der Webapp erstellt. Dies erlaubte frühes Feedback zu den geplanten Funktionen und deren Darstellung. Auch konnte damit Kontinuität im Interface sichergestellt werden und es wurden Probleme frühzeitig aufgezeigt, welche in iterativen Verbesserungen beseitigt werden konnten.

Folgend werden die Wireframes beschrieben, sowie die möglichen Interaktionen, welche der Benutzer auf der jeweiligen Seite ausführen kann.

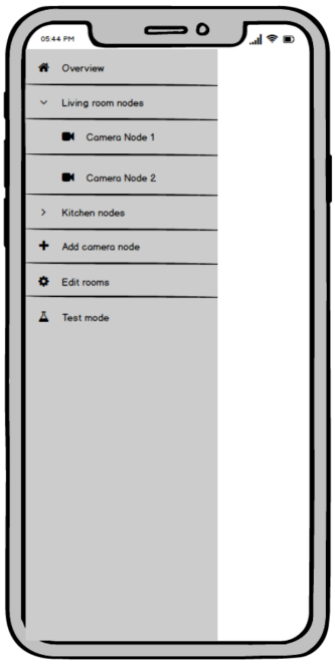


Abbildung 36 Wireframe des Hauptmenüs.

Mit dem Hauptmenü in Abbildung 36 kann der Benutzer die Webapp navigieren und die verfügbaren Seiten aufrufen. Um die verschiedenen Tracking Nodes zu steuern und zu kalibrieren, wird für jede Tracking Node ein Menüpunkt hinzugefügt und diese dem zugewiesenen Raum untergeordnet. Bei einem Klick auf einen solchen Menüpunkt wird die dazugehörige Tracking Node Detailseite aufgerufen.

Das Hauptmenü kann mit einem Klick auf das Menü-Icon, welches im Header von jeder Seite sichtbar ist, angezeigt und wieder versteckt werden.

3.9.4 Übersicht

Die Übersichtsseite in Abbildung 37 ist die erste Seite, welche dem Benutzer bei einem konfigurierten System angezeigt wird. Auf der Übersichtsseite der Webapp kann das Real-Stereo Balancing-System ein- und ausgeschalten werden. Ebenfalls wird bei aktiviertem Balancing-System die aktuelle Lautstärke der verschiedenen Lautsprecher angezeigt.

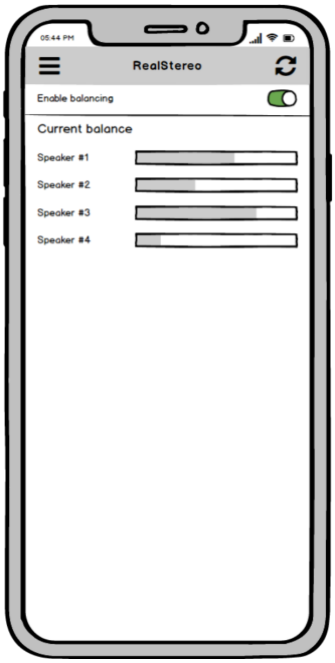


Abbildung 37 Wireframe der Übersichtsseite.

3.9.5 Tracking Node Detailseite

Auf der Tracking Node Detailseite kann der Name, der zugewiesene Raum und der Algorithmus für das Personentracking für die Tracking Node angepasst werden. Wie in Abbildung 38 ersichtlich, wird ebenfalls eine Live-Vorschau der Kamera angezeigt, welche bei aktivem Personentracking auch Informationen über die Personenerkennung und die Leistung enthält. Dies ermöglicht dem Benutzer eine genauere Platzierung der

Tracking Nodes, da er sofort den sichtbaren Bereich überprüfen kann. Schlussendlich kann die Tracking Node auf der Detailseite auch vom System entfernt werden.

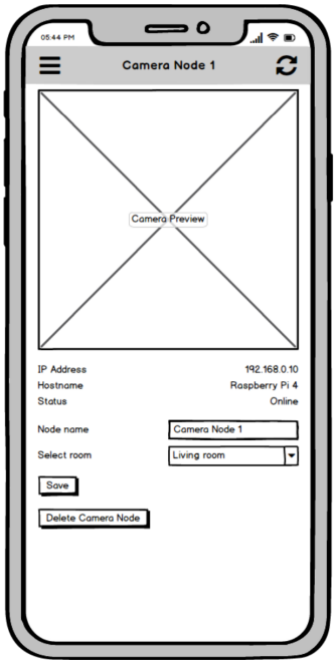


Abbildung 38 Wireframe der Tracking Node Detailseite.

3.9.6 Tracking Node hinzufügen

Diese Seite ermöglicht dem Benutzer, alle entdeckten Tracking Nodes, welche noch nicht von der Coordinator Node zum System hinzugefügt wurden, einzusehen und sie auf Wunsch mit einem einzelnen Klick zum System hinzufügen. Das Hinzufügen ist durch das Plus-Icon möglich, welches in Abbildung 39 zu sehen ist.

Darauf wird der Benutzer auf die Detailseite, welche in Kapitel 3.9.5 beschrieben wurde, weitergeleitet. In dieser Detailseite kann die neu hinzugefügte Node gleich einem Raum zugeordnet werden und ist somit einsatzbereit.

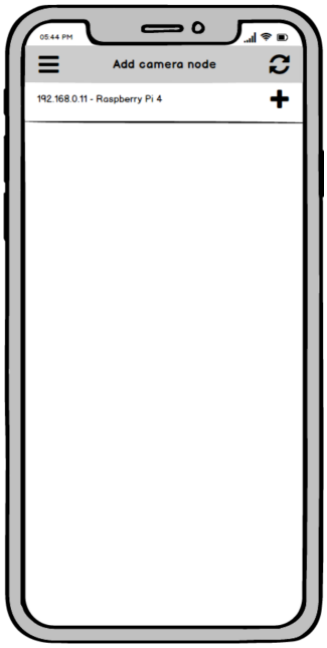


Abbildung 39 Wireframe Tracking Node hinzufügen.

3.9.7 Räume bearbeiten

Die Seite listet alle vom Benutzer hinzugefügten Räume auf und erlaubt es diese zu entfernen oder die dazugehörige Raum Detailseite aufzurufen. Ebenfalls kann hier vom Benutzer ein neuer Raum erstellt werden.

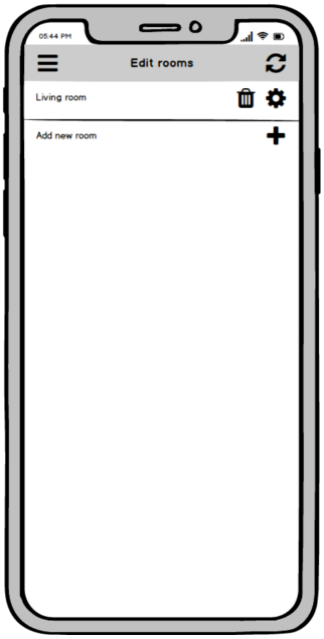


Abbildung 40 Wireframe Räume bearbeiten.

3.9.8 Raum Detailseite

Auf der Raum Detailseite der Abbildung 41 kann der Name und die zugewiesenen Lautsprecher des dazugehörigen Raums verändert werden. Ebenfalls wird hier die Raumkalibration gestartet und der Benutzer wird laufend über den Fortschritt und die Resultate des Kalibrationsprozesses informiert. Um Platz zu sparen ist der Fortschritt jedoch nur sichtbar, wenn eine Kalibration gestartet wird.

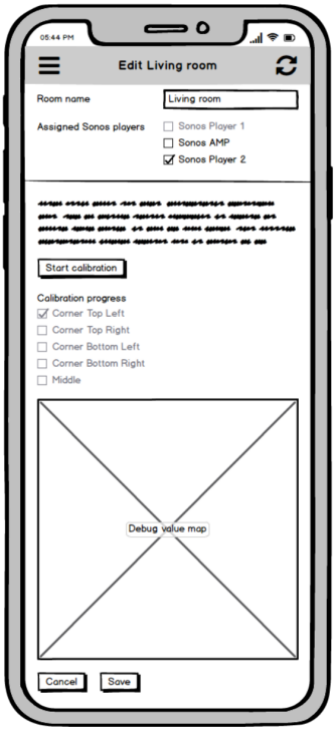


Abbildung 41 Wireframe Raum Detailseite.

3.9.9 Testmodus

In Abbildung 42 wird die Testmodus Seite aufgezeigt, auf welcher die Ergebnisse des Balancing getestet und somit die Leistung des Real-Stereo Systems bewertet werden. Im Testmodus wird ein weisses Rauschen, welches ebenfalls für die Raumkalibration verwendet wird, auf allen Lautsprechern des Raumes wiedergegeben und das Balancing System aktiviert. Durch das Messen der Lautstärke an mehreren Positionen im Raum kann anschliessend festgestellt werden, ob das System wie erwartet die Lautstärke an allen Positionen normalisiert. Die Messwerte werden grafisch auf einer Karte eingezeichnet, um einen einfacheren Vergleich von mehreren Positionen im Raum zu ermöglichen.

Ebenfalls wird auf der Seite des Testmodus ein Spektrumanalysator hinzugefügt, womit der Frequenzbereich des eingebauten Mikrofons grafisch dargestellt wird.

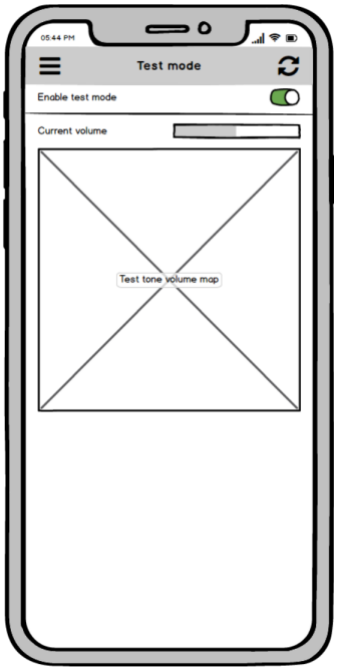


Abbildung 42 Wireframe Testmodus.

3.10 Performance Optimierungen

Aufgrund der einschränkenden Performance im Embedded Umfeld wurden mehrere Möglichkeiten zur Performance Optimierung analysiert.

Die **erste Optimierung** wird durch den Quad-Core-Prozessor ermöglicht, welcher in der verwendeten Version des Raspberry Pis eingesetzt wird. Um den Prozessor besser auszunutzen, wird die Applikation auf mehrere Prozesse aufgeteilt. Die Prozessarchitektur ist in Abbildung 43 beschrieben.

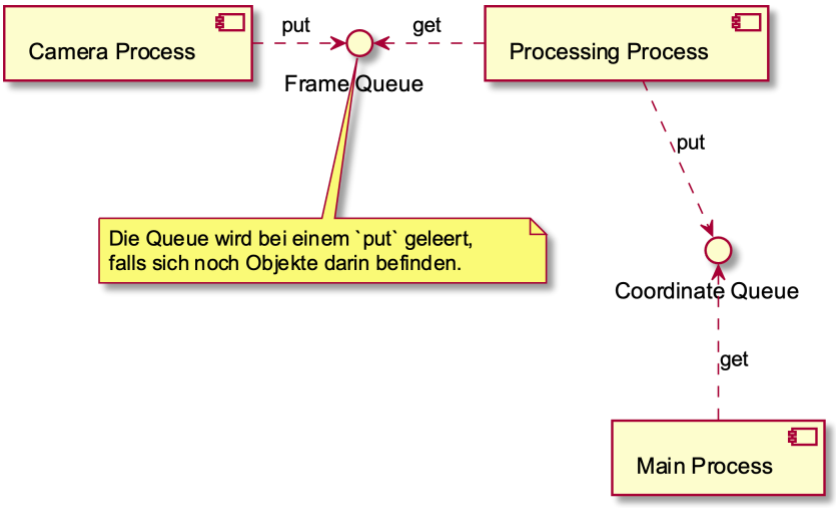


Abbildung 43 Prozessarchitektur.

Der Kamera Prozess öffnet eine Verbindung zum Kameramodul und erhält laufend das aktuelle Bild. Dieses wird in eine Queue eingefügt, auf welche der Processing Prozess ebenfalls Zugriff hat. Befindet sich beim Einfügen in die Queue noch das zuletzt erhaltene Frame darin, wird die Queue zuerst gelöscht. Dies verhindert, dass veraltete Frames verwendet werden, weil von der Kamera mehr Bilder zurückgegeben werden als vom Processing Prozess verarbeitet werden können. Der Processing Prozess führt danach den aufwändigsten Schritt durch, die Personenerkennung. Das Resultat wird wiederum in eine Queue gespeichert, welches dadurch zum Hauptprozess gelangt. Im Hauptprozess werden die restlichen Bereiche von Real-Stereo ausgeführt, wozu unter anderem die Kommunikation mit den Nodes im Netzwerk, das Balancing der Sonos™ Lautsprecher, und der HTTP-Server für das Web Interface gehören.

Für die Kommunikation zwischen den Prozessen wird auf Queues gesetzt, welche für eine Multiprocessing-Architektur von Python bereitgestellt werden [23]. Diese bringen den Vorteil mit, dass der empfangende Prozess auf ein neues Objekt warten kann, aber der sendende Prozess während der Verarbeitung des anderen Prozesses nicht warten muss. Dadurch kann bereits das nächste Frame entgegennehmen kann, während die Verarbeitung noch läuft.

Zusätzlich zu der Multiprocessing-Architektur wurde nach Möglichkeiten zur Optimierung der Personenerkennungs-Algorithmen gesucht, da diese den aufwändigsten Teil der Applikation beinhalten. Recherchen haben ergeben, dass es mit einem eigenen und für den Raspberry Pi optimierten Build von OpenCV möglich ist, die Performance um bis zu 50% zu verbessern [24]. Die tatsächliche Verbesserung ist jedoch von den verwendeten Algorithmen und Funktionen abhängig.

Für die **zweite Optimierung** werden demnach drei optionale Module, welche beim normalen Build von OpenCV nicht verwendet wurden, eingebunden. Das VFPV3 [25] Modul von Arm sorgt für die grösste Verbesserung. Der in den Raspberry Pis verbaute Arm Prozessor verfügt über sehr performante und effiziente Floating Point Optimierungen. Durch das Modul werden diese in den OpenCV-Algorithmen verwendet und bieten besonders Verbesserungen bei grossen Datensätzen und unvorhersehbaren Zugriffen [25]. Diese kommen häufig bei Algorithmen zum Einsatz, welche auf neuronalen Netzen basieren. Weitere Optimierungen bringt Neon, welches ebenfalls von Arm bereitgestellt wird und eine Erweiterung für Arm Prozessoren ist, die unter anderem Video- und Image-Processing beschleunigt [26]. Das dritte Modul sind die Thread Building Blocks (TBB) von Intel®. Diese ermöglichen das einfache Parallelisieren von ansonsten sequenziellen Algorithmen [27] und somit die schnellere Ausführung.

4 Resultate

In diesem Abschnitt werden die Ergebnisse der Arbeit untersucht. Auch werden die Unterschiede der endgültigen Implementierung im Vergleich mit der geplanten Methodik in Abschnitt 3 analysiert.

4.1 System

Wie in Abschnitt 3.1 erläutert, wurde der Raspberry Pi 4 Model B als Embedded Plattform und das passende Kamera-Modul ausgewählt. Um die komplette Einheit möglichst kompakt zu halten, wurde entschieden, das Kamera-Modul direkt auf der Raspberry Pi Leiterplatte mit doppelseitigem Klebeband zu installieren. Das kurze Flachbandkabel, welches das Kamera-Modul mit dem Raspberry Pi verbindet, unterstützt dieses Vorhaben. Für die Installation wird der Raspberry Pi mit dem installierten Kamera-Modul im offiziellen Raspberry Pi 4 Gehäuse [28] montiert. Damit wird das Gerät zum einen von äusseren Einflüssen geschützt und ist zum anderen anschaulicher für den Endbenutzer. Die Position des Kamera-Moduls, welche in Abbildung 44 dargestellt ist, ermöglicht eine Installation im Gehäuse, ohne dass dieses stark modifiziert werden muss.

Abbildung 44 Raspberry Pi mit installiertem Kameramodul.

Die einzig benötigte Modifikation im Gehäuse ist ein Loch im Deckel mit einem etwa 2cm grossen Durchmesser. Ein komplett installierter Raspberry Pi mit Kamera-Modul im modifizierten Gehäuse ist in Abbildung 45 abgebildet.

Abbildung 45 Raspberry Pi mit Deckel.

Die Kosten für ein minimales Real-Stereo System, mit einer Coordinator und einer Tracking Node ohne Sonos™ Lautsprecher, ist in Tabelle 2 aufgeführt. Ebenfalls wurden Links für die Artikel beim verwendeten Online-Shop aufgelistet.

Tabelle 2 Preis für benötigte Komponenten.

Menge	**Artikel**	**Preis (einzeln)**
2	Raspberry Pi 4 Model B 2GB https://www.pi-shop.ch/raspberry-pi-4-model-b-2gb	38.90
2	Raspberry Pi Camera (I), Fisheye Lens https://www.pi-shop.ch/raspberry-pi-camera-i-fisheye-lens-fischauge	29.90
2	Raspberry Pi 4 Gehäuse https://www.pi-shop.ch/raspberry-pi-4-gehaeuse-schwarz-grau	8.90
	Total	77.70 x 2 = 155.40

4.2 Applikation

Aufgrund der guten Performance und Kompatibilität mit Schnittstellen in allen benötigten Bereich, wurde das Backend der Applikation mit Python umgesetzt. Die zuvor definierten Methoden in Kapitel 3 vereinfachten dabei die Implementation und konnten wie definiert übernommen werden. In den folgenden Unterkapiteln werden die verschiedenen Bereiche der Applikation genauer erläutert.

4.2.1 User Interface

Die meisten Teile des User Interfaces konnten, wie in den Wireframes im Abschnitt 3.9 geplant, umgesetzt werden. Grössere Unterschiede gibt es vor allem auf der Raum Detailseite, auf welcher der Kalibrationsprozess redesignt wurde, um eine intuitivere und verständlichere Bedienung zu ermöglichen. Dieses Ziel wird erreicht, indem eine klarere Darstellung der benötigten Schritte während der Raumkalibration eingeführt wird und die Verwendung von zusätzlichen Kontrollelementen es dem Benutzer erlaubt, den Prozess zu beeinflussen.

Im Folgenden sind Screenshots von allen wichtigen User Interface Seiten und Elementen aufgeführt und deren Funktionen werden erläutert.

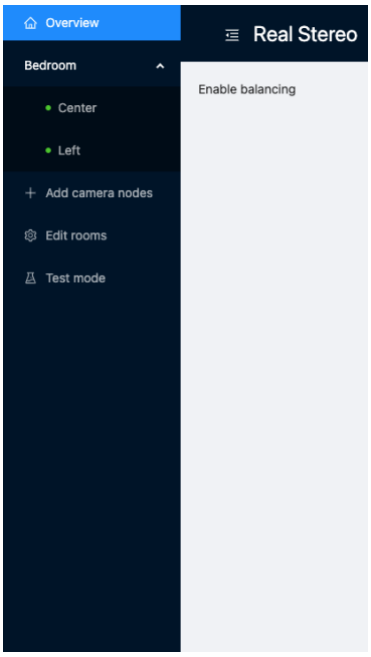


Abbildung 46 Screenshot des Hauptmenüs.

Das Hauptmenü, welches in Abbildung 46 gezeigt wird, erlaubt das einfache Erreichen von allen wichtigen Seiten. Durch das Menü-Icon, welches im Header sichtbar ist, kann das Hauptmenü von allen Seiten aus erreicht werden.

Es werden ebenfalls alle konfigurierten Räume direkt in der Navigation aufgelistet. Durch das Aufklappen von einem Raum werden dessen zugeordneten Nodes angezeigt. Mit einem Punkt links vom Namen wird der Status angezeigt. Ist die Tracking Node online, wird dieser grün dargestellt. Falls die Node nicht erreichbar ist, wird der Status rot. Durch einen Klick auf eine Node oder den Raum gelangt man auf die entsprechende Detailseite.

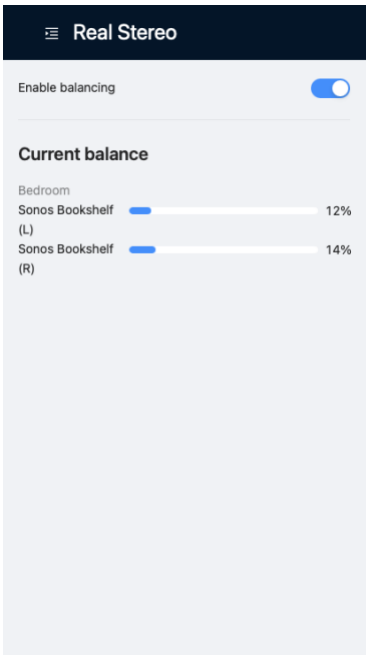


Abbildung 47 Screenshot der Übersichtsseite.

Die Übersichtsseite ist der Einstiegspunkt in die Applikation und wird in Abbildung 47 gezeigt. Sie ist schlicht gehalten und ermöglicht den einfachen Zugriff auf die wichtigste Funktion, welches das Aktivieren und Deaktivieren des Balancings ist. Wenn das Balancing aktiviert ist, werden alle Lautsprecher und deren eingestellte Lautstärke in Echtzeit aufgelistet.

4.2.2 Konfiguration

Für die Konfiguration des Real-Stereo Systems wurden, wie in Abschnitt 3.9 definiert, mehrere Seiten entwickelt. Die *Tracking Node hinzufügen* Seite der Webapp, in Abbildung 48 sichtbar, ermöglicht das Hinzufügen einer neuen Tracking Node zum Real-Stereo System. Dies ist durch einen Klick auf die blaue Schaltfläche mit einem Plus-Symbol möglich.

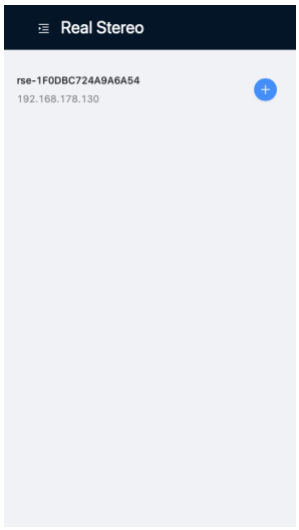


Abbildung 48 Screenshot Tracking Node hinzufügen.

Auf der *Tracking Node Detailseite* können anschliessend die Einstellungen für ausgewählte Tracking Node angepasst werden. Dazu zählt der Name der Tracking Node, der zugewiesene Raum und der Algorithmus, welcher für das Personen Tracking verwendet werden soll. Eine Live-Vorschau der Kamera im oberen Teil der Seite ermöglicht dem Benutzer eine einfache und genaue Platzierung der Tracking Node. Im unteren Teil der Seite können die Änderungen gespeichert werden oder die Tracking Node vom Real-Stereo System entfernt werden. Ein Screenshot der Seite ist in Abbildung 49 abgebildet.

Abbildung 49 Screenshot der Tracking Node Detailseite.

Auf der *Räume bearbeiten* Seite, welche in Abbildung 50 gezeigt wird, ist eine Liste der vom Benutzer erstellten Räume sichtbar. Mithilfe der Schaltfläche mit dem Zahnrad-Symbol kann die dazugehörige *Raum Detailseite* geöffnet werden. Durch Klicken auf die rote Schaltfläche mit dem Abfalleimer-Symbol kann ein Raum und die dazugehörige Konfiguration und Kalibration gelöscht werden. Am Ende der Liste befindet sich eine blaue Schaltfläche mit dem Plus Symbol, durch welche ein neuer Raum vom Benutzer erstellt werden kann.

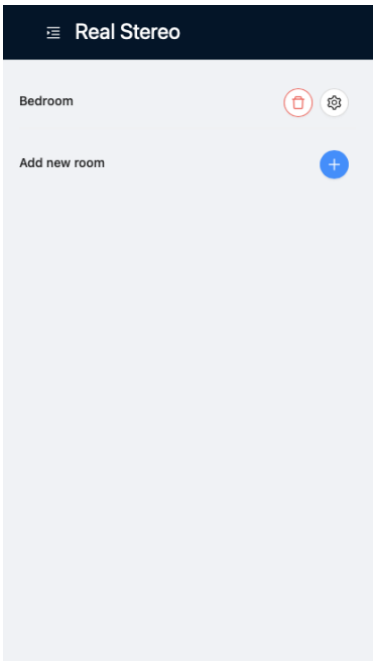


Abbildung 50 Screenshot Räume bearbeiten.

4.2.3 Raum Kalibration

Die Kalibration stellt die Basis für die spätere Interpolation dar. Anhand der während der Kalibration aufgenommene Lautstärke an einzelnen Punkten im Raum, kann durch Interpolation die wahrgenommene Lautstärke im ganzen Raum berechnet werden. Diese wird für das Balancing benötigt, um alle Lautsprecher auf die aktuelle Position abzustimmen.

Die Kalibration muss für jeden Raum einmal durchgeführt werden. Falls Tracking Nodes oder Lautsprecher dem Raum hinzugefügt oder entfernt werden, muss die Kalibration erneut durchgeführt werden.

Auf der Detailseite des Raumes kann die Kalibration gestartet werden. Einen kurzen Text weist den Benutzer ein und erklärt den Ablauf. Ist die Kalibration aktiv, ist im unteren Teil der Seite, welche auch in Abbildung 51 zu sehen ist, der Raum auf einem kleinen Quadrat abgebildet. Darin wird die aktuelle Position des Benutzers und die

Positionen, an welchen bereits die Kalibration durchgeführt worden ist, eingezeichnet. Rechts daneben sind die verschiedenen Schritte und der Fortschritt aufgelistet.

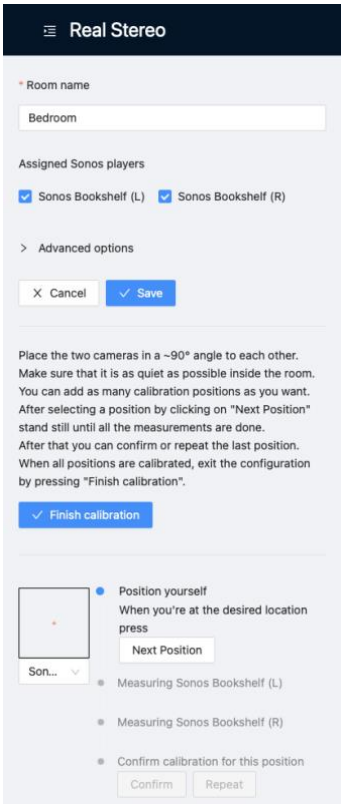


Abbildung 51 Screenshot Raum Detailseite.

Die Kalibration läuft folgendermassen ab.

1. Die Kalibration wird mit einem Klick auf «Start calibration» gestartet. Es werden das Raumabbild und die einzelnen Schritte eingeblendet.
2. Ist der Benutzer an der gewünschten Position, kann er durch den «Next Position» Button signalisieren, dass die Kalibration für die nächste Position durchgeführt werden soll.
3. Es wird ein weisses Rauschen für jeweils ungefähr fünf Sekunden auf allen Lautsprechern nacheinander abgespielt und die Lautstärke aufgenommen.
4. Der Medianwert der Lautstärke von jedem Lautsprecher wird abgespeichert.
5. Der Benutzer muss die Kalibration für die aktuelle Position entweder bestätigen oder wiederholen. Eine Wiederholung kann beispielsweise sinnvoll sein, wenn unerwartet andere laute Geräusche das Rauschen gestört haben oder das Mikrofon verdeckt wurde. Für eine Wiederholung wird erneut bei 3. fortgefahren.
6. Wird die Position bestätigt, wird diese in der Raumkarte abgebildet.

7. Der Benutzer kann nun zur nächsten Position gehen, an der er die Kalibration durchführen möchte. In diesem Fall wird bei 2. fortgefahren. Damit später eine zuverlässige Interpolation möglich ist, sollten mindestens 4 Positionen in den Ecken des Raumes durchgeführt werden. Es können jedoch beliebig viele weitere Positionen kalibriert werden.
8. Soll die Kalibration abgeschlossen werden, kann dies mit dem «Finish calibration» Button erreicht werden. Alle Kalibrationsdaten werden nun im Dateisystem abgespeichert.

4.2.4 Kamera Kalibration

Mit jeder Real-Stereo Installation werden Standard-Kalibrationsdaten für die Kamera mitgeliefert. Vergleiche der bestellten Kameras haben jedoch ergeben, dass deren Herstellung nicht exakt ist und es Unterschiede gibt. Die Kameralinsen sind nicht immer genau in der Mitte des Objektivs platziert, was zu unterschiedlichen grossen Bereichen in den Ecken führt, die kein Bild zurückgeben. Aus diesem Grund ist es den Benutzern möglich, die Kalibration pro Kamera selbst erneut durchzuführen, sollte die Standard-Kalibration keine guten Resultate liefern. Die Standard-Kalibration wurde initial mit dem gleichen Kalibrationsprozess erstellt, welche den Benutzern zur Verfügung steht. Der Prozess wird in der Abbildung 52 aufgezeigt.

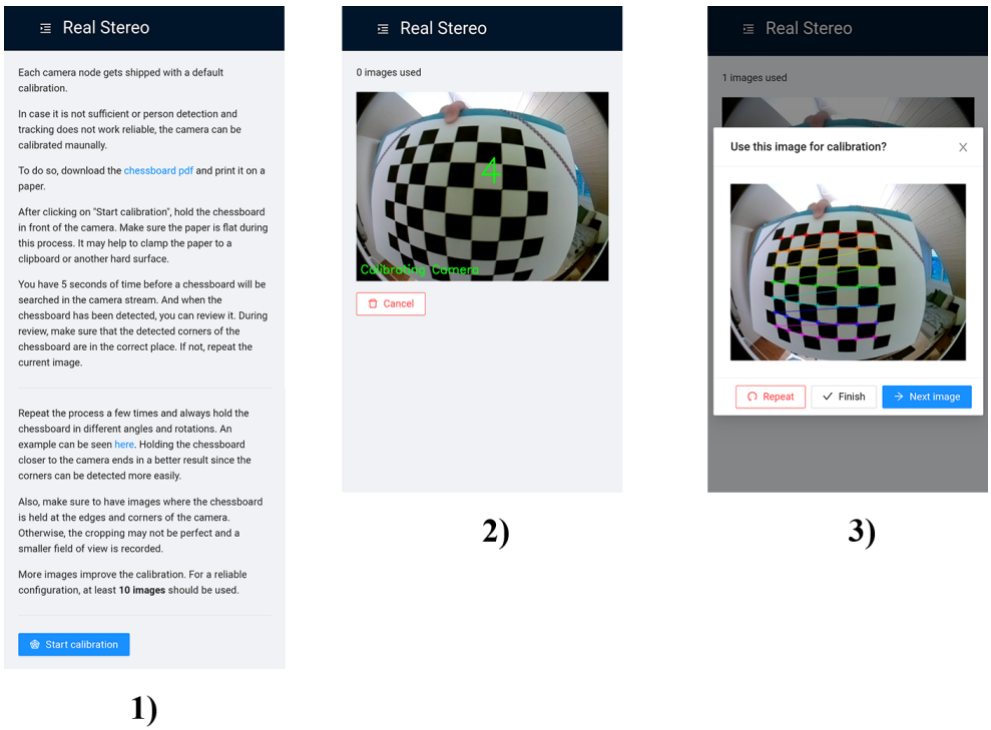


Abbildung 52 Ablauf des Kamera-Kalibrierungs-Prozess.

Zu Beginn der Kalibration erhält der Benutzer alle benötigten Informationen. Darunter befindet sich auch ein Link, um das Schachbrettmuster als PDF herunterzuladen. Dieses muss ausgedruckt werden und wird in den folgenden Schritten verwendet. Als zweites wird der Benutzer aufgefordert, das Schachbrettmuster in die Kamera zu halten. Der Benutzer hat dafür 5 Sekunden Zeit, um sich und das Muster wie gewünscht zu positionieren. Diese 5 Sekunden werden in der Mitte des Bildes als Countdown angezeigt. Sind diese abgelaufen, wird im Kamerabild nach dem Muster gesucht. Sobald dieses gefunden wurde, wird es dem Benutzer präsentiert. Er hat nun die Möglichkeit, das Ergebnis zu validieren. Sollte es entweder ganz oder auch nur in einzelnen Punkten nicht mit dem Muster übereinstimmen, kann das aktuelle Bild wiederholt aufgenommen werden. Stimmt jedoch alles überein, kann mit dem nächsten Bild fortgefahren werden. Für eine genaue Kalibration werden mindestens 10 Bilder aus unterschiedlichen Blickwinkeln und Entfernungen empfohlen. Der Benutzer hat aber jederzeit die Möglichkeit, die Kalibration zu beenden oder auch abzubrechen.

4.2.5 Testmodus

Der Testmodus erlaubt es, die Effektivität des Systems zu überprüfen. Ursprünglich wurde er für die Entwicklung und Tests dieser Bachelor Thesis erschaffen, kann aber auch von Benutzern verwendet werden, um die Kalibration zu verifizieren. Funktioniert das System perfekt, sollte im ganzen Raum die gemessene Lautstärke gleich laut sein.

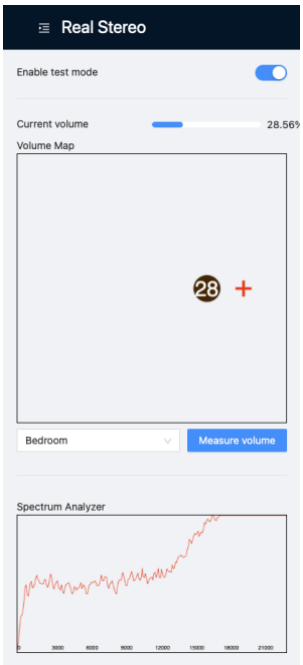


Abbildung 53 Screenshot Testmodus.

In Abbildung 53 ist das Interface, welches der Benutzer während dem Testmodus sieht, sichtbar. Mit der ersten Schaltfläche kann der Testmodus aktiviert und deaktiviert werden. Gleich darunter wird zu jeder Zeit die gemessene Lautstärke angezeigt. Das zentrale Element des Testmodus ist die Raumkarte. Darin wird der ganze Raum auf ein Quadrat abgebildet und die gemessene Lautstärke an allen Messpunkten eingezeichnet. Die aktuelle Position des Benutzers ist mit einem roten Kreuz ebenfalls darin eingezeichnet. Der Spektrum Analyzer am Ende der Seite stellt das gemessene Audiosignal auf einem Frequenzbereich von 0-21'000 Hertz dar. Genaue Analysen damit sind für den Endbenutzer zu technisch. Es kann jedoch verwendet werden, um zu verifizieren, dass das verwendete Mikrofon funktioniert. Während der Implementation wurde es zudem verwendet, um verschiedene Rauschen zu vergleichen und Bewertungsfilter anzupassen.

Ein Testlauf dieses Modus läuft wie folgt ab.

1. Der Benutzer aktiviert den Testmodus.
2. Der Benutzer sieht im User Interface seine aktuelle Position im Raum, welche durch ein rotes Kreuz gekennzeichnet ist.
3. Befindet sich der Benutzer an der gewünschten Position, kann er die Messung dieser Position durch einen Klick auf den Button «Measure volume» starten.
4. Das gleiche weisse Rauschen wie in der Raumkalibration wird auf allen Lautsprechern abgespielt.
5. Die Lautstärke wird nun gemessen, bis ein weiteres Positionsupdate von der Webapp empfangen wird. Anschliessend wird der Medianwert gespeichert, in der Raumkarte an der aktuellen Position eingetragen und das Rauschen gestoppt.
6. Der Vorgang kann für eine beliebige Anzahl an weiteren Punkten wiederholt werden. In diesem Fall wird bei 3. weitergefahren.

4.3 Tests

Um die Qualität der Umsetzung und die Effektivität zu verifizieren, wurden zwei Arten von Tests durchgeführt. Zum einen wurde das Balancing im Raum getestet. Hierfür wurde der in Kapitel 4.2.5 erläuterte Testmodus verwendet, um die wahrgenommene Lautstärke an verschiedenen Punkten im Raum zu vergleichen. Zum anderen wurde die Performance der verfügbaren Personenerkennungs-Algorithmen analysiert, damit später eine Empfehlung für verschiedene Anwendungsfälle herausgegeben werden kann.

4.3.1 Balancing

Mit Hilfe des Testmodus wurde die wahrgenommene Lautstärke im Raum, und somit die Qualität des Balancings getestet. Hierfür wurde ein Rauschen auf den Lautsprechern

abgespielt und die Lautstärke an mehreren Positionen gemessen. Für den Vergleich wurde der Test je einmal mit aktiviertem und deaktiviertem Balancing durchgeführt.

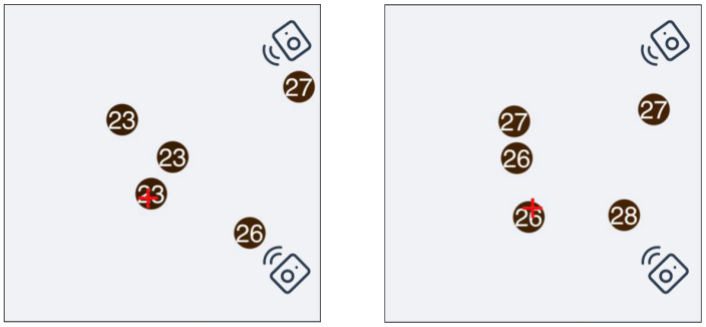


Abbildung 54 Resultat des Balancing-Tests verglichen mit deaktiviertem Balancing.

In der Abbildung 54 ist das Resultat des Testmodus ersichtlich. Die Position der Lautsprecher wurden zusätzlich nachträglich als Icon in die Resultate eingezeichnet.

Während das Balancing deaktiviert war, wurde die Verteilung im Raum nicht gleichmässig gemessen. Direkt vor den Lautsprechern wurden Werte von 26 und 27 gemessen, wobei die Werte weiter entfernt im Raum nur noch 23 betrugen. Während das Balancing jedoch aktiviert war, befanden sich alle gemessenen Werte in einem Bereich von 26-28.

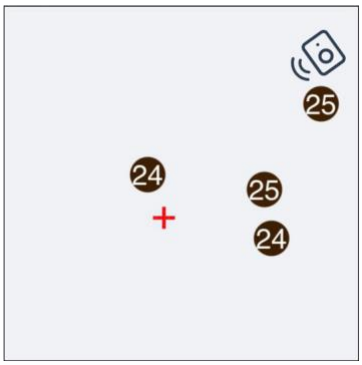


Abbildung 55 Resultat des Balancing-Tests mit nur einem Lautsprecher.

Derselbe Test wurde erneut durchgeführt, jedoch nur noch mit einem Lautsprecher. Auch in diesem Fall sollte die Lautstärke im ganzen Raum gleich laut gemessen werden, wenn das System korrekt funktioniert.

Die Resultate dieses Tests werden in Abbildung 55 aufgezeigt. Die Position des Lautsprechers wurde ebenfalls wieder zusätzlich eingezeichnet. Es wurden an allen Positionen Werte im Bereich von 24-25 gemessen.

4.3.2 Personenerkennung

Die Personenerkennung wurde anhand von zwei Merkmalen getestet. Als Erstes wird die Performance wird in FPS bewertet, was widerspiegelt, wie oft pro Sekunde ein Resultat vom Algorithmus vorliegt. Zusätzlich wird als Zweites gemessen, wie oft Personen korrekt erkannt worden sind und wie viele Objekte fälschlicherweise als Personen erkannt wurden.

Tabelle 3 Performancevergleich der Personenerkennungsalgorithmen.

Algorithmus	FPS vor Optimierungen	FPS nach Optimierungen
Histogram of oriented gradients	0.6	0.7
YOLOv3	1.0	2.0
Motion Detection	5.1	5.4

In Tabelle 3 wird die Performance jeweils vor und nach den Optimierungen, welche in Kapitel 3.10 aufgezeigt wurden, gemessen. Für das Resultat wurde jeweils die durchschnittliche FPS über eine Minute verwendet. Es ist ersichtlich, dass der YOLOv3 Algorithmus am meisten von den Optimierungen profitiert und genau doppelt so viele Bilder pro Sekunde verarbeiten kann wie vor den Optimierungen. Beim den Histogram of oriented gradients und Motion Detection Algorithmen sind nur kleine Verbesserungen zu verzeichnen.

Nach den Optimierungen hat der Motion Detection immer noch eine 2.7-mal bessere Performance als YOLOv3 und eine etwa 7.7-mal bessere als der Histogram of oriented gradients Algorithmus.

Tabelle 4 Vergleich erkannte Personen und False-Positives in 10 Bildern.

Algorithmus	Erkannte Personen	False-Positives
Histogram of oriented gradients	7	3
YOLOv3	8	0
Motion Detection	9	3

Für den zweiten Vergleich in Tabelle 4 wurden die Resultate der Algorithmen in 10 verschiedenen Situationen analysiert. Dabei befand sich die Person an verschiedenen Orten in mehreren Testräumen und wurde teilweise durch Hindernisse verdeckt. Fotos der Situationen, sowie die Resultate der Algorithmen, sind im Anhang unter 7.2.2 aufgeführt.

Der Motion Detection Algorithmus konnte in 9 von 10 Bildern die Person erkennen. YOLOv3 erzielte ebenfalls ein gutes Resultat mit 8 erkannten Personen. Einzig wenn

sich die Person nicht stark vom Hintergrund und den Gegenständen abgehoben hat, oder mit grellem Licht von hinten beleuchtet wurde, konnte die Person nicht erkannt werden. Der Histogram of oriented gradients Algorithmus erzielte mit 7 von 10 Personen das schlechteste Resultat in diesem Vergleich. Auch in Situationen, in denen die Testperson klar und unverdeckt sichtbar war, konnte diese nicht erkannt werden. Ebenfalls wurde im Vergleich mit den False-Positives kein gutes Resultat erzielt. Dieser Wert gibt an, in wie vielen Bildern die Person an einem falschen Ort erkannt wurde, wobei 0 das beste Resultat darstellt. Alle drei False-Positives vom Histogram of oriented gradients Algorithmus stammen aus einem Testraum mit vielen verschiedenen Objekten. Weil Motion Detection nur Bewegungen und keine Personen sucht, wurde ebenfalls ein Fernseher mit laufendem Programm und Schatten der Personen fälschlicherweise erkannt. Mit keinem einzigen False-Positive erzielt YOLOv3 in diesem Test das beste Resultat.

5 Diskussion

In diesem Abschnitt werden die erreichten Resultate dieser Bachelorarbeit analysiert und mit einem Rückblick auf die ursprüngliche Zielsetzung verglichen. Ebenfalls wird mit einem Ausblick untersucht, welche weiteren Schritte nach dem Vollenden dieser Arbeit möglich sind.

5.1 Resultate

Die Resultate der im Kapitel 4.3 durchgeführten Tests erfüllen alle Erwartungen und gesetzten Ziele. Spezifisch werden nun auf die Resultate des Balancings und der Personenerkennungs-Algorithmen eingegangen.

5.1.1 Balancing

Die Resultate der Tests zeigen klar die Effektivität des Systems. Mit aktiviertem Balancing weichen alle gemessenen Werte ± 1 von der Durchschnittslautstärke ab, was den Erwartungen entspricht. Die geringen Abweichungen können auf Messungenauigkeiten und Rundungen zurückgeführt werden, da sie in den Tests für den Benutzer nicht hörbar waren.

Es hat sich auch gezeigt, dass das System bereits bei nur einem Lautsprecher wie gewünscht funktioniert und die Lautstärke im ganzen Raum gleich laut wahrgenommen wird. Somit kann der Einsatzort auch auf Räume ausgeweitet werden, die nur über einen Lautsprecher verfügen.

5.1.2 Personenerkennung

Bezüglich der Performance haben sowohl YOLOv3 als auch Motion Detection gute Resultate erzielt. Mit 2 FPS von YOLOv3 kann die Lautstärke zweimal pro Sekunde angepasst werden. Dies ist für die Anwendung von Real-Stereo ausreichend, da erwartet werden kann, dass sich eine Person innerhalb eines Raumes mit normaler Geschwindigkeit fortbewegt. Bei einer Gehgeschwindigkeit von 4 km/h sind es pro Sekunde 1.11 Meter. Mit einer Rate von 2.0 FPS kann demnach alle 0.55 Meter die Lautstärke neu eingestellt werden. Die 5.4 FPS von Motion Detection sind eine Verbesserung, aber nicht zwingend notwendig für ein ausgeglichenes Hörgefühl. Im Falle von Histogram of oriented gradients wäre ein Balancing jedoch nur höchstens alle 1.5 Meter möglich, was als ungenügend angesehen wird.

Gleiches spiegelt sich bei der Zuverlässigkeit der Algorithmen wider. Histogram of oriented gradients erkennt am wenigsten Personen korrekt und hat gleichzeitig, zusammen mit Motion Detection, am meisten False-Positives. Aus diesem Grund ist der Histogram of oriented gradients Algorithmus nicht zu empfehlen und nur einzusetzen,

wenn die anderen zwei Algorithmen im verwendeten Raum und Umfeld keine guten Resultate erzielen.

Motion Detection erkannte zwar mehr Personen korrekt als YOLOv3, lieferte aber auch mehr False-Positives. Im Vergleich sind die False-Positives mehr zu gewichten, weil sich dadurch die Lautstärke im Raum merklich ändert, wenn eine zusätzliche Person fälschlicherweise an einem anderen Ort im Raum erkannt wird. Wird eine Person hingegen nur nicht erkannt, ist dies weniger zu gewichten. Minimale Bewegungen können dazu führen, dass im nächsten Frame die Person wieder erkannt wird. Ebenfalls kann davon ausgegangen werden, dass sich die Person öfters am gleichen Ort befindet, als dass sie sich bewegt. Somit ist es nicht zwingend erforderlich, diese in jedem Frame zu erkennen.

Aufgrund dieser Gewichtung bei der Zuverlässigkeit und der guten Resultate in der Performance, hat sich YOLOv3 als geeignetster Algorithmus für die Personenerkennung herausgestellt. Dieser wird somit empfohlen und als Standard festgelegt. Sollte er jedoch in einem Raum keine guten Resultate liefern, da beispielsweise viele Objekte die Sicht verdecken, kann auf Motion Detection zurückgegriffen werden, welcher ebenfalls gute Resultate erzielt hatte.

5.2 Rückblick Zielsetzung

Die in Abschnitt 1.2 aufgeführte Zielsetzung wurde durch die vorliegende Bachelorarbeit erreicht und durch zusätzliche Funktionen und Konfigurationsmöglichkeiten erweitert. Das Real-Stereo System lokalisiert mittels mehreren Tracking Nodes den Hörer im Raum und balanciert die Lautstärke von mehreren Sonos™ Lautsprechern, sodass der Zuhörer von allen Positionen aus die Lautstärke gleich laut wahrnimmt. Die Steuerung der Lautsprecher und das Tracking des Zuhörers werden dabei durch ein modulares und einfach konfigurierbares System von Tracking und Coordinator Nodes durchgeführt. Die Hardware der Nodes ist darauf ausgelegt, kostengünstig und performant die gewünschte Funktion durchzuführen. Dadurch kann der Benutzer das Real-Stereo System einfach durch weitere Tracking Nodes erweitern. Die kabellosen Kommunikationstechnologien der ausgewählten Hardware wurden in der Software verwendet, um eine möglichst einfache Installation des Real-Stereo Systems zu ermöglichen.

5.3 Ausblick

Während es noch weiteres Verbesserungspotential gibt, kann das Real-Stereo System als marktreif angesehen werden. Die Nutzerfreundlichkeit und Performance des Systems ist für den täglichen Einsatz durch Laien geeignet. Die Unterstützung des weitverbreiteten Sonos™ Lautsprechersystems in Real-Stereo ermöglicht die Weiterverwendung eines bestehendes Lautsprechersystems für den Benutzer.

Als nächster Schritt könnten die Details zur möglichen Vermarktung, Produktion und Vertrieb ausgearbeitet werden.

6 Verzeichnisse

6.1 Literaturverzeichnis

[1] M. Berchtold und C. Wanner, «Real-Stereo», ZHAW Zürcher Hochschule für Angewandte Wissenschaften, Winterthur, 2020.

[2] J. u. F. A. Redmon. (2018). *YOLOv3: An Incremental Improvement* [Online]. URL: https://pjreddie.com/darknet/yolo/ [Stand: 18.05.2021]

[3] Raspberry Pi Foundation. (o. J.). *Buy a Raspberry Pi 4 Model B* [Online]. URL: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/ [Stand: 19.05.2020]

[4] Nvidia. (o. J.). *Jetson Modules* [Online]. URL: https://developer.nvidia.com/embedded/jetson-modules [Stand: 19.05.2021]

[5] aiohttp maintainers. (06.03.2021). *Welcome to AIOHTTP* [Online]. URL: https://docs.aiohttp.org/en/stable/ [Stand: 18.05.2021]

[6] Waveshare. (o. J.). *RPi Camera (I), Fisheye Lens* Waveshare, [Online]. URL: https://www.waveshare.com/RPi-Camera-I.htm [Stand: 17.05.2021]

[7] M. Jouni. (12.01.2013). *Linux WPA/WPA2/IEEE 802.1X Supplicant* [Online]. URL: https://w1.fi/wpa_supplicant/ [Stand: 28.05.2021]

[8] Raspberry Pi Foundation. (28.05.2020). *Setting up a wireless LAN via the command line* [Online]. URL: https://www.raspberrypi.org/documentation/configuration/wireless/wireless-cli.md [Stand: 28.05.2021]

[9] S. Pigeon. (o. J.). *Grey Noise* [Online]. URL: https://www.audiocheck.net/testtones_greynoise.php [Stand: 10.06.2021]

[10] S. Pigeon. (o. J.). *White Noise* [Online]. URL: https://www.audiocheck.net/testtones_whitenoise.php [Stand: 10.06.2021]

[11] Mozilla and individual contributors. (19.02.2021). *AudioContext* [Online]. URL: https://developer.mozilla.org/en-US/docs/Web/API/AudioContext [Stand: 19.05.2021]

[12] Mozilla and individual contributors. (30.03.2021). *AnalyserNode* [Online]. URL: https://developer.mozilla.org/en-US/docs/Web/API/AnalyserNode [Stand: 19.05.2021]

[13] H. R. Straub. (26.12.2020). *Rechnen mit Frequenzen und Intervallen* [Online]. URL: https://hrstraub.ch/rechnen-mit-frequenzen-und-intervallen/ [Stand: 10.06.2021]

[14] E. Sengpiel. (01.06.2014). *Schallmessung (Geräuschmessung)* [Online]. URL: http://www.sengpielaudio.com/Rechner-dba-spl.htm [Stand: 19.05.2021]

[15] F. Di Tommaso und F. Dell' Accio, «Scattered data interpolation by Shepard's like methods: Classical results and recent advances», *Dolomites Research Notes on Approximation,* Nr. 9, S. 32-44, 2016.

[16] Sonos Inc.. (o. J.). *Sonos Developer* [Online]. URL: https://developer.sonos.com/build/direct-control/control/ [Stand: 26.05.2021]

[17] SoCo. (o. J.). *SoCo (Sonos Controller)* [Online]. URL: https://github.com/SoCo/SoCo [Stand: 26.05.2021]

[18] Google Inc.. (o. J.). *Protocol Buffers | Google Developers* [Online]. URL: https://developers.google.com/protocol-buffers [Stand: 23.05.2021]

[19] Facebook Inc.. (o. J.). *React - A JavaScript library for building user interfaces* [Online]. URL: https://reactjs.org [Stand: 21.05.2021]

[20] D. Arrachequesne. (o. J.). *Socket.IO* [Online]. URL: https://socket.io [Stand: 21.05.2021]

[21] Microsoft. (o. J.). *TypeScript: Typed JavaScript at Any Scale* [Online]. URL: https://www.typescriptlang.org [Stand: 27.05.2021]

[22] M. a. i. contributors. (19.02.2021). *MediaDevices.getUserMedia()* [Online]. URL: https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices/getUserMedia [Stand: 23.05.2021]

[23] Python Software Foundation. (2021). *multiprocessing — Process-based parallelism* [Online]. URL: https://docs.python.org/3/library/multiprocessing.html#multiprocessing.Queue [Stand: 25.05.2021]

[24] A. Rosebrock. (09.10.2017). *Optimizing OpenCV on the Raspberry Pi* [Online]. URL: https://www.pyimagesearch.com/2017/10/09/optimizing-opencv-on-the-raspberry-pi/ [Stand: 25.05.2021]

[25] Arm Ltd.. (o. J.). *Floating Point* [Online]. URL: https://developer.arm.com/architectures/instruction-sets/floating-point [Stand: 25.05.2021]

[26] Arm Ltd.. (o. J.). *Neon* [Online]. URL: https://developer.arm.com/architectures/instruction-sets/simd-isas/neon [Stand: 25.05.2021]

[27] Intel Corporation. (19.05.2021). *Intel® oneAPI Threading Building Blocks* [Online]. URL: https://software.intel.com/content/www/us/en/develop/tools/oneapi/components/onetbb.html [Stand: 25.05.2021]

[28] Totonic GmbH. (o. J.). *Raspberry Pi 4 Gehäuse, Schwarz/Grau* [Online]. URL: https://www.pi-shop.ch/raspberry-pi-4-gehaeuse-schwarz-grau [Stand: 31.05.2021]

[29] Focus Online. (07.02.2020). *Günstige Mini-PCs für Multimedia, Smart Home und Co.: Die stärksten Modelle im Überblick* [Online]. URL: https://www.focus.de/digital/computer/raspberry-pi-und-alternativen-guenstige-mini-pcs-fuer-multimedia-smart-home-und-co-die-staerksten-modelle-im-ueberblick_id_11637348.html [Stand: 31.05.2021]

6.2 Abbildungsverzeichnis

6.3 Tabellenverzeichnis

6.4 Abkürzungsverzeichnis

Abkürzung	Bedeutung
AIOHTTP	Asynchronous I/O HTTP, eine Library, welches einen Webserver basierend auf dem asyncio Modul von Python zur Verfügung stellt.
DHCP	Dynamic Host Configuration Protocol ist ein Netzwerkprotokoll, welches die Zuweisungen der IP-Adressen innerhalb eines Netzwerkes managed.
FPS	Frames per second ist eine Messeinheit, die angibt, wie oft pro Sekunde ein Bild verfügbar ist oder verarbeitet werden kann.
GPU	Graphical processing unit ist ein für Bild- und Videoverarbeitung optimierter Prozessor.
RTMP	Real-Time Messaging Protokoll ist ein Echtzeit Protokoll und ursprünglich für das Streaming von Video und Audio entwickelt.
SSID	Service Set Identifier bezeichnet den Namen eines Netzwerkes.
UPnP	UPnP ist ein Service Discovery Protokoll, welches das Entdecken von anderen Geräten ermöglicht.
WPA	Wi-Fi Protected Access ist ein Sicherheitsstandard für kabellose Netzwerke und definiert die Authentifizierung und Verschlüsselung.

7 Anhang

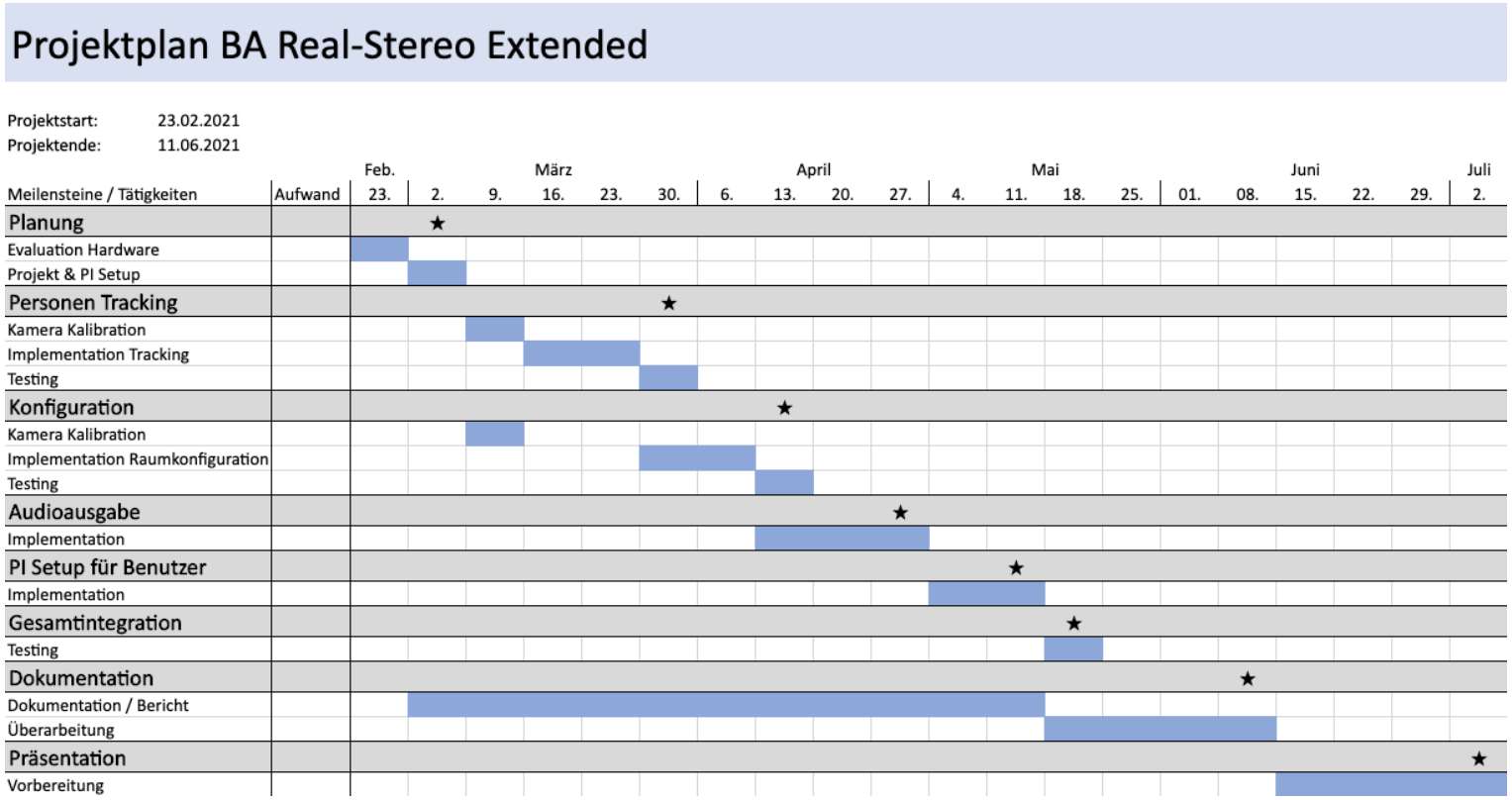
7.1 Projektmanagement

7.1.1 Offizielle Aufgabenstellung

In der Projektarbeit Real-Stereo wurde eine Applikation entwickelt, um die Lautstärke mehrerer Audio Kanäle aufgrund der physischen Position des Zuhörers im Raum zu steuern. In dieser Bachelor Arbeit soll dieses Projekt, welches als eine reine Software Lösung implementiert wurde, in ein eigenständiges Hardware- und Software-System übertragen werden. Dies soll eine effektive und praktikable Nutzung im Alltag ermöglichen.

1. In der Projektarbeit angesprochene, mögliche Verbesserungen (wie etwa die Unterstützung von mehreren Personen) sollen implementiert werden.
2. Der existierende Balancing-Algorithmus soll auf einer passenden Hardware Plattform implementiert werden. Hierfür muss dieser eventuell in eine andere Programmiersprache übersetzt werden.
3. Es muss sichergestellt werden, dass die Performance auf der gewählten Hardware-Plattform genügend ist.
4. Das User-Interface muss an die neue Plattform angepasst werden, eventuell auf ein Web-basiertes UI, so dass keine zusätzlichen Bildschirme und Eingabegeräte benötigt werden.
5. Es sollen auch andere Audio Geräte, wie zum Beispiel Sonos, für das Balancing unterstützt werden.

7.1.2 Zeitplan



7.1.3 Arbeitszeiten

Arbeitszeiten von Cyril Wanner

Datum	Aufwand	Total	Beschreibung Tätigkeit
26.02.21	3.00	3.00	Vergleich Mikrocomputer & Kamera
27.02.21	1.50	4.50	Vergleich Mikrocomputer & Kamera
01.03.21	2.00	6.50	Projektplan
01.03.21	1.00	7.50	Web Audio Context Tests
02.03.21	1.00	8.50	Meeting
02.03.21	5.00	13.50	Python Projekt setup
03.03.21	2.00	15.50	OpenCV setup
04.03.21	2.00	17.50	Dev environment improvement
05.03.21	3.00	20.50	Spezifikation Web Protokoll
05.03.21	3.00	23.50	API Update mit neuem Protokoll
06.03.21	6.00	29.50	Kamera / Tracking / Web Server Implementierung
07.03.21	4.00	33.50	Kamera / Tracking / Web Server Implementierung
08.03.21	2.00	35.50	GitHub Issue #11 behoben, reviews
09.03.21	3.50	39.00	Meeting, Bestellliste Hardware
09.03.21	5.50	44.50	Config handling, rooms api controller
10.03.21	5.50	50.00	API Architektur verbessert, fehlende Endpunkte
13.03.21	6.00	56.00	Cluster Protokoll
15.03.21	4.00	60.00	Cluster Protokoll: Service acquisition & release
16.03.21	1.50	61.50	Meeting, Bugfix
18.03.21	4.00	65.50	Material abholen, BA Inputs, Testing recording
19.03.21	2.50	68.00	Sonos setup, testing
20.03.21	6.00	74.00	PI setup, installation & development scripts
23.03.21	5.00	79.00	Meeting, testing, asyncio
24.03.21	5.00	84.00	Asyncio, camera
25.03.21	5.00	89.00	Camera
26.03.21	3.00	92.00	Camera finish, testtone recording testing
27.03.21	1.00	93.00	Recording testing, android fixes
28.03.21	6.00	99.00	Camera calibration
29.03.21	4.00	103.00	Camera calibration
30.03.21	1.50	104.50	Meeting, Calibration improvements
31.03.21	4.50	109.00	HoG Person Detection, Review
01.04.21	1.50	110.50	Detection improvement, temperature tests
06.04.21	1.00	111.50	Meeting
06.04.21	4.00	115.50	Implement multiprocessing
07.04.21	3.00	118.50	Multiprocessing fixes
10.04.21	3.00	121.50	Detection algorithms (architecture)
11.04.21	3.00	124.50	YOLOv3 object detection
12.04.21	3.00	127.50	YOLOv3 object detection & select algorithm per node
13.04.21	1.00	128.50	Meeting
13.04.21	4.00	132.50	People detection based on motion
18.04.21	3.00	135.50	Improvements, coordinate calculation
20.04.21	1.00	136.50	Meeting
25.04.21	4.00	140.50	Coordinate calculation, tracking
26.04.21	3.00	143.50	Coordinate calculation, tracking
26.04.21	1.50	145.00	Optimizing opencv
27.04.21	4.00	149.00	Optimizing opencv, config.json bugfix
28.04.21	4.00	153.00	Bugfixing opencv optimizations
29.04.21	1.50	154.50	Reviews
30.04.21	1.50	156.00	Coordinate implementation during calibration
01.05.21	3.00	159.00	SSL error, volume interpolation
02.05.21	6.00	165.00	Volume interpolation, speaker balancing
03.05.21	4.50	169.50	Allow user volume changes, reviews
04.05.21	1.00	170.50	Meeting
05.05.21	3.50	174.00	Bugfixes, minor improvements, reviews
06.05.21	3.50	177.50	Bugfixes, improvements
07.05.21	3.00	180.50	Session renewal fix, ad hoc network
08.05.21	6.00	186.50	Ad hoc network

Datum	Aufwand	Total	Beschreibung Tätigkeit
09.05.21	4.00	190.50	Ad hoc network
10.05.21	8.00	198.50	Ad hoc network
11.05.21	1.00	199.50	Meeting
12.05.21	2.00	201.50	Fix opencv optimization
13.05.21	2.00	203.50	Tests, Bericht
16.05.21	2.00	205.50	Bugfixing
17.05.21	3.50	209.00	Testing, Bericht (Kameras)
18.05.21	3.50	212.50	Bericht (Kameras)
19.05.21	4.00	216.50	Bericht (Personenerkennung), Bugfixes
21.05.21	4.00	220.50	Bericht (Lokalisierung)
23.05.21	2.00	222.50	Bericht (Protokoll)
24.05.21	4.00	226.50	Bericht (Protokoll)
25.05.21	3.00	229.50	Meeting, Bericht (diverse Überarbeitungen, Performance Optimierungen)
27.05.21	4.00	233.50	Bericht (Balancing, Performance Optimierungen)
28.05.21	2.50	236.00	Bericht (AdHoc Netzwerk)
30.05.21	2.00	238.00	More test mode settings
31.05.21	6.00	244.00	Bericht (Theoretische Grundlagen, Resultate, Diskussion, Überarbeitungen)
01.06.21	6.00	250.00	Testing INIT, Vorbereitungen
03.06.21	2.00	252.00	Schreibberatung, Testing bugfixes
04.06.21	5.00	257.00	Bericht (Resultate, Personenerkennung tests)
05.06.21	5.50	262.50	Bericht (Resultate, Diskussion)
06.06.21	2.00	264.50	Bericht (Überarbeitung)
07.06.21	2.00	266.50	Bericht (Überarbeitung)
08.06.21	4.00	270.50	Bericht (Überarbeitung)
09.06.21	4.00	274.50	Bericht (Überarbeitung)
10.06.21	6.00	280.50	Bericht (Überarbeitung)
11.06.21	6.00	286.50	Bericht (Finalisierung, Abgabe)
27.06.21	3.00	289.50	Vorbereitung Präsentation (geplant)
28.06.21	3.00	292.50	Vorbereitung Präsentation (geplant)
29.06.21	3.00	295.50	Vorbereitung Präsentation (geplant)
30.06.21	3.00	298.50	Vorbereitung Präsentation (geplant)
01.07.21	3.00	301.50	Vorbereitung Präsentation (geplant)
02.07.21	2.00	303.50	Präsentation (geplant)

Arbeitszeiten von Marc Berchtold

Datum	Aufwand	Total	Beschreibung Tätigkeit
23.02.21	2.50	2.50	Meeting Protokoll & Research Audioausgabe (Toslink & HDMI)
24.02.21	2.00	4.50	Vergleich Audioausgabe
01.03.21	2.50	7.00	Research Test Ton Aufnahme
02.03.21	0.50	7.50	Meeting
02.03.21	2.50	10.00	Webapp React basic setup
03.03.21	2.50	12.50	Webapp Wireframes erste Version
04.03.21	1.50	14.00	Webapp Wireframes Revision nach Feedback & Diskussion
05.03.21	3.00	17.00	Webapp sidenav & basic layout implementiert
07.03.21	5.00	22.00	Webapp router & socket.io architecture
08.03.21	4.50	26.50	Webapp Socket.io Architektur Revision mit intelligentem Verbindungsaufbau
09.03.21	1.00	27.50	Meeting & Protokoll
09.03.21	4.00	31.50	Webapp EditRooms markup
10.03.21	4.50	36.00	Webapp EditRoom markup & logic
11.03.21	3.00	39.00	Webapp add speakers to rooms
13.03.21	3.50	42.50	Webapp add speakers to rooms & overview page
14.03.21	2.00	44.50	Webapp nodes list
15.03.21	3.00	47.50	Webapp edit & delete nodes
16.03.21	5.00	52.50	Webapp TestMode markup & AudioContext Recherche und Code
17.03.21	4.00	56.50	Webapp VolumeMeter finished
18.03.21	3.50	60.00	Installation Sonos Lautsprecher & Recherche Sonos Control API
20.03.21	1.50	61.50	Kamera Plazierung geplant & Loch für Kamera in Gehäuse gebohrt
21.03.21	2.50	64.00	Loch für Kamera in Gehöuse gebohrt 2 & Raspberry Pis aufgesetzt
21.03.21	3.50	67.50	Sonos discovery
22.03.21	4.00	71.50	Sonos discovery und Weiterleitung in Repository
23.03.21	3.00	74.50	Sonos Adapter pattern

Datum	Aufwand	Total	Beschreibung Tätigkeit
23.03.21	0.50	75.00	Meeting & Protokoll
23.03.21	6.00	81.00	Sonos Adapter pattern & Volumen in Stereo Pairs anpassen
24.03.21	1.00	82.00	Code review asyncio
25.03.21	4.00	86.00	Webapp Spectrum Analyzer &Kalibrationston Research
26.03.21	5.00	91.00	Webapp Kalibrationston Research & bessere Volumenmessung
29.03.21	2.00	93.00	Research Kalibrationston Generation
30.03.21	1.50	94.50	Meeting, Code review & Weisses Rauschen mit Audacity generiert
31.03.21	3.00	97.50	Wiedergabe Kalibrationston
01.04.21	1.50	99.00	Code review person detection
04.04.21	2.50	101.50	Heat spreader gebastelt um Überhitzung zu reduzieren, nicht gut genug ohne zusätzlichen Lüfter
05.04.21	4.00	105.50	Mit Raumkalibration angefangen, Klassen für die Speicherung der Kalibrationspunkte
06.04.21	3.50	109.00	Raumkalibration
07.04.21	4.00	113.00	Raumkalibration Flowchart & Architektur
08.04.21	4.50	117.50	Raumkalibration Start & automatisches Client update
09.04.21	3.00	120.50	Raumkalibration Tracking in bestimmten Momenten aktivieren & deaktivieren
11.04.21	5.50	126.00	Raumkalibration Webapp Komponente & aktuelle Position der Person anzeigen
12.04.21	6.00	132.00	Raumkalibration Alle Lautsprecher kalibrieren und Fortschritt Benutzer anzeigen
13.04.21	0.50	132.50	Meeting
13.04.21	4.50	137.00	SSL Zertifikat automatisch generieren
14.04.21	1.50	138.50	Code review motion detection & Überlegungen zu HTTPS Verbindung
19.04.21	4.50	143.00	Proxy Kamera Stream & Assets über Master Node für HTTPS Verbindung
20.04.21	1.00	144.00	Meeting & Protokoll
20.04.21	4.50	148.50	Raumkalibration Kontrollfluss Änderungen & HTTP Server für Sonos Testsound
25.04.21	6.00	154.50	Raumkalibration abspeichern & laden, Position wiederholen oder bestätigen
26.04.21	4.50	159.00	Raumkalibration mehr User Feedback & Median verwenden für Lautstärke & Code review
27.04.21	1.50	160.50	Meeting & OpenCV Kompilierung für Raspberry Pis
28.04.21	2.00	162.50	OpenCV Kompilierung Fehlerbehebung
29.04.21	1.50	164.00	Bugfix Node & Raum Formular
30.04.21	2.00	166.00	Bugfix Node wird nicht korrekt gelöscht auf Anfrage
30.04.21	2.50	168.50	Raumkalibration bisherige Kalibrationspunkte auf der Karte anzeigen
02.05.21	1.50	170.00	Raumkalibration bisherige Kalibrationspunkte nummerieren
03.05.21	5.00	175.00	Balancing testen, bugfix, Kalibrationslautstärke anpassbar, Sonos automatisch gruppieren für Kalibration
04.05.21	4.00	179.00	Interpolation auch im Webapp berechnen & Funktion welche sie visuell darstellt
08.05.21	3.00	182.00	Interpolation bei Kalibration auf Karte anzeigen, Benutzer Lautsprecher für die Darstellung wählen lassen
09.05.21	4.00	186.00	Testmodus Kalibrationston starten & stoppen und Position an webapp senden, wenn testmodus aktiv ist
10.05.21	4.50	190.50	Testmodus Volumen aufnehmen, ausrechnen & in Karte einzeichnen
15.05.21	3.50	194.00	Bugfix Grosse Verzögerung bis Testton wiedergegeben wird
15.05.21	3.00	197.00	Bugfix Ausgesteckte Sonos Lautsprecher von Liste entfernen
16.05.21	2.00	199.00	Bericht Zusammenfassung & Management Summary
17.05.21	3.00	202.00	Bugfix Falsche Volumen bei Kalibration mit Stereo Sonos Lautsprechern & weitere Bugfixes
18.05.21	3.00	205.00	Bericht Management Summary & Embedded Platform, Bugfix ausgesteckte Sonos Lautsprecher entfernen
19.05.21	4.00	209.00	Bericht Embedded Platform Tracking Node & Raumkalibration Testgeräusch Aufnahme
20.05.21	3.00	212.00	Bericht Testgeräusch
21.05.21	3.50	215.50	Bericht User Interface
23.05.21	4.00	219.50	Bericht User Interface Zertifikat & Screenshots & Bilder formatieren für User Interface
24.05.21	2.50	222.00	Bericht Vorwort & Resultate User Interface
26.05.21	3.50	225.50	Bericht Sonos Schnittstelle, Ausgangslage & Zielsetzung
27.05.21	3.50	229.00	Bericht User Interface Kommunikationsprotokoll

Datum	Aufwand	Total	Beschreibung Tätigkeit
31.05.21	3.00	232.00	Code review TestModus & Bericht Resultate System & Diskussion Rückblick & Ausblick
02.06.21	5.50	237.50	Testen vor Ort INIT TD und Vorbereitungen
03.06.21	3.00	240.50	Schreibberatung Feedback, Verbesserungen Schreibberatung Zusammenfassung & Abstract
04.06.21	3.00	243.50	Verbesserungen von Schreibberatung
06.06.21	3.50	247.00	Resultate Konfiguration & Verbesserungen von Schreibberatung
08.06.21	3.00	250.00	Verbesserungen von Schreibberatung
09.06.21	2.50	252.50	Verbesserungen im Bericht & Fast Fourier Transformation Theorie
10.06.21	4.00	256.50	Korrekturlesen Bericht & Veröffentlichung von Vorschau im BAWebTool
11.06.21	5.00	261.50	Bericht Abgabe & letzte Verbesserungen
27.06.21	3.00	264.50	Vorbereitung Präsentation (geplant)
28.06.21	3.00	267.50	Vorbereitung Präsentation (geplant)
29.06.21	3.00	270.50	Vorbereitung Präsentation (geplant)
30.06.21	3.00	273.50	Vorbereitung Präsentation (geplant)
01.07.21	3.00	276.50	Vorbereitung Präsentation (geplant)
02.07.21	2.00	278.50	Präsentation (geplant)

7.1.4 Besprechungsprotokolle

Die Besprechungsprotokolle können im Ordner *documents/meeting_protocols* der Abgabe oder des GitHub Repository des Projektes (https://github.com/ItsEcholot/real-stereo-extended) gefunden werden. Die Protokolle sind jeweils im PDF Format abgelegt und hier als Übersicht aufgelistet.

```
documents/
├─ meeting_protocols/
│  ├─ Beschlussprotokoll 23.02.2021.pdf
│  ├─ Beschlussprotokoll 02.03.2021.pdf
│  ├─ Beschlussprotokoll 09.03.2021.pdf
│  ├─ Beschlussprotokoll 16.03.2021.pdf
│  ├─ Beschlussprotokoll 23.03.2021.pdf
│  ├─ Beschlussprotokoll 30.03.2021.pdf
│  ├─ Beschlussprotokoll 06.04.2021.pdf
│  ├─ Beschlussprotokoll 13.04.2021.pdf
│  ├─ Beschlussprotokoll 20.04.2021.pdf
│  ├─ Beschlussprotokoll 27.04.2021.pdf
│  ├─ Beschlussprotokoll 04.05.2021.pdf
│  ├─ Beschlussprotokoll 11.05.2021.pdf
│  ├─ Beschlussprotokoll 25.05.2021.pdf
```

7.2 Weiteres

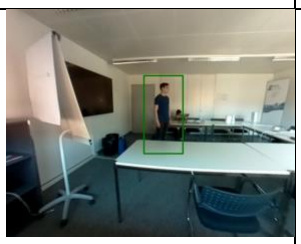
7.2.1 Quellcode

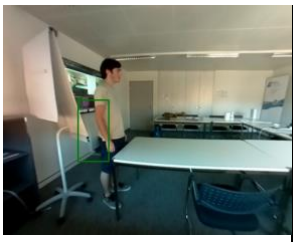
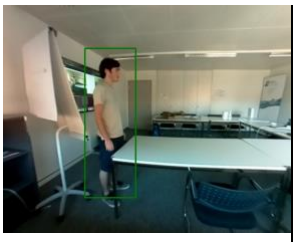
Der Quellcode ist im Ordner *code* der Abgabe abgelegt oder kann im GitHub Repository des Projektes (https://github.com/ItsEcholot/real-stereo-extended) eingesehen werden. Die Unterordner sind hier aufgelistet und deren Inhalt kurz erläutert.

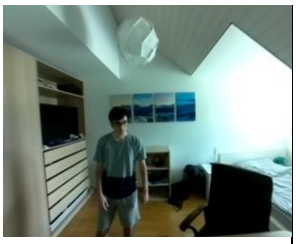
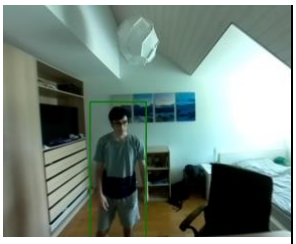
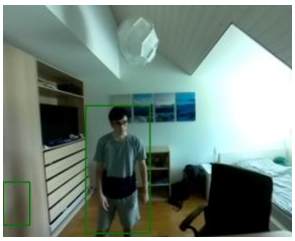
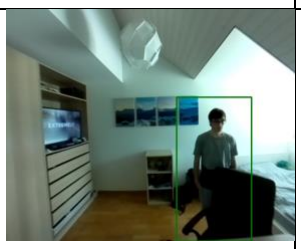
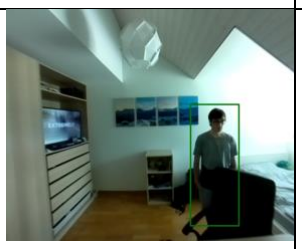
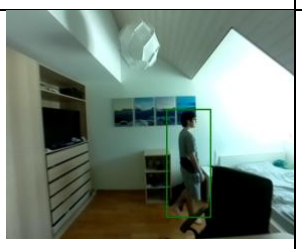
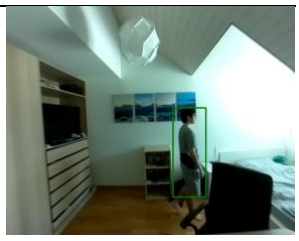
```
backend/                        Code für das Backend
├── assets/                     Schachbrettmuster, Rauschen,
│   │                           Standard-Kamera-Kalibration
│   ├── yolo/                   Konfigurationsdaten für YOLO
├── src/
│   ├── api/                    Web API für das Frontend
│   │   ├── controllers/
│   ├── balancing/              Interpolation & Berechnungen
│   ├── config/                 Speichert & Lädt die
│   │                           Konfiguration
│   ├── models/                 Datenstrukturen
│   ├── networking/             AdHoc Netzwerk, WLAN-
│   │                           Verbindungs-Informationen
│   ├── protocol/               Netzwerk-Protokoll zwischen
│   │   │                       Real-Stereo Nodes
│   │   ├── master/             Coordinator Node
│   │   ├── slave/              Tracking Node
│   ├── repositories/           Datenstrukturen
│   ├── sonos/                  Sonos-Schnittstelle
│   ├── tracking/               Personenerkennung & -tracking
documents/                      Spezifikationen
frontend/                       Code für das Frontend
├── public/                     Statische Dateien
├── src/
│   ├── assets/                 Bilder
│   ├── components/             Wiederverwendbare Komponenten
│   │   ├── Calibration/
│   │   ├── Header/
│   │   ├── Layout/
```

```
│   │   ├── MainMenu/
│   │   ├── NodeSetupType/
│   │   ├── RequireSetup/
│   ├── pages/                  Aufrufbare Seiten
│   │   ├── AddCameraNodes/
│   │   ├── CalibrateCamera/
│   │   ├── EditNode/
│   │   ├── EditRoom/
│   │   ├── EditRooms/
│   │   ├── Overview/
│   │   ├── Setup/
│   │   ├── SetupFinished/
│   │   ├── TestMode/
│   ├── services/               Schnittstellen zur API
scripts/                        Scripts zum Setup vom
│                               Raspberry Pi und OpenCV
├── config/                     Configs für Scripts & Services
wireframes/                     Frontend Wireframes
```

7.2.2 Resultate Personenerkennung

	Histogram of oriented gradients	YOLOv3	Motion Detection
Situation 1			
	Person erkannt: Ja False-Positives: 0	Person erkannt: Ja False-Positives: 0	Person erkannt: Ja False-Positives: 0
Situation 2			
	Person erkannt: Ja False-Positives: 0	Person erkannt: Ja False-Positives: 0	Person erkannt: Ja False-Positives: 0

Situation 3	 Person erkannt: Ja False-Positives: 0	 Person erkannt: Ja False-Positives: 0	 Person erkannt: Ja False-Positives: 0
Situation 4	 Person erkannt: Ja False-Positives: 0	 Person erkannt: Ja False-Positives: 0	 Person erkannt: Ja False-Positives: 0
Situation 5	 Person erkannt: Nein False-Positives: 1	 Person erkannt: Nein False-Positives: 0	 Person erkannt: Ja False-Positives: 0
Situation 6	 Person erkannt: Ja False-Positives: 1	 Person erkannt: Ja False-Positives: 0	 Person erkannt: Ja False-Positives: 0
Situation 7	 Person erkannt: Ja False-Positives: 1	 Person erkannt: Nein False-Positives: 0	 Person erkannt: Nein False-Positives: 1

Situation 8			
	Person erkannt: Nein False-Positives: 0	Person erkannt: Ja False-Positives: 0	Person erkannt: Ja False-Positives: 1
Situation 9			
	Person erkannt: Ja False-Positives: 0	Person erkannt: Ja False-Positives: 0	Person erkannt: Ja False-Positives: 1
Situation 10			
	Person erkannt: Nein False-Positives: 0	Person erkannt: Ja False-Positives: 0	Person erkannt: Ja False-Positives: 0