Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering

InIT Institut für angewandte
Informationstechnologie

Projektarbeit Informatik

Real-Stereo

Autoren	Marc Berchtold Cyril Wanner
Hauptbetreuung	Jürgen Spielberger

Datum	17.12.2020

Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering

Erklärung betreffend das selbständige Verfassen einer Projektarbeit an der School of Engineering

Mit der Abgabe dieser Projektarbeit 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 Projektarbeit 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:

Zürich, 17.12.2020 .....

Unterschriften:


.....


.....

.....

Das Original dieses Formulars ist bei der ZHAW-Version aller abgegebenen Projektarbeiten zu Beginn der Dokumentation nach dem Titelblatt mit Original-Unterschriften und -Datum (keine Kopie) einzufügen.

Zusammenfassung

Verschiedene Technologien wie Stereo- und Surround-Sound wurden erfunden, um das Hörerlebnis von verschiedenen Medien so realistisch wie möglich zu gestalten. Diese Technologien setzen aber implizit voraus, dass die verschiedenen Lautsprecher die richtigen Positionen im Raum, und vor allem eine einheitliche Distanz zum Hörer einnehmen.

In dieser Arbeit wird ein System ausgearbeitet, welches die Lautstärke der einzelnen Lautsprecher, und damit das Balancing, kontinuierlich an die Position des Hörers im Raum anpasst. Hierzu wird diskutiert, welche technischen Ansätze für die Umsetzung in Frage kommen. Die vielversprechendste Lösung wird als Software für PCs umgesetzt.

Dabei werden für die Umsetzung wichtige Grundlagen und Konzepte erarbeitet. Dazu zählen vor allem die Personenerkennung und Positionierung im zweidimensionalen Raum mittels digitalen Farbkameras, die Algorithmen, um die Lautstärken pro Lautsprecher anhand der Personenposition zu berechnen, und die Windows APIs für die Anwendung der berechneten Volumen-Werte.

Das Resultat dieser Projektarbeit ist ein Windows-Programm, welches mittels 2 USB-Webcams die Position des Hörers kontinuierlich ermittelt und die Audio-Kanäle des am Computer angeschlossenen Wiedergabegeräts anpasst, um eine uniforme Lautstärke für jeden Audio-Kanal zu ermöglichen. Die erarbeiteten Algorithmen und Lösungswege bilden eine Grundlage, auf welcher in einer weiterführenden Arbeit eine allgemeinere Lösung für den einfachen Einsatz im Alltag mit verschiedenen Audio-Geräten erarbeitet werden kann.

Abstract

Different technologies like Stereo- and Surround-Sound have been invented to create a realistic listening experience with different media formats. These technologies implicitly require that the used speakers are placed at the correct positions and, even more importantly, at the correct and uniform distance to the listener.

This paper describes a system, that adjusts the individual volumes, and thus the balancing to the spatial position of the listener in the room. To accomplish this, the different applicable technologies and strategies are discussed. The most promising solution is implemented as a software for PCs.

The fundamentals and concepts which are essential for the implementation are compiled. Special focus is put on the person detection and positioning in a two-dimensional space using digital color cameras, the algorithms which are used to calculate the volume values for each speaker using the current position of the person and the Windows APIs used to apply the calculated volume values.

The result of this project is a Windows application, which uses 2 USB-Webcams to continuously calculate the position of the listener and to adjust the audio channels of the audio playback device connected to the computer, thus creating a uniform loudness for each audio channel. The compiled algorithms and solutions pose a basis for future work, which could provide a more general applicable solution and more easy day-to-day usage with different audio devices.

Management Summary

Bei Real-Stereo handelt es sich um ein neues Produkt, welches ein weitverbreitetes Problem im Audiosektor zu lösen versucht. Verschiedene populäre Technologien wie Stereo- und Surround-Sound setzen voraus, dass der Hörer eine einheitliche Distanz zu den Lautsprechern einnimmt, um korrekt zu funktionieren und die angepriesene Leistung zu erreichen. Moderne Technologien, wie die Personenerkennung mithilfe von Farbkameras, werden eingesetzt, um ein intelligentes System zu entwickeln, welches automatisch die verschiedenen Lautstärken der Lautsprecher anpasst, um die vom Hörer wahrgenommene Lautstärke an allen Positionen im Raum uniform zu halten.

Durch die Entwicklung dieser innovativen Lösung für dieses Problem kann gegenüber der Konkurrenz ein grosser technischer Vorsprung erarbeitet werden.

Auch wenn hier als Erstes nur eine PC-basierte Lösung erstellt wurde, welche nur beschränkt als fertiges Produkt für den täglichen Einsatz angesehen werden kann, stellt es doch eine solide Basis für die untersuchten Technologien dar. Auch wird hiermit das Konzept bewiesen und in einem voll funktionsfähigen Rahmen präsentiert.

Inhaltsverzeichnis

1 Einleitung

Dieses Kapitel schildert die Ausgangslage der Arbeit und zeigt die vereinbarten Ziele auf. Ausserdem werden verwandte Arbeiten aufgeführt.

1.1 Ausgangslage

Personenerkennung und -lokalisierung ist ein Forschungsgebiet mit stetig zunehmender Beliebtheit. Es existieren bereits Lösungsansätze basierend auf verschiedenen Technologien, welche das Problem in unterschiedlichen Umgebungen zu lösen versuchen. Für diese Arbeit sind speziell Lösungen interessant, die für eine Erkennung innerhalb von Räumen optimiert wurden. Oftmals existiert jedoch das Problem, dass Personen die ganze Zeit für alle Kameras oder Sensoren sichtbar sein müssen. In einem realistischen Umfeld ist es aber möglich, dass eine Person durch Hindernisse verdeckt wird oder, aufgrund von verwinkelten Räumen, nur auf einer Kamera sichtbar ist. Diese Gegebenheiten sollen für diese Projektarbeit berücksichtigt und verbessert werden.

Auch bei Stereo- und Surround-Sound findet viel Innovation statt, obwohl dieses Gebiet bei weitem länger existiert. Gerade kombiniert mit künstlicher Intelligenz gibt es einige Forschungsarbeiten, welche die Wahrnehmung von mehrkanäligen Sound erforschen und verbessern.

Kombinationen der beiden Gebiete sind jedoch selten und für das spezifische Problem, welches sich diese Projektarbeit annimmt, wurden bisher keine Arbeiten publiziert. Es soll deshalb ein neuer Ansatz gesucht werden, welcher die beiden Gebiete mit Berücksichtigung von deren Eigenheiten optimal verbindet.

1.2 Verwandte Arbeiten

Bereits im Jahr 2010 veröffentlichten Forscher der Tsinghua Universität in Peking mit dem Paper *A Robust Approach for Person Localization in Multi-camera Environment* eine Lösung, um eine Person mit mehreren Kameras exakt lokalisieren zu können. Dabei wird die Person durch Computer Vision in allen Bildern erfasst und die Berechnung der Position auf ein lineares Optimierungsproblem heruntergebrochen, so dass keine vorgängige Kalibrierung benötigt wird [1].

Eine weitere Möglichkeit zur Lokalisierung wurde etwa von forschenden Mitarbeitenden der technischen Universität von Košice in der Slowakei präsentiert. Dabei wurde die Ultra-Wideband-Technologie in Echtzeit und in verschiedenen Szenarien eingesetzt, um sowohl direkt sichtbare, aber auch von Hindernissen verdeckte Personen zu erkennen und die Resultate zu präsentieren [2].

Den Versuch, Image Processing mit Audio zu kombinieren, unternahmen vier Forscher der Tokyo University of Science mit dem Paper *Combination of microphone array processing and camera image processing for visualizing sound pressure distribution*. Darin wird ein Vorgehen beschrieben, welches Objekte im dreidimensionalen Raum von Kameras lokalisieren und dessen Lautstärke messen kann.

Die Umsetzung unterscheidet sich in der Art, wie das Audio aufgenommen wird, grundlegend von dieser Projektarbeit. Es wurden 64 Richtmikrofone statisch in einem Array positioniert und mit den Lokalisierungsinformationen der Kameras ergänzt, so dass die Lautstärke an einem beliebigen Punkt aus der Ferne gemessen werden kann [3].

Aufgrund der grossen Anzahl benötigter Mikrofone im letzten Ansatz, ist dieses System aber nicht alltagstauglich und in gewöhnlichen Wohnungen einsetzbar. Es zeigt aber detailliert auf, wie mit einer Stereo-Kamera und Triangulation die für das Mikrofon-Array benötigten Lokalisierungsdaten berechnet werden können.

1.3 Zielsetzung

Ziel dieser Projektarbeit ist das Entwickeln eines Prototyps, welcher eine Person innerhalb eines Raumes kontinuierlich und in Echtzeit lokalisieren kann und damit die anfangs konfigurierten Lautsprecher balanciert, damit ein möglichst gleichmässiges Hörerlebnis entsteht. Dies wird definiert, in dem alle Lautsprecher vom Zuhörer als gleich laut wahrgenommen werden sollen, unabhängig von dessen aktueller Position.

Ein weiteres Ziel ist, das System möglichst einfach konfigurieren zu können. Dabei wird die Lautstärke an mehreren Positionen im Raum mit einem Mikrofon gemessen und zusammen mit den berechneten Koordinaten abgespeichert.

Während das Programm aktiv ist, soll die Position einer Person verfolgt werden und alle konfigurierten Lautsprecher stetig aufeinander abgestimmt werden. Dies soll mit einer Interpolation zwischen den aus der Konfiguration gespeicherten Werte geschehen.

Der Umfang dieser Projektarbeit beschränkt sich auf eine reine PC-Lösung, um die erarbeiteten Konzepte und Ideen und deren Umsetzung zu prüfen. Zusätzlich soll bereits recherchiert werden, ob es damit möglich ist, auch beliebte Sound-Systeme, wie beispielsweise Sonos [4], anzusteuern.

In einer fortführenden Arbeit könnte die Funktionalität auf eben genannte Systeme erweitert werden, sowie die ganze Applikation in ein eigenständiges System übersetzt werden. Mögliche Verbesserungen und Erkenntnisse dieser Arbeit sollen ebenfalls berücksichtigt werden.

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

2 Theoretische Grundlagen

Für ein besseres Verständnis der kommenden Kapitel werden in den folgenden Absätzen erforderliche technische Grundlagen erläutert.

2.1 Personenerkennung und Lokalisierung

Um die verschiedenen Lautsprecher aufgrund der aktuellen Position einer Person im Raum aufeinander abstimmen zu können, ist eine Erkennung und Lokalisierung derer notwendig. Dies soll in Echtzeit und mit möglichst wenig Aufwand geschehen. In diesem Abschnitt werden die Technologien der in Frage kommenden Lösungen beschrieben.

2.1.1 Computer Vision

Basierend auf verschiedenen, zur Verfügung stehenden Algorithmen, können mithilfe von Computer Vision auf einem Bild Personen erkannt werden. Als Resultat werden Bereiche zurückgegeben, in welchen diese erkannt wurden, zusammen mit einem Score zwischen 0 und 1. Der Score sagt aus, mit welcher Sicherheit die Entscheidung getroffen wurde.

Ist eine Person erkannt, ist die Lokalisierung auf einer eindimensionalen Ebene durch den Mittelpunkt des ermittelten Bereiches möglich. Für die Erweiterung auf den zweidimensionalen Raum ist eine zweite Kamera erforderlich. Dabei wird je eine Kamera für die Bestimmung der Position auf einer Achse verwendet.

2.1.2 Ultra-Wideband

Die Ultra-Wideband-Technologie ist auf eine hochpräzise Lokalisierung eines Senders innerhalb von Räumlichkeiten spezialisiert. Dabei wird die Flugzeit von den ausgesendeten Signalen der Sender an mehreren Basisstationen aufgezeichnet und anschliessend mittels Triangulation die Position in einem zwei- oder dreidimensionalen Raum bestimmt.

Aufgrund der Breite des verwendeten Frequenzbereichs und der starken Impulse, funktioniert das System auch mit Hindernissen zwischen Sender und Basisstationen einwandfrei.

2.2 Interpolation der Lautstärke

Während der Kalibrierung eines Raumes wird die Lautstärke von jedem Lautsprecher an mehreren Punkten gemessen und abgespeichert. Darauffolgend soll anhand dieser weniger Werte die Lautstärke im ganzen Raum interpoliert werden.

Aufgrund des zweidimensionalen Raumes, in welchem sich das Projekt befindet, ist eine multivariate Interpolation gesucht, welche eine Funktion mit mehreren Unbekannten interpoliert. Diese wird in den kommenden Abschnitten beschrieben.

3 Methoden

In diesem Kapitel wird beschrieben, welche Methoden angewendet wurden, um die gesetzten Ziele zu erreichen. Dabei wird sowohl auf die Softwarearchitektur, als auch verwendete Algorithmen eingegangen. Ebenfalls werden die verschiedenen Lösungsansätze erläutert und Entscheidungen begründet.

3.1 Wiedergabeart

Für die Wiedergabe von Medien gibt es im Kontext dieser Arbeit drei Methoden, welche besonders geeignet erscheinen und geprüft werden. Dabei soll ermöglicht werden, verschiedene Medien vom Real-Stereo-Steuerungsprogramm "balanciert" wiederzugeben. Ebenfalls soll die gewählte Methodik keine zu hohe Komplexität erfordern, um den Fokus des Projekts nicht zu beeinträchtigen. Im Folgenden werden alle berücksichtigten Methoden kurz vorgestellt und die Entscheidung begründet.

3.1.1 Lokale Wiedergabe von Audio durch Real-Stereo

Bei dieser Methode werden Audio-Dateien im MP3 Format lokal auf einem Computer mittels der entwickelten Steuerungsapplikation wiedergegeben. Hierbei kann der Lautstärke-Pegel der Kanäle direkt beeinflusst werden, indem deren Datenströme direkt bearbeitet werden. Es existiert eine Vielzahl von Audio Libraries für die weitverbreitetsten Programmiersprachen, welche eine solche Echtzeit-Modifikation vereinfachen [5].

Da die gesamte Audio Pipeline vom Playback der Audiodatei bis zur angepassten Ausgabe im entwickelten Steuerungsprogramm abläuft, sind, bis auf die verwendeten Bibliotheken, keine direkten Abhängigkeiten oder Systemvoraussetzungen vorhanden. Dieser Ansatz setzt aber auch voraus, dass das Real-Stereo Steuerungsprogramm die Fähigkeit, populäre Audioformate abzuspielen, besitzen muss. Ebenfalls wäre es somit nicht möglich, die Anpassungen für andere Applikationen, welche Audio wiedergeben, vorzunehmen. Somit wäre der Benutzer bei der Wiedergabe an die Real-Stereo Applikation gebunden.

3.1.2 Steuerung von lokaler Wiedergabe mittels Betriebssystem APIs

In diesem Ansatz werden API-Schnittstellen des Betriebssystems verwendet, um die Lautstärke der einzelnen Audio Kanälen zu steuern und somit das Balancing zu erreichen. Da diese Einstellungen auf Betriebssystem-Ebene durchgeführt werden, kann der Ton von allen Quellen gesteuert werden. Diese Art der Steuerung erfordert aber auch eine betriebssystemspezifische Implementierung und eine Lösung wäre nicht ohne weiteres auf anderen Betriebssystemen lauffähig. Dies schliesst auch die Verwendung von generischen und plattformunabhängigen Programmiersprachen wie Java für diese Lösung aus, da diese keinen ausreichenden Zugriff auf solche APIs ermöglichen. Eine solche API für Windows Betriebssysteme stellt Beispielsweise die "Core Audio API" [6] dar.

3.1.3 Steuerung von Wireless Wiedergabe über Sonos

Diese Methode wurde in Betracht gezogen, um die kabellose Wiedergabe über das Sonos Soundsystem zu steuern. Durch Recherche der von Sonos bereitgestellten API-Dokumentationen, insbesondere die Control-API [7], konnte aber zu diesem Zeitpunkt kein offiziell unterstützter Weg gefunden werden, die Lautstärke einzelner Kanäle direkt zu steuern. Die einzige Möglichkeit, dies umzusetzen, wäre für jeden Lautsprecher separate Sonos Geräte zu verwenden, welche dann einzeln angesteuert werden können. Sonos Geräte, die aus mehr als einem Lautsprecher bestehen, oder externe, an Sonos angeschlossene Lautsprecher, sind über diese API nicht separat ansteuerbar.

3.1.4 Auswahl

Es wurde die Entscheidung getroffen, die Wiedergabelautstärke, wie in Kapitel 3.1.2 beschrieben, über die Betriebssystem API zu realisieren. Dafür ausschlaggebend war, dass es möglich sein soll, alle Sounds, die von Windows und den verwendeten Applikationen ausgegeben werden, zu balancieren. Der Benutzer soll bei der Wiedergabe nicht eingeschränkt werden, was bei der in 3.1.1 erläuterte Methode der Fall wäre. Ausserdem sollen für den Prototyp möglichst keine Einschränkungen bei anzuschliessenden Lautsprecher getroffen werden. Mit dieser Möglichkeit werden alle von Windows erkannten Geräte unterstützt, was die grösste mögliche Menge ist.

3.2 PC Applikation

Die Applikation, welche für diese Projektarbeit erarbeitet wird, soll für Windows in .NET Core mit C# entwickelt werden. Dies hat sich einerseits aus den Anforderungen, eine reine PC-Lösung zu entwickeln, und andererseits aus Recherchen über mögliche Schnittstellen ergeben.

Da für die Steuerung der einzelnen Audio-Channels low-level Betriebssystem APIs genutzt werden müssen, schränkt dies die Auswahl an möglichen Programmiersprachen bereits ein und viele plattformübergreifende Programmiersprachen, wie etwa Java, verfügen nicht über die erforderlichen APIs. Windows gibt sich in diesem Bereich als offenes Betriebssystem und stellt eben diese APIs über die .NET Umgebung zur Verfügung. Weiter ist .NET Core mit Unterstützung für C# eine relativ neue Version der beliebten Entwicklungsumgebung von Microsoft, erstmals plattformunabhängig und open source [8]. Zudem erfreut sich .NET Core zunehmend an Beliebtheit in den Entwicklercommunities [9].

Aus all diesen Gründen scheint die Zielplattform Windows, zusammen mit C# in .NET Core, die passendste Basis für Real-Stereo zu sein.

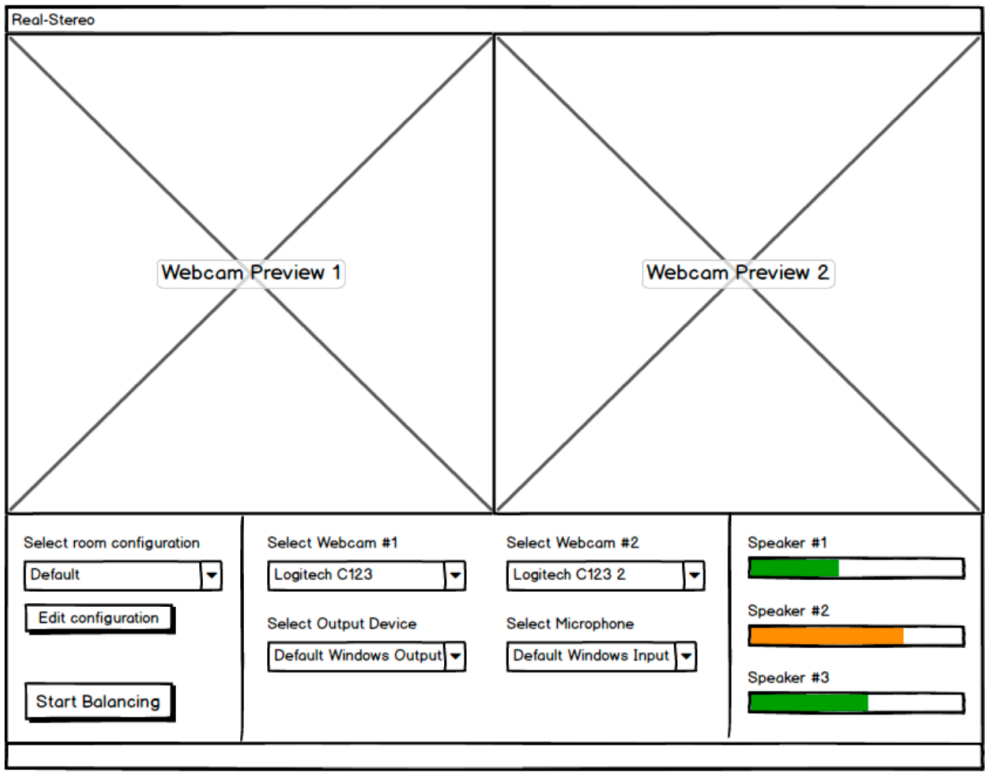

Abbildung 1 Hauptfenster Wireframe

Für die Steuerung der Applikation wird ein GUI mithilfe des Windows Presentation Foundation [10] und der Markup Sprache XAML [11] aufgebaut. Hierzu wurden als erstes Wireframes erstellt, welche die gewünschte Funktionalität und ihre Umsetzung mit Widgets vorzeigen. Das Wireframe für das Hauptfenster der Applikation ist in Abbildung 1 dargestellt. In dem Hauptfenster sind die Bilder der beiden Kameras sichtbar, sowie die aktuelle Lautstärke pro Audio-Channel. Ausserdem kann die verwendete Konfiguration und Hardware ausgewählt werden.

3.3 Kalibration und Raumkonfiguration

Mittels der Raumkonfiguration soll der zu überwachende Raum möglichst einfach und schnell konfiguriert und kalibriert werden können, sodass später eine zuverlässige Lokalisierung von Personen und Audioausgabe gewährleistet sei. Die fertige Raumkonfiguration wird abgespeichert, wodurch der Prozess für jeden Raum nur einmal durchgeführt werden muss. Bei der Konfiguration wird der Benutzer von der Applikation unterstützt und geleitet. Das Wireframe des hierzu konzipierten GUIs ist in Abbildung 2 abgebildet.

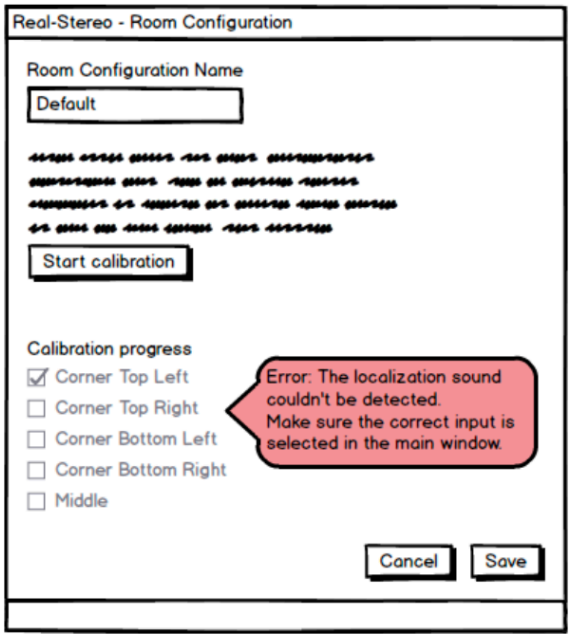

Abbildung 2 Konfiguration Wireframe

Im Folgenden werden die Schritte für eine komplette Raumkonfiguration aufgelistet.

3.3.1 Ablauf

I. Es werden zwei Kameras im Raum platziert und mit dem Computer verbunden. Ein 90° Winkel zwischen den beiden Kameras gilt als optimal.

II. Die Applikation wird gestartet und die angeschlossenen Kameras werden in der Applikation ausgewählt.

III. Das Audio Ein- (Mikrofon) und Ausgabegerät (Lautsprecher) werden in der Applikation im Hauptfenster ausgewählt.

IV. Eine neue Konfiguration wird erstellt, wobei der Benutzer einen Namen setzen kann, sodass er später zwischen verschiedenen Raumkonfigurationen auswählen kann.

V. Die Person wird vom Programm nun angewiesen, sich nacheinander an mindestens vier Positionen im Raum mit einem Mikrofon zu positionieren. Die vier Positionen sollten in den Ecken des physischen Raumes sein, jedoch auch möglichst von beiden Kameras gesehen werden. Ist ein Raum nicht rechteckig und hat mehr als vier Ecken, sollen die vier Ecken gewählt werden, die am weitesten aussen sind.

VI. Für jede Position wird anschliessend folgender Ablauf durchgeführt:
   a. Erkennt die Kamera die Person und es wird für 3 Sekunden keine Bewegung festgestellt, wird die Konfiguration für die aktuelle Position gestartet.
   b. Es wird ein Ton in verschiedenen Lautstärken jeweils 2 Sekunden lang auf jedem Lautsprecher nacheinander abgespielt. Durch die unterschiedlichen Lautstärken des Testtons, kann danach abgeschätzt werden, wie gross eine

Veränderung der Kanal-Lautstärke ausfallen muss, um die gewünschte Änderung der wahrgenommenen Lautstärke zu erreichen.

c. Für jeden Lautsprecher wird die Lautstärke des Testtons gemessen und der Mittelwert zusammen mit der Position der Person im Raum abgespeichert.

d. Tritt ein Fehler auf (bsp. Person bewegt sich, während der Testton wiedergegeben wird), wird in der Applikation ein Fehler angezeigt und je nach Fehler wird entweder die ganze Konfiguration oder die der aktuellen Position wiederholt.

e. Sind noch nicht alle vier Positionen konfiguriert, wird die Person angewiesen, sich zur nächsten Position zu begeben und der Prozess startet erneut bei Schritt *a.* Sind die vier Ecken konfiguriert, können eine beliebige Anzahl an weiteren Positionen konfiguriert werden, um beispielsweise eine bessere Berechnung in der Raummitte zu ermöglichen. In diesem Fall wird ebenfalls bei Schritt *a.* der Prozess neu gestartet.

VII. Die Raumkonfiguration wird abgespeichert und kann anschliessend fürs Balancing ausgewählt und vom Programm verwendet werden.

3.4 Audio Geräte API

Bei Windows Betriebssystemen ist die Aufzählung und Kontrolle der angeschlossenen Audiogeräte über die Core Audio API möglich. Obwohl das Ansprechen dieser low-level Schnittstelle mittels eines C# Programms durchführbar ist, sind alle damit erstellten Ressourcen nicht von C# selber verwaltet und werden bei der Garbage collection nicht berücksichtigt. Eine Abstraktionsschicht in Form einer C# Programmbibliothek erleichtert die Verwendung der Core Audio API sehr. Eine solche Programmbibliothek für die Core Audio API wurde in NAudio [12] gefunden.

3.4.1 Auflistung von Ein- und Ausgabegeräten

Damit der Benutzer das Ein- und Ausgabegerät für die Verwendung im Real-Stereo Programm auswählen kann, müssen die am Computer angeschlossenen Audiogeräte aufgezählt werden. Ebenfalls müssen die Anzahl Kanäle und ihre aktuelle Lautstärke bestimmt werden.

Dies kann mit der *MMDeviceEnumerator* Klasse erreicht werden. Anhand des Gerätezustands und der Datenrichtung (Ein- oder Ausgabegerät), können mit der *MMDeviceEnumerator* Klasse die Audiogeräte gefiltert werden und ein Array von *MMDevice* Instanzen, die die Audiogeräte repräsentieren, zurückgegeben werden.

Diese Instanzen der *MMDevice* Klasse beinhalten ein *AudioEndpointVolume* Feld, über welches ein Array der Kanäle und ihrer aktuellen Lautstärke ausgelesen werden kann.

3.4.2 Steuerung der Kanallautstärke

Über die *MMDevice* Klasse und das *AudioEndpointVolume* Feld kann nicht nur die aktuelle Lautstärke eines Kanals ausgelesen, sondern auch geändert werden. Der neue

Wert wird von der NAudio Bibliothek direkt an die Windows Core Audio API übergeben und somit betriebssystemweit auf das gewählte Audio-Ausgabegerät angewendet.

3.4.3 Wiedergabe des Testtons

Mithilfe des *SampleProviders* Namespaces, welcher im *Wave* Namespace in der NAudio Bibliothek gefunden werden kann, kann ein Sinuston-Generator instanziiert werden, welcher den benötigten Testton von 2000hz generiert. Für die Wiedergabe wird ein 2 Sekunden langer Testton mithilfe dieses Generators erstellt und der *WasapiOut* Klasse auf dem vom Benutzer gewählten *MMDevice* ausgegeben.

Das Windows Betriebssystem stellt mit der Windows Audio Session API (WASAPI) [13] eine Schnittstelle zur Verfügung, um Datenströme zwischen Audiogeräten und Applikation einfach zu erstellen.

3.4.4 Aufnahme starten

Um die Lautstärke des Testtons zu bestimmen, muss für die Dauer des Tons eine Aufnahme gestartet werden und die aufgenommenen Samples gesammelt werden.

Bei der Aufnahme kümmert sich WASAPI um das Datenformat und liefert bei jeder Hardwarekonfiguration Samples im Float Format. Diese Schnittstelle ist in abstrahierter Form in der NAudio Bibliothek im Namespace *Wave* verfügbar. Nach dem Start der Aufnahme auf einem *MMDevice* liefert die NAudio Bibliothek Daten an einen *EventHandler*, in welchem die erhaltenen Samples weiterverwendet werden können.

3.4.5 Aufnahme filtern

Beim Testton handelt es sich um einen 2000hz Sinuston. Um die Lautstärke möglichst akkurat bestimmen zu können, müssen die Hintergrundgeräusche in der Aufnahme so weit möglich reduziert werden.

Mithilfe der NAudio Bibliothek und dem *Dsp* Namespace können Tief- und Hochpassfilter erstellt werden. Diese werden auf die aufgenommenen Samples angewendet. Um die Funktionalität mit inakkuraten Mikrofonen zu ermöglichen, wird der Tiefpassfilter auf 1900hz und der Hochpassfilter auf 2100hz eingestellt. Somit sollte der Testton ungehindert passieren können und die meisten anderen Geräusche jedoch entfernt werden.

3.4.6 Durchschnittslautstärke der Aufnahme bestimmen

Nachdem die Audio Samples im *EventHandler* mithilfe des Tief- und Hochpassfilters bearbeitet wurden, werden sie zu einer Liste hinzugefügt. Dadurch kann am Ende des Testtons die durchschnittliche Lautstärke berechnet werden, indem der Absolutwert aller Samples aufaddiert, und durch die Anzahl der Samples geteilt wird.

3.5 Personenerkennung

Aufgrund der einfacheren Umsetzung und Bedienung wird für die Personenerkennung auf Computer Vision gesetzt. Ultra-wideband hätte den grossen Nachteil, dass der Zuhörer ständig einen Sender bei sich tragen müsste, was für den privaten Gebrauch umständlich und deswegen nicht geeignet ist.

Für die Umsetzung wird, die auf Computer Vision spezialisierte und weit verbreitete Library, OpenCV [14] verwendet, welche Interfaces für C++, Python, Java und MATLAB zur Verfügung stellt. Um die Funktionen auch aus C# aufrufen zu können, wird zusätzlich der .NET Wrapper Emgu CV [15] eingesetzt. Als Wrapper ist er eine Schnittstelle zwischen C# und der C++ Version von OpenCV und sorgt dafür, dass während der Benutzung normale C# Datenstrukturen verwendet werden können, während Emgu CV diese intern zu C++ Typen und Funktionsaufrufe umwandelt.

Abbildung 3 Beispiel Histogram of oriented gradients. Quelle: angepasst von [16].

Die eigentliche Erkennung einer Person basiert auf der *Histogram of oriented gradients (HOG)* Methode, welche bereits in OpenCV als dedizierte *HOGDescriptor* Klasse implementiert ist. Diese teilt ein Bild in viele kleine Quadrate auf und wandelt diese in Gradienten um, wie in Abbildung 3 zu sehen ist. Darin ist es einfacher, bekannte Muster zu suchen, was im Falle der Personenerkennung einzelne Körperteile sind [17]. So ist es auch möglich, den Zuhörer zu erkennen, wenn er nur teilweise sichtbar ist.

Evaluierte Alternativen zum Histrogram of oriented gradients Algorithmus waren der *Haar-cascade Classifier* und *Motion Detection*. Während der Haar-cascade Classifier auf einem Machine Learning-Modell basiert und in einem Bild zuvor trainierte Features erkennen und Personen zuordnen kann [18], werden mit Motion Detection in zwei aufeinander folgenden Bilder Unterschiede und somit Objekte, die sich bewegt haben, erkannt [19]. Die Nachteile der beiden letzten Verfahren sind, dass entweder nur genau trainierte Personenmerkmale erkannt werden und es schwierig wird, den Zuhörer aus anderen Perspektiven oder teilweise verdeckt zu identifizieren, oder die

Lichtverhältnisse und Kameraqualität eine zu grosse Rolle spielen und die Schritte, um Rauschen im Bild zu eliminieren, pro Raum und Tageszeit angepasst werden müssten.

3.5.1 Auflistung von Kameras

Emgu CV verwendet standartmässig die *Microsoft Media Foundation* API [20], um einen Videostream von den am Computer angeschlossenen Kameras zu erhalten. Um die korrekte Auswahl der Kamera zu ermöglichen und den Gerätenamen abzurufen, wird die Bibliothek *MediaFoundation* [21] verwendet. Mit der Funktion *EnumVideoDeviceSources* kann eine Liste aller am Computer angeschlossenen Kameras abgerufen werden. Die Kameras in der Liste sind in der gleichen Reihenfolge, und somit unter dem gleichen Index, wie sie anschliessend nach der Auswahl durch den Benutzer an Emgu CV weitergegeben werden können. Mit der Funktion *GetAllocatedString* und dem Attribut *MF_DEVSOURCE_ATTRIBUTE_FRIENDLY_NAME* [22] als Argument kann der Gerätename abgerufen werden. Er wird anschliessend dem Benutzer angezeigt wird.

3.6 Reduktion False-Positives

Keine Personenerkennung ist perfekt und es wird immer eine gewisse Anzahl an False-Positives geben: Objekte die fälschlicherweise als Person erkannt wurden. Diese gilt es zu minimieren, um eine möglichst robuste Lösung präsentieren zu können. Dafür wurden drei verschiedene Methoden, welche im Folgenden erläutert werden, angewendet und kombiniert.

3.6.1 Score

Die bereits in OpenCV implementierte Histogram of oriented gradients-Methode zur Personenerkennung gibt als Resultat mehrere Bounding-Boxen, Rechtecke die Personen umfassen, mit einem dazugehörigen Score zurück. Der Score gibt an, wie sicher sich der Algorithmus ist, dass in der Bounding-Box auch tatsächlich eine Person ist.

Abbildung 4 Verschiedene Score Threshold

Um die False-Positives bereits einschränken zu können, werden als erstes durch einen Threshold zu kleine Scores schon am Anfang verworfen. Oftmals ist sich der Algorithmus auch bei einer korrekt erkannten Person nicht sicher, ob er richtig liegt,

besonders wenn diese nur von der Seite oder hinten sichtbar ist. Obwohl der maximale Score 1.0 beträgt, sind deshalb in diesen Fällen Scores zwischen 0.1 und 0.2 keine Seltenheit. Die Abbildung 4 zeigt auf, welche Auswirkungen verschiedene Thresholds in der Testumgebung haben. Die erkannten Bounding-Boxen sind dabei als blaue Rechtecke eingezeichnet. Wenn nur Scores von 0.1 oder höher berücksichtigt werden, können bereits viele False-Posivites ausgeschlossen werden. Um nicht zu viele korrekt erkannte Personen auszuschliessen, wurde der Threshold deshalb auf 0.1 gesetzt, auch wenn damit nicht alle False-Positives eliminiert werden können, sondern nur jene mit einem sehr niedrigen Score.

Nur die Bounding-Boxen, mit einem genug hohen Score, werden für die nächsten Schritte berücksichtigt.

3.6.2 Grouping

Es ist möglich, dass eine Person nicht von einer einzelnen Bounding-Box umschlossen wird, sondern dass mehrere Körperteile als einzelne Bounding Boxen und somit separate Personen erkannt werden. Das ist vor allem problematisch, weil der Prototyp nur auf eine Person im Raum ausgelegt ist und die Position der Person in diesem Fall auch nicht genau bestimmt werden kann.

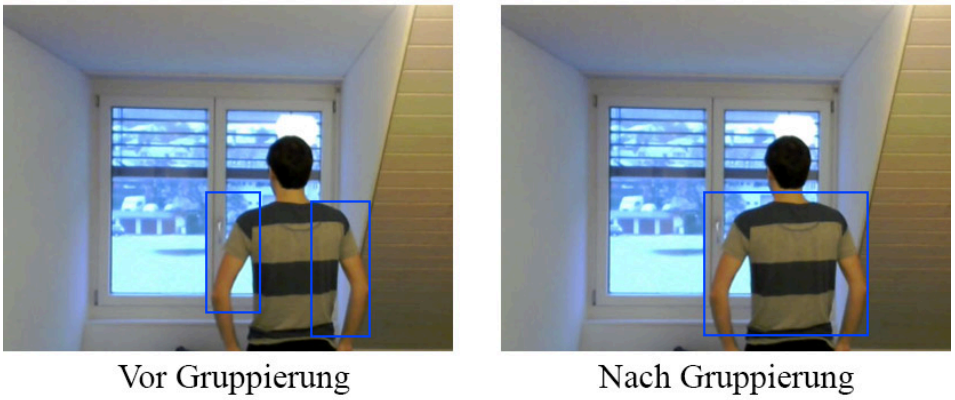

Abbildung 5 Zwei Bounding-Boxen (links) werden neu zu einer grossen (rechts) kombiniert

Um dieses Problem zu lösen, werden mehrere Bounding-Boxen zu einem neuen Rechteck zusammengefasst, wenn diese nicht weiter voneinander entfernt sind als ein definierter Wert. Diese Schwelle wurde auf 50 Pixel gesetzt. In der Abbildung 5 ist das Resultat dieser Methode ersichtlich. Mit den zwei blauen Bounding-Boxen im linken Bild sind zwar beide Arme korrekt vom Histogram of oriented gradients-Algorithmus identifiziert worden, jedoch nicht die Person als Ganzes. Die Gruppierung fasst diese zu einer neuen Bounding-Box zusammen, wie im rechten Bild ersichtlich, und verwendet nur noch jene für die nächsten Schritte.

3.6.3 Tracking

Ein weiteres aufgetauchtes Problem ist, dass False-Positives nur in einzelnen Frames erkannt werden, während es im Frame zuvor und danach wieder korrekt ist. Dies kann durch Rauschen, Lichtreflektionen oder andere Unregelmässigkeiten entstehen.

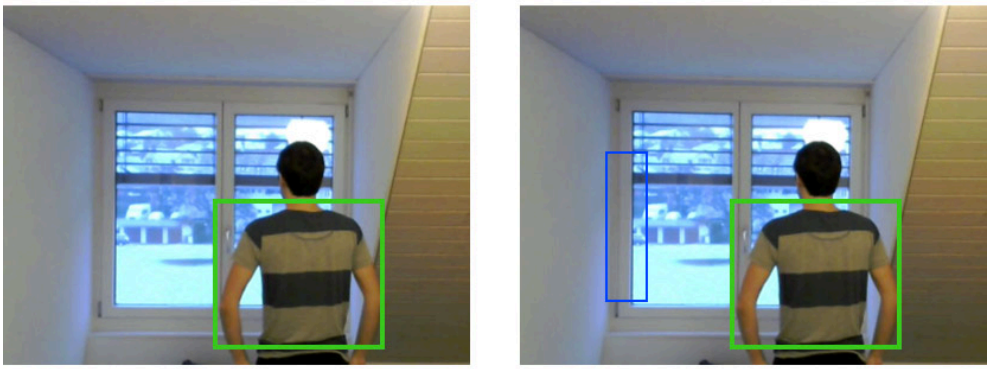

Abbildung 6 Eine neue False-Positive Bounding-Box im zweiten Frame (blau) wird ignoriert

Das Problem wird durch ein Tracking gelöst. Eine Bounding-Box muss deshalb in mindestens drei aufeinanderfolgenden Frames ungefähr am gleichen Ort sein. Die Positionen der verschiedenen Frames müssen sich also überlappen und die mögliche Verschiebung ist damit relativ zu deren Grösse. Ist dies gegeben, wird die Bounding-Box als Person markiert und als grünes Rechteck dargestellt. Abbildung 6 zeigt ein Beispiel, in dem im zweiten Frame fälschlicherweise eine neue Bounding-Box erkannt wird. Weil es im Frame davor an dieser Position jedoch noch keine Erkennung gab, wird diese nicht als Person markiert. In den weiteren Schritten werden nur die grünen, als Person markierten Bounding-Boxen berücksichtigt. Die verworfenen, in der Abbildung blau dargestellten Rechtecke, sind nur zu Debugging-Zwecke weiterhin sichtbar.

Zusätzlich soll das Problem gelöst werden, dass auch Koordinaten bestimmt werden können, wenn die Person kurzzeitig von einem Hindernis verdeckt wird oder aufgrund von verwinkelten Räumen auf einer Kamera nicht mehr sichtbar ist.

Abbildung 7 Kann keine Person erkannt werden, wird deren vorherige Position benutzt

Im Beispiel der Abbildung 7 hat die Person das Bild auf der rechten Seite verlassen. In diesem Fall wird die zuletzt bekannte und als Person markierte Bounding-Box als Grundlage für die Berechnungen gebraucht. Dies führt trotzdem noch zu guten Resultaten, da sich die Person in den meisten Fällen nicht weit davon aufhält und ebenfalls von der gleichen Seite später das Bild wieder betritt und dies somit die nächstmögliche Bestimmungsposition ist. Zur Veranschaulichung wird in diesem Fall die Bounding-Box mit einem dunkleren Grün dargestellt.

3.7 Lokalisierung

Ist eine Person erkannt worden, muss sie in einem zweidimensionalen Koordinatensystem, welches den Raum abbildet, lokalisiert werden. Dafür werden zwei Webcams genutzt, welche in einem 90° Winkel aufeinander aufgestellt sind und möglichst grossflächig den Raum abdecken.

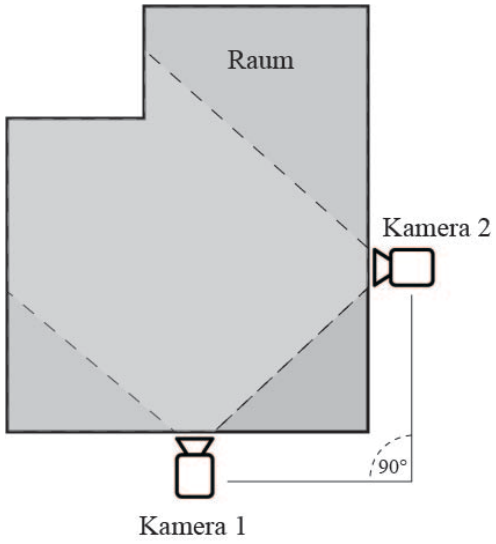

Abbildung 8 Beispielaufbau eines Raumes

In der Abbildung 8 ist ersichtlich, wie ein solcher Aufbau aussehen kann und welche Bereiche des Raumes von den Kameras abgedeckt werden. Dies hängt jedoch auch von der Brennweite der verwendeten Kameras ab.

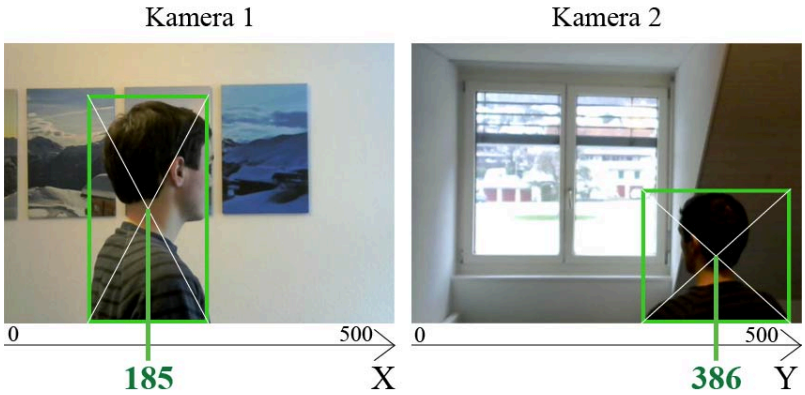

Abbildung 9 Aus den beiden Kameras wird eine X- und Y-Koordinate berechnet

Jede Kamera berechnet nun die Koordinate einer Achse, wie in Abbildung 9 zu sehen ist. Dafür wird jeweils der Mittelpunkt auf der horizontalen Achse der, von der Personenerkennung, zurückgegebenen Bounding-Box genommen und auf eine Skala von 0 bis 500 skaliert. Wird dies von beiden Kameras gemacht, werden zwei Koordinaten als Resultat erhalten – von der Kamera 1 die X-Koordinate und von der Kamera 2 die Y-Koordinate - und der ganze Raum kann auf ein Koordinatensystem von 500 mal 500 abgebildet werden.

3.8 Audio Balancing

Als letzter Schritt wird das eigentliche Balancing durchgeführt. Dafür wird anhand der bekannten, aufgenommenen Lautstärken bestimmter Positionen aus dem Konfigurationsschritt, für jeden Lautsprecher die wahrgenommene Lautstärke des ganzen Raumes berechnet. Diese Berechnung wird mit Hilfe von Shepard's Methode für verstreute Daten durchgeführt [23]. Die Methode gewichtet die bereits bekannten Punkte anhand der Entfernung zum aktuell zu berechnenden Punkt, interpoliert zu allen bekannten Punkten und berechnet den finalen Wert unter Berücksichtigung aller Gewichte.

Nicht nur aufgrund der Möglichkeit, die Abstufung mit einem Parameter ändern zu können und somit für jede Umgebung zu optimieren, eignet sich diese Interpolationsmethode für Real-Stereo. Auch kommt diese Methode im Gegensatz zu Alternativen [24] [25] mit verstreuten, nicht regulär rechteckig angeordneten Startpunkten zurecht. Sie ist ausserdem speziell für die räumliche Interpolation entwickelt worden [23].

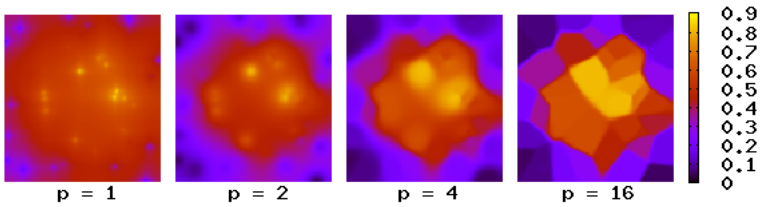

Abbildung 10 Shepard's Methode mit verschiedenen Werte für Parameter p. Quelle: [26]

Die Berechnung der Lautstärke kann mit Hilfe von Shepard's Methode in folgender Formel dargestellt werden.

$$Volume = \frac{\sum_{i=0}^{N} R_i * \sqrt{(X_i - TX)^2 + (Y_i - TY)^2}^{p}}{\sum_{i=0}^{N} \sqrt{(X_i - TX)^2 + (Y_i - TY)^2}^{p}}$$

Dabei ist N die Anzahl gegebener Punkte, R die aufgenommene Lautstärke an jenem gegebenen Punkt, X, respektive Y, die Koordinaten des gegebenen Punktes und TX und TY die Koordinaten für den Punkt, der interpoliert werden soll. Der Parameter p spielt bei der Gewichtung der einzelnen gegebenen Punkte eine Rolle und entscheidet, wie fliessend die Übergänge sind. In Abbildung 10 sind verschiedene Werte von p und deren Auswirkung ersichtlich. Für Real-Stereo gilt es nun, den geeigneten Wert für Parameter p zu finden.

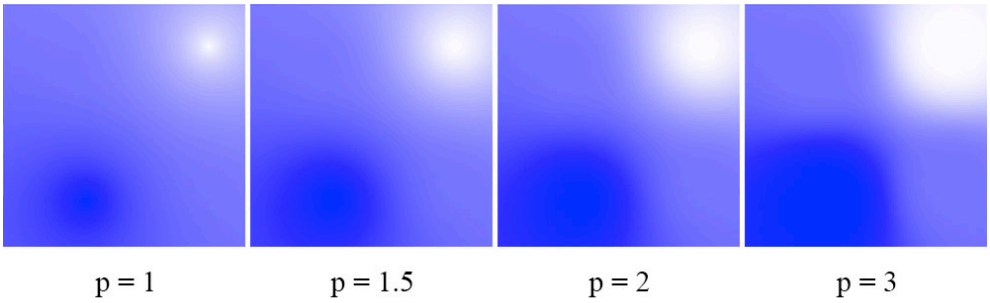

Abbildung 11 Test mit verschiedenen Werten für Parameter p von Shepard's Methode in Real-Stereo

Dafür wurden von einem Testraum, in dem die Lautstärke an vier Positionen aufgenommen wurde, alle möglichen Koordinaten mit verschiedenen Parameterwerten interpoliert, wie es in Abbildung 11 ersichtlich ist. An den dunkelblauen Stellen ist die wahrgenommene Lautstärke laut, an den weissen Stellen leise. Um ein optimales Hörerlebnis bieten zu können, sollen die Übergänge möglichst fliessend und nicht hart sein, da ansonsten eine Änderung der Lautstärke abrupt wahrgenommen wird. Deshalb fällt bereits Wert 3 weg und auch der Wert 2 ist an der Grenze. Bei einem Wert von 1 sind die Extreme jedoch zu klein und es hat im ganzen Raum fast die gleiche Lautstärke. Aus dem Grund ist dieser auch nicht optimal.

Als Kompromiss wird deshalb der Wert 1.5 für die Interpolation genommen, bei welchem der Übergang optimal erscheint.

Der Applikation ist nun bekannt, wie laut der Lautsprecher an jeder Position im Raum vom Zuhörer wahrgenommen wird. Jedoch muss für eine korrekte Ansteuerung auch bekannt sein, wie viel eine Änderung der Lautstärke am wahrgenommenen Wert ändert. Für diesen Zweck wird zusätzlich berechnet, wie viel eine 1%-Änderung der Lautstärke für den Hörer verändert. Dies wurde in der Folgenden Formel berechnet.

$$Diff = \frac{Volume - HalfVolume}{(SpeakerLevelFull - SpeakerLevelHalf) * \frac{1}{2}}$$

Dabei ist $Volume$ das Resultat der vorherigen Interpolation und $HalfVolume$ der gleiche Wert, jedoch von der zweiten Aufnahme während der Konfiguration, bei der die Hälfte der Lautstärke gemessen wurde. $SpeakerLevelFull$ und $SpeakerLevelHalf$ geben an, wie Laut die Lautsprecher bei beiden Messungen eingestellt waren.

Mit diesen beiden Werten kann nun berechnet werden, wie Laut ein Lautsprecher für jede mögliche Position eingestellt werden muss, was die nächste Formel darstellt.

$$Level = InitialLevel + \frac{TargetVolume - Volume}{Diff}$$

$InitialLevel$ ist die ursprüngliche eingestellte Lautstärke des Lautsprechers, wobei $TargetVolume$ für die Lautstärke steht, welche vom Benutzer an jeder Position im Raum gleich wahrgenommen werden soll und setzt sich mit dem am lautesten gemessenen Wert der Konfiguration gleich. Dabei sind $Volume$ und $Diff$ die Resultate der vorherigen Gleichungen für die aktuelle Position.

4 Resultate

Das Resultat der Projektarbeit ist ein als .exe ausführbarer, fertiger Prototyp. In diesem Kapitel werden die einzelnen Teile der Applikation aufgezeigt und mit zusätzlichen Daten erklärt. Ausserdem wird auf interessante Stellen vom Code eingegangen.

4.1 Applikation

Das entstandene Hauptfenster der Applikation entspricht der im Wireframe (Abbildung 1 Hauptfenster Wireframe) geplanten Benutzeroberfläche und erlaubt die Kontrolle der Applikation. Im Folgenden wird genauer auf die einzelnen Komponenten des Hauptfensters, welches in Abbildung 12 gezeigt wird, eingegangen.

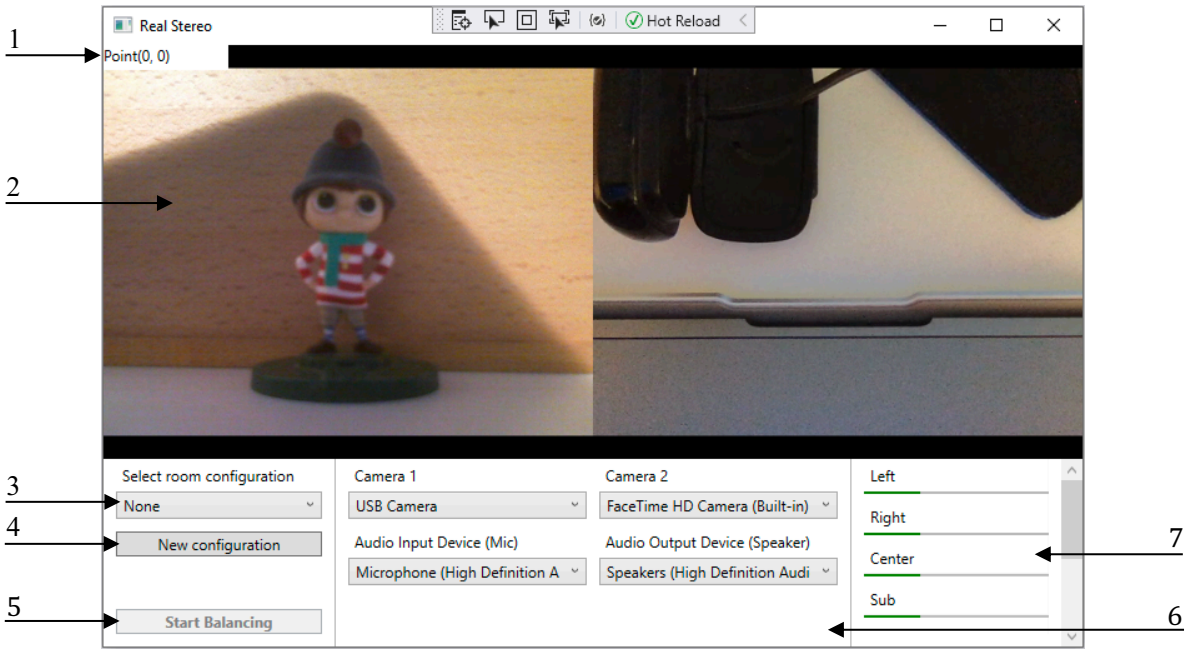

Abbildung 12 Hauptfenster

1. Die aktuell berechneten Koordinaten der Person werden im Format *Point(x,y)* dargestellt.
2. Vorschau von Kamera 1 (links) und von Kamera 2 (rechts). Ebenfalls werden hier farbige Rahmen um erkannte Objekte gezeichnet. Blau steht für Regionen, in welchen potentielle neue Personen erkannt wurden und grün für Personen, die bereits im letzten Frame erkannt wurden.
3. Im Dropdown werden alle gespeicherten Raumkonfigurationen angezeigt. Der Benutzer kann durch Anklicken einer Raumkonfiguration diese laden.
4. Mit diesem Knopf wird das Fenster geöffnet, durch welches eine neue Konfiguration vom Benutzer erstellt werden kann.
5. Durch Klicken dieser Schaltfläche wird die aktuell ausgewählte Raumkonfiguration angewendet und die Applikation beginnt mit dem Balancing

anhand der Position des Hörers. Der Knopf ist deaktiviert, solange keine gültige Raumkonfiguration ausgewählt ist.

6. Dieser Bereich erlaubt die Auswahl von Ein- und Ausgabegeräten sowie der zwei Kameras. Die Liste der auswählbaren Geräte wird beim Öffnen der Dropdowns aktualisiert, sodass auch Geräte, welche nach dem Starten der Applikation angeschlossen wurden, verwendet werden können.
7. Die Fortschrittsbalken zeigen die aktuelle Lautstärke der verschiedenen Kanäle des aktuell ausgewählten Audioausgabegeräts an. Durch einen Doppelklick auf den Namen eines Kanals wird dessen aktuelle, kalibrierte Interpolationswerte in einem neuen Fenster grafisch dargestellt, was für Debugging-Zwecke nützlich ist.

4.2 Konfiguration

Mit einem Klick auf die Schaltfläche 4 der vorherigen Abbildung 12 öffnet sich das Konfigurationsfenster, welches in Abbildung 13 ersichtlich ist. Das Konfigurationsfenster ermöglicht und leitet die Erstellung einer neuen Raumkonfiguration.

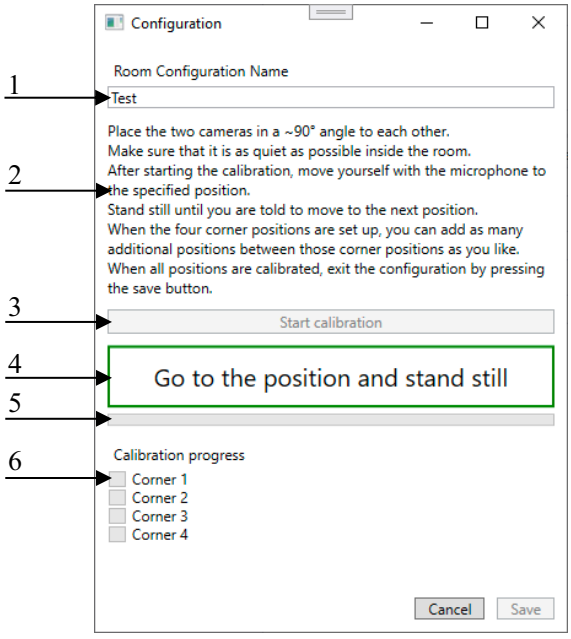

Abbildung 13 Konfigurationsfenster

1. Der gewünschte Name der neuen Raumkonfiguration kann hier vom Benutzer eingegeben werden.
2. Der generelle Ablauf der Kalibration und die korrekte Platzierung der zwei Kameras wird dem Benutzer in dieser Textbox erklärt. Ebenfalls wird darauf hingewiesen, dass mehr als die mindestens benötigten vier Punkte zur Kalibration hinzugefügt werden können.
3. Der Kalibrationsprozess kann mit diesem Knopf gestartet werden.

4. In diesem Textfeld erscheinen Anweisungen, welche der Benutzer durchführen muss, um die Kalibration erfolgreich abzuschliessen. Auch werden hier eventuelle Fehlermeldungen angezeigt.
5. Die Lautstärke des zuletzt gemessenen Testtons wird in diesem Fortschrittsbalken grafisch dargestellt.
6. Der Fortschritt für die mindestens benötigten vier Punkte wird hier dargestellt.

Nach erfolgreichem Abschluss des Kalibrationsprozesses kann mit der *Save* Schaltfläche die Raumkonfiguration abgespeichert werden und ist dann sofort für das Balancing verfügbar.

Die für die Raumkonfiguration verwendete Klassenarchitektur ist in Abbildung 14 dargestellt.

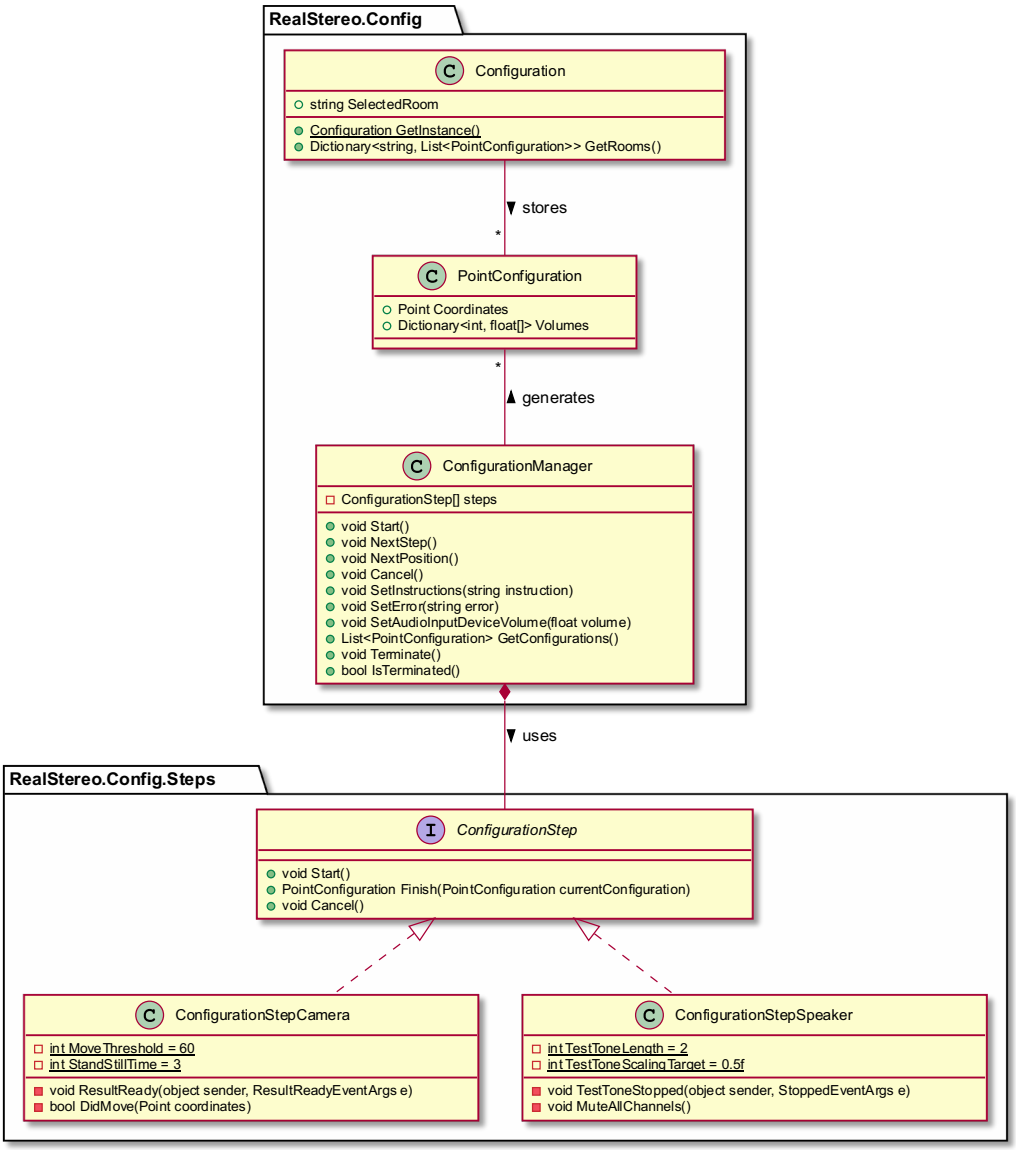

Abbildung 14 Klassendiagramm des Konfigurationsteil

Dabei ist hervorzuheben, dass für die einzelnen Schritte der Konfiguration das Strategy und Composite Pattern vereint verwendet werden. Die *ConfigurationManager* Klasse verwaltet in dem privaten Feld *Steps* die einzelnen Schritte des Ablaufs und ruft diese dynamisch der Reihe nach auf. Somit kann der ganze Konfigurationsablauf in Zukunft problemlos erweitert werden, sollten für Erweiterungen noch zusätzlich Schritte benötigt werden.

Die *ConfigurationStepCamera* und *ConfigurationStepSpeaker* implementieren die beiden für den Prototyp erforderlichen Schritte. Der Konfigurationsschritt für die Kamera gibt die Koordinaten und die für den Speaker gemessene Lautstärke zurück, sofern die Person genug lange stillsteht. Die Parameter, welche in der Implementation verwendet werden und das Resultat verbessern können, sind jeweils in Klassenkonstanten gespeichert, was ein einfaches Anpassen ermöglicht.

Die Klasse *Configuration* ist dafür zuständig, die Konfiguration aus der JSON-Datei zu lesen und in Java Objekte umzuwandeln und umgekehrt diese bei einer Änderung wieder abzuspeichern. Dafür wird die Newtonsoft Json.NET Library [27] verwendet. Damit von überall aus der Applikation die aktuelle Konfiguration gelesen werden kann und nicht jeweils die Daten neu gelesen und deserialisiert werden müssen, wurde das Singleton Pattern auf diese Klasse angewendet.

4.3 Lautstärkenregelung

Wie bereits in Abschnitt 3.8 erläutert, wurde Shepard's Methode für verstreute Daten angewendet, um die wahrgenommene Lautstärke an jedem Punkt im Raum zu interpolieren.

Damit nicht für jede Position neu interpoliert werden muss und somit mehr Ressourcen für die aufwändige Personenerkennung verwendet werden können, werden zu Beginn bereits die Werte für den ganzen Raum berechnet. Damit dies nicht zu viel Speicher beansprucht, wird der Raum auf ein 100 mal 100 Koordinatensystem herunterskaliert. Mit immer noch 10'000 möglichen Koordinaten bleibt die Abstufung trotzdem klein genug, so dass das Hörgefühl nicht beeinträchtigt wird.

In der folgenden Abbildung 15 ist das Klassendiagramm für den Teil von Real-Stereo zu sehen, der für das Balancing zuständig ist.

Unter dem *RealStereo.Balancing.Tracking* Namespace wird die Personenerkennung zusammengefasst. Die Klasse *Camera* liest dabei ein Frame von der ausgewählten Kamera und lässt in diesem Personen erkennen. Dies wird von der *PeopleDetector* Klasse, welche den Histogram of oriented gradients Algorithmus benutzt und die Parameter für einfaches Anpassen als Klassenkonstante deklariert, durchgeführt. Danach werden diese in das Frame eingezeichnet und in Koordinaten umgewandelt.

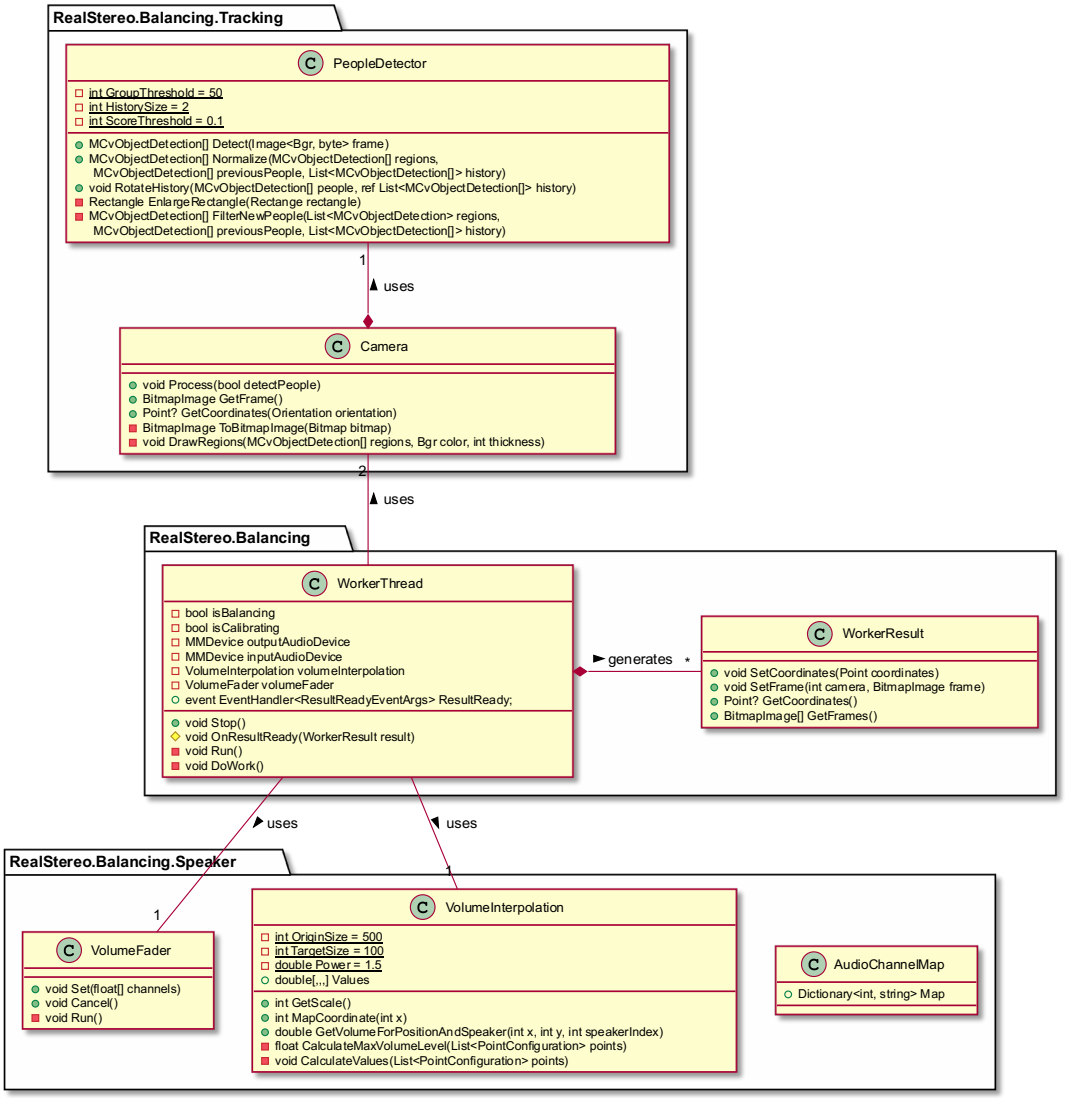

Abbildung 15 Klassendiagramm des Audio-Balancings

Im *RealStereo.Balancing.Speaker* Namespace findet die Interpolation der Lautstärke statt. Ebenfalls sorgt die als Thread implementierte Klasse *VolumeFader* dafür, dass die Lautstärke über einen kurzen Zeitraum von ein paar hundertstel Sekunden an den neuen Wert angepasst wird, damit ein stufenloser Übergang möglich ist und das Hörerlebnis verbessert wird. Die *AudioChannelMap* Klasse wird vom GUI verwendet und wandelt eine Channel Id in Text um, beispielsweise wird der ID 1 der Channel *left* zugeordnet.

Schliesslich wird alles im Namespace *RealStereo.Balancing* zusammengeführt und orchestriert. Die Hauptklasse darin – *WorkerThread* – wird als Thread im Hintergrund gestartet, damit das grafische Interface nicht beeinträchtigt wird. Sie sorgt dafür, dass in der richtigen Reihenfolge und nur wenn nötig, Personen erkannt, lokalisiert und die Lautsprecher aufeinander abgestimmt werden. Zusätzlich stellt sie ein Event zur Verfügung, welches alle aktuellen Informationen veröffentlicht. Die Daten des Events werden vom GUI verwendet, um dieses zu updaten.

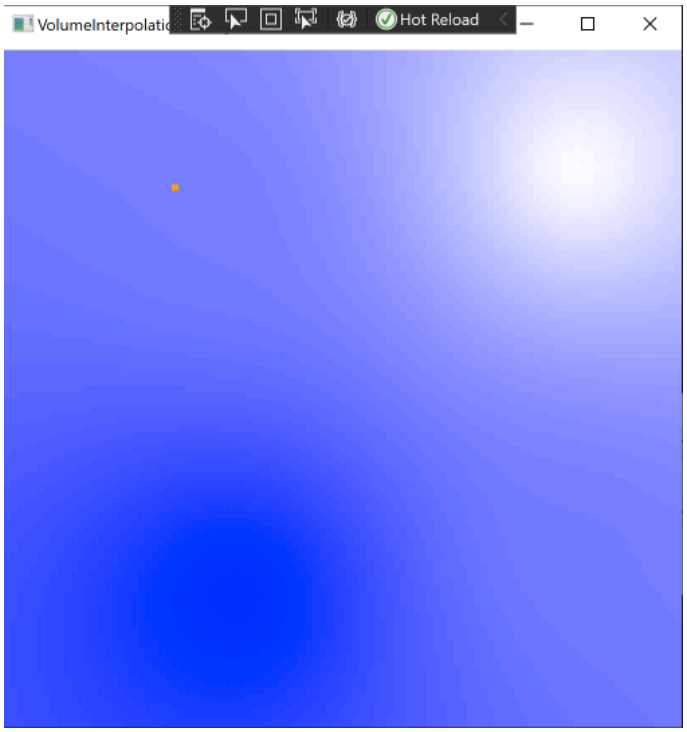

Abbildung 16 Debugging-Fenster für Interpolation

Um das Debugging zu erleichtern, wurde ein Fenster erstellt, welches die interpolierte, wahrgenommene Lautstärke darstellt. Dies ist in Abbildung 16 zu sehen. In den blauen Bereichen wird die Lautstärke als laut wahrgenommen, in den weissen Bereichen leise. Der orange Punkt, in der Abbildung im Bereich links oben, stellt die aktuelle Position des Zuhörers dar. Da dieses Fenster nur für Debugging-Zwecke ausgelegt ist, ist es nur versteckt durch einen Doppelklick auf einen Channel im Hauptfenster in Abbildung 12 zu öffnen und es wurde auf Beschreibungen verzichtet.

4.4 Tests

Der fertige Prototyp wurde einem Test unterzogen, welcher möglichst nahe an einen realen Einsatz kommen soll. Dafür wurde der Versuch in einem etwa $25.5m^2$ grossen Raum aufgebaut.

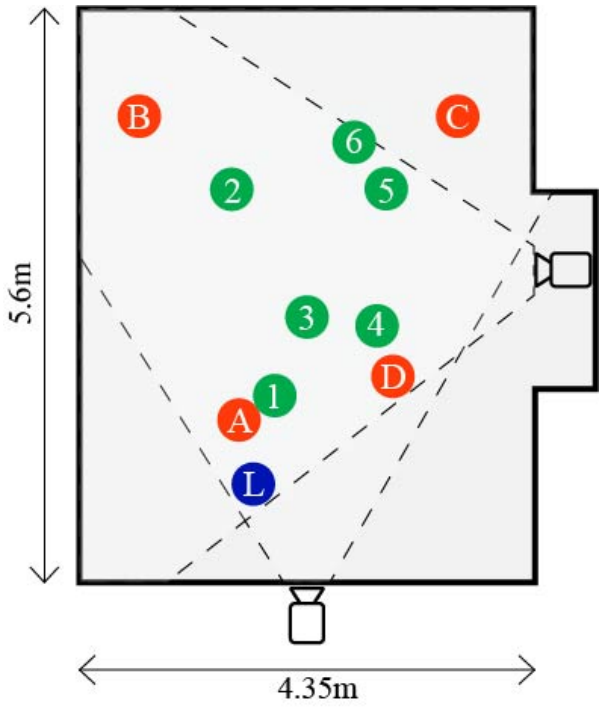

Abbildung 17 Testaufbau

In Abbildung 17 ist der Aufbau ersichtlich. Anschliessend wurde die Konfiguration an den vier roten Punkten durchgeführt. Um das Balancing zu testen, wurde nun an mehreren zusätzlichen Positionen (in Abbildung 17 grün markiert) die Lautstärke eines gleichbleibenden Tones gemessen. Am blauen «L»-Punkt ist der Lautsprecher positioniert. Für den Test wurde nur ein Lautsprecher verwendet, da es so einfacher ist die Lautstärke, und somit korrekte Funktionalität, zu testen. Funktioniert die Applikation korrekt, sollte die gemessene Lautstärke etwa an allen Punkten gleich laut sein. Für die Konfiguration und die Messung wurde das Mikrofon eines Beyerdynamics MMX2 USB Headsets [28] und zwei Logitech C270 Webcams [29] verwendet.

Messpunkt	Koordinaten	Gemessene Lautstärke
A	174, 97	0.266
B	148, 382	0.234
C	427, 419	0.197
D	430, 126	0.241
1	191, 122	0.331
2	195, 361	0.382
3	278, 219	0.335
4	381, 201	0.379
5	409, 379	0.368
6	392, 411	0.403

Tabelle 1 Messpunkte mit deren Koordinaten und Lautstärke

In Tabelle 1 sind alle Testwerte aufgelistet. Die Messpunkte beziehen sich auf Abbildung 17 und die Koordinaten sind die tatsächlich von der Applikation gemessenen und für die Interpolation verwendeten Koordinaten. Für die vier roten Konfigurationspunkte ist die gemessene Lautstärke der Wert des Testtons, welcher auch abgespeichert wurde. Für die grünen Messpunkte ist der Wert die tatsächlich aufgenommene Lautstärke während des Tests, welche in einem perfekten Szenario genau gleich sein sollten. Die mögliche Lautstärke liegt im Bereich von 0.0 – 1.0, wobei 1.0 am lautesten ist.

Die gemessene Lautstärke der Konfigurationspunkte kann jedoch nicht direkt mit denen der Testpunkte verglichen werden, weil die Lautstärke während den Tests normalisiert wird und für die Tests die tatsächlich gemessene Lautstärke verwendet wird.

Der Test zeigt, dass die Lautstärke der Messpunkte ungefähr gleich ist. Die maximale Abweichung beträgt 0.072. An den Punkten, die näher beim Lautsprecher sind, wurde tendenziell eine tiefere Lautstärke gemessen als bei den weiter entfernten Punkte.

Die gemessene Lautstärke ist jedoch kein exakter Wert, wie zusätzliche Tests ergeben haben. Bei der vom Mikrofon zurückgegebene Lautstärke existiert eine gewisse Schwankung, obwohl für die gemessene Lautstärke der Mittelwert von 5 Sekunden genommen wurde. Die Schwankung existiert jedoch auch bei einer Stille und gleichbleibender Position und ist somit auf das Mikrofon zurückzuführen.

5 Diskussion

5.1 Resultate

Während den Tests hat sich abgezeichnet, dass bei Punkten, welche sich näher am Lautsprecher befinden, die Lautstärke zu leise ist. Dies wurde jedoch während der Entwicklung nicht in allen Räumen beobachtet. Es lässt sich darauf schliessen, dass der Balancing-Algorithmus weit entfernte Punkte im Testraum zu hoch gewichtet hat und dieser noch Verfeinerung und mehr Tests in verschiedenen Räumen braucht.

Ausserdem wurde, wie in den Resultaten erläutert, eine gewisse Unregelmässigkeit bei der aufgenommenen Lautstärke erkannt. Auch bei gleichbleibender Position und unveränderter Lautstärke, beziehungsweise auch bei Stille, variiert die aufgenommene Lautstärke teilweise erheblich. Würde das verwendete Mikrofon eine bessere Qualität aufweisen, könnten auch die Tests genauer durchgeführt werden. Weil das gleiche Mikrofon auch für die Konfiguration verwendet wurde, wäre es auch möglich, dass in diesem Schritt bereits eine fehlerhafte Lautstärke abgespeichert wurde und dies Auswirkungen auf die Testwerte hatte.

Auch trotz leichten Schwankungen in den Testwerten sind sie aussagekräftig, da sie eine klare Tendenz aufzeigen: eine ungefähr gleichbleibende Lautstärke im ganzen Raum. Dies entspricht auch den Erwartungen, was aufzeigt, dass die grundsätzliche Idee erfolgreich umgesetzt werden konnte.

Für eine weiterführende Arbeit sollte dieser Teil aber auf jeden Fall verbessert werden. Mit einem qualitativ hochwertigeren Aufnahmegerät kann sowohl die Konfiguration genauer durchgeführt werden, als auch die Tests, welche für eine Verbesserung der verwendeten Algorithmen zentral sind. Es wäre auch möglich, gewisse Parameter der Algorithmen konfigurierbar zu machen, wenn sie pro Raum unterschiedliche Resultate bewirken.

5.2 Rückblick Zielsetzung

Die in Kapitel 1.3 festgelegte Zielsetzung wurde durch die vorliegende Projektarbeit erreicht. Die Applikation Real-Stereo lokalisiert den Hörer im Raum mithilfe von zwei Kameras kontinuierlich und balanciert die Lautstärke der vom Benutzer ausgewählten Lautsprecher mithilfe der implementierten Algorithmen, sodass diese vom Zuhörer von allen Positionen gleich laut wahrgenommen werden. Die für das Windows Betriebssystem entwickelte Programm steuert die Audio Kanäle mithilfe von Windows APIs. Dadurch werden alle Audioquellen und Wiedergabeprogramme, welche unter Windows laufen, unterstützt. Dies ermöglicht die grösstmögliche Flexibilität, welche mit einer reinen PC-Lösung erreicht werden kann.

Die Konfiguration der Applikation wird erleichtert, indem dem Benutzer genaue Anweisungen angezeigt werden. Damit ist die Durchführung der Raumkonfiguration intuitiv und schnell durch neue Benutzer erlernt. Der aktuelle Fortschritt und neue Anweisungen werden sofort aktualisiert und prominent dargestellt.

5.3 Ausblick

Obwohl die Zielsetzung als erfüllt angesehen werden kann, gibt es doch noch viele Verbesserungen, welche in einem weiterführenden Projekt durchgeführt werden könnten. Vor allem ist das aktuelle Produkt eine reine PC-Implementation, wodurch andere Audioquellen, wie zum Beispiel Fernseher oder dedizierte CD-Spieler, nicht angesteuert werden können. Durch die reine PC-Lösung wird auch die alltägliche Benutzung erschwert, weshalb ein unabhängiges System vorgeschlagen wird. Weil in diesem keine Steuerung mehr über die Windows Audio Ausgabe möglich ist, müssen Alternativen gefunden werden. Die am vielversprechendste Alternative ist die Ansteuerung von wireless Sound-Systemen. Öffentliche, vom Hersteller dokumentierte APIs würden die Implementierung erleichtern.

Auch ist durch den Emgu CV .NET Wrapper die Geschwindigkeit der OpenCV Modelle für die Personenerkennung stark beeinträchtigt. Dieser unterstützt kein Multithreading und hat generell einen grossen Overhead. Durch die Nutzung der Python-Version von OpenCV könnte die offizielle Version verwendet werden, wodurch die Leistung stark verbessert werden könnte, wie bereits erste Vergleiche aufgezeigt haben.

6 Verzeichnisse

6.1 Literaturverzeichnis

[1] H. Di, L. Sun, L. Tao und G. Xu, «A Robust Approach for Person Localization in Multi-camera Environment», in *20th International Conference on Pattern Recognition*, Istanbul, 2010.

[2] D. Kocur, T. Porteleky und M. Švecová, «UWB Radar Testbed System for Localization of Multiple Static Persons», in *IEEE SENSORS*, Montreal, 2019.

[3] M. Ding, M. Goseki, H. Mizoguchi und H. Takemura, «Combination of microphone array processing and camera image processing for visualizing sound pressure distribution», in *IEEE International Conference on Systems, Man, and Cybernetics*, Anchorage, 2011.

[4] Sonos. (o. J.). *Sonos | Wireless Speakers and Home Sound Systems* [Online]. URL: https://www.sonos.com/ [Stand: 07.12.2020]

[5] Oracle. (o. J.). *Package javax.sound.sampled* [Online]. URL: https://docs.oracle.com/javase/7/docs/api/javax/sound/sampled/package-summary.html [Stand: 07.12.2020]

[6] Microsoft Corporation. (05.12.2018). *IAudioEndpointVolume:: SetChannelVolumeLevel method (endpointvolume.h)* [Online]. URL: https://docs.microsoft.com/en-us/windows/win32/api/endpointvolume/nf-endpointvolume-iaudioendpointvolume-setchannelvolumelevel [Stand: 06.12.2020]

[7] Sonos. (o. J.). *Control API* [Online]. URL: https://developer.sonos.com/reference/control-api/ [Stand: 06.12.2020]

[8] I. Landwerth. (12.11.2014). *.NET Core is Open Source* [Online]. URL: https://devblogs.microsoft.com/dotnet/net-core-is-open-source/ [Stand: 07.12.2020]

[9] S. Khan. (13.08.2019). *What Stats and Surveys are saying about .Net Core in 2020* [Online]. URL: https://dottutorials.net/stats-surveys-about-net-core-future-2020/ [Stand: 07.12.2020]

[10] Microsoft. (18.07.2019). *What is Windows Presentation Foundation (WPF .NET)* [Online]. URL: https://docs.microsoft.com/en-us/dotnet/desktop/wpf/overview/?view=netdesktop-5.0 [Stand: 09.12.2020]

[11] Microsoft. (12.03.2020). *XAML overview (WPF .NET)* [Online]. URL: https://docs.microsoft.com/en-us/dotnet/desktop/wpf/fundamentals/xaml?view=netdesktop-5.0 [Stand: 09.12.2020]

[12] M. Heath. (o. J.). *NAudio: Audio and MIDI library for .NET* [Online]. URL: https://github.com/naudio/NAudio [Stand: 07.12.2020]

[13] Microsoft. (31.05.2018). *About WASAPI* [Online]. URL: https://docs.microsoft.com/en-us/windows/win32/coreaudio/wasapi [Stand: 07.12.2020]

[14] OpenCV. (o. J.). *About OpenCV* [Online]. URL: https://opencv.org/about/ [Stand: 07.12.2020]

[15] Emgu CV. (23.11.2020). *Main Page* [Online]. URL: http://www.emgu.com/wiki/index.php/Main_Page [Stand: 07.12.2020]

[16] M. Theiler, Artist, *HOG scikit-image Angela Merkel.* [Art]. 2018.

[17] C. Tomasi. (o. J.). *Histograms of Oriented Gradients* [Online]. URL: https://www2.cs.duke.edu/courses/fall15/compsci527/notes/hog.pdf [Stand: 07.12.2020]

[18] W. Berger. (14.01.2018). *Deep Learning Haar Cascade Explained* [Online]. URL: http://www.willberger.org/cascade-haar-explained/ [Stand: 07.12.2020]

[19] A. Rosebrock. (25.05.2015). *Basic motion detection and tracking with Python and OpenCV* [Online]. URL: https://www.pyimagesearch.com/2015/05/25/basic-motion-detection-and-tracking-with-python-and-opencv/ [Stand: 07.12.2020]

[20] Microsoft. (31.05.2018). *Microsoft Media Foundation* [Online]. URL: https://docs.microsoft.com/en-us/windows/win32/medfound/microsoft-media-foundation-sdk [Stand: 09.12.2020]

[21] Snarfle. (21.11.2016). *NuGet MediaFoundation* [Online]. URL: https://www.nuget.org/packages/MediaFoundation/ [Stand: 09.12.2020]

[22] Microsoft. (31.05.2018). *MF_DEVSOURCE_ATTRIBUTE_FRIENDLY_NAME attribute* [Online]. URL: https://docs.microsoft.com/en-us/windows/win32/medfound/mf-devsource-attribute-friendly-name [Stand: 09.12.2020]

[23] 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.

[24] E. J. Kirkland, «Bilinear Interpolation», in *Advanced Computing in Electron Microscopy*, o. O., Springer, 2010, S. 261-263.

[25] o. V. (o. J.). *2-D Interpolation* [Online]. URL: http://fourier.eng.hmc.edu/e176/lectures/ch7/node7.html [Stand: 08.12.2020]

[26] H.-K. Nienhuys, Artist, *Shepards interpolation method, sampling a smooth function.* [Art]. 2012.

[27] Newtonsoft. (o. J.). *Popular high-performance JSON framework for .NET* [Online]. URL: https://www.newtonsoft.com/json [Stand: 10.12.2020]

[28] Headset.net. (19.08.2019). *Beyerdynamic MMX 2 USB-Gaming-Headset Experteneinschätzung* [Online]. URL: https://www.headset.net/beyerdynamic/mmx-2/ [Stand: 11.12.2020]

[29] Logitech. (o. J.). *Logitech C270 HD Webcam* [Online]. URL: https://www.logitech.com/en-ch/product/hd-webcam-c270 [Stand: 11.12.2020]

6.2 Abbildungsverzeichnis

6.3 Tabellenverzeichnis

6.4 Abkürzungsverzeichnis

Abkürzung	Bedeutung
HOG	Histogram of oriented gradients
WASAPI	Windows Audio Session API

7 Anhang

7.1 Projektmanagement

7.1.1 Offizielle Aufgabenstellung

Stereo-Sound (Stereofonie) gibt es seit bald 100 Jahren. Für einen optimalen Höreindruck müssen aber eine ganze Reihe von Bedingungen erfüllt sein. Abgesehen vom Aufstellungsort und dem akustischen Einfluss durch den Raum ist vor allem wichtig, dass die Distanz zu den beiden Lautsprechern identisch ist. Leider ist dies selten gegeben, und völlig unmöglich, wenn man sich gleichzeitig im Raum bewegt.

In dieser Arbeit soll mittels Kameras die Position des Zuhörers im Raum kontinuierlich bestimmt werden und die Lautstärke der beiden (oder mehr) Lautsprecher so angepasst werden, dass sie gleich Laut bei gleichbleibender, wahrgenommener Gesamt-Lautstärke durch den Zuhörer wahrgenommen werden. Dadurch soll der Stereo-Eindruck grundlegend verbessert werden.

Um die räumlichen Aspekte genügend zu berücksichtigen, muss es möglich sein, das System "anzulernen", so dass es für verschiedene Positionen im Raum weiss, wie die akustischen Verhältnisse sind (z.B. durch Ausmessen mit einem Mikrofon). Für Positionen dazwischen muss dann interpoliert werden.

In der Projektarbeit soll zunächst mit einer reinen PC-Lösung das Konzept (Positionsbestimmung mit Kameras, Raumvermessung, Lautstärkeregelung) geprüft und verbessert werden. Dabei kann auch untersucht werden, ob und wie bestehende Systeme (z.B. Sonos: https://developer.sonos.com/) dank bestehenden APIs durch diesen Ansatz erweitert werden können. In einer möglichen Bachelor-Arbeit könnte dann ein "reales" System entwickelt werden.

Folgende Komponenten sollen in einem ersten Schritt betrachtet werden:

1. Konfiguration der Anwendung
2. Positionsbestimmung
3. Lautstärkeregelung

7.1.2 Zeitplan

Projektstart: 14.09.2020
Projektende: 19.12.2020

Meilensteine / Tätigkeiten	Aufwand	September 22.	29.	Oktober 06.	13.	20.	27.	November 03.	10.	17.	24.	Dezember 01.	08.	15.	19.	?
Planung		*														
Evaluation Personen Track	6															
Evaluation Konfiguration	6															
Evaluation Audioausgabe	6															
Konfiguration									*							
Konfiguration Raum	40															
Konfiguration Speaker	20															
Testing	20															
Personen Tracking						*				*						
Vergleich Libraries	20															
Implementation	60															
Testing	20															
Audioausgabe											*					
Implementation	20															
Gesamtintegration												*				
Testing	20															
Dokumentation															*	
Dokumentation / Bericht	60															
Überarbeitung	20															
Präsentation																*
Vorbereitung	20															

7.1.3 Arbeitszeiten

Die Folgende Tabelle zeigt die Arbeitszeiten und Aufteilung im Team auf. Marc Berchtold hat sich dabei auf das Recording und Balancing spezialisiert, während Cyril Wanner den Fokus auf die Personenerkennung und das Tracking gelegt hat. Code-Reviews sorgten dafür, dass alle Mitglieder einen Überblick über das ganze Projekt hatten und sich gegenseitig unterstützen konnten. Ideen und Lösungswege wurden dabei jeweils zu zweit besprochen, bevor sie implementiert wurden.

Datum	Beschreibung	Zeit (in h)	Wer
16.09.20	Recherche zu Indoor Localization / Suche nach UWB Dev Modulen	3.00	Marc
19.09.20	Recherche zu Indoor Localization mit Computer Vision	3.00	Cyril
20.09.20	Projektplan	2.00	Cyril
20.09.20	Recherche und Dokumentation zu UWB Indoor Localization	2.00	Marc
22.09.20	Meeting & Planung	1.00	Cyril
22.09.20	Meeting & Planung	1.00	Marc
23.09.20	Evaluation Webcam	2.00	Cyril
25.09.20	Suche nach alternativer Webcam	1.00	Marc
27.09.20	Konfigurations Konzept	2.00	Cyril
28.09.20	Diskussion Konfigurations Konzept	1.00	Marc
28.09.20	Evaluation Audio Playback	2.00	Marc
28.09.20	Diskussion, Ergänzung, Code Beispiele	2.00	Cyril
29.09.20	Meeting + Protokoll & Planung	1.00	Cyril
29.09.20	Meeting & Planung	1.00	Marc
03.10.20	Evaluation Library (OpenCV)	3.50	Marc
05.10.20	Evaluation Library (Audio)	3.50	Marc
05.10.20	Evaluation Erkennungsmodell	3.00	Cyril
06.10.20	Setup Dev Environment	3.50	Cyril
06.10.20	Meeting + Protokoll & Planung	1.00	Marc
06.10.20	Meeting & Planung	1.00	Cyril
10.10.20	Project setup, Emgu integration into WPF .NET Core	5.50	Cyril
10.10.20	Wireframes Main & Configuration Window	3.50	Marc
11.10.20	Person detection	5.00	Cyril
12.10.20	Webcam Auswahl Dropdowns	4.00	Marc
12.10.20	Improve person detection, calculate coordinates	4.00	Cyril
13.10.20	Meeting + Protokoll & Planung	1.00	Cyril
13.10.20	Meeting & Planung	1.00	Marc
16.10.20	Improve person detection	3.00	Cyril
17.10.20	Layout & Audio device selector	4.50	Marc
18.10.20	Code review	1.00	Cyril
19.10.20	GUI Implementations	4.00	Cyril
20.10.20	Multiple Improvements	3.50	Cyril
20.10.20	Meeting & Planung	1.00	Cyril
20.10.20	Meeting + Protokoll & Planung	1.00	Marc
24.10.20	Rewrite camera list with MSMF & audio channel names	5.00	Marc
24.10.20	Multithreading	4.00	Cyril
25.10.20	Multithreading	3.50	Cyril
27.10.20	Meeting & Planung	1.00	Marc
29.10.20	Meeting + Protokoll & Planung	1.00	Cyril
29.10.20	Configuration process	4.50	Cyril
30.10.20	Camera configuration	3.50	Cyril
31.10.20	Audio configuration, play test sound on each channel	5.00	Marc
01.11.20	Audio configuration, record sound & interpret raw PCM stream	5.00	Marc
05.11.20	Audio configuration, record test tone fixed	3.50	Marc
06.11.20	Configuration improvements	3.50	Cyril
07.11.20	Configuration improvements	2.50	Cyril
09.11.20	Save/load config from disk	4.00	Cyril
09.11.20	Low & high-pass filter for recording test tone	3.50	Marc
10.11.20	Meeting & Planung	1.00	Cyril
10.11.20	Meeting + Protokoll & Planung	1.00	Marc
14.11.20	Audio testing, window close bugfix	3.50	Cyril

Datum	Beschreibung	Zeit (in h)	Wer
15.11.20	Volume interpolation (first version)	6.00	Cyril
16.11.20	Test tone recording volume boost	4.50	Marc
17.11.20	Meeting & Planung	1.00	Marc
17.11.20	Meeting + Protokoll & Planung	1.00	Cyril
22.11.20	Improve interpolation	3.50	Cyril
23.11.20	Implement balancing from calculated interpolation	3.00	Marc
24.11.20	Meeting + Protokoll & Planung	1.00	Marc
24.11.20	Meeting & Planung	1.00	Cyril
24.11.20	Improve calculation & debugging experience	3.50	Cyril
29.11.20	Testing & improvements	4.00	Cyril
01.12.20	Meeting & Planung	1.00	Marc
01.12.20	Meeting + Protokoll & Planung	1.00	Cyril
05.12.20	Zusammenfassung & Abstract	4.00	Marc
05.12.20	Improvements	4.00	Cyril
06.12.20	Refactoring	2.50	Cyril
06.12.20	Theoretische Grundlagen - Wiedergabe	2.50	Marc
06.12.20	Testing	3.00	Cyril
06.12.20	Bericht - Layout	1.50	Cyril
07.12.20	Bericht - Methoden	4.50	Marc
07.12.20	Bericht - Einleitung, theoretische Grundlagen, Methoden	6.00	Cyril
08.12.20	Meeting + Protokoll & Planung	1.00	Marc
08.12.20	Meeting & Planung	1.00	Cyril
08.12.20	Bericht - Methoden	5.00	Marc
08.12.20	Bericht - Methoden	4.00	Cyril
09.12.20	Bericht - Methoden, Resultate	5.50	Cyril
09.12.20	Bericht	3.50	Marc
10.12.20	Bericht - Resultate, Diskussion, Management Summary	5.00	Marc
10.12.20	Bericht - Methoden, Resultate	4.50	Cyril
11.12.20	Bericht - Methoden, Resultate	4.50	Cyril
12.12.20	Bericht - Resultate, Diskussion	4.00	Cyril
12.12.20	Bericht - Rückblick, Ausblick	3.00	Marc
13.12.20	Bericht - Anhang	2.00	Marc
13.12.20	Bericht - Überarbeitung	3.50	Cyril
14.12.20	Bericht - Überarbeitung	2.00	Cyril
15.12.20	Bericht - Überarbeitung	4.00	Cyril
15.12.20	Bericht - Überarbeitung	4.50	Marc
15.12.20	Präsentation	2.00	Marc
16.12.20	Improve code style & documentation	4.00	Cyril
16.12.20	Bericht - Überarbeitung	3.00	Marc
17.12.20	Bericht - Überarbeitung	4.50	Marc
17.12.20	Bericht - Überarbeitung	3.00	Cyril
17.12.20	Vorbereitung Präsentation & Demo	3.00	Marc
17.12.20	Vorbereitung Präsentation & Demo	4.00	Cyril
18.12.20	Vorbereitung Präsentation & Demo	4.00	Cyril
18.12.20	Vorbereitung Präsentation & Demo	4.00	Marc
19.12.20	Präsentation & Demo	1.50	Cyril
19.12.20	Präsentation & Demo	1.50	Marc

7.1.4 Besprechungsprotokolle

Die Besprechungsprotokolle für alle wöchentlichen Meetings können im PDF Format im *meeting_protocols* Ordner des GitHub Repository des Projektes (https://github.com/cyrilwanner/real-stereo) gefunden werden.

Die verfügbaren Besprechungsprotokolle sind hier aufgelistet.

```
meeting_protocols/
├── Beschlussprotokoll 15.09.2020.pdf
├── Beschlussprotokoll 22.09.2020.pdf
├── Beschlussprotokoll 29.09.2020.pdf
├── Beschlussprotokoll 06.10.2020.pdf
├── Beschlussprotokoll 13.10.2020.pdf
├── Beschlussprotokoll 20.10.2020.pdf
├── Beschlussprotokoll 27.10.2020.pdf
├── Beschlussprotokoll 10.11.2020.pdf
├── Beschlussprotokoll 17.11.2020.pdf
├── Beschlussprotokoll 24.11.2020.pdf
├── Beschlussprotokoll 01.12.2020.pdf
└── Beschlussprotokoll 08.12.2020.pdf
```

7.2 Weiteres

7.2.1 Quellcode

Der Quellcode der Software ist im GitHub Repository des Projektes (https://github.com/cyrilwanner/real-stereo) abgelegt.

Die Verzeichnisstruktur des Quellcodes ist hier als Übersicht angehängt.

```
.
├── RealStereo/
│   ├── Balancing/
│   │   ├── Speaker/
│   │   │   ├── AudioChannelMap.cs
│   │   │   ├── VolumeFader.cs
│   │   │   └── VolumeInterpolation.cs
│   │   ├── Tracking/
│   │   │   ├── Camera.cs
│   │   │   └── PeopleDetector.cs
│   │   ├── WorkerResult.cs
│   │   └── WorkerThread.cs
│   ├── Config/
│   │   ├── Steps/
```

```
│   │   │   ├── ConfigurationStep.cs
│   │   │   ├── ConfigurationStepCamera.cs
│   │   │   └── ConfigurationStepSpeaker.cs
│   │   ├── Configuration.cs
│   │   ├── ConfigurationManager.cs
│   │   ├── PointConfiguration.cs
│   │   └── TestTone.cs
│   ├── Ui/
│   │   ├── ConfigurationWindow.xaml
│   │   ├── ConfigurationWindow.xaml.cs
│   │   ├── MainWindow.xaml
│   │   ├── MainWindow.xaml.cs
│   │   ├── VolumeInterpolationDebugWindow.xaml
│   │   └── VolumeInterpolationDebugWindow.xaml.cs
│   ├── App.xaml
│   ├── App.xaml.cs
│   ├── AssemblyInfo.cs
│   └── RealStereo.csproj
├── diagrams/
├── meeting_protocols/
├── wireframes/
├── .editorconfig
├── .gitignore
├── README.md
└── RealStereo.sln
```