

Hochschule für angewandte Wissenschaften Coburg  
Fakultät Elektrotechnik und Informatik

Studiengang: Informatik

**Communication Systems – Projekt Smart Audio**

Marvin Roth

Patrick Schneider

Yannick Helling

Stephan Wachter

Abgabe der Arbeit: 17. August 2020

Betreut durch:

Prof. Dr. Matthias Mörz, Hochschule Coburg

# Inhaltsverzeichnis

[Inhaltsverzeichnis 2](#_Toc48486176)

[Abbildungsverzeichnis 4](#_Toc48486177)

[Tabellenverzeichnis 5](#_Toc48486178)

[Codeverzeichnis 6](#_Toc48486179)

[1 Einleitung 7](#_Toc48486180)

[2 Theoretische Grundlagen 8](#_Toc48486181)

[2.1 Komponentenauswahl 8](#_Toc48486182)

[2.1.1 PIR-Sensor 8](#_Toc48486183)

[2.1.2 Radar-Modul 8](#_Toc48486184)

[2.1.3 Bluetooth-Beacon 9](#_Toc48486185)

[2.1.4 Kamera 9](#_Toc48486186)

[2.1.5 Laserschranke 10](#_Toc48486187)

[2.1.6 Ultraschallsensoren 10](#_Toc48486188)

[2.1.7 Fazit 10](#_Toc48486189)

[2.2 Die WiringPi Bibliothek 11](#_Toc48486190)

[2.3 Distanzmessung mit dem HC-SR04 Ultraschallmodul 13](#_Toc48486191)

[2.4 MQTT Kommunikation 17](#_Toc48486192)

[2.5 SmartAudio MQTT Library 18](#_Toc48486193)

[3 Hardwareaufbau 21](#_Toc48486194)

[3.1 Anschluss der Sensoren 21](#_Toc48486195)

[3.2 Anordnung der Sensoren 23](#_Toc48486196)

[4 Software 25](#_Toc48486197)

[4.1 Volumio 25](#_Toc48486198)

[4.2 Skripte zur automatischen Installation der benötigten Bibliotheken 25](#_Toc48486199)

[4.3 Installation von Snapcast für Volumio 26](#_Toc48486200)

[4.4 Client 27](#_Toc48486201)

[4.4.1 Durchgangsdetektion und Serverkommunikation 27](#_Toc48486202)

[4.4.2 Fehler und -behebung 31](#_Toc48486203)

[4.4.3 Buildskript zur Kompilierung der Client-Software 32](#_Toc48486204)

[4.4.4 Einrichten des Bluetooth Audio Outputs 33](#_Toc48486205)

[4.4.5 Hostname, SSID, WLAN beim Booten, VNC und SSH 33](#_Toc48486206)

[4.4.6 Autostart der Software 36](#_Toc48486207)

[4.5 Master 37](#_Toc48486208)

[4.5.1 MQTT Auswertesoftware 37](#_Toc48486209)

[4.5.2 Auswertung der Durchgangsdaten mit Reaktion 39](#_Toc48486210)

[4.6 Verworfene Konzepte und Problematiken 40](#_Toc48486211)

[4.6.1 Verwendung von Volumio auf allen Geräten 40](#_Toc48486212)

[4.6.2 Steuerung von Volumio 41](#_Toc48486213)

[4.6.3 Verwurf von Paho-MQTT als MQTT-Library 41](#_Toc48486214)

[4.6.4 Auslesen der Konfiguration aus Ini-Dateien schlägt fehl 42](#_Toc48486215)

[4.6.5 Leistungsgrenzen des Raspberry Pi Zero W 42](#_Toc48486216)

[5 Zusammenfassung und Ausblick 43](#_Toc48486217)

[5.1 Zusammenfassung 43](#_Toc48486218)

[5.2 Ausblick 43](#_Toc48486219)

[Literaturverzeichnis 45](#_Toc48486220)

[Anhang A 1. Ehrenwörtliche Erklärung 47](#_Toc48486221)

[Anhang A 2. GetDistance-Methode 48](#_Toc48486222)

[Anhang A 3. Sicherheitsdistanzmessungen 50](#_Toc48486223)

[Anhang A 4. Skript InstallRequirementsMaster.sh 52](#_Toc48486224)

# Abbildungsverzeichnis

[Abb. 1: Raspberry Pi Pintabelle von „gpio readall“ 13](#_Toc48486225)

[Abb. 2: HC-SR04 Sender und Empfänger mit Objekt 13](https://d.docs.live.net/360032c51c1a6c1d/Dokumente/projektbericht.docx#_Toc48486226)

[Abb. 3: Anschlussplan des HC-SR04 [TutRasp] 14](https://d.docs.live.net/360032c51c1a6c1d/Dokumente/projektbericht.docx#_Toc48486227)

[Abb. 4: Schematischer Aufbau der MQTT Kommunikation 17](#_Toc48486228)

[Abb. 5: Anschluss zwei HC-SR04 mit 16 Kabeln 22](#_Toc48486229)

[Abb. 6: Anschluss zwei HC-SR04 mit 8 Kabeln 23](#_Toc48486230)

[Abb. 7: HC-SR04 mit geringem Abstand montieren 24](#_Toc48486231)

[Abb. 8: Improvisierte Aufhängung mit ca. 160° Winkel 24](#_Toc48486232)

[Abb. 9 Funktionsweise Sensorauswertung 29](#_Toc48486233)

[Abb. 10: In- / Out-Fehler und korrekte Messungen 31](#_Toc48486234)

[Abb. 11: Testergebnis erster zehn Stunden Test 32](#_Toc48486235)

[Abb. 12: Testergebnis zweiter zehn Stunden Test 32](#_Toc48486236)

[Abb. 13: Raspi-Config 36](#_Toc48486237)

[Abb. 14: Veranschaulichung einer eingehenden MQTT Nachricht 38](#_Toc48486238)

[Abb. 15: Raumplan im Test 39](#_Toc48486239)

# Tabellenverzeichnis

[Tab. 1 Hardwareliste 21](#_Toc48486240)

# Codeverzeichnis

[Code 1: Distanzmessung mit dem HC-SR04 in C 16](#_Toc48486241)

[Code 2: Ausschnitt aus dem Installationsskript 26](#_Toc48486242)

[Code 3: Initialisierung 28](#_Toc48486243)

[Code 4: Durchgangsdetektion 31](#_Toc48486244)

[Code 5: build.sh 33](#_Toc48486245)

[Code 6: hostname.sh 34](#_Toc48486246)

[Code 7: Automatische Konfiguration des WLAN 35](#_Toc48486247)

[Code 8: Inhalt mosquitto.service 36](#_Toc48486248)

[Code 9: Inhalt startsmartaudio.sh 37](#_Toc48486249)

[Code 10: Mainloop des Masters (Ausschnitt) 38](#_Toc48486250)

[Code 11: config.ini auf dem Master zur Raumdefinition 40](#_Toc48486251)

[Code 12: Grundlegender Aufbau eines Volumio Kommandozeilenbefehls 41](#_Toc48486252)

# Einleitung

Dieser Bericht fasst die Ergebnisse und Erkenntnisse des Moduls Communication Systems aus dem Wintersemester 2019/2020 zusammen. Als Ziel wurde die Entwicklung eines „intelligenten“ Audiosystems gewählt. Das heißt konkret, dass das Audiosystem auf Wunsch nur in solchen Räumen Audiodateien wiedergibt, in welchen auch Personen anwesend sind. In jedem Raum soll ein Client verfügbar sein, welcher einerseits mit einem Server verbunden ist, von welchem er einen Audiostream (d. h. eine Audioquelle wie z. B. Musik vom USB-Stick, Internetradio, usw…) erhält, und andererseits mit einem Audiowiedergabegerät verbunden ist, auf welchem der Audiostream wiedergegeben wird.

1. Theoretische Grundlagen

In diesem Kapitel werden einige theoretische Grundlagen behandelt, wie der Aufbau der verwendeten Bibliotheken für die Nutzung der Sensoren, der Erfassung der Distanzen und MQTT. Ebenfalls werden kurz mögliche in Frage kommende Komponenten bezüglich deren Funktionsweise näher erläutert. Dieses Kapitel liefert lediglich Vorwissen, das zum besseren Verständnis der Arbeit beiträgt.

## Komponentenauswahl

Ein wichtiger Teil des Smart-Audio Systems ist die Erkennung von Personen, die einen Raum betreten oder sich darin aufhalten. Deshalb soll nun erörtert werden mit Hilfe welcher Technologie ein solcher Vorgang detektiert werden kann. Nach einigen Recherchen wurde die Auswahl auf folgende Verfahren eingeschränkt: Pyroelektrischer Infrarot Sensor (PIR-Sensor), Radar-Modul, Bluetooth-Beacon, Kamera, Laserschranke, Ultraschallsensoren und Kombinationen daraus.

### PIR-Sensor

Der PIR-Sensor wird als Bewegungsmelder verwendet, bei dem Unterschiede in der Infrarotstrahlung eines Raumes als Bewegung wahrgenommen werden. Für den Einsatz dieses Sensors spricht der geringe Preis, der bei nur wenigen Euro liegt, sowie die Möglichkeit auch kleine Bewegungen wahrzunehmen, jedoch können auch auch Haustiere oder Kinder erkannt werden, die das Audiosystem nicht registrieren soll. Der Winkel des Sensors beträgt zwar 135 Grad allerdings kann es in bestimmten Raumkonstellationen zu toten Winkeln kommen, welche durch weitere Sensoren kompensiert werden müssten, was wieder den Preis nach oben treibt. Ein anderer Nachteil ist das kein Raumübergang detektiert werden kann, da nur Bewegungen aber keine Richtungen erkannt werden, weshalb stilles Liegen als leerer Raum angesehen werden könnte, die Musik deaktiviert wird und man sich bewegen muss damit sie wieder startet.

### Radar-Modul

Das Radar-Modul wird ebenfalls zur Bewegungsdetektion verwendet doch im Gegensatz zum PIR-Sensor, der Infrarotlicht verwendet, nutzt das Radar-Modul Mikrowellen welche die Messungen noch empfindlicher machen, so werden nicht nur Infrarotquellen erkannt sondern auch andere Arten von Bewegungen wie wehende Vorhänge oder sogar Fußgänger die auf der Straße vorbeilaufen. Durch den höheren Preis und durch die komplexere Art das Radar-Modul anzusprechen bzw. auszuwerten, schneidet das Radar-Modul im Vergleich zum PIR-Sensor schlechter ab.

### Bluetooth-Beacon

Die Beacon Technologie funktioniert nach dem Prinzip, dass ein im Raum platziertes Gerät (Beacon) ein kontinuierliches BLE-Signal (Bluetooth Low Energy) sendet das z.B. von einem Smartphone empfangen wird und aus der Signalstärke die Entfernung vom Beacon zum Empfängergerät berechnet. Der Vorteil dieses Verfahrens ist, dass keine weiteren Kabel benötigt werden, da jedes Gerät das BT 4.0 unterstützt als Bluetooth-Beacon verwendet werden kann. So bietet sich der Raspberry Pi als Bluetooth-Beacon an, da er nicht nur über ein Bluetooth-Modul verfügt, sondern auch das empfangene Signal verarbeiten kann. So gut dies auch klingt gibt es auch einige Nachteile: Bei der Beacon-Technologie muss man stets das Smartphone mitführen, wenn man das System nutzen möchte. Ein Bluetooth-Beacon kann allerdings höchstens eine Distanz mit einer Genauigkeit von 10-50 Metern zu einem Empfängergerät ermitteln. Für eine genauere Positionsbestimmung müsste Bluetooth-RSSI oder Bluetooth-Fingerprinting angewendet werden. Dadurch lässt sich die Genauigkeit auf 3-10 Meter verbessern [DIVA, Tabelle 1.1]. Mit einer Anzahl von 6 Bluetooth-Beacons lässt sich die Genauigkeit auf 1,3 – 1,9 Meter verbessern. [DIVA, Tabelle 5.1] Der mit der Beacon-Technologie verbundene Initialisierungsaufwand und die Anzahl der Beacons, die man benötigt um relativ genau bestimmen zu können ob sich tatsächlich jemand im Raum aufhält oder nicht lies uns zu dem Schluss kommen, dass die Technologie für das Projekt eher ungeeignet ist. Die Tatsache, dass Geräte und keine Personen geortet werden ist das K.O. Kriterium.

### Kamera

Mit einer Kamera könnte man durch Gesichtsdetektion, mit mehr Aufwand möglicherweise eine Körperdetektion, genau feststellen ob sich eine Person im Raum befindet und somit Haustiere oder Sensorfehler ausschließen. Doch wie beim PIR-Sensor gibt es tote Winkel, wodurch mehrere Kameras vonnöten wären, was sich wiederum im Preis niederschlägt. Außerdem stellt sich die Frage des Datenschutzes (mit dem Betreten des Raumes muss einer Erfassung per Kamera zugestimmt werden), da die wenigsten Leute permanente Kameraüberwachung in ihrem Haus haben wollen. Der Rechenaufwand ist ebenfalls nicht zu unterschätzen da die aufgenommenen Bilder noch verarbeitet werden müssen, um eine Person zu registrieren was möglicherweise die Leistung eines Raspberry Pi übersteigt.

### Laserschranke

Bei einer Laserschranke werden zwei Laser parallel im Türrahmen angebracht. Durch die Reihenfolge, in welcher die Laser durchbrochen werden, lässt sich die Richtung bestimmen in der die Tür durchschritten wurde. Der größte Vorteil der Laserschranke ist die Genauigkeit und Reichweite (Türbreite) in der detektiert werden kann. Außerdem gibt es keine toten Winkel da der Sensor in der Tür angebracht wird. Doch gibt es auch hier Nachteile, wie das Gesundheitsrisiko das entstehen kann, wenn man die Laser in der nähe der Augen verwendet. Bei stärkerem Licht mit höherer Intensität ist eine Schädigung der Augen (bei direktem Blick in das Licht) möglich. Ausnahme ist ein Laser der Klasse 1 welcher wiederum teurer als die ohnehin kostspieligen Laser einer höheren Klasse sind. Selbst die teuren Laser haben nur eine Lebensdauer von 6.000 Stunden, wodurch man einmal im Jahr einen Satz Laser kaufen müsste. Ein weiterer Grund, der gegen eine Laserschranke spricht, ist die Notwendigkeit eines Gegenstücks, dass auf der anderen Seite des Rahmens angebracht werden muss wodurch die Möglichkeit besteht das sich der Laser verschiebt, nicht mehr auf das Gegenstück trifft und somit keine Messungen mehr stattfinden können.

### Ultraschallsensoren

In der verwendeten Konfiguration werden die Ultraschallsensoren ebenso wie die Laser für eine Schranke verwendet. Die Montage am Türrahmen erfolgt in der Reihenfolge, in der die Sensoren anschlagen kann und ein Durchgang ermittelt wird. Der Nachteil an den Ultraschallsensoren ist, dass sie nicht wie der Laser in einer dünnen geraden Linie ausstrahlen, sondern in einer Kegelform. So kann es zu Überschneidungen kommen, wenn die Sensoren zu nahe beieinander angebracht werden. Außerdem wird durch die Kegelform die Überschneidungsfläche größer, je weiter die Messung, d. h. je breiter die Türe ist, erfolgen soll. Doch im Gegensatz zum Laser haben die Ultraschallsensoren eine lange Lebensdauer, kosten gerade mal wenige Euro und brauchen kein empfangendes Gegenstück, um zu funktionieren was sie, bis auf die Genauigkeit, dem Laser in fast allen Belangen überlegen machen.

### Fazit

Am Ende wurde sich für den Ultraschallsensor entschieden, da er keinen toten Winkel hat und die komplette Türbreite erfasst wie der PIR-Sensor oder das Radar-Modul, außerdem ist er günstiger als eine Kamera oder ein sicherer Laser mit hoher Lebensdauer. Zwar ist die Genauigkeit nicht so hoch wie bei einem Laser, aber immer noch im vertretbaren Rahmen und um ein vielfaches genauer als der Beacon weshalb der Ultraschallsensor mit dem geringen Preis die optimale Lösung ist. Die benötigte Hardware für den gewählten Aufbau wird in Tab. 1 aufgelistet (siehe Kapitel 3.1).

## Die WiringPi Bibliothek

Die erste Version der Bibliothek WiringPi wurde für den Raspberry Pi A1 und B1 geschrieben, um den Zugriff auf die 26 unterschiedlichen GPIO Pins zu erleichtern[WiringPi]. In der darauffolgenden Iteration wurde es ermöglicht alle 40 Pins des Raspberry Pi A2 und B2 zu verwenden. WiringPi enthält ein Command-Line Werkzeug namens gpio die es auch ermöglicht die Bibliothek in Shell-Skripten zu verwenden. Darunter auch der Befehl „gpio readall“ die eine Tabelle in der Command-Line erzeugt, welche alle Pins in physischer Anordnung (für den Raspberry Pi Zero), BCM\_GPIO Nummerierung und WiringPi Nummerierung auflistet.

Um die Funktionen der WiringPi Bibliothek zu nutzen muss man zu Beginn des Programms eine der 4 Setupfunktionen aufrufen, die festlegen welche Pinnummerierung man verwenden möchte.

* int WiringPiSetup(void)

wird verwendet, um die Pinnummerierung zu verwenden wie sie von WiringPi vorgegeben wird.

* int wiringPiSetupGpio(void)

Verwendet die BCM\_GPIO Pinnummerierung.

* int wiringPiSetupPhys(void)

Legt die physische Anordnung der Pins als Nummerierung fest, beginnend mit Pin 1 als der obere Linke 3,3V Pin. Pin 2 ist der gegenüberliegende 5V Pin und fortführend in der nächsten Reihe.

* int wiringPiSetupSys(void)

Die letzte Setupfunktion greift auf das Interface /sys/class/gpio zu und entnimmt die Nummerierung aus dieser Datei.

Da wie der Name GPIO schon sagt, können die Pins als Input oder Output genutzt werden, um die Richtung des Pins festzulegen. Hierzu verwendet man die Funktion pinMode:

* void pinMode(int pin, int mode)

pin ist die Nummer des Pins je nachdem welche Setupfunktion aufgerufen wurde.

mode hat 4 Möglichkeiten. INPUT, OUTPUT, PWM\_OUTPUT (nur für Hardware PWM Pins) und GPIO\_CLOCK (nur Pins die ein Clock-Signal generieren).

Der Status eines Pins wird überprüft indem man die Funktion

* int digitalRead(int pin)

verwendet. Sie gibt die Werte 1 für HIGH, und 0 für LOW, zurück.

Um einen Wert auf einen Pin zu legen gibt es

* void digitalWrite(int pin, int value)

pin steht für den Pin dessen Wert gesetzt werden soll.

value ist entweder HIGH oder LOW für 1 oder 0.

Da es auch manchmal von Nöten sein kann auf etwas zu warten, bietet WiringPi auch hier eine Möglichkeit, um das Programm zu pausieren.

* void delay (unsigned int howLong)

Pausiert das Programm für die Anzahl an Millisekunden, die in howLong festgelegt wurden.

* void delayMicroseconds(unsigned int howLong)

Wenn die Genauigkeit von Millisekunden nicht mehr ausreicht kann auf diese Funktion zurückgegriffen werden, die das Programm für die angegebene Anzahl an Mikrosekunden pausiert.

WiringPi verfügt noch über weitere Funktionen wie zum Beispiel PWM Unterstützung, eine SPI und I2C Bibliothek, doch diese werden in diesem Tutorial nicht erläutert da sie im Projekt Smart Audio keine Verwendung finden.

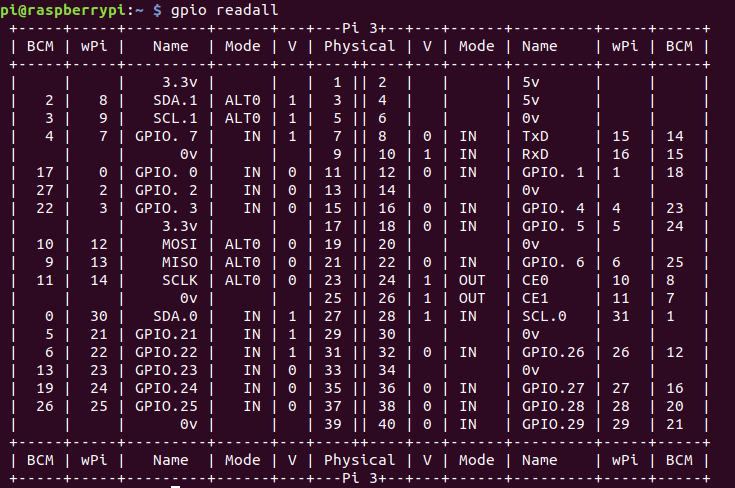


Abb. 1: Raspberry Pi Pintabelle von „gpio readall“

## Distanzmessung mit dem HC-SR04 Ultraschallmodul

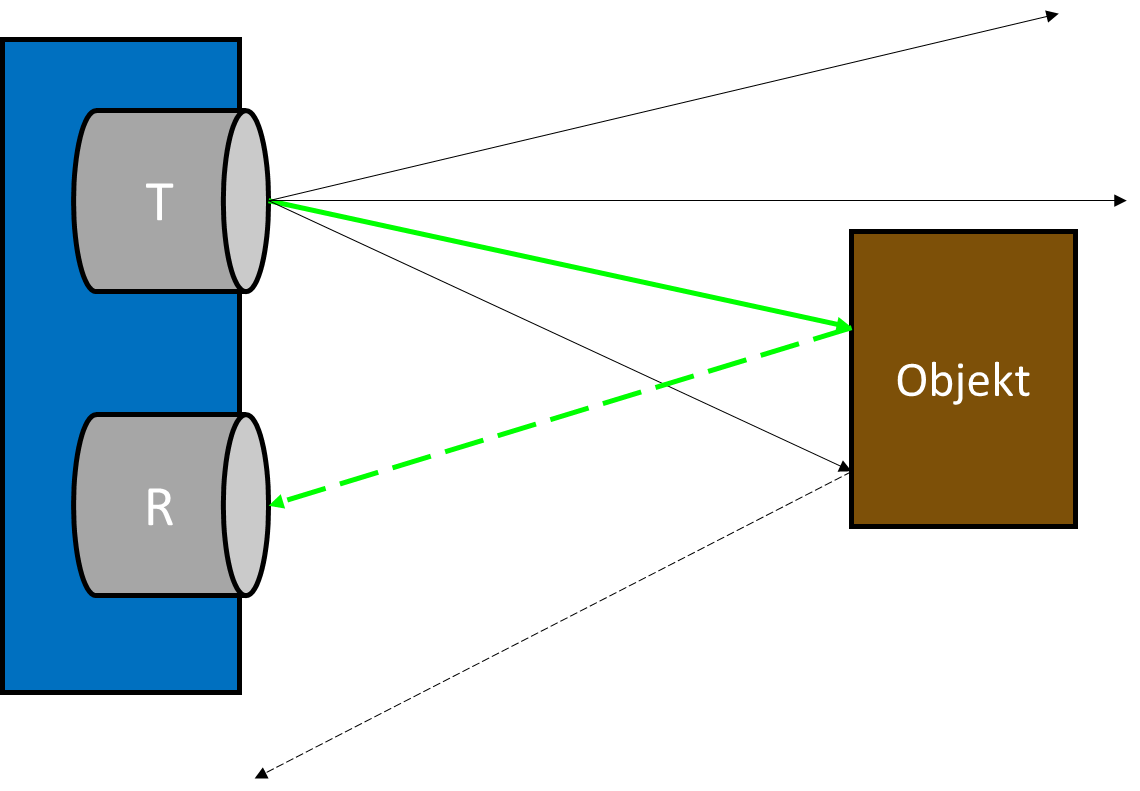
Das HC-SR04 Ultraschallmodul ist ein Modul mit einem Ultraschallsender und einem Ultraschallempfänger sowie den vier Pins VCC, Trig, Echo und GND. Mit ihm werden in diesem Projekt die Durchgänge und Durchgangsrichtungen von Personen in Türen detektiert. Dafür muss zunächst die Distanzmessung mit dem HC-SR04 implementiert werden.

Abb. 2: HC-SR04 Sender und Empfänger mit Objekt

Der HC-SR04 sendet ein Ultraschallsignal aus, welches vom Empfänger nach einer gewissen Zeitdauer wieder empfangen werden kann. Mit diesem zeitlichen Versatz zwischen dem Senden und Empfangen des Signals und der Schallgeschwindigkeit kann mithilfe der Formel

die Entfernung zu einem Objekt hinreichend genau bestimmt werden.

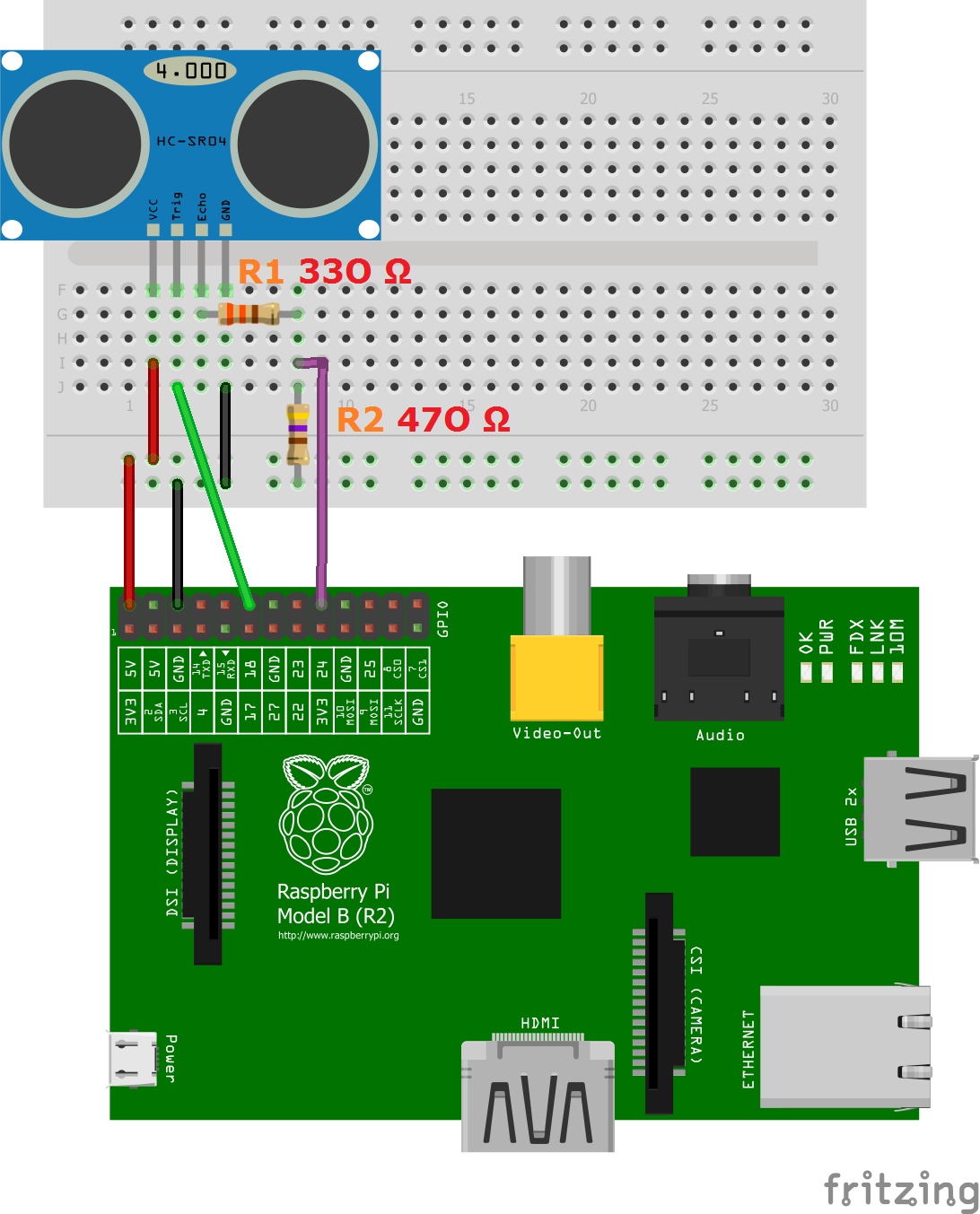
Um zu erhalten muss der Sensor richtig angesteuert werden. Hierzu wird der GND-Pin des Sensors mit GND am Raspberry Pi verbunden, VCC mit 5V, Trig und Echo mit je einem I/O-Pin (hier 18 und 24), siehe Abb. 3.

Abb. 3: Anschlussplan des HC-SR04 [TutRasp]

Der Widerstand R1 (Abb. 3) mitsamt des Widerstands R2 (Abb. 3) dient als Spannungsteiler um die zulässigen 3,3 Volt der GPIO-Pins des Raspberry Pis nicht zu überschreiten, während R2 als Pulldown-Widerstand fungiert [TutRasp]. Die Werte der Widerstände wurden übernommen und mit der Formel für die Ausgangsspannung von Spannungsteilern   
wurde die Spannung berechnet, welche an den GPIO-Pins anliegt, wobei *V1* = 5V, *R1* = 330 und *R2* = 470 ist. *Vout* ist dann 2,94 V, was ausreichend unter den zulässigen 3,3 V liegt. Der Messvorgang wird mit einem 10µs H-Signal an Trig, gefolgt von einer fallenden Flanke gestartet. Nach 250µs wird ein 200µs Burst-Signal ausgesendet. Danach liegt an Echo ein H-Signal an und sobald ein akustisches Echo vom HC-SR04 registriert wird fällt Echo wieder auf einen L-Pegel. Unser ist also gleich der Dauer des H-Signal an Echo. Nach dem „Triggern“ speichern wir also den Zeitpunkt [Microcontroller], wenn Echo auf H steigt und anschließend wieder auf L sinkt, um damit zu berechnen (Code 1).

#include <wiringPi.h>

#include <time.h>

#include <stdlib.h>

#include <stdio.h>

#include <unistd.h>

long getDistance(int trigger, int echo) {

long startM;

long stopM;

double ellapse;

long dist;

//trigger kurz an und aus schalten

digitalWrite(trigger, HIGH);

usleep(100);

digitalWrite(trigger, LOW);

//Startzeiten

startM = stopM = clock();

//Startzeit speichern

while (digitalRead(echo) == 0) {

startM = clock();

}

startBreak = clock();

while (digitalRead(echo) == 1 ) {

stopM = clock();

}

//distanz berechnen

ellapse = (double)(stopM - startM) / CLOCKS\_PER\_SEC;

dist = (ellapse \* 34320) / 2;

return dist;

}

int main() {

printf("Start\n");

//INITIALISIERUNG DER PINS

//------------------------------------

wiringPiSetup();

int trigger1 = 1; //GPIO 18

int echo1 = 5; //GPIO 24

pinMode(trigger1, OUTPUT);

pinMode(echo1, INPUT);

//------------------------------------

while (1){

getDistance(trigger1, echo1);

}

printf("End");

return 0;

}

Code 1: Distanzmessung mit dem HC-SR04 in C

Die Methode clock() liefert die Anzahl der Takte, weshalb (ellapse), nachdem der Start vom Stoppwert abgezogen wurde, noch durch CLOCKS\_PER\_SEC geteilt werden muss um die Zeit in Sekunden zu erhalten. Die hier geschriebene Methode getDistance(int trigger, int echo) liefert die Distanz auf ganze Zentimeter gerundet zurück.

## MQTT Kommunikation

Ein Bild, das Karte enthält.

Automatisch generierte Beschreibung

Abb. 4: Schematischer Aufbau der MQTT Kommunikation

In diesem Projekt müssen alle miteinander verbundenen Clients untereinander kommunizieren und Kommandos bzw. die Ergebnisse der Sensoren (Person ein-/ausgetreten) an den Masterclient übermitteln. Hierzu ist es nicht erforderlich ein eigenes Server-Client System zu implementieren, sondern ein einfacher Austausch von Strings welche anschließend in der eigens entwickelten Software durch Parsing ausgewertet und dann die gewünschten Befehle ausgeführt werden. Ein schnelles und einfaches Mittel um Strings auszutauschen ist MQTT. Nach [MQTT] ist MQTT ist ein leichtgewichtiges Publish/Subscribe Nachrichtenprotokoll zur Verwendung in vielerlei Szenarien. Das Prinzip basiert auf die Idee, dass sich Abonnenten (Subscriber) in Themen, sogenannte Topics, einschreiben. Jeder Teilnehmer kann entweder nur Nachrichten empfangen oder Senden, aber auch beides gleichzeitig. Ebenfalls ist es gestattet Messages ohne ein Abonnement des jeweiligen Topics zu schreiben. Verwaltet wird die Kommunikation mittels Broker, welcher sich um die Anmeldung der Clients, der Einschreibung in Topics und die Zustellung bzw. Verwaltung der Messages kümmert. Die Nachrichten wiederum können in den 3 Quality-Of-Service Stufen 0, 1 und 2 unterteilt werden:

* **0:** Der Broker/Client versendet die Nachricht ohne Bestätigung.
* **1:** Der Broker/Client versendet die Nachricht min. einmal und bestätigt den Erhalt.
* **2:** Der Broker/Client versendet die Nachricht exakt einmal und garantiert den Erhalt durch einen Handshake

Für das SmartAudio Projekt ist es von Relevanz, dass Sensorinformationen genau einmal ausgetauscht werden, da sonst eine präzise Detektion der Anzahl der sich im Raum befindenden Personen nicht mehr gewährleistet ist und das System fehlerhaft arbeitet. Aus diesem Grund werden Nachrichten nur mit der QoS-Stufe 2 ausgetauscht. Der dadurch entstehende höhere Datenverkehr belastet zwar die Netzwerkauslastung etwas mehr, ist aber für den Einsatz im Haushalt irrelevant.

Für die Softwareschnittstelle wurde ursprünglich die Open-Source Library Paho-MQTT benutzt, welche zwar in vielen Programmiersprachen verfügbar und in stetiger Weiterentwicklung ist, wurde aber durch libmosquitto [LIBMOSQUITTO] ersetzt. Der Grund hierfür ist, dass diese Library für den Raspberry Pi direkt als passende ARM Version aus den Repositories installiert werden kann. Für weitere Informationen siehe hierzu auch Kapitel 4.6.3. Für die Masterkommunikation wurde [MQTTPython] und für die Slaveanbindung [MQTTC] verwendet. Um eine einfachere auf das Projekt zugeschnittene Schnittstelle zu verwenden, wurde eine eigene kleine Bibliothek basierend auf libmosquitto für den Client entwickelt, siehe hierzu Abschnitt 2.5.

## SmartAudio MQTT Library

Die eigens entwickelte MQTT Library soll einen einheitlichen, einfachen Zugriff auf die darunter liegende Bibliothek libmosquitto bilden. Da als Programmiersprache C (statt C++) verwendet wird, gibt es nicht die Möglichkeit den Zugriff mittels Klassen zu erstellen und Attribute zu kapseln. Daher verwendet die C Schnittstelle das Präfix ‘SA’ um öffentlich nutzbare Funktionen zu kennzeichnen.

Um die Bibliothek zu benutzen ist es zwingend erforderlich diese mit SAInitialize zu initialisieren. Ebenso verpflichtend ist das Abwarten des Verbindungsaufbaus sowie sämtliche Subscriptions in Topics, da der Broker sonst die Verbindung als fehlerhaft kennzeichnet sobald mehrere Requests gleichzeitig behandelt werden sollen. Dies erreicht man durch den Aufruf der SAWait\* Funktionen welche wiederum eine internalWait Funktion mit dem passenden Flag aufrufen und eine definierte Zeit lang auf die Änderung des Flags warten (warten mit Timeout). Die Bibliothek besteht aus folgenden Methoden:

* void SAInitializeMQTT(const char\* address, const char\* clientid, void (\*messageHandler))

Initialisiert sämtliche Flags, setzt den MessageHandler und verbindet sich mit dem MQTTBroker

* SAInitializeIni(const char\* filename, void (\*messageHandler), void \*config)

Initialisiert, bzw. liest die Konfigurationseinstellungen aus der Ini-Datei. Zu diesen Einstellungen gehören

* + Name des clients, welcher gleichzeitig auch als Topic in das sich eingeschrieben werden soll dient
  + Adresse des Brokers (Hostname oder IP)
  + Streamingtopic („None“ falls der Client nicht fürs streaming zuständig ist)
* SASendMessage(const char\* message, const char\* topicName)

Sofern eine Verbindung zum MQTT Broker besteht, wird eine Nachricht an diesen ins jeweilige Topic gesendet.

* SASubscribeToTopic(const char\* topicName)

Sofern eine Verbindung besteht, schreibt sich der client in das gewünschte Topic ein.

* on\_message(struct mosquitto \*mosq, void \*obj, const struct mosquitto\_message \*message)

Eventhandler, welcher bei eingehenden Nachrichten aufgerufen wird und den internen (in SAInitialize assoziierten) message handler aufruft.

* on\_disconnect(struct mosquitto \*mosq, void \*userData, int rc)

Eventhandler, welcher bei der Schließung der Verbindung aufgerufen wird.

* on\_subscribe(struct mosquitto \*mosq, void \*userData, int messageID, int qosCount, const int \*grantedQoS)

Eventhandler, welcher bei der Einschreibung in ein Topic aufgerufen wird.

* on\_connect(struct mosquitto \*mosq, void \*obj, int result)

Eventhandler, welcher bei erfolgreicher Verbindungsherstellung aufgerufen wird.

* void cleanup()

Schließt offene Verbindungen und beendet den snapclient, falls dieser zuvor gestartet wurde. Diese Funktion wird zum Einen beim Beenden der Software aufgerufen, zum Anderen falls der User (oder ein anderer Prozess) die Software mit dem Signal SIGKILL (oder STRG + C) beendet.

* void CheckSnapclientStatus()

Threadsichere Ermittlung ob der Snapclient gestartet oder beendet werden muss. Diese Funktion muss in der Hauptschleife regelmäßig aufgerufen werden. Standardmäßig wird der Snapclient mit dem Hostnamen „volumiomaster“ aufgerufen.

Achtung: Der Raspberry Pi Zero W benötigt etwas Zeit um den Snapclient zu beenden.

* void on\_disconnect(struct mosquitto \*mosq, void \*userData, int rc)

Wird bei jeglicher Beendigung der Verbindung aufgerufen. Gibt aus, ob der Verbindungsabbruch das Resultat einer gewollten oder fehlerhaften Beendigung war.

* short SAIsSubscribedAndConnected()

Gibt zurück ob der Client verbunden und in die jeweiligen Topics eingeschrieben ist.

# Hardwareaufbau

Nachfolgend wird der Hardwareaufbau und die benötigten Komponenten beschrieben.

|  |  |  |
| --- | --- | --- |
| **Bezeichnung** | **Anzahl** | **Rolle** |
| Raspberry Pi 4 B | 1 | Master |
| Raspberry Pi Zero WH | 3 | Slave |
| HC-SR04 | 6 | Ultraschallsensor |
| Jumperkabel m-f | 24/48 | Verbindung |
| Widerstand 330 Ω | 6 | Widerstand |
| Widerstand 470 Ω | 6 | Widerstand |
| LogiLink Lautsprecher | 3 | Audiowiedergabegerät |

Tab. 1 Hardwareliste

Der Raspberry Pi 4 B wird als Master verwendet und die 3 Raspberry Pi Zero WH als Slaves, an die je 2 Ultraschallsensoren (HC-SR04) angeschlossen werden, weswegen von diesen sechs Stück benötigt werden. Je nachdem wie die Sensoren angeordnet werden, sind 24 (Abb. 6) oder 48 m-f Jumperkabel (Abb. 5) nötig. Die Widerstände sind aus Hardwaregründen erforderlich, siehe hierzu auch 2.3. Die Lautsprecher können durch andere Bluetooth-fähige Lautsprecher ersetzt werden.

## Anschluss der Sensoren

Da zwei Sensoren pro Slave benötigt werden, folgt nun noch eine Erklärung zum Anschlussplan. Neben dem Namen, an dem ein jeweiliger Pin des Sensors angeschlossen wird, wird auch die physikalische Nummer (PHY) angegeben (vergleiche Abb. 1). Die Sensoren können auch getauscht werden. Wichtig ist nur, dass Trigger und Echo eines Sensors einmal an die Pins mit der physikalischen Adresse 12 und 18 und einmal an 11 und 22 angeschlossen werden, da diese im Code jeweils zu einem Sensor gehören. Vcc muss an einen 5V Pin und Ground (GND) an einen GND Pin. Wenn andere Pins für Trigger und Echo gewählt werden, muss der Quellcode entsprechend angepasst werden.

Sensor 1:

* Trigger wird an GPIO 18/PHY 12 angeschlossen (linkes weißes Kabel)
* Echo wird an GPIO 24/PHY 18 angeschlossen (linkes schwarzes Kabel)
* Vcc wird an 5V/PHY 2 angeschlossen (linkes rotes Kabel)
* GND wird an GND/PHY 6 angeschlossen (linkes braunes Kabel)

Sensor 2:

* Trigger wird an GPIO 17/PHY 11 angeschlossen (rechtes weißes Kabel)
* Echo wird an GPIO 25/PHY 22 angeschlossen (rechtes schwarzes Kabel)
* Vcc wird an 5V/PHY 4 angeschlossen (rechtes rotes Kabel)
* GND wird an GND/PHY 20 angeschlossen (rechtes braunes Kabel)

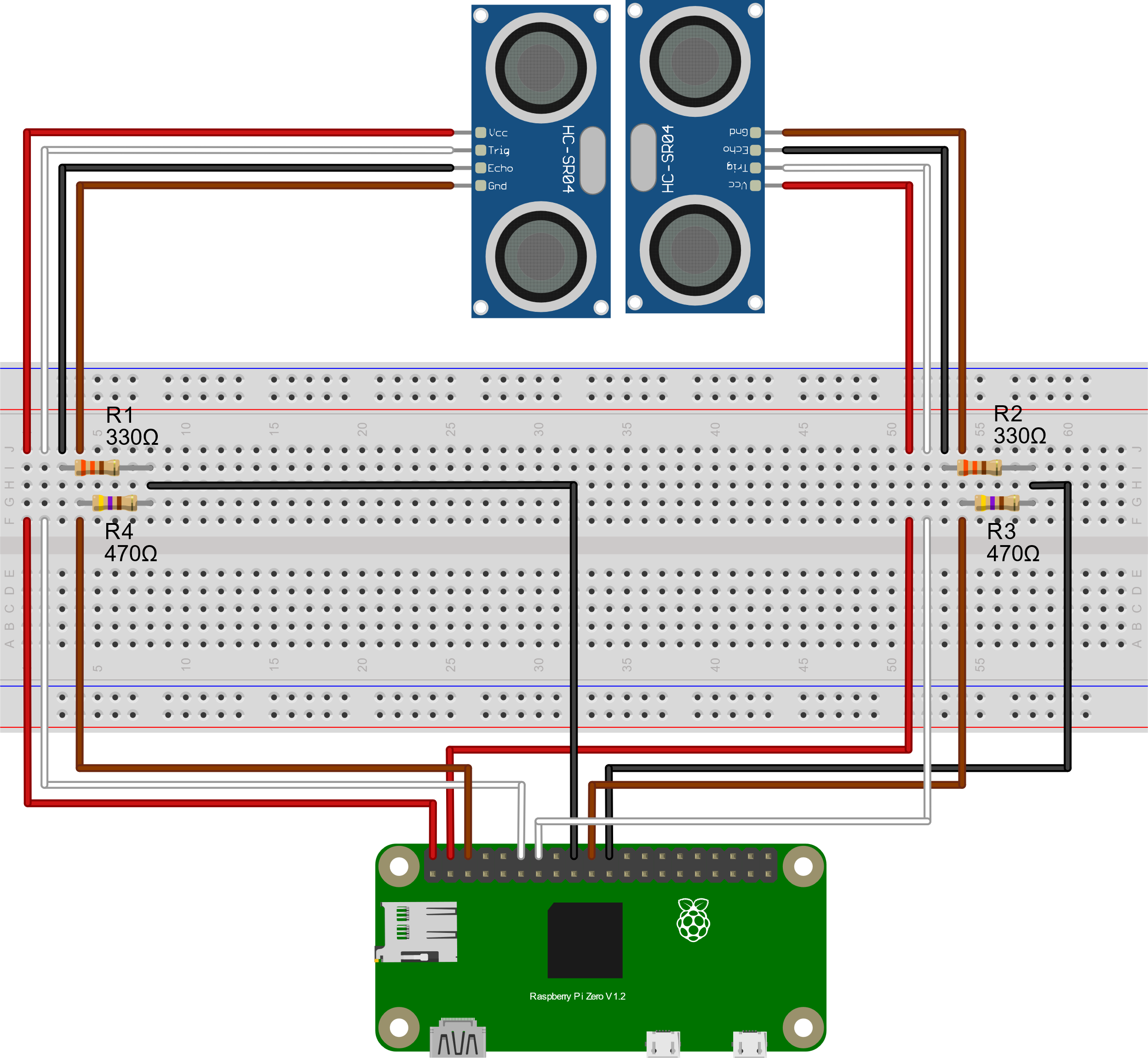


Abb. 5: Anschluss zwei HC-SR04 mit 16 Kabeln

Dieser Anschlussplan hat den Vorteil, dass die Sensoren flexibler angeordnet werden können, um Überschneidungen der Schallkegel zu vermeiden und so die korrekte Detektion zu optimieren (siehe 3.2). Allerdings benötigt man die doppelte Anzahl an Jumperkabeln.

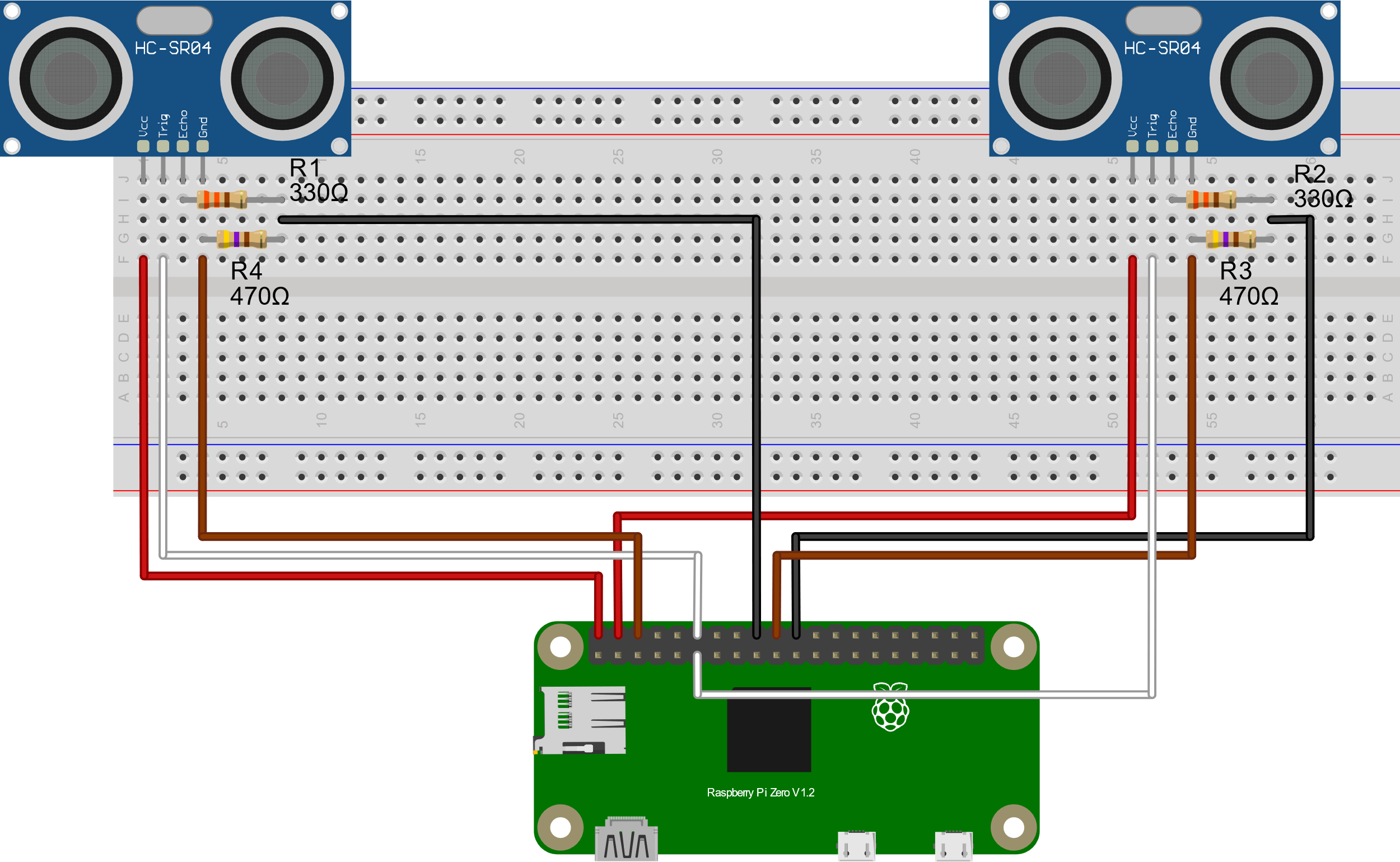


Abb. 6: Anschluss zwei HC-SR04 mit 8 Kabeln

Dieser Anschlussplan hat den Vorteil, dass nur 8 Jumperkabel benötigt werden. Allerdings lassen die Sensoren sich in ihrer Anordnung nicht verändern.

## Anordnung der Sensoren

Da die Sensoren einen Messwinkel von 15° haben, ist eine parallele Anordnung nur mit gewissem Abstand sinnvoll. Zum Beispiel überschneiden sich die Kegel der Sensoren nach ca. 15 cm, wenn sie nur 4 cm voneinander parallel stehen. Das ist relevant, weil die abwechselnden Messungen unserer Sensoren so schnell erfolgen, dass es sich im Verhältnis der langsamen Bewegungsgeschwindigkeit eines durchschreitenden Menschen so verhält, als ob sie gleichzeitig messen würden. Das hat zur Folge, dass es dem Zufall gleicht welche Durchgangsrichtung die beiden Sensoren detektieren, wenn sie zu nah aneinander und parallel platziert werden und somit beide in die gleiche Richtung (ohne Winkelunterschied zwischen beide Sensoren) einen Ultraschallkegel emittieren. Als Lösung ist ein ausreichender Abstand denkbar, aber dann könnten die Sensoren nicht mehr kompakt angebracht werden. Alternativ können die Sensoren so in einem Winkel zueinander angebracht werden, dass sich die Messkegel nicht überschneiden. Dies wäre in diesem Fall ein Winkel <= 165° (Abb. 7). Im praktischen Versuch (Abb. 8) wurden 160° verwendet

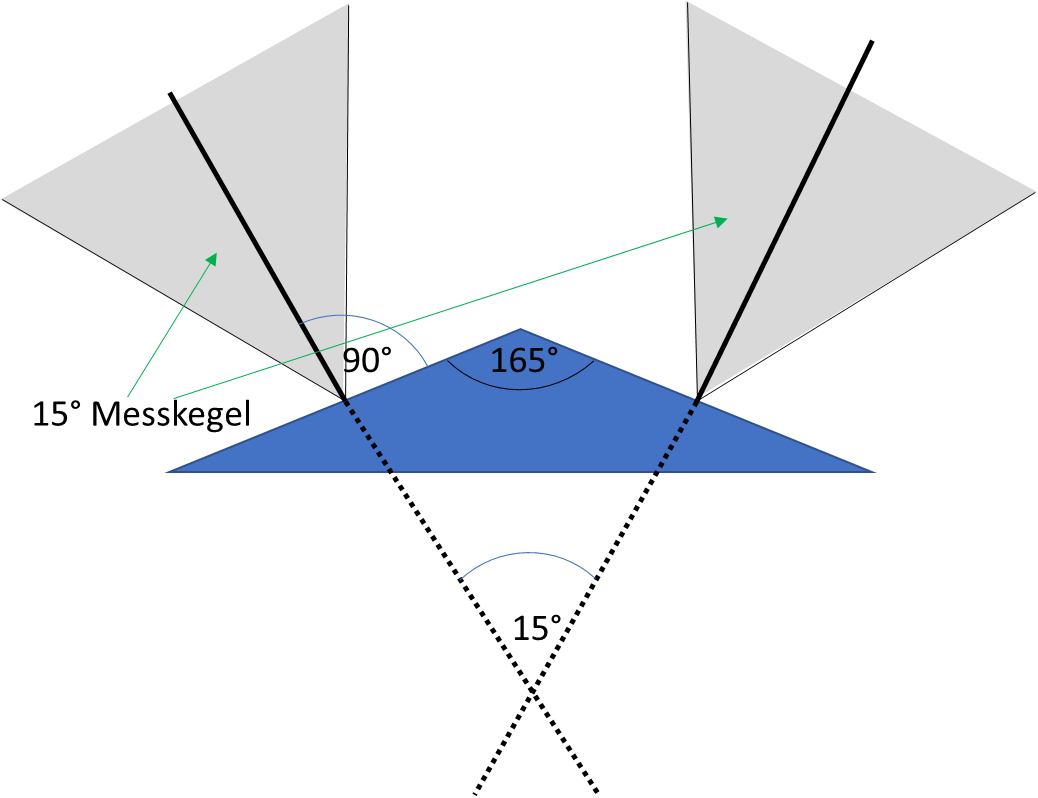


Abb. 7: HC-SR04 mit geringem Abstand montieren

Ein Bild, das drinnen, Tisch, Schreibtisch, Computer enthält.

Automatisch generierte Beschreibung

Abb. 8: Improvisierte Aufhängung mit ca. 160° Winkel

# Software

Hier wird die Installation und Einrichtung der verwendeten Software erläutert. Außerdem wird der Zweck der jeweils verwendeten Komponente erklärt und erläutert, warum diese gewählt wurde.

## Volumio

Das frei verfügbare Volumio [VolumioPi] ist das verwendete Betriebssystem für den Master, welches für das Audiostreaming auf allen verbundenen Clients sorgt. Dieses Betriebssystem wurde gewählt, da dieses eine einfache Bedienung, automatische Erzeugung eines Hotspots falls keine Netzwerkverbindung zur Verfügung steht (bspw. um notfalls ohne Monitor auf die Einstellungen über das Webinterface zugreifen zu können) und eine großes Sortiment an frei verfügbaren Plugins, welche das System um Funktionalitäten erweitert, anbietet. Zudem ist es Open-Source und für jedermann im Detail der Quellcode einsehbar und für den Raspberry Pi 4 verfügbar. Alternativen wie RuneAudio sind leider noch nicht hierfür zur Verfügung da diese softwaretechnisch noch nicht angepasst wurden. Was aber bei den allermeisten gleich ist, ist die Verwendung des MPD-Servers (Music Player Daemon) um Musik wiederzugeben, d. h. Volumio wäre durchaus tauschbar, sofern die Systemanforderungen bzw. die Verfügbarkeit angepasst werden. Zum Installieren des Systems ist es nötig, eine genügend groß dimensionierte SD-Karte (Anforderungen bitte unter [VolumioPi] nachschlagen) zu verwenden und das heruntergeladene Image für den Raspberry Pi bspw. mittels BalenaEtcher (Windows) oder dd (Linux) auf die SD-Karte zu flashen. Danach die SD-Karte in den Raspberry Pi stecken und diesen in Betrieb nehmen. Anschließend muss sich über WLAN mit Volumio verbunden werden. Hierzu sucht man, nachdem das Betriebssystem hochgefahren ist, das dazugehörige WLAN mit der SSID „Volumio“. Das dazugehörige Passwort war zum Einrichtungszeitpunkt „volumio2“. Zum Schluss über einen Webbrowser die URL <http://volumio.local> aufrufen und die Anweisungen auf dem Bildschirm befolgen. Als Hostname bitte „volumiomaster“ verwenden. Die initiale Einrichtung ist abgeschlossen.

## Skripte zur automatischen Installation der benötigten Bibliotheken

Um die Software ordnungsgemäß in Betrieb nehmen zu können, wurden Installationsskripte geschrieben welche die benötigten Libraries automatisch herunterladen und installieren. Ein Installationsschritt in diesem Skript sieht wie folgt aus:

echo

echo "Mosquitto:"

if [ $(dpkg-query -W -f='${Status}' mosquitto 2>/dev/null | grep -c "ok installed") -eq 0 ];

then

echo "Installing Mosquitto"

sudo -s apt-get install -y Mosquitto

else

echo "Mosquitto already installed"

fi

Code 2: Ausschnitt aus dem Installationsskript

Es existieren zwei Shellskripte, nämlich „InstallRequirementsMaster.sh“ (für den Master, siehe auch ) und „InstallRequirementsSlave.sh“ (für den Slave/Client). Beide installieren nach dem gleichen Schema mehrere Softwarekomponenten aus dem Raspbian Repository. Diese Installationen müssen einmalig durchgeführt werden. Lediglich der Snapserver für Volumio muss gesondert installiert werden.

## Installation von Snapcast für Volumio

Snapcast ist keine eigenständige Software, sondern interagiert mit dem in Volumio, welches ein einfaches Plugin-Installationssystem bietet, integrierten MPD-Player. Ebenso erlaubt es die Verteilung eines von Volumio verteilten Streams auf mehrere verbundene Clients zu verteilen (Multicast). Snapcast bietet eine echtzeitfähige Audioübertragung, d.h. dass der Audiostream nahezu Latenzfrei übertragen wird, weswegen herkömmliche Audioübertragungsverfahren (welche erst Audio puffern und einfach diesen Puffer abspielen) ausscheiden. Dieses hingegen synchronisiert die Zeit mit dem Server bzw. fragt diese ab und spielt die entsprechenden erhaltenen mit Zeitstempel versehenen Audiosamples ab. Abweichungen werden durch Entfernung bzw. Duplizierung von Samples korrigiert um synchron zu bleiben. Da das Pluginrepository für Volumio nicht immer auf dem aktuellsten Stand ist, ist es ratsam die benötigten Dateien aus dem Git-Repository herunterzuladen und diese zu installieren. Für die aktuellsten Installationsanweisungen und weitere detailliertere Informationen über Snapcast bitte [Snapcast] konsultieren. Zunächst müssen die Dateien heruntergeladen werden. Dazu öffnet man ein Terminal im /home/-Verzeichnis und führt folgende Schritte durch:

Das Git-Repository zunächst klonen:

git clone https://github.com/Saiyato/volumio-snapcast-plugin.git

Ins Verzeichnis wechseln:

cd volumio-snapcast-plugin

Die Datei volumio-snapcast-plugin.zip entfernen:

rm -f volumio-snapcast-plugin.zip

Plugininstallation starten und alles bestätigen

volumio plugin install

## Client

### Durchgangsdetektion und Serverkommunikation

Um einen Durchgang zu detektieren und die Durchgangsrichtung zu unterscheiden werden zwei Ultraschallsensoren und ein Raspberry Pi Zero W verwendet. Auf dem Raspberry läuft die DetectionOfHumenFlowDirection.c, die für die Datenauswertung der beiden Sensoren und die Nachrichtenübermittlung an ein MQTT-Topic zuständig ist.

Zuerst werden die Pins wie in Grundlagen mit WiringPi, danach eine Verbindung zum MQTT-Broker initialisiert. Hierzu wird mit ini\_parse(…) die IP-Adresse des Brokers sowie ein Topic aus einer config.ini Datei gelesen und in config gespeichert. Anschließend wird mit SAInitialize(…) eine Verbindung zum Broker hergestellt (Code 3).

typedef struct

{

const char\* IP; //IP-Adresse des MQTT-Broker

const char\* ID; //Sensor ID

} configuration;

int main() {

printf("Start\n");

//INITIALISIERUNG DER PINS

//------------------------------------

wiringPiSetup();

int trigger1 = 1; //GPIO 18

int trigger2 = 0; //GPIO 17

int echo1 = 5; //GPIO 24

int echo2 = 6; //GPIO 25

pinMode(trigger1, OUTPUT);

pinMode(trigger2, OUTPUT);

pinMode(echo1, INPUT);

pinMode(echo2, INPUT);

//------------------------------------

//INITIALISIERUNG DER SERVERVERBINDUNG

//------------------------------------

//Topic for Sensor + and -

configuration config;

if (ini\_parse("config.ini", iniHandler, &config) < 0) {

printf("Can't load 'config.ini'\n");

return 1;

}

const char\* topic = config.ID;

const char\* serverName = "PLACEHOLDER";

//Verbindung mit SAInitialize initialisieren

printf("initializing\n");

SAInitialize(config.ID, serverName, 0);

printf("done\n");

//Auf Verbindung warten

SAWaitForConnection();

printf("connection established\n");

//------------------------------------

Code 3: Initialisierung

Durch minDistance (Code 4) wird eine Mindestdistanz beschrieben innerhalb welcher ein Durchgang stattfinden muss um vom Programm als valide angesehen zu werden. Nun werden die beiden Sensoren abwechselnd mit getDistance (Code 4) abgefragt. Sollte einer der beiden Sensoren einen Wert kleiner als minDistance zurückliefern, so wird der entsprechend andere Sensor 5-mal nacheinander abgefragt. Sollte hier auch jemand näher als minDistance sein, hat ein Durchgang stattgefunden und es wird eine Nachricht mit „in“ oder „out“ an das Topic gesendet (Abb. 9).

Ein Bild, das Tastatur, Computer enthält.

Automatisch generierte Beschreibung

Abb. 9 Funktionsweise Sensorauswertung

Damit nach einem erfolgreichen Durchgang nur eine Nachricht gesendet wird, wird die Hilfsvariable trigger verwendet, mit der wir nach Durchgang so lange in einer while-Schleife bleiben, bis keiner der beiden Sensoren jemanden näher als minDistance erkennt.

int minDistance = 50; //ab wann wird Person detektiert in cm

int trigger = 0; //wurde ein Durchgang detektiert?

while (1){

/\*Wenn sich noch jemand im Sensorbereich aufhält, nachdem ein

Durchgang detektiert wurde, so warten wir, bis er sich aus dem

Sensorbereich entfernt hat\*/

if(trigger == 1){

while(getDistance(trigger1, echo1) <= minDistance

|| getDistance(trigger2, echo2) <= minDistance){;}

trigger = 0;

}

/\*abwechselnde distanzmessung auf Sensor A und B\*/

long distance\_a = getDistance(trigger1, echo1);

long distance\_b = getDistance(trigger2, echo2);

/\*wenn Sensor a eine Person detektiert hat, so wird Sensor b

5 mal abgefragt.

Wenn mindestens 1 von den 5 Messungen kleiner als minDistance ist,

so wird eine Nachricht mit "+" in unser MQTT-Topic gesendet\*/

if (distance\_a < minDistance) {

long init = minDistance + 1;

long liste[5] = { init, init, init, init, init };

for (unsigned int i = 0; i < (sizeof(liste) / sizeof(\*liste)); i++) {

long addDistance = getDistance(trigger2, echo2);

liste[i] = addDistance; i++;

}

for (unsigned int i = 0; i < (sizeof(liste) / sizeof(\*liste)); i++) {

if (liste[i] < minDistance) {

trigger = 1;

SASendMessage("+", topic);

break;

}

}

}

/\*Sensor B analog zu

\* -Sensor a, nur wird "-" gesendet\*/

else if (distance\_b < minDistance) {

long init = minDistance + 1;

long liste[5] = { init, init, init, init, init };

for (unsigned int i = 0; i < (sizeof(liste) / sizeof(\*liste)); i++) {

long addDistance = getDistance(trigger1, echo1);

liste[i] = addDistance; i++;

}

for (unsigned int i = 0; i < (sizeof(liste) / sizeof(\*liste)); i++) {

if (liste[i] < minDistance) {

trigger = 1;

SASendMessage("-", topic);

break;

}

}

}

}

printf("End");

return 0;

}

Code 4: Durchgangsdetektion

### Fehler und -behebung

Nach ersten Tests des Gesamtsystems fielen größere Fehler auf, die vermutlich Hardwarebedingt auftraten. Die genauen Ursachen konnten nicht identifiziert werden, allerdings wurden die Symptome durch Codeanpassungen behoben. Unabhängig davon, welcher der beiden HC-SR04 für eine Distanzmessung verwendet wurde, blieb das Programm in einer Dauerschleife hängen. Außerdem erfolgten Detektionen von Durchgängen, obwohl keine Durchgänge erfolgten.

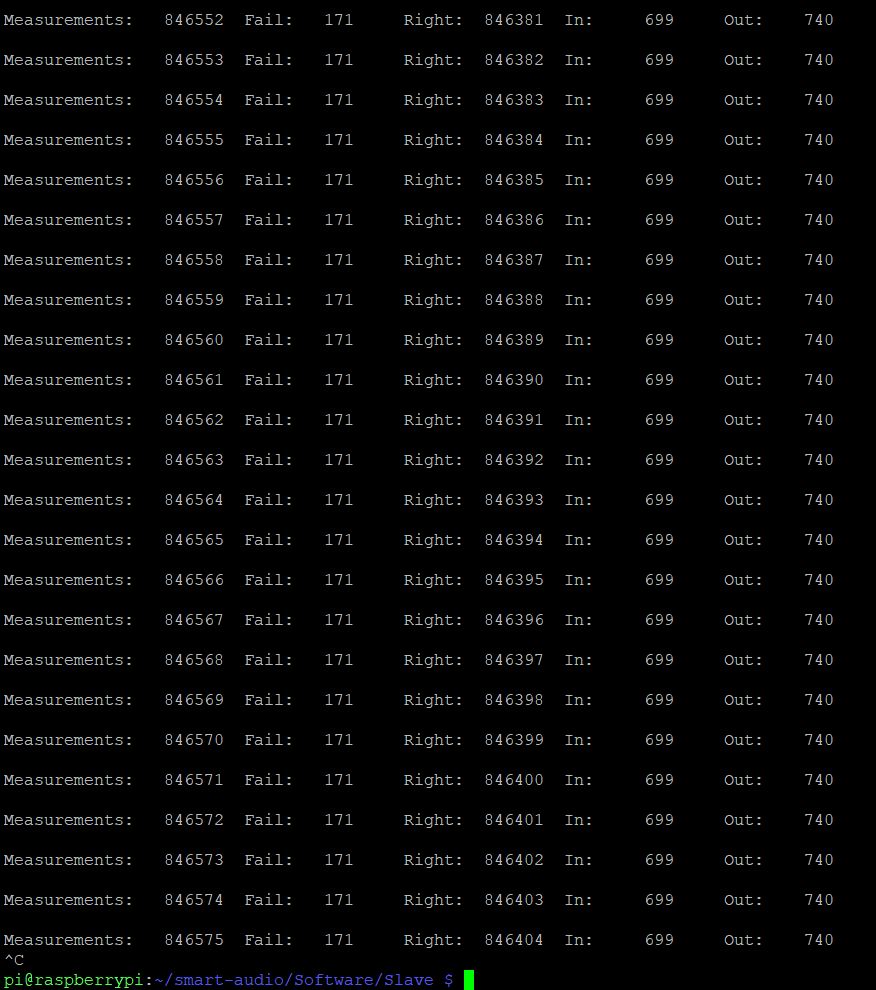


Abb. 10: In- / Out-Fehler und korrekte Messungen

Abb. 10 zeigt die gesamten Messungen (Measurements), die Dauerschleifenfehler (Fail) sowie die korrekten Messungen (Right). Zusätzlich wurden In- und Out-Detektionen gezählt, jedoch nicht als Fehlmessungen gewertet. Die Dauerschleifenfehler wurden behoben, indem nach einer Wartezeit von über 2 ms die Schleife abbricht und die Messfunktion einen Wert von 1000 cm zurückliefert, was weit über dem möglichen Messbereich der HC-SR04 liegt. Außerdem wurden Werte unter 3 cm als Rückgabewerte ausgeschlossen. In einem solchen Fall wird ebenfalls 1000 cm zurückgegeben. Da im Falle einer falschen In- oder Out-Detektionen beide Sensoren versagen müssen, wurde eine weitere Distanzmessung des jeweiligen Sensors zum Bestätigen des Messwerts eingebaut. Anschließend wurde ein ca. zehnstündiger Test gefahren.

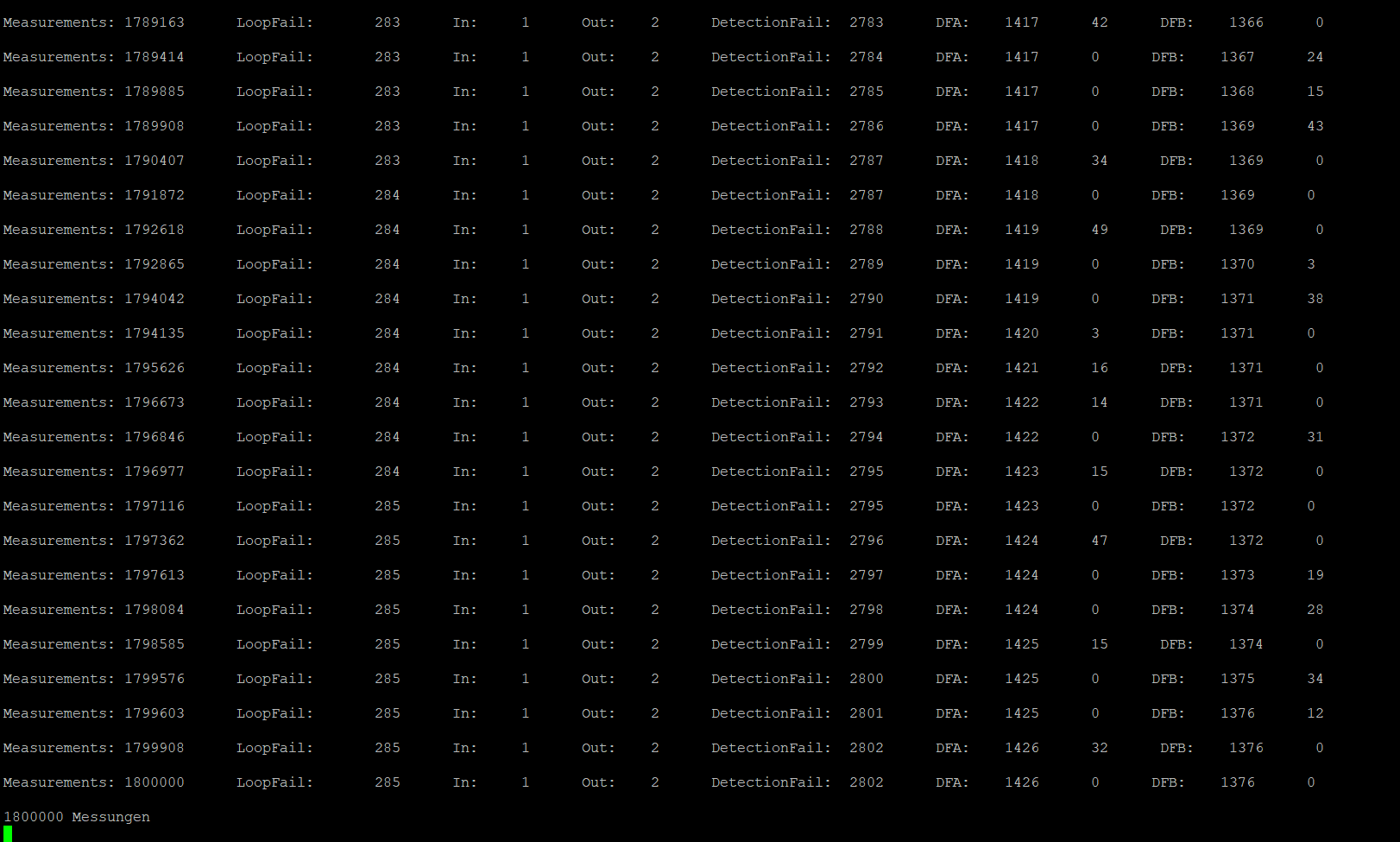


Abb. 11: Testergebnis erster zehn Stunden Test

In Abb. 11 sehen Sie das Testergebnis. Nach 18.000.000 Distanzmessungen wurden ein Fehlerhafter Durchgang in die eine (In) und zwei in die andere Richtung (Out) detektiert. Dieses Ergebnis stellt zwar eine deutliche Verbesserung dar, allerdings ist es, gemessen an der gewünschten dauerhaften Laufzeit des Systems, nicht akzeptabel. Durch eine weitere Sicherheitsmessung konnte das Ergebnis auf 0 fehlerhafte Messungen auf gleicher Laufzeit verbessert werden. Siehe In und Out in Abb. 12.

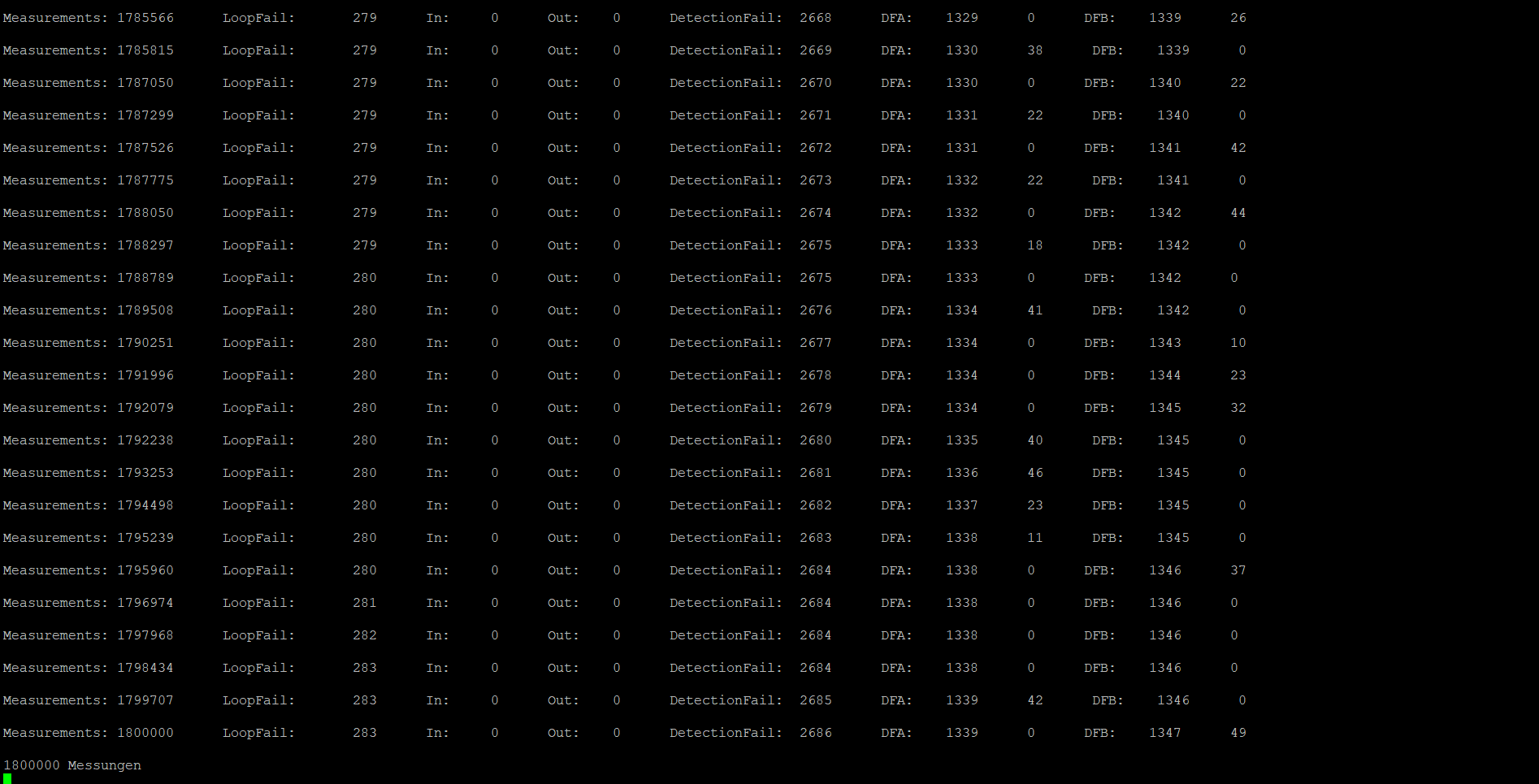


Abb. 12: Testergebnis zweiter zehn Stunden Test

Den Code für die abgeänderte getDistance-Methode finden Sie im Anhang A2 und den Code für die abgeänderte Durchgangsdetektion für eine der beiden Durchgangsrichtungen finden Sie im Anhang A 3.

### Buildskript zur Kompilierung der Client-Software

Zur Slavesoftware wird ein Buildskript zur Kompilierung der Client-Software beigelegt, welches alle benötigten Libraries, nämlich WiringPi, libmosquitto und den Ini-Parser, einbindet:

#!/bin/bash

rm -f ./\*.o

gcc -c ini.c -o ini.o

gcc -c communication.c -o comm.o

gcc -D\_GNU\_SOURCE -c DetectionOfHumanFlowDirection.c -o dohfd.o

gcc -o smartaudioclient ini.o dohfd.o comm.o -lwiringPi -lmosquitto -lpthread

Code 5: build.sh

Im Wesentlichen wird aufgefordert bash als Standard Shell zu verwenden und alle vorhandenen Objectfiles mittels „rm -f ./\*.o“ zu löschen. Anschließend wird die Software mit gcc -c kompiliert. Posix kompatible Sprachfeatures werden über den switch „D\_GNU\_SOURCE“ eingebunden, d. h. spezielle Funktionen die nicht zum Standard C Sprachrepertoire gehören werden hierdurch ermöglicht [GNUPOSIX]. Zum Schluss werden die Objectfiles mit WiringPi und Mosquitto (MQTT) durch gcc -o gelinkt.

### Einrichten des Bluetooth Audio Outputs

Auf dem Debian Buster Betriebssystem (mit GUI) vom 05.02.2020 ist es recht leicht ein Bluetooth Gerät als Audio Output einzurichten, da im Gegensatz zu Volumio Debian etwas aktueller ist und somit auch diverse Fehlerbehebungen (bspw. verwenden von Bluetooth als Audioausgabe) beinhaltet. Oben in den Bluetooth Einstellungen muss das Gerät gesucht und verbunden werden. Das Pairing und Trusten wird automatisch ausgeführt. Die im Projekt verwendeten Lautsprecher wurden auch automatisch als Audiowiedergabegerät erkannt. Wenn man im Audiomenü nun den Lautsprecher auswählt, so wird dieser als Standardwiedergabegerät eingestellt. In älteren Versionen funktioniert das noch nicht automatisch und auch nicht über die GUI des Betriebssystems. Das Problem lässt sich beheben in dem man in der auf der SD-Karte befindlichen Datei /boot/config.txt die Zeile „dtparam=audio=on“ auskommentiert, damit die Standardwiedergabegeräte (HDMI und/oder Klinke) deaktiviert werden und so das Bluetooth Gerät als letzte Wahl übrigbleibt und gewählt wird.

### Hostname, SSID, WLAN beim Booten, VNC und SSH

**Hostname**

Auf den Clients wurde ein Skript eingerichtet um den Hostnamen über die SD-Karte ändern zu können. Um dieses Skript aufzurufen ist die Anpassung der Datei „autostart“ unter /etc/xdg/lxsession/LXDE-pi/autostart nötig. Nachfolgende Zeile ist einzufügen:

@sudo -s sh /home/pi/hostname.sh

Diese Zeile ruft das Skript hostname.sh im Homeverzeichnis des Users „pi“ mit erhöhten Rechten auf. Das Shellscript selbst prüft ob eine Datei namens „newhost“ im Bootverzeichnis (/boot) der SD-Karte liegt, falls ja, dann auf Leerheit prüfen und ignorieren falls die Datei keinen Inhalt enthält, ansonsten die Änderungen mit „hostnamectl set-hostname $NEWHOST“ durchführen und den Client neu starten. Zum Schluss wird die Datei aus /boot entfernt.

if [ -f /boot/newhost ]

then

#check if file is empty

if [ -s /boot/newhost ]

then

OLDHOST=`cat /etc/hostname`

NEWHOST=`cat /boot/newhost`

echo "Changing $OLDHOST to $NEWHOST..."

echo $NEWHOST > /etc/hostname

hostnamectl set-hostname $NEWHOST

rm /boot/newhost

shutdown -r now

else

rm /boot/newhost

fi

fi

Code 6: hostname.sh

**SSID festlegen**

Da der Raspberry Pi Zero nur Mini-USB Anschlüsse anbietet, gestaltet sich die Ansteuerung per Tastatur schwierig. Deshalb ist es einfacher eine WLAN-Verbindung herzustellen und auf den Raspberry Pi Zero mittels VNC und SSH zuzugreifen.

Für die WLAN Konfiguration bietet Raspbian bereits von Haus aus die Möglichkeit benötigte WLAN Daten, also SSID und Passwort, im /boot Verzeichnis zu hinterlegen. Die dafür anzulegende Datei lautet „wpa-supplicant.conf“:

country=DE

ctrl\_interface=DIR=/var/run/wpa\_supplicant GROUP=netdev

update\_config=1

network={

ssid="SSID des Routers"

scan\_ssid=1

psk="WPA2 Key"

key\_mgmt=WPA-PSK

}

Code 7: Automatische Konfiguration des WLAN

**WLAN beim Boot ermöglichen**

Damit die Software automatisch starten und sich mit dem MQTT Broker verbinden kann ist es zwingend erforderlich dass eine Netzwerkverbindung beim Booten des Raspberry Pi Zero W besteht. Hierfür muss im Tool „raspi-config“ eine entsprechende Konfiguration vorgenommen werden. Bitte beachten das je nach Distribution raspi-config erst nachinstalliert werden muss. In diesem Fall wurde die Konfiguration auf einem Raspbian Buster System vorgenommen. Sollte man mit der GUI arbeiten, dann muss zunächst ein Terminal geöffnet werden, ansonsten einfach die nachfolgenden Schritte in der Kommandozeile durchführen. Als erstes wird die Konfigurationssoftware über „raspi-config“ aufgerufen. Ggf. sind erhöhte Rechte erforderlich welche mittels sudo erhalten werden können. Danach ist der Menüpunkt 4 „Wait for Network at boot“ auszuwählen. Im Anschlussdialog selektiert man den zweiten Punkt „Slow wait for network connection before completing boot“. Abschließend speichert und beendet man die Konfiguration mit „Finish“ und startet den Rechner neu. Jetzt wird die Netzwerkverbindung beim boot erzwungen, sofern verfügbar.

**SSH**

Die SSH Konfiguration gestaltet sich sehr einfach. Hierzu muss einfach im /boot Verzeichnis eine Datei namens „ssh“ angelegt werden und der Zugriff per SSH ist gestattet. Im Standardfall kann nun eine Verbindung mittels

ssh pi@raspberrypi

hergestellt werden. Je nach vorher gesetztem Hostnamen muss „raspberrypi“ durch den neu gewählten Namen ersetzt werden.

**VNC**

Damit einfach auf die grafische Oberfläche zugegriffen werden kann, gestaltet sich die Möglichkeit per VNC als sehr komfortabel. Es existieren mehrere VNC-Server Hersteller, welche ihre Software für den Raspberry Pi vertreiben wie z. B. TightVNC. Der VNC-Server RealVNC ist allerdings bereits in Raspbian enthalten [VNC] und wird deshalb benutzt. Zunächst muss aber SSH aktiviert worden sein um VNC freizuschalten. Dazu stellt man wie oben beschrieben eine SSH Verbindung her und startet die Raspberry Pi Konfigurationsoberfläche über „sudo raspi-config“:

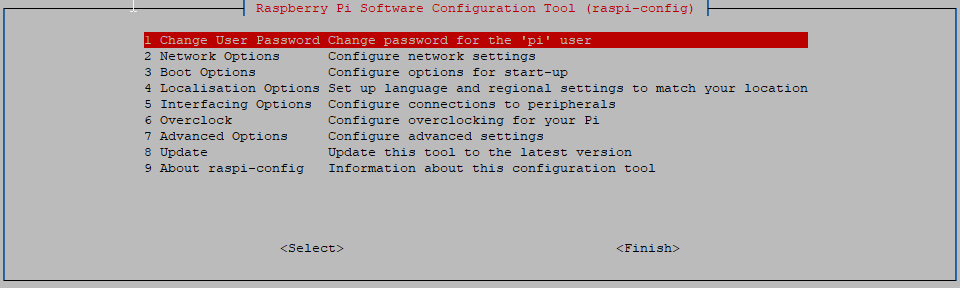


Abb. 13: Raspi-Config

Nun wählt man die Option „Interfacing Options“ und dann „VNC“. Bei der anschließenden Nachfrage ob dies aktiviert werden soll, diese mit „Yes“ und „Ok“ bestätigen. Jetzt kann mit einem VNC Client wie zum Beispiel VNC Viewer darauf zugegriffen werden.

### Autostart der Software

Damit die Software nicht immer von Hand neu gestartet werden muss, werden Startskripte eingerichtet, welche die Konfiguration und den Start der Kommunikation initiieren. Zunächst muss der MQTT-Broker Mosquitto (auf dem Master) automatisch gestartet werden. Hierfür legt man sich eine Datei mosquitto.service mit folgendem Inhalt unter /etc/systemd/system an:

[Unit]

Description=Mosquitto MQTT Broker

After=xdk-daemon.service

After=network.target

[Service]

ExecStart=/usr/sbin/mosquitto

ExecReload=/bin/kill -HUP $MAINPID

restart=always

[Install]

WantedBy=multi-user.target

Code 8: Inhalt mosquitto.service

Hierbei handelt es sich um eine systemd Servicedatei, welche für den autostart verantwortlich ist. Unter „Unit“ wird ein Name für den Service unter „Description“ vergeben. After beschreibt nach welchen Services der zu startende Service aufgerufen werden soll. Das Ziel wird anschließend unter „Service“ beschrieben. ExecStart benötigt den Pfad zur Software. ExecReload sorgt für den Neustart des Services mittels einem Signal „Hangup“. Install sorgt für den Autostart bei einem bestimmten Runlevel, hier das „multi-user.target“. Je nach Mosquittoversion kann es zu Problemen beim Autostart kommen. Um dies zu beheben deaktiviert man das Logfile oder erteilt dem Logfile entsprechende Schreibrechte. Um das Logfile zu deaktivieren muss die Datei /etc/mosquitto/mosquitto.conf an der Stelle logfile abgeändert werden. Bitte hierfür die Dokumentation zu Mosquitto konsultieren da sich die Konfigurationsdatei mit den Versionen entsprechend ändern kann.

Damit die Smartaudiosoftware automatisch startet, legt man eine Datei namens startsmartaudio.sh im /home/pi/ Verzeichnis mit dem folgenden Inhalt an:

#!/bin/bash

sleep 5

lxterminal –working-directory=“/home/pi/smart-audio/Slave/“ –command=“./smartaudioclient“

Code 9: Inhalt startsmartaudio.sh

Nun muss noch der Autostart erfolgen durch modifizieren von /etc/xdg/lxsessions/LXDE-Pi/Autostart. Folgende Zeile muss eingefügt werden:

sh /home/pi/startsmartaudio.sh

## Master

### MQTT Auswertesoftware

Die eigentliche Steuerung der Clients erfolgt vom Master aus. Hierfür wurde ein Pythonskript „mqttserver.py“ entworfen um alle eingehenden Sensordaten auszuwerten und mittels kleiner Datenbank (realisiert als Dictionaries) die aktuelle Anzahl der Personen pro Raum festhält.

client = mqtt.Client()

client.on\_connect = on\_connect

client.on\_message = on\_message

client.connect(MQTT\_SERVER, 1883, 60)

client.loop\_forever()

Code 10: Mainloop des Masters (Ausschnitt)

Die Mainloop des Masters besteht lediglich aus dem Anlegen des Clients, der Assoziierung nötiger Events und dem Starten der MQTT-Loop um die Kommunikation mit dem (auf dem Master befindlichen) Server zu starten. Die Adresse des MQTT-Servers wird über eine Inidatei, geparst mit der Library Configparser, ermittelt, welche im Unterkapitel zum Parsen der Inidatei erklärt wird. Bei jeder eingehenden Nachricht wird ein Event „on\_message“ des Skriptes ausgelöst und die Nachricht verarbeitet (Eine Veranschaulichung bietet Abb. 14).

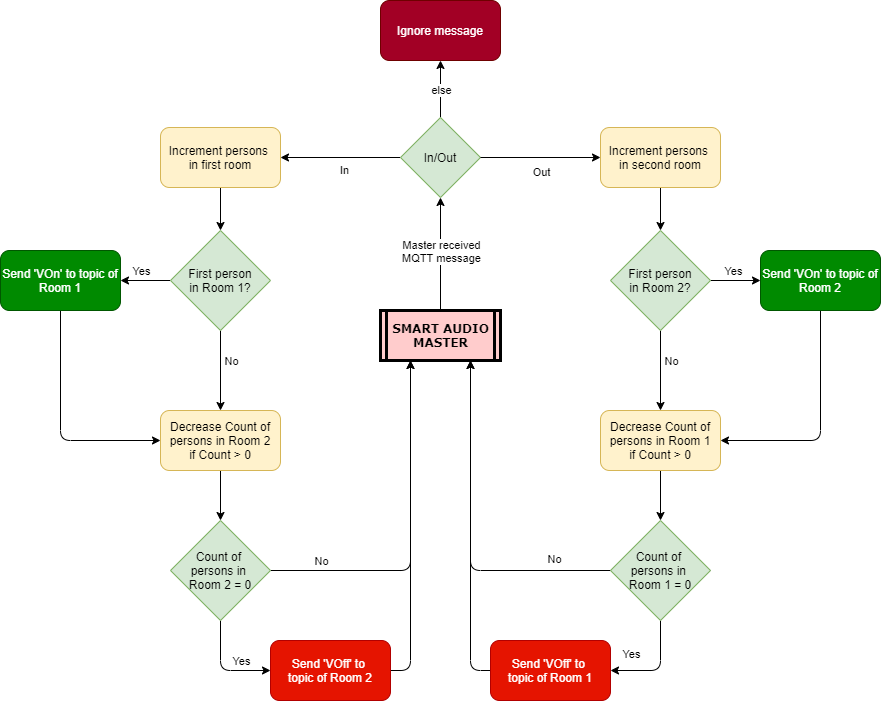


Abb. 14: Veranschaulichung einer eingehenden MQTT Nachricht

Unterschieden wird zwischen „in“ und „out“. Prinzipiell werden die Durchgänge im Skript als Dictionaries festgehalten, welche je nach eingehender Nachricht die Ziffer, also die Anzahl der Personen im Raum, um 1 inkrementiert oder dekrementiert. Im on\_message-Event wird bei jeder eingehenden Nachricht geprüft ob jemand ein- oder ausgetreten ist und die passenden Variablen modifiziert. Essentiell ist hierbei „PersonsInRoom“, „SensorsToRoom“ und „RoomToStream“. Bei Empfang einer „in“-Nachricht von einem Sensor wird nach den entsprechenden Räumen gesucht. Im ersten Raum wird die Personenanzahl inkrementiert und im zweiten Raum dekrementiert falls sie nicht schon 0 ist. Bei einer „out“-Nachricht entsprechend umgedreht. Wenn in einem Raum mehr als 0 Personen sind, dann wird das zugehörige Topic für den Volume-Befehl herausgesucht und dorthin „VOn“ gesendet. Falls 0 Personen in einem Raum gezählt werden, dann wird „VOff“ gesendet.

### Auswertung der Durchgangsdaten mit Reaktion

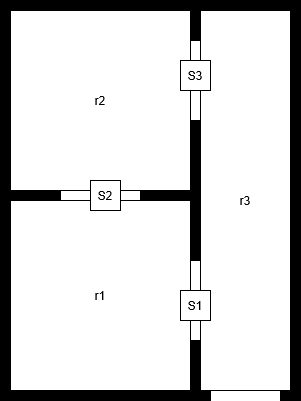


Abb. 15: Raumplan im Test

Die Durchgangsdaten werden momentan mit „in“ bzw. „+“ und „out“ bzw. „-„ unterschieden, was aber lediglich als Fallunterscheidung zwischen den beiden verschiedenen Durchgangrichtungen dient. Andere Bezeichnungen sind auch möglich. Da ein Sensorpaar im Türrahmen angebracht werden soll muss nicht nur die Durchgangsrichtung unterschieden werden, sondern den Sensoren müssen die beiden anliegenden Räume zugeordnet werden. Für die Auswertung dieser Daten ist die mqttserver.py auf dem Master verantwortlich. Sie erhält aus den Topics der einzelnen Sensoren die Durchgangsdaten und aus einer config.ini (Code 11) die Zuordnung zu den Räumen. Hier liegen dann auch die Topics in die ein Volumebefehl gesendet werden muss, falls sich Personen in einem Raum befinden.

[SensorToRoom];welche räume hängen am Sensor

smartaudio/s1=r1,r3

smartaudio/s2=r1,r2

smartaudio/s3=r2,r3

[RoomToStream];Topic für laut/leise kommando

r1=smartaudio/s1

r2=smartaudio/s2

r3=smartaudio/s3

Code 11: config.ini auf dem Master zur Raumdefinition

Die Konvention für die Sektion [SensorToRoom] lautet:

[*Name des MQTT-Topics des Sensors*]=r[*Raumnummer*],r[*Raumnummer*]

Die Konvention für die Sektion [RoomToStream] lautet:

r[*Raumnummer*]=[*Name des MQTT-Topics des Streamenden Pis*]

Zu beachten ist, wie bereits oben beschrieben, dass der erste Raum (hier bspw. r1) bei einer in-Nachricht inkrementiert und der zweite (hier bspw. r3) dekrementiert wird. Das config.ini Beispiel ist für den getesteten Raumplan (Abb. 15) ausgelegt. Bei anderer Raum- und Sensorverteilung sieht die Datei entsprechend anders aus.

## Verworfene Konzepte und Problematiken

Dieses Unterkapitel beschreibt in Betracht gezogene Konzepte, welche aber aufgrund von Kompatibilitätsgründen oder Komplexitätsgründen verworfen worden sind.

### Verwendung von Volumio auf allen Geräten

Ursprünglich sollte sowohl auf dem Master als auch auf den Slaves Volumio benutzt werden. Es hat sich herausgestellt, dass, zumindest im Moment, die Nutzung eines Volumiomasters und ein von Volumio unabhängiges Gerät als praktikabler herausstellt, da Volumio noch auf ‚Debian Jessie‘ basiert und noch nicht auf die aktuellere Version ‚Debian Buster‘. Die Verwendung von Volumio auf den Slaves gestaltet sich als problematisch da der Bluetoothtreiber im Gegensatz zur aktuellsten Raspbian Version mehr Abhängigkeiten und Einrichtungsarbeit erfordert. Deshalb wurde es erforderlich die Steuerbefehle bei den Clients etwas abzuändern und die Audiosteuerung mittels Volumiobefehlen durch Prozesssteuerung zu ersetzen. Natürlich können auch die im nächsten Unterkapitel erläuterten Volumiobefehle verwendet werden, sobald die benötigten Treiber aktualisiert wurden und Volumio wiederverwendet wird. Ansonsten muss der jeweilige Client lediglich WiringPi und eine MQTT Anbindung mitsamt Snapclientinstallation anbieten, damit dieser den Stream des Masters empfängt und auf die Personendetektion reagieren kann.

### Steuerung von Volumio

Volumio bietet vielfältige Möglichkeiten die auf dem Client laufende Software über Schnittstellen zu steuern. Angeboten wird die REST API, WebSockets API und ein Command Line Client ([VolumioAPI]). Der Einfachheit halber wird die Steuerung des Command Line Client gewählt, da die Kommandos direkt von den Streamingclients ausgeführt werden müssen. Weiterhin muss keine weitere Kommunikationsimplementierung über curl oder SocketIO stattfinden, was die Komplexität und die Fehleranfälligkeit erheblich reduziert.

Der Aufbau eines Kommandozeilenbefehls gestaltet sich wie folgt (siehe [VolumioAPI], Abschnitt API, Command Line Client):

volumio <argument1> <argument2>

Code 12: Grundlegender Aufbau eines Volumio Kommandozeilenbefehls

"Argument1” erwartet den zu modifizierenden Steuerbefehl und "Argument2” den dazugehörigen Parameter. Für die Lautstärkesteuerung ist das Argument “Volume” zuständig. Der dazugehörige Parameter ist “mute”, für die Stummschaltung bzw. “unmute" für Reaktivierung der Lautstärke. Vorteilhafter ist aber die Pausierung und Wiederaufnahme des erhaltenen Audiostreams. Die Abschaltung und Anschaltung des Audiostreams wirkt sich positiv auf den Datenverkehr innerhalb des Netzwerkes aus, da nur Daten an die Clients gestreamt werden, sofern dies erforderlich ist und der Stream nicht dauerhaft aktiv, wenn auch im stummen Zustand, sein muss. Die dafür erforderlichen Kommandos lauten

“volumio play” (Stream abspielen) und “volumio stop” (Stream stoppen). Aufgerufen werden diese Befehle innerhalb der clientsoftware über Systemaufrufe:

system(“volumio play”); bzw. system(“volumio stop”);

### Verwurf von Paho-MQTT als MQTT-Library

Ursprünglich sollte die recht verbreitete Bibliothek „Paho-MQTT“ genutzt werden, jedoch ist für die C-Bibliothek eine Kompilierung für jede neue Version nötig, da keine ARM-Bibliothek als Release auf den Repositories von Raspbian existiert. Des Weiteren müssten auch die Abhängigkeiten für diese Bibliothek installiert werden. Die Nutzung von Mosquitto bietet eine einfache Anbindung an den Mosquitto-MQTT Server und ist zudem auf den öffentlichen Repositories für ARM verfügbar und muss einfach nur einmalig installiert werden. Weiterhin sind auch spezielle Compile-Switches (Pfadangabe) nicht mehr nötig, da Mosquitto dann direkt auf dem Zielsystem existiert und eingebunden werden kann.

### Auslesen der Konfiguration aus Ini-Dateien schlägt fehl

Um die benötigte Konfiguration des Masters einzulesen wurde die Python-Bibliothek [Configparser] verwendet. Da während der Entwicklung unserer Software ein Update des Configparsers erfolgte, wurde plötzlich das Auslesen der Konfiguration mit einem Fehler zur falschen Bibliotheksbenutzung quittiert. Abhilfe schafft das Installationsskript des Masters, welches den Configparser in der für dieses Projekt passenden Version 4.0.2 installiert.

### Leistungsgrenzen des Raspberry Pi Zero W

Der Raspberry Pi Zero W ist zwar ein relativ leistungsfähiger Mini Rechner, dennoch ist er im Verhältnis zu seinen größeren Vertretern doch eher schwach. So kann es durchaus passieren, dass er etwas schwach wird, sobald ein per Snapclient aufgefangenes Audiosignal über Bluetooth wiedergegeben soll. Zum Einen ist die Auswertesoftware aktiv, zum Anderen der Snapclient der konstant bzw. in Echtzeit Daten empfangen und diese wiedergeben muss. Sobald das WLAN-Signal etwas schwächer wird kann es vorkommen, dass der Stream und ggf. die MQTT-Verbindung verloren gehen.

# Zusammenfassung und Ausblick

Abschließend soll nochmal das gesamte Projekt mit den Ergebnissen kurz zusammengefasst und ein Ausblick hinsichtlich der Ausbaufähigkeit gegeben werden.

## Zusammenfassung

Dieses Projekt ermöglicht eine automatisiertes Musikstreaming in Räumen mit anwesenden Personen. Dieser, wenn auch etwas instabiler, Prototyp ermöglicht einen ersten Ansatz im Bereich der Heimautomation, eine intelligente Musiksteuerung zu einem relativ geringen Preis einzurichten. Günstige Komponenten gestatten es die Audiosteuerung auf frei verfügbarer Software zu entwickeln und diese zu erweitern. Die Durchgangsdetektion wurde mit zwei Ultraschallsensoren realisiert und funktioniert weitgehend stabil. Hierzu mussten hardwarebedingte Störanfälligkeiten mit C programmatisch behoben werden. Mit Python wurde eine Auswertungs- und Steuerungssoftware geschrieben, die einen Musikstream in den einzelnen Räumen an oder aus schaltet. Diese erhält und sendet ihre Daten über eine Kommunikationsstruktur zwischen den einzelnen Räumen, die mittels Raspberry Pis und MQTT aufgebaut wurde. Hierzu mussten einige Anpassungen der Softwarebibliotheken vorgenommen werden. Für die Auswahl der Musik wird das frei verfügbare Betriebssystem „Volumio“ verwendet, dass eine Weboberfläche zur Musiksteuerung im lokalen Netz hostet. Um die Musik als Stream ins lokale Netz zu broadcasten wird die kostenlose Software „Snapcast“ verwendet. Um die Musik abspielen zu können wurden günstige Bluetooth-Lautsprecher mit den Raspberry Pis verbunden, welche auch die Durchgangsdetektion erledigen.

## Ausblick

Mit den bereits vorhandenem Entwicklungsstand wäre eine Weiterentwicklung nicht nur hinsichtlich der Fehleranfälligkeit, sondern auch eine Erweiterung der bisherigen Features denkbar. Mit entsprechenden Komponenten wäre es möglich ein noch intelligenteres System zu entwerfen, welches die Anwesenheit von Personen im Raum unabhängig von den Durchgängen detektiert. Hierbei ist jedoch eine präzise Anpassung der Software und eine gute Vorüberlegung nötig um nicht Fremdeinwirkungen, wie Hitze durch Heizungen, Bewegungen von Vorhängen oder Geräusche durch Gegenstände (Mobile) als Anwesenheit von Menschen zu interpretieren. Ebenso sind Haustiere ein möglicher Faktor, welcher Audiosysteme automatisch Musik im Raum abspielen lassen würde. Der Verwendungszweck der selbstentwickelten Software ist aber nicht rein an die Audiotechnik gebunden. So könnten Daten wie Anzahl der Personen zu bestimmten Uhrzeiten in einer Datenbank gespeichert werden. Wenn man diese Daten über einen gewissen Zeitraum sammelt und auswertet, könnte das als gute Grundlage z.B. für ein intelligentes Heizungssystem verwendet werden, welches dafür sorgt, dass zu Stoßzeiten ein Raum gut beheizt wird und mittels statistischer Auswertung (in Abhängigkeit wieviel ein Zimmer prozentual am Tag besucht wird) demnach die Heizleistung unterschiedlich regelt.

# Literaturverzeichnis

[Configparser] configparser – Configuration file parser; <https://docs.python.org/3/library/configparser.html> (Zugriff: 10.06.2020)

[DIVA] J. LARSSON - Distance Estimation and Positioning Based on Bluetooth Low Energy Technology; <https://www.diva-portal.org/smash/get/diva2:859549/FULLTEXT01.pdf> (Zugriff: 12.08.2020)

[GNUPOSIX] Feature Test Macros (The GNU C Library); <http://www.gnu.org/software/libc/manual/html_node/Feature-Test-Macros.html> (Zugriff: 02.08.2020)

[Libmosquitto] libmosquitto man page; <https://mosquitto.org/man/libmosquitto-3.html> (Zugriff 09.06.2020)

[Microcontroller] Ultraschall Sensor HC-SR04 und kompatible Ultraschall-Module;

[https://www.mikrocontroller-elektronik.de/ultraschallsensor-hc- sr04/](https://www.mikrocontroller-elektronik.de/ultraschallsensor-hc-sr04/) (Zugriff: 17.01.2020)

[MQTT] MQTT man page;

<http://mosquitto.org/man/mqtt-7.html> (Zugriff: 13.02.2020)

[MQTTC] GitHub eclipse/paho.mqtt.c; An Eclipse Paho C client library for MQTT for Windows, Linux and MacOS;

<https://github.com/eclipse/paho.mqtt.c>

[MQTTPython] Paho-Mqtt – PyPI;

<https://pypi.org/project/paho-mqtt/> (Zugriff: 13.02.2020)

[Pulldown] Pullup- und Pulldown Widerstand;

<https://rn-wissen.de/wiki/index.php/Pullup_Pulldown_Widerstand> (Zugriff: 17.01.2020)

[Snapcast] Saiyato/volumio-snapcast-plugin; Volumio 2 SnapCast plugin, to easily manage SnapCast functionality; <https://github.com/Saiyato/volumio-snapcast-plugin> (Zugriff 12.06.2020)

[Sparkfun] Ultrasonic Ranging Module HC - SR04;

<https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf> (Zugriff: 04.06.2020)

[TutRasp] Entfernung messen mit Ultraschallsensor HC-SR04 – Raspberry Pi;

<https://tutorials-raspberrypi.de/entfernung-messen-mit-ultraschallsensor-hc-sr04/> (Zugriff: 17.01.2020)

[VNC] VNC (Virtual Network Computing) – Raspberry Pi Documentation; <https://www.raspberrypi.org/documentation/remote-access/vnc/> (Zugriff: 02.08.2020)

[VolumioAPI] Volumio Documentation; Abschnitt API; <https://volumio.github.io/docs/index.html> (Zugriff: 24.01.2020)

[VolumioPi] Download Volumio Audiophile Music Player for Raspberry Pi;

<https://volumio.org/get-started/> (Zugriff: 11.02.2020)

[WiringPi] Wiring Pi; GPIO Itnerface library for the Raspberry Pi; <http://www.wiringpi.com> (Zugriff 05.08.2020)

1. Ehrenwörtliche Erklärung

**Ehrenwörtliche Erklärung**

Wir versichern hiermit, dass die Projektarbeit mit dem Titel

Smart Audio

|  |
| --- |
| Realisierung eines intelligenten Audiosystems |

selbständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie nicht an anderer Stelle als Prüfungsarbeit vorgelegt wurde.

|  |  |  |
| --- | --- | --- |
| Ort |  |  |
| Datum |  | Unterschrift |

1. GetDistance-Methode

long getDistance(int trigger, int echo) {

long startBreak = 0;

long startM = 0;

long stopM = 0;

double ellapse = 0;

long dist = 0;

//20ms nach trigern darf erneut gemessen werden

long beginMeasuring = 0;

long endMeasuring = 0;

double waitTime = 0;

//trigger kurz an und aus schalten

digitalWrite(trigger, HIGH);

delayMicroseconds(100);

digitalWrite(trigger, LOW);

beginMeasuring = clock();

//Startzeiten

startM = stopM = clock();

//Startzeit speichern

while (digitalRead(echo) == 0 && waitTime < 0.02) {

endMeasuring = startM = clock();

waitTime = (double)(endMeasuring-beginMeasuring)/CLOCKS\_PER\_SEC;

//wenn in Schleife gefangen (noramlerweise dauert das keine 2 millisiekunden)

if(waitTime >= 0.02){

///DEBUG\*/ LF++;

break;

}

}

while (digitalRead(echo) == 1){

stopM = clock();

}

endMeasuring = clock();

//jetzt noch warten bis Sensor für nächste Messung bereit ist

waitTime = (double)(endMeasuring-beginMeasuring)/CLOCKS\_PER\_SEC;

while (waitTime < 0.02 ){

endMeasuring = clock();

waitTime = (double)(endMeasuring-beginMeasuring)/CLOCKS\_PER\_SEC;

}

//distanz berechnen

ellapse = (double)(stopM - startM) / CLOCKS\_PER\_SEC;

dist = (ellapse \* 34320) / 2;

///DEBUG\*/ M++;

//negative Messungen gibts nicht und näher als 3 cm kann HC-SR04 nicht messen

//also wird 1000cm zurückgegeben

if(dist < 3) dist = 1000;

return dist;

}

1. Sicherheitsdistanzmessungen

if (distance\_a < minDistance) {

///DEBUG\*/ DA = (int) distance\_a;

counter = 0;

//Fehlmessungen ausschließen indem a nochmals abgefragt wird

for(i = 0; i <(sizeof(l) / sizeof(\*l)); i++){

l[i] = init;

}

for(i = 0; i <(sizeof(l) / sizeof(\*l)); i++){

addDistance = getDistance(trigger1, echo1);

l[i] = addDistance;

}

for(i = 0; i <(sizeof(l) / sizeof(\*l)); i++){

if(l[i] < minDistance) counter++;

}

if(counter >=2){

/\*liste initialisieren\*/

for (i = 0; i <(sizeof(liste) / sizeof(\*liste)); i++){

liste[i] = init;

}

for (i = 0; i < (sizeof(liste) / sizeof(\*liste)); i++) {

addDistance = getDistance(trigger2, echo2);

liste[i] = addDistance;

}

for (i = 0; i < (sizeof(liste) / sizeof(\*liste)); i++) {

if (liste[i] < minDistance) {

// printf("\nin\n");

///DEBUG\*/ Iold = I++;

trigger = 1;

SASendMessage("in", topic);

break;

}//ENDIF

}//ENDFOR

}//ENDIF

1. Skript InstallRequirementsMaster.sh

echo "Checking requirements for Smart-Audio Master"

echo

echo "Mosquitto:"

if [ $(dpkg-query -W -f='${Status}' mosquitto 2>/dev/null | grep -c "ok installed") -eq 0 ];

then

echo "Installing Mosquitto"

sudo -s apt-get install -y Mosquitto

else

echo "Mosquitto already installed"

fi

echo

echo "Python-Pip:"

if [ $(dpkg-query -W -f='${Status}' python-pip 2>/dev/null | grep -c "ok installed") -eq 0 ];

then

echo "Installing python-pip"

sudo -s apt-get install -y python-pip

else

echo "Python-Pip already installed"

fi

echo

echo "Paho-MQTT:"

if [ $(pip list | grep -c "paho-mqtt") -eq 0 ];

then

echo "Installing Paho-MQTT"

sudo -s pip install paho-mqtt

else

echo "Paho-MQTT already installed"

fi

echo

echo "Configparser 4.0.2:"

if [ $(pip list | grep -c "configparser") -eq 0 ];

then

echo "Installing Configparser"

sudo -s pip install 'configparser==4.0.2'

else

echo "Configparser already installed"

fi