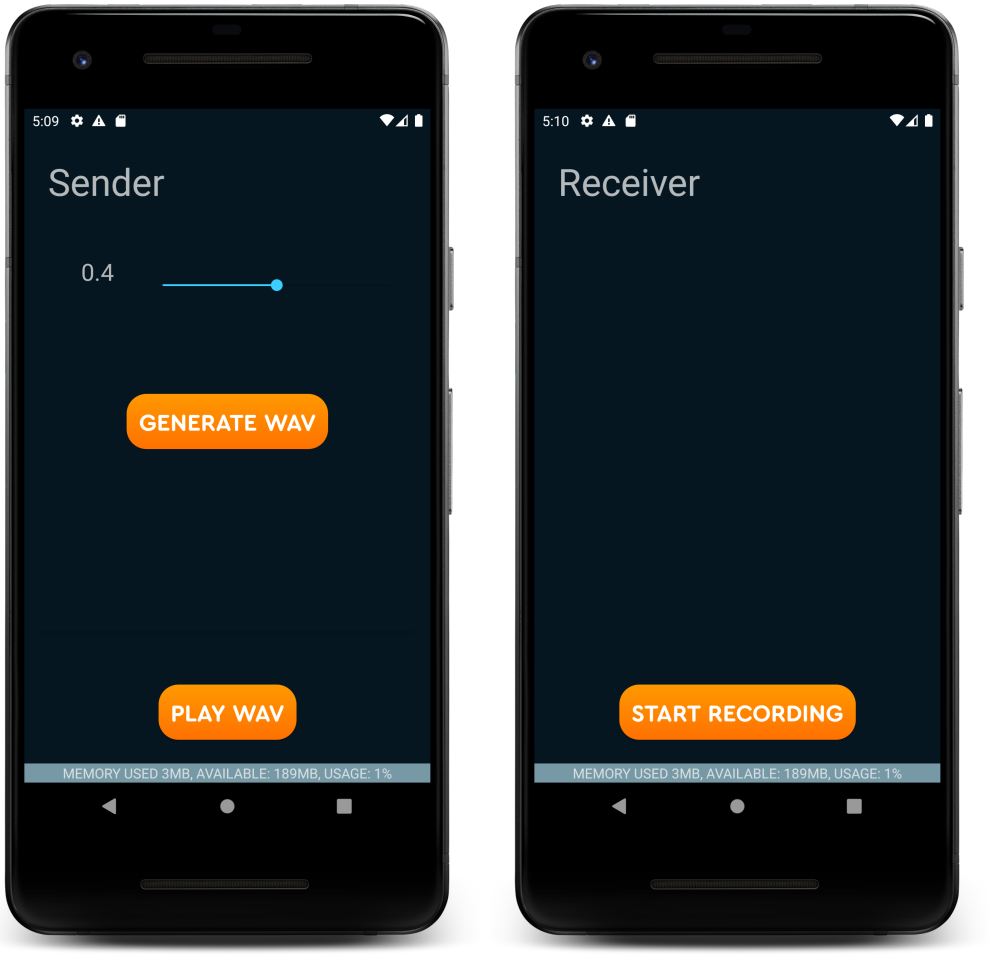


Applikation för testning av ultraljudskommunikation på Android-enheter

Examensarbete inom Data- och Informationsteknik

Erik Börne
Rebecca Carlson

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2021

EXAMENSARBETE

Applikation för testning av ultraljudskommunikation på Android-enheter

Erik Börne
Rebecca Carlson

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2021

Applikation för testning av ultraljudskommunikation på Android-enheter
Erik Börne
Rebecca Carlson

Examinator: Jonas Almström Duregård, Institutionen för data- och informationsteknik
Handledare: Ulf Norell, Institutionen för data- och informationsteknik
Företagshandledare: Viktor Åkerskog, Sleep Cycle AB

Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola / Göteborgs Universitet
412 96 Göteborg
Telefon: 031-772 1000

Omslag: *Skärmdump från 'Sender' och 'Reciever'-delarna i applikationen*

Institutionen för Data- och Informationsteknik
Göteborg 2021

Sammanfattning

Företaget Sleep Cycle AB har som mål att genom sin app hjälpa sina användare att sova bättre om natten. De gör detta genom att analysera deras olika sömnfaser och att väcka dom på morgonen när de går in i lätt sömn. Om man däremot är fler än en person som sover tillsammans kan det påverka sömnanalysen så att den blir mindre exakt. För bättre sömnanalys har de en funktion på iPhone som kopplar ihop alla mobiler på samma WiFi, avgör vilken mobil som uppfattat rörelsen starkast och därför sannolikt är dess källa. De är nyfikna på ifall det är möjligt att vidareutveckla denna funktion genom att istället koppla ihop mobilerna i samma rum via ultraljud istället för igenom WiFi.

För att utforska denna möjlighet var projektets mål att utveckla en Androidapplikation för att testa ultraljudskommunikation mellan mobila enheter. Syftet var att avgöra ifall man kunde utföra kommunikation som både var funktionell och icke-hörbar för människor.

Under utvecklingen var det främst fokus på integrationen av den utomstående signalbehandlingskoden i Python som hanterar genereringen och tolkningen av ultraljudet. Det genomfördes även funktionalitets- och hörseltester för att komma fram till rimliga slutsatser kring frågeställningarna.

Resultaten visade på att mobiltelefoner har tillräckligt med processeringskraft för att kunna generera och tolka ultraljud. Däremot så krävs det specifika omständigheter för rummets storlek och akustik för ett lyckad ultraljudskommunikation. Andra omständigheter såsom Androids inbyggda upplösningssbegränsningar på WAV-filer, problem som är specifika till vissa mobilmodeller, samt hårdvarubegränsningar gör att det är omöjligt att dra en slutsats kring detta för tillfället och kräver fortsatt undersökning.

Baserat på resultaten från hörseltesterna som utfördes kunde man dra slutsatsen att människor inte kan höra det ultraljud som användes under funktionalitetstesterna. Dessa resultat visar att det finns möjligheter att sänka frekvensen och öka amplituden i hopp om att förbättra kommunikationens funktionalitet, samtidigt som det fortfarande är utanför människans hörselspann.

Keywords: Android, Python, Ultraljud, Ultraljudskommunikation.

Förord

Vi vill säga ett stort tack till våra handledare Ulf Norell och Viktor Åkerskog för deras hjälp och rådgivning under projektets gång. Samt till Viktors barn som deltog som hörseltestdeltagare. Vi vill även tacka Mikael Kågebäck för hans assistans i samband med signalbehandlingenskoden och allmänt krångel med Python.

Tack även till våra partners som har varit givmilda med att låna ut sina mobiltelefoner i samband med funktionalitetstesterna.

Erik Börne & Rebecca Carlson, Gothenburg, March 2021

Innehåll

Figurer **x**

Tabeller **xiii**

1 Introduktion **1**
1.1 Bakgrund . . . 1
1.2 Syfte . . . 1
1.2.1 Frågeställningar . . . 1
1.2.2 Mål . . . 2
1.3 Avgränsningar . . . 2

2 Teori **3**
2.1 Ultraljud . . . 3
2.2 Amplitudmodulering . . . 3
2.3 WAVE (WAV) . . . 3
2.4 LPCM . . . 4
2.5 NumPy . . . 4
2.6 SciPy . . . 4
2.7 BLAS . . . 4
2.8 LAPACK . . . 5
2.9 python-for-android . . . 5
2.10 Chaquopy . . . 5

3 Metod **7**
3.1 Signalbehandlingskod i Python . . . 7
3.1.1 Refaktorering av Pythonkod . . . 8
3.1.2 Python på Android . . . 8
3.1.2.1 python-for-android . . . 8
3.1.2.2 Chaquopy . . . 9
3.1.3 Uppspelning av ultraljud . . . 10
3.1.3.1 Audiotrack . . . 11
3.1.4 Mottagande av ultraljud . . . 11
3.2 Applikationens upplägg och funktion . . . 12
3.3 Tester . . . 12

4 Resultat **13**
4.1 Applikationen . . . 13
4.1.1 Sender . . . 13
4.1.2 Receiver . . . 14
4.1.3 Design . . . 14
4.2 Testresultat . . . 15
4.2.1 Funktionalitetstest . . . 15
4.2.2 Hörseltest . . . 18

5 Diskussion **19**
5.0.1 Funktionalitetstester . . . 19
5.0.2 Hörseltester . . . 20
5.0.3 Upplösningsbegränsnigar på WAV-filer . . . 20
5.0.4 Ultraljudets frekvensvärde . . . 21
5.0.5 Processering . . . 21
5.0.6 Mobilspecifika problem . . . 21
5.0.7 Etik och hållbarhet . . . 22
5.0.7.1 Människopåverkan . . . 22
5.0.7.2 Integritet . . . 22
5.0.7.3 Hållbarhet . . . 22
5.0.8 Funktionalitet baserat på användaren . . . 23

6 Slutsats **25**

Referenser **27**

Figurer

3.1 Pythonkodens beteende 7
3.2 Verktygskedjor 9

4.1 Sender skärmdump 13
4.2 Reciever skärmdump 14
4.3 Amplitudfaktor = 0,4 16
4.4 Amplitudfaktor = 0,1 17
4.5 Amplitudfaktor = 0,5 17

Tabeller

Tester som ska lyckas 1 15
Tester som ska lyckas 2 16
Tester som *inte* ska lyckas 16

1

Introduktion

1.1 Bakgrund

Företaget Sleep Cycle AB har utvecklat en app, Sleep Cycle, som hjälper sina användare att sova bättre om natten genom att analysera deras olika sömnfaser och att väcka dom på morgonen när de går in i lätt sömn. Om man däremot är fler än en person som sover tillsammans kan det påverka sömnanalysen så att den blir mindre exakt. För att motverka detta har de implementerat en funktion på iPhone som kopplar ihop alla mobiler på samma WiFi, avgör vilken mobil som uppfattat rörelsen starkast och därför sannolikt är dess källa. Detta förbättrar noggrannheten för båda användarna.

1.2 Syfte

Syftet med projektet är att ta reda på ifall det är möjligt att vidareutveckla denna funktion genom att koppla ihop användarnas mobiler via ultraljudskommunikation istället för genom WiFi. Detta ska säkerställa att endast mobilerna i samma rum kopplas ihop och ska även motverka onödig kommunikation över ett nätverk. Det sistnämnda blir speciellt problematiskt när flera enheter kommunicerar via ett delat nätverk såsom på hotell eller i studentkorridorer.

1.2.1 Frågeställningar

De frågeställningar som ska besvaras under projektet är:

- Har en mobiltelefon tillräcklig prestanda för att genomföra den signalbehandling som krävs för ultraljudskommunikation?
- Är det möjligt att, med tillräckligt liten felmarginal, säkerställa att två enheter befinner sig i samma rum med hjälp av ultraljud?
- Kan en välfungerande ultraljudskommunikation utföras på ett sätt som inte kan höras av människor?

1.2.2 Mål

För att besvara frågeställningarna ovan ska vi utveckla en Androidapp som underlättar testning av ultraljudskommunikation. Applikationen ska ha två funktioner: att kunna skicka och tolka ultraljud. Med hjälp av appen ska man dels kunna avgöra ifall prestandan i en telefon är tillräcklig för den processesering som behöver göras, både för genereringen av ultraljudet samt att kunna ta emot och tolka data.

Appen ska även kunna användas för att testa ifall enheter som är i samma rum kan kommunicera med varandra och samtidigt exkludera enheter utanför rummet. Under dessa tester ska vi även determinera vilken frekvens och amplitud på ljudet som blir optimal för just detta.

För att avgöra om människor kan höra ultraljudet kommer vi att läsa tidigare studier som har gjorts angående människors uppfattning av ultraljud samt utföra egna mindre hörseltester.

1.3 Avgränsningar

Projektet kommer inte att innefatta utveckling av att kunna ta emot, behandla, eller tolka ljud för att koppla ihop mobilenheter. Vi kommer att använda oss av redan färdig pythonkod från Sleep Cycle som hanterar just detta.

Vi kommer inte att göra egna undersökningar angående huruvida ultraljud påverkar människor eller djur i stora drag. Vi kommer endast att testa ifall vi och några få från Sleep Cycle kan höra ultraljuden som krävs för att koppla ihop enheter eller inte.

2

Teori

I detta kapitel återfinns ett uppslag av definitioner och beskrivningar tillhörande koncept och begrepp som används inom denna rapport.

2.1 Ultraljud

Ultraljud definieras som ljudvågor med frekvenser som överstiger den övre tröskeln för mänsklig hörsel. För yngre individer brukar denna tröskel ligga kring 20kHz. Inom Android finns flaggor, så kallade "properties", som indikerar om enhetens mikrofon eller högtalare har stöd för frekvenser som närmar sig ultraljud. Omfånget detta syftar till i Android 11 sträcker sig från 18,5kHz till 20kHz.[1]

2.2 Amplitudmodulering

Mer uppmärksammat som AM i allmänbruk, används vanligen för att sända ut radiosändningar. Det går ut på att man kodar in information av olika former i amplituden på radiovågen, för att sedan på mottagarsidan ta emot och tolka informationen.

2.3 WAVE (WAV)

Filformatet Waveform Audio File Format, som förkortas till WAVE eller alternativt WAV på grund av dess filändelse. Det en kontainertyp som kan lagra ljuddata som spelats in med olika kodekar och metainformation om detta ljud. Filformatet är en implementation av Resource interchange file format (RIFF), som definierades av IBM och Microsoft.[2] Det vanligaste kodeken som används tillsammans med WAVE behållaren är LPCM.

2.4 LPCM

Linear pulse-code modulation är ett kodek som representerar en samplad analog signal. Att den är linjär innebär att den samplar signalen med jämna mellanrum. När man har samplat signalen kvantiseras det till ett digitalt värde. Valet av datatyp påverkar därför kvalitén på den digitala ljudströmmen, då en datatyp med högre upplösning kan beskriva signalen med högre precisson.

Några vanligt förekommande datatyper som används med LPCM är: 16-bitar integers, som används för bland annat CD-skivor.[3] 24-bitar integers, som används i DVDs och Blu-ray Discs.[4] Även 32-bitar och till och med 64-bitar förekommer.

Det finns även starka fördelar med att använda flyttal istället för heltal som datatyp, då man med dessa kan skala ner signalen utan att tappa precision. Nackdelen med detta är att operationer med flyttal är mer kostsamma.[5]

2.5 NumPy

NumPy är ett Python-bibliotek som förser användaren med diverse matematiska verktyg. Dessa inkluderar bland annat skapandet av multidimensionella arrayer, utförandet av snabba operationer på arrayer, grundläggande linjär algebra, med mera.[6]

2.6 SciPy

SciPy är ett Python-bibliotek som bygger vidare på det förnämnda NumPy-biblioteket genom att förse användaren men allt fler matematiska verktyg. Några av dessa innefattar integration, Fouriertransformer, och signalbehandling. Det sistnämnda är det som används mycket i detta projekt.[7]

2.7 BLAS

Basic Linear Algebra Subprograms är en samling rutiner som behandlar de grundläggande operationerna för vektorer och matriser. BLAS är skrivet i Fortran, det finns ett gränssnitt för att anropa BLAS via C som kallas CBLAS. Då BLAS är väloptimerat och effektivt i sina operationer används det ofta som en viktig byggsten i mer omfattande program, ett exempel på detta är LAPACK.[8]

2.8 LAPACK

Linear Algebra Package är en uppsättning rutiner som andvänds för att lösa linjära ekvationssystem, utföra minsta-kvadraten-metoden på ett linjärt ekvationssystem, ta fram egenvärden och singulärvärden. För att göra detta optimerat så andvänds BLAS som ett underliggande lager.[9]

2.9 python-for-android

python-for-android är ett open-source byggverktyg som möjliggör för apputveckling till Android i Python. Detta åstadkoms genom att kompilera ner Pythoninterpretatorn, alla dess dependencies, de paket som andvänds i programmet (bibliotek) samt Pythonkoden som ska utföras, sedan paketeras detta i en Androidapp tillsammans med en bootstrap.[10]

För att få tillgång till paket behöver man sätta upp recept. Recept har i uppgift att instruera verktyget vart man hämtar paketet och, baserat på hur modulen är uppbyggd, även hur man bygger det. Den enklaste formen av recept är paket som endast är skrivna i Python, då dessa endast behövs förses med en nedladdningslänk. För moduler som innehåller kompilerade komponenter, såsom SciPy och NumPy, behöver receptet även se till att dessa komponenter kompileras mot den önskade arkitekturen. Många av de vanligaste paketen till Python har redan färdiga byggrecept. NumPy är ett utav dessa, dock så saknas SciPy.[11]

2.10 Chaquopy

Chaquopy är ett closed-source program gör det möjligt att exekvera Pythonkod på Android. Likt python-for-android, kompileras allt som behövs för att koden ska kunna köras på Android. Chaquopy tillåter att man blandar Pythonkod med Java och Kotlin i applikation, något som åstadkoms med API:er finns i båda riktningar.

Installationsprocessen är snabb och smärtfri då detta kan göras genom Android Studios föredragna byggsystem, Gradle. Många av de mest använda paketen finns redan tillgängliga, bland dessa återfinns bland annat både NumPy och SciPy.[12]

3

Metod

I detta kapitel redovisas genomförandet av projektet. Först beskrivs applikationens back-end med fokus på hur Pythonkoden integrerades i applikationen. Därefter beskrivs applikationens funktion och visuella upplägg, och sist de tester som utfördes för att komma fram till projektets slutsatser.

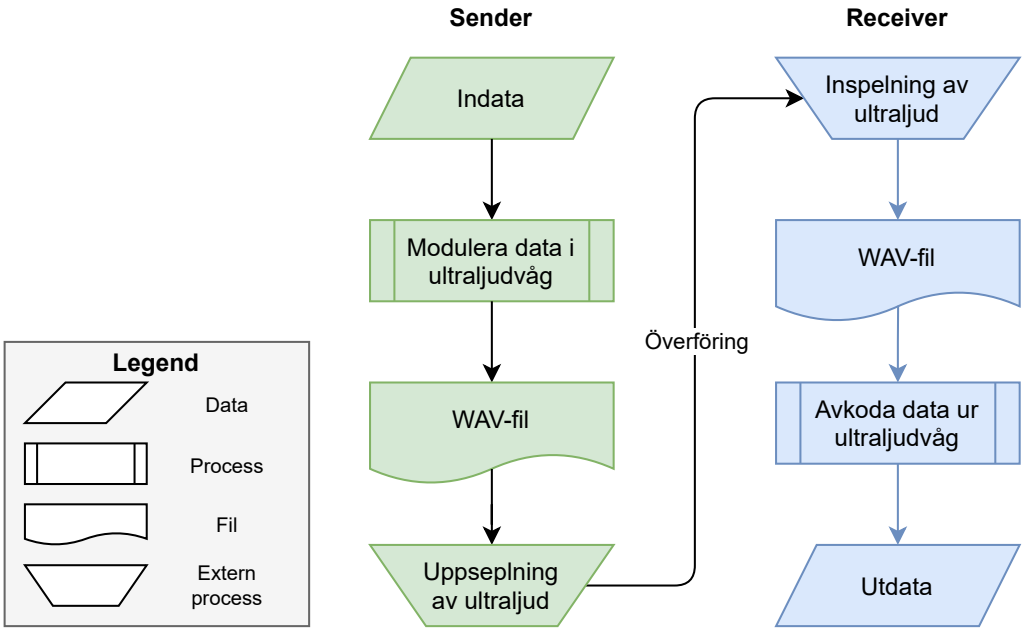
3.1 Signalbehandlingskod i Python

Första steget i processen var att integrera Pythonkoden från Sleep Cycle med applikationen. Pythonkodens funktionalitet kan beskrivas kortfattat på följande vis.

Givet en lista med heltal av en bestämd storlek, kan koden generera en ultraljudsvåg som är amplitudmodulerad med den önskade datan. Därefter kan man på mottagarsidan ta emot ultraljudsvågen och tolka datan.

På så sätt kan man jämföra listan med heltal som man skickade med den som man tar emot. Om listorna innehåller samma värden så har kommunikationen fungerat.

Figure 3.1: Pythonkodens beteende



Sleep Cycle hade tidigare lyckats kommunicera mellan två datorer som utför processen ovan. Utmaningen i detta projekt var att man nu inte kunde köra Pythonkoden direkt, utan behövde kalla på den genom Androidapplikationen. Detta visade sig komma med ett flertal utmaningar som kommer att tas upp.

3.1.1 Refaktorering av Pythonkod

Pythonkoden delades upp i två delar, sender() och receiver(), som hanterade varsin del av koden för sig, till skillnad från tidigare när det allt- kördes i samma metod. sender() hanterade genereringen av ultraljuds-filen och receiver() hanterade tolkningen av det mottagna ultraljudet. Man kan se denna uppdelning i figur 3.1.

Detta gjordes för att underlätta senare i projektet när man enbart vill kalla på en specifik metod från Androidapplikationen beroende på vilken "roll" som enheten har i kommunikationen.

3.1.2 Python på Android

Utveckling av Androidapplikationer sker ofta i programmeringsspråken Java och Kotlin men kan även ske i C och C++ via Androids NDK. För att kunna exekvera kod som är skriven i Python mot Androidplattformen prövades två program som tillåter detta. Den första var 'python-for-android' som var ett förslag som gavs av Sleep Cycle i början av projektet. Den andra var 'Chaquopy' som hittades i samband med informationssökning om python-for-android.

För att kunna behandla ultraljud behövs det ett flertal moduler utöver standardbibliotek för Python. Det som behövs är NumPy och SciPy som till stor del består av färdigkompilerade moduler som är skrivna i programmeringsspråk som är snabbare än Python. För att bygga SciPy modulen så behöver man två Fortran bibliotek: LAPACK och dess underliggande program BLAS.

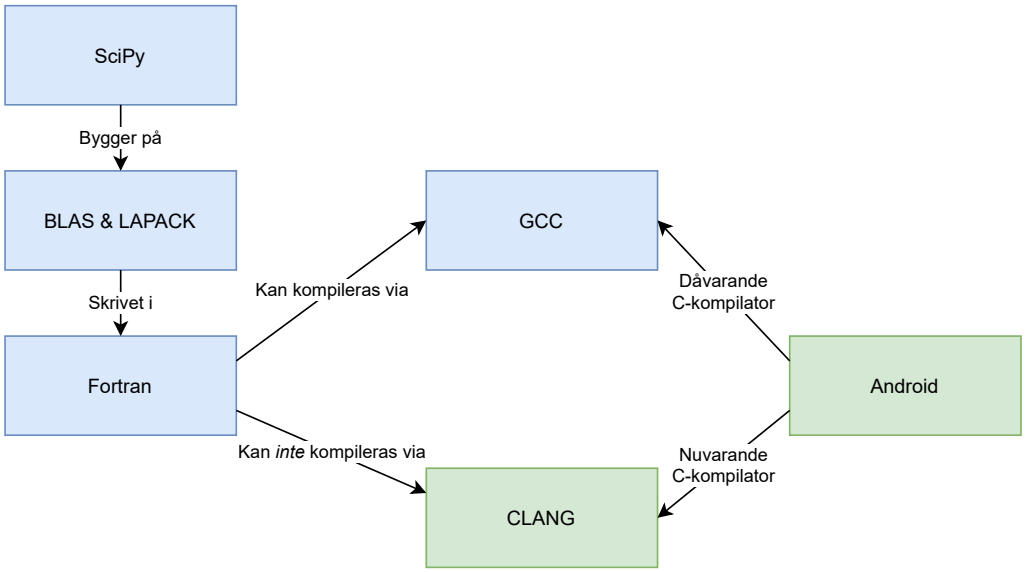
3.1.2.1 python-for-android

När python-for-android hade installerats behövde man specificera vilka komponenter man behövde för sitt projekt. Dessa komponenter byggs av recepten som tas upp i sektion 2.9. De flesta recept som behövdes för att bygga de Pythonpaketen som användes av Pythonkoden var redan fördefinierade. Det enda paket som inte hade ett byggrecept, var SciPy, som därmed behövdes ett nytt recept för detta paket.

I det nya receptet specificeras först en URL som pekade mot källkoden för SciPy. Därefter behövde man förbereda krosskompilering riktad mot de tre instruktionsseten som Android platformen främst består utav. Den bredast täckande är den äldre Armeabi-v7a, en 32-bitars repotoire för både 32 och 64-bitars Arm-processorer. Den andra är Arm64-v8a som endast kan köras på 64-bitars Arm-processorer. Den sista är x86 som är ett 32-bitar instruktionsset som typiskt är riktad mot PCs.

BLAS och LAPACK behövde också byggas så att SciPy kunde anropa dom. För att kompilera dessa Fortranbibliotek behövdes det en speciallanpassad GCC-toolchain. Syftet med detta var att, utöver C-kompilatorn, även aktivera GCCs Fortrankompilator i den nya verktygkedjan, vilket då skulle tillåta för krosskomilering av Fortrankoden. Informationen om detta visade sig dock vara utdaterad, då Android har gått över till att använda CLANG som kompilator och denna hade inte liknande stöd för Fortran. [13]

Figure 3.2: Verktygskedjor



Processen för detta var långgående och resulterade enbart i ett flertal förvirrande felmeddelanden. Då det inte var tydligt vad problemet var och för mycket tid hade ägnats åt det, bestämdes det att det var dags att byta byggprogram från python-for-android. Valet på nytt byggprogram föll på Chaquopy som hittades under felsökningsprocessen.

3.1.2.2 Chaquopy

För att installera Chaquopy behövde man, i Gradle-byggskriptet för projektet, lägga till dess förvar av källkod[1] och registrera det som ett underliggande program[2]. Sedan behövde man, i byggfilen för modulen, specificera vilka Pythonpaket man vill installera samt vilka instruktionsset applikationen ska bygga mot. Alla Pythonpaket som behövdes fanns tillgängliga och redo för att användas.

Pythonkoden placerades i en projektmapp med namnet 'python', där Chaquopy letade efter Pythonmoduler. En viktig sak var att se till att Pythoninterpretatorn endast startades exakt en gång under appens livscykel. Vid det här tillfället kunde man anropa Pythonkoden från applikationen och då börja fokusera på uppspelningen och mottagandet av ultraljud.

[1] "repository"
[2] "dependency"

3.1.3 Uppspelning av ultraljud

När det gick att kalla på Pythonkoden via Androidapplikation var det fortfarande problem med uppspelningen av ultraljudet inne i appen. Däremot, när den genererade WAV-filen spelades upp utanför projektet, genom en vanlig mediaspelare på datorn till exempel, så kunde mottagarsidan på mobilen ta emot och tolka datan. Så man kunde dra slutsatsen att det inte var något fel med själva ljudfilen utan det var något bekymmer med Android.

Problemet visade sig vara att Android enbart stödjer WAV-filer i formatet 16-bitars Int medan den WAV-fil som genererades var i 64-bitars Float.[15] Ljudvågen som genererar i Pythonkoden representeras av en lista av värden. Ett enstaka element i listan kan ta upp till den specificerade storleken i minnet, dvs 2^{16} för 16 bitar eller 2^{64} för 64 bitar.

För att lösa detta problem behövde WAV-filen konverteras till rätt storlek så att den blir kompatibel med Android. I Pythonkoden genereras WAV-filen med hjälp av `scipy.wavfile.write`[14] och den bestämmer automatiskt storleken och formatet på filen baserat på den indata den får, det vill säga listan av värden. För att utföra konverteringen så behöver man ändra elementen i listan så att de uppfyller kraven för det format man är ute efter. I koden nedan är `y` en NumPy-array som innehåller de värden som representerar ljudvågen som ändras för att uppfylla kraven för att kunna generera 16-bitars Int WAV-fil.

```
y = y*amp*32767          (1)
y = np.around(y)         (2)
y = y.astype(np.int16)   (3)
```

`(1)` `y` multipliceras med en amplitudfaktor för att anpassa ljudstyrkan. För WAV-filer av typen Float är max- och minimumvärdena på elementen ± 1.[14] För att de ska hamna inom spannet av storleken för en signerad 16-bitars Int så multipliceras varje element med $2^{16}/2$. `(2)` Varje element i `y` avrundas till närmsta heltal eftersom att decimalvärdesiffrorna kommer att tappas i konverteringen till Int i nästa steg. `(3)` Varje element i `y` konverteras till 16-bitars Int och arrayen kommer därmed att kunna generera en WAV-fil i det önskade formatet.

3.1.3.1 Audiotrack

När den genererade ultraljudsfilen hade konverterats till ett format som Android stödjer kunde man spela upp den i applikationen. Däremot så kunde fortfarande inte mottagarenheten ta emot data när uppspelningen skedde genom en annan mobilenhet. Dessutom när man spelade upp ultraljudet via applikationen så lät det annorlunda mot hur det borde ha gjort. Ultraljudet borde inte vara hörbart men ljudet som spelades upp sprakade och brusade.

Fram till denna punkt i projektet hade uppspelningen varit genom MediaPlayer[16], så det bestämdes att försöka testa två nya mediaspelare. Den första var ExoPlayer [17] som är den mediaspelare som används av till exempel YouTube och den andra var AudioTrack [18] som är en mer lågnivå mediaspelare och ligger i grunden av de tidigarenämnda MediaPlayer.

Kommunikationen fungerade när ultraljudet spelades upp via AudioTrack. Anledningen varför det fungerade via denna mediaspelare istället för via MediaPlayer är fortfarande oklar. Teorin är att MediaPlayer spelar upp med en lägre sample rate istället för 48kHz som drog ner frekvensen på ljudet och därmed förvrängde datan inbäddad i ultraljudet.[15]

3.1.4 Mottagande av ultraljud

För att kunna ta emot det genererade ultraljudet behövde man först kunna spela in ljud från omgivningen. Detta gjordes genom att använda AudioRecorder i Android. Den blev initierad till att spela in ljudet som 16-bitars LPCM i endast en ljudkanal. När inspelningen var avklarad behövde man synliggöra den till Pythondelen. Detta löstes genom att lagra undan den inspelade datan i applikationens interna minnesutrymme och skicka vidare dess adress via anropet till Python.

Ett problem som uppstod med detta var att Pythonkoden förväntade sig en WAV-fil för att hämta ljudströmmen. Android har dock inget inbyggt sätt att generera WAV-filer så det var något som var behövdes lösas innan man kunde analysera ljudet som hade spelats in. Lösningen var att skapa hjälpobjektet WavRecorder som använde sig av tidigarenämnda AudioRecord. Dess uppgift var att spela in en ljudfil i samma kodek, det vill säga 16-bitar PCM, paketera ner den i en WAV-behållare, och sedan placera den i enhetens minne.

WavRecorders funktion skalades därefter tillbaka, från att paketera WAV-filer till att endast lagra undan det inspelade ljudet direkt i sitt PCM-16 format. Anledningen till detta är att den metadata som lagras i WAV-filen var inte nödvändig utöver samplingsfrekvensen. Men eftersom denna är konstant definierad som 48000 blir även den onödig. För att kunna tolka datan som lagras måste den läsas in med samma endian och förstås samma datatyp som den lagrades i.

3.2 Applikationens upplägg och funktion

Liksom signalbehandlingskoden så är också applikationen visuellt och kodmässigt primärt uppdelad i två delar - Sender och Reciever. Allting utgår ifrån en mainActivity eller aktivitet vilket kan liknas vid en main()-method i andra sammanhang.[21] Denna aktivitet agerar som en global komponent för applikationen och hanterar bytet emellan fragmenten Sender och Reciever. Ett fragment kan beskrivas som en seperat UI-komponent som ligger nedanför den förnämnda Activity-komponenten i hierarkin.[22]

När applikationen startar så hamnar användaren automatiskt i Sender-fragmentet. En ultraljudsfil genereras automatiskt genom att kalla på sender()-delen i signalbehandlingskoden i Python. Ultraljudet kan spelas upp från samma fragment. Användaren kan välja att generera en ny ultraljudsfil som ersätter den äldre.

Från Receiver-fragmentet kan man via en knapptryckning spela in ljud från mobilens omgivning. Som beskrivet i 3.1.4, så sparas den inspelade ljudströmmen som en binärfil och skickas därefter till signalbehandlingskoden i Python för att avkodas. När processeringen är klar så skrivs datan ut på skärmen så att användaren kan avgöra ifall kommunikationen har lyckats eller inte.

3.3 Tester

För att testa applikationen utfördes två olika slags tester: funktionalitets- och hörseltester. Funktionalitetstesterna var för att verifiera att kommunikation mellan två mobilenheter var möjlig samt att determinera att den fungerar under rätt omständigheter - att enbart två mobiler i samma rum ska kunna kommunicera och exkludera andra enheter i samma byggnad. Dessa tester delades ytterligare upp i två kategorier: tester som skulle lyckas och som *inte* skulle lyckas. Detta var för att kunna verifiera kriteriet som angavs tidigare.

Testerna utfördes till att börja med i grupprum i Chalmers egna lokaler och flyttades därefter till ett verkligt sovrum för anledningar som beskrivs senare i resultatdelen.

Hörseltesterna utfärdades på två vis. Det första testtillfället sammanföll samtidigt som funktionalitetstesterna pågick. I och med att utvecklarna skickade och tog emot det genererade ultraljud via applikationen så blev de naturligt utsatta för ljudet själva under tiden. Det andra testtillfället skedde genom att en anställd på Sleep Cycle laddade ner applikationen och spelade upp ljudet för sina barn och rapporterade resultaten.

4

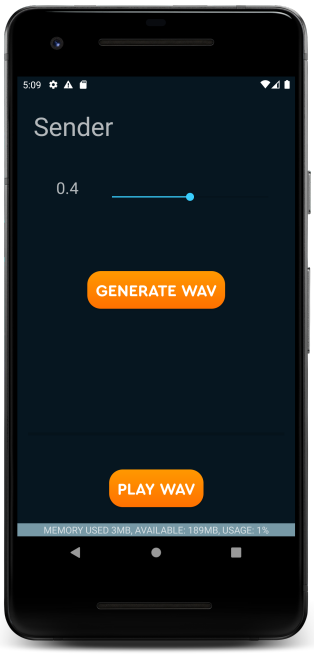
Resultat

Nedan beskrivs projektets slutprodukt i form av den färdiga Androidapplikationen. Därefter redovisas resultaten från testerna som utfärdades med hjälp av applikationen.

4.1 Applikationen

Som beskrivet i metodkapitlet ovan så är appen uppdelad i två delar, Sender och Receiever, som man byter till beroende på vilken kommunikationsroll som enheten har. Man kan enkelt byta emellan dom genom att svepa antingen till höger eller vänster över skärmen. När man startar appen hamnar man automatiskt i Senderdelen. Något som båda delarna har gemensamt är att de har en "header" som skriver ut processens minnesförbrukning.

4.1.1 Sender



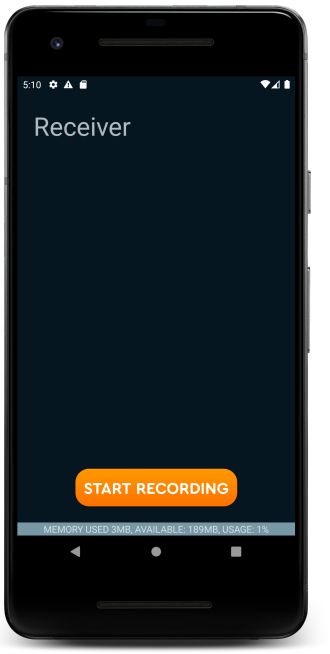
Den övre delen av Sender hanterar ultraljudsgeneringen och den nedre hanterar uppspelningen. Längst upp så kan man anpassa det amplitudvärd som man vill att ultraljudet ska genereras med. De anpassas via en slider till höger om rutan där värdet står.

Därefter kan man generera en ny ultraljudsfil genom att trycka på knappen nedan. Detta kör sender()-delen i Pythonkoden igen med det nya värdet och sparar över den tidigare ultraljudsfilen i mobilens eller emulatorns minne.

På den nedre delen av Sender finns en knapp vars funktion både startar och stoppar den nuvarade uppspelningen. Eftersom vi alltid spelar upp hela ultraljudsfilen under en testomgång så behövs ingen pausknapp. Det finns också en förloppsindikator[1] så att användaren kan se när ultraljudet är färdiguppspelad.

[1] "progress bar"

4.1.2 Receiver



Längst ner i Receiver finns en knapp som påbörjar en inspelning via mikrofonen. När man startar inspelningen så blir en timer synlig som visar hur länge den har pågått. För att stoppa inspelningen så trycker man på samma knapp som tidigare. Därefter påbörjas avkodningen av det inkommande ljudet som beskrivs i metodkapitlet.

När processeringen är avklarad så visas datan som extraherats från ultraljudet på den översta delen av sidan i form av en textsträng. Den korrekta testdatan ser ut på följande vis, det vill säga en lista med stigande värden som skickas tre gånger:

$$[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]$$
$$[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]$$
$$[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]$$

Om detta skrivs ut så har kommunikationen lyckats. I vissa fall så lyckas endast en eller två av listorna tolkas av mottagaren och detta antal redovisas senare i testresultaten. Om ingenting tas emot eller lyckats processeras på mottagarsidan skrivs det ut en tom lista, som ser ut på följande vis:

$$[\,]$$

Alternativt kan en lista med felaktiga och slumpmässiga värden skrivas ut. Detta beskrivs nedan i testresultaten som korrupt data och beror framförallt på störningar.

4.1.3 Design

Applikationens designval och upplägg som redovisas i styckena ovan är gjorda från ett främst utilitärt perspektiv. Detta beror på att appens funktion är för att testa ultraljudskommunikation och kommer enbart att användas i utvecklings- och testningssammanhang och inte av några faktiska slutanvändare.

Däremot så implementerades funktioner i applikationen som inte är nödvändiga men som underlättar för användarna. De mest betydelsefulla är de när applikationen ger relevant visuell respons baserat på användarens knapptryckningar. När man genererar en ny WAV-fil så säger appen till när den är färdiggenererad och klar att användas. Det finns en förloppsindiktator och en timer för respektive uppspelningen och inspelningen. De sistnämnda elementen eliminerar att personerna som utför testarna behöver gissa när allt är klart, trots att ett ultraljudstest enbart tar några sekunder att slutföra. Men det gör att nya användare som använder appen förstår vad som pågår utan att till exempel veta i förväg att ett ultraljudsklipp är 5 sekunder långt och tar så lång tid att spela upp.

Färgschemat och fonten är baserad direkt på Sleep Cycles egna design för att hålla en någorlunda enhetlig stil med deras huvudapplikation. Detta var dock inte ett primärfokus i projektet.

4.2 Testresultat

Med hjälp av den färdiga applikationen så utfärdades tester för att utvärdera appens funktionalitet och för att försöka komma fram till optimala värdena för ultraljudskommunikationen. Kriterierna för det sistnämnda var med omtanke på både kommunikationens prestanda samt människors uppfattning av ultraljudet.

4.2.1 Funktionalitetstest

Under den första testomgången låg fokuset på ultraljudskommunikationens funktionalitet. Som det nämndes tidigare så delades testerna upp i två grupper: tester där kommunikationen ska lyckas och där den ska misslyckas. Varje test utfärdades även tre gånger för att säkerställa att resultatet var korrekt.

Tester som ska lyckas			
Test specifikation	**Test 1**	**Test 2**	**Test 3**
Mobiler bredvid varandra	3/3	2/3	3/3
Mobiler, 2 meters avstånd (50% volym)	0/3	0/3	0/3
Mobiler, 2 meters avstånd (80% volym)	4/3*	3/3	3/3
Mobiler, 2 meter, (80% volym), annan testlokal	0/3	korrupt	3/3
Mobiler, 2 meter, (80% volym) + bakgrundsljud	0/3	0/3	korrupt

*Togs emot 4 korrekta meddelanden istället för 3. Troligtvis pga restdata från en tidigare testsession.

Som man tydligt kan se i tabellen ovan så hade testerna blandade resultat. De första tre testerna utfördes i samma lokal och hade främst lyckade resultat. Efter volymhöjningen från 50% till 80% så kunde mobilerna med hög framgång kommunicera.

Därefter flyttades testandet till rummet bredvid och där utfärdades de återstående testerna som vi ville skulle lyckas. Däremot, kunde inte mobilerna kommunicera lika väl som tidigare, trots att de var placerade på samma avstånd, riktning, och volym som i de föregående testerna. Anledningen till detta var ytterst oklar. Rummen var både till synes lika i storlek och det fanns inga märkbara ljudstörningar i det andra rummet som inte också var närvarande i det bredvid. Det enda som skilde rummen åt var att i det andra så fanns det fler gardiner som kan ha påverkat rummets akustik. Detta var endast en teori för tillfället som vi hoppades inte var korrekt. Slutmålet var trots allt att mobilerna skulle kunna kommunicera i ett verkligt sovrum, som i sin tur troligtvis skulle innebära närvaron av allt fler ljudabsorberande material.

För de återstående funktionalitetstesterna byttes testningslokalen till ett riktigt sovrum för att ge mer pålitliga testresultat samt att förhoppningsvis kunna avgöra vilka faktorer som negativt påverkade appens funktionalitet.

Tester som ska lyckas			
Test specifikation	**Test 1**	**Test 2**	**Test 3**
Mobiler bredvid varandra	0/3	0/3	1/3
Mobiler, 2 meters avstånd (50% volym)	0/3	0/3	0/3
Mobiler, 2 meters avstånd (80% volym)	0/3	0/3	0/3
Mobiler, 2 meter, (80% volym) + bakgrundsljud	0/3	0/3	0/3

Tester som inte ska lyckas			
Test specifikation	**Test 1**	**Test 2**	**Test 3**
Genom en stängd dörr	0/3	0/3	0/3
Mobiler i rum bredvid varandra (öppna dörrar)	0/3	0/3	0/3
Mobiler i rum bredvid varandra (stängda dörrar):	0/3	0/3	0/3
Mobiler i stort rum, 15 meters avstånd	0/3	korrupt	0/3

Som man ser i tabellen 'Tester som ska lyckas' så hade testerna negativa resultat och man kunde därmed anta med stor sannolikhet att en lyckad kommunikation var mycket beroende på rummets storlek och akustik.

Det gjordes ett försök att öka amplituden, och därmed ljudstyrkan, för att se ifall det påverkade testresultaten. När man gjorde detta så blev ultraljudet väldigt högljutt och brusigt. För att lista ut varför det ändrades så drastiskt så grafades de olika signalerna för att se ifall det var någon skillnad i ultraljudet eller om det var till exempel, mobilens hårdvarubegränsning som gjorde att ljudet ändrades. Det blev dock tydligt i graferna vad problemet var.

Figure 4.3: Amplitudfaktor = 0,4

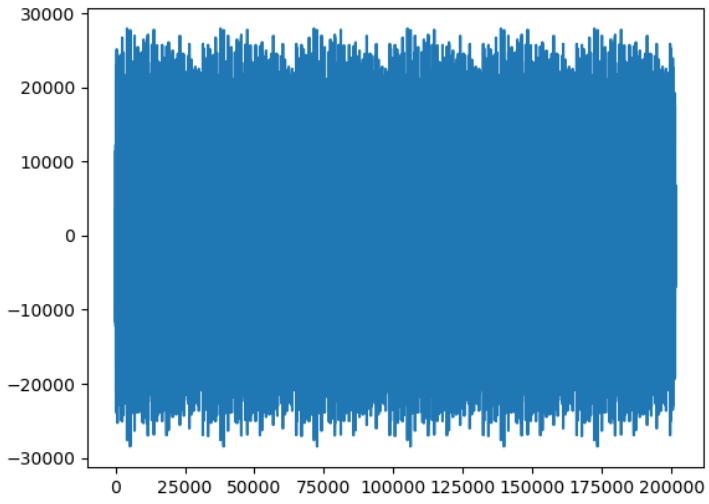


Figure 4.4: Amplitudfaktor = 0,1

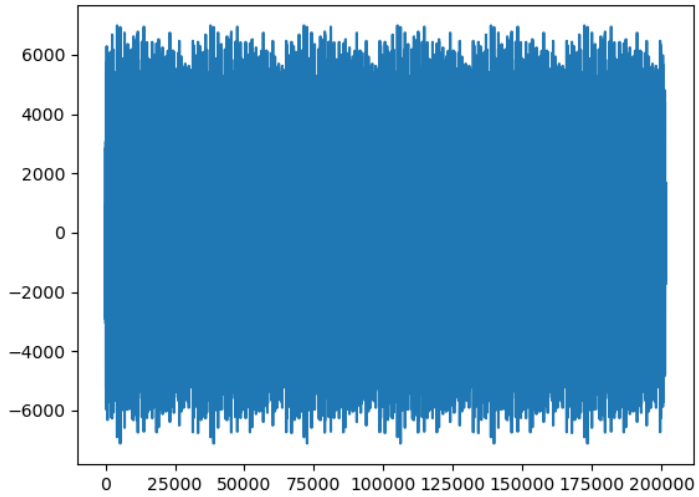
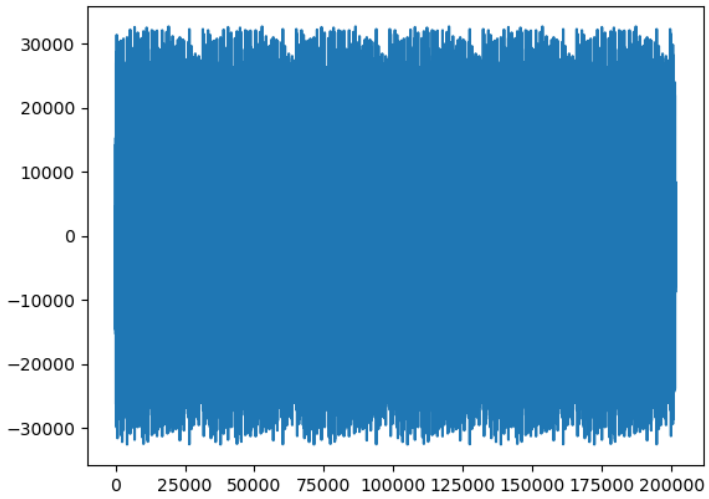


Figure 4.5: Amplitudfaktor = 0,5



Figur 4.1 visar den originella ultraljudsvågen som kommunikationen testades med. Figur 4.2 visar en ny genererad ultraljud när amplitudfaktorn sänks från 0,4 till 0,1. Som man kan se, så har de två ljudvågorna samma form. Den enda skillnaden är det värdena på Y-axeln som representerar amplituden. Därmed innehåller de samma data.

I Figur 4.3, som representerar en ultraljudsvåg genererad med en ökad amplitudfaktor 0,5 så ser man att vågen har en annan form. Värdet på Y-axeln verkar skära av strax efter ±30000 . Anledningen till detta visade sig vara att maxstorleken på en signerad 16-bitars Int är $2^{16}/2$, alltså ±32768, och blir därmed max- och minimumvärdet i Y-axeln. Detta innebär att på grund av Androids upplösningsbegränsningar på WAV-filer så var det därmed omöjligt att öka amplituden för att testa ifall en ökad ljudstyrka hade påverkat kommunikationen.

4.2.2 Hörseltest

Under tiden som funktionalitetstesterna utfördes så blev samtidigt vi som utförde dom utsatta för ultraljudet. Vi är i åldrarna 22 och 24 år och uppfattade inga märkbara ljud. När samma ultraljud spelades upp för yngre personer med bättre hörselsensitivitet [25], ålder 4 och 6, så kunde de inte heller höra något.

Baserat på dessa resultat så kunde man dra slutsatsen att ultraljudet med standardvärdena på både amplituden och frekvensen inte kunde uppfattas av människor.

5

Diskussion

I kapitlena ovan har vi beskrivit med vilka metoder vi uppnått våra resultat. Med hjälp av applikationen som vi utvecklat har vi också verifierat dess prestanda med både funktionalitets- och hörseltester. Dock finns alltid rum för att debattera testernas validitet och därmed projektets slutsatser.

5.0.1 Funktionalitetstester

En av anledningarna till möjligtvis vilja ifrågasätta testresultaten är avsaknaden av tillräckligt varierade testmiljöer och mobilenheter. Det sistnämnda berodde på att det endast fanns tillgång till ett fåtal mobiler, det vill säga, de som ägs av utvecklarna och deras partners. Denna resursbegränsing hade varit lättare att åtgärda om den nuvarande Covid-19-pandemin inte var ett bekymmer och man därmed hade haft tillgång till vänner och bekantas mobiler. I senare vidareutveckling så bör också applikationens funktionalitet testas med fler välkända Androidmobiler som inte är budgetvarianter. Man kan däremot argumentera att om det funkar på mobilenheter som är "sämre" så bör det inte vara ett problem med de återstående.

Som det nämndes tidigare i resultatkapitlet kunde man inte heller dra någon slutsats kring varför kommunikationen fungerade i vissa rum och inte i andra. Det föreslogs att det kunde bero på rummets storlek och akustik men mer noggrant än så kunde inte determineras. I den första testomgången var skillnaderna mellan testlokalerna minimala utöver gardinerna som fanns i rummet där kommunikationen fallerade. På grund av detta grundades teorin att närvaron av ljudabsorberande material dämpade ultraljudet såpass att det inte kunna tas upp av mottagarenheten. Däremot, under en rådspråkning med Mikael Kågebäck, som är mer erfaren med att arbeta med signaler, kom han med argumentet att färre hårda ytor skulle innebära en förbättrad kommunikation. Detta, enligt honom, skulle innebära en minskad risk för reflektioner av ultraljudet som stör och tar ut varandra. I teorin verkar detta rimligt men går emot de resultat som observerades. I framtiden behövs mer utförliga tester som bland annat tar med infallsvinken av ljudkällan i beräkningen för att undersöka vidare kring detta.

5.0.2 Hörseltester

Under projektets gång kunde man inte komma fram till optimala värdena för en fungerande ultraljudsskommunikation. På grund av detta var det också svårt att försöka dra några slutsatser genom att applicera samma värden i hörseltesterna. För ifall inte grundfunktionaliteten finns där, vad spelar det för roll om människor kan höra det eller inte? I ett optimalt sammanhang där det fanns mer tid till vidareutveckling, hade man velat utföra hörseltester med ultraljud som uppfyller funktionalitetskraven till en bättre grad. Därefter skulle man kunna dra slutsatser genom att dra samband till exempel, mellan ålder och amplitud på det uppspelade ultraljudet och se vart gränsen går när de kan höra ljudet. Men på grund av dessa brister nöjde vi oss med att testa ultraljudet med standardvärdena med de yngre deltagarna.

Som det sades i resultatkapitlet, visade det sig att varken de äldre eller yngre hörseltestdeltagarna kunde uppfatta ultraljudet när det spelades upp. Detta gör att det finns möjligheter att sänka frekvensen och öka amplituden på ultraljudet, som troligtvis skulle förbättra funktionaliteten. Det kanske visar sig att de optimala värdena ligger på en nivå som är hörbar för yngre personer. Man kan då argumentera att det är främst vuxna som använder Sleep Cycle-appen, eller som åtminstone är i en levnadssituation som kräver att två mobiler i samma rum behöver kommunicera med varandra. Det är en avvägning man behöver göra i den utvecklingsfasen.

5.0.3 Upplösningsbegränsnigar på WAV-filer

Som beskrivet i resultatkapitlet, så kom det fram till slutsatsen att det inte var möjligt att öka ultraljudets amplitud på grund av Androids upplösningsbegränsning på WAV-filer. En tänkbar lösning för att motverka detta är att använda Android Open Source, som stöttar diverse audiofiler med högre resolution.[23] Detta skulle bland annat innebära i detta projekt att omvandlingen från Float till Int i signalbehandlingskoden inte skulle vara nödvändig. Däremot, så kunde vi inte komma fram till ifall att kommunikation skulle ha bättre framgång även med ökad amplituden, med tanke på andra begränsningar i till exempel mobilens hårdvara.

5.0.4 Ultraljudets frekvensvärde

Hörsel- och funktionalitetstesterna utfördes med ultraljud genererade med de satta standardvärdena. När det visade sig att funktionaliteten inte var pålitlig under alla omständigheter utfördes tester för att undersöka ifall en ökad amplitud skulle påverka kommunikationen. Detta visade sig vara en återvändsgränd som beskrivet tidigare i rapporten. Däremot, på grund av tidsbrist kunde inte projektet utforska huruvida funktionaliteten påverkas av ett förändrat värde på ultraljudets frekvens.

Projektet resulterade i en färdig applikation som kan utföra ultraljudskommunikation under vissa omständigheter och som inte är hörbar för människor. Trots att standardvärdet på frekvensen troligtvis inte är optimal uppfyller den ändå de kriterier som sattes. Så baserat på det faktum kändes det rimligt att påstå att frekvensen är bra nog under omständigheterna. Däremot krävs mer tid för att justera balansen mellan värdet på amplituden och frekvensen för att kunna förbättra funktionaliteten.

5.0.5 Processering

Proceduren att generera ett nytt ultraljud, spela upp ljudet och sedan ta emot det sker enbart på några sekunder. Däremot tar slutprocesseringen av det mottagna ultraljudet ett bra tag att avkoda och skriva ut resultatet. Till exempel, för ett 7 sekunders klipp så tar det ungefär 30 sekunder att avkoda det. Denna avkodningstid har en kvadratisk tillväxt och blir därmed väldigt stor på kort tid. Detta är troligtvis knutet till att avkodningen sker nere i Pythonkoden och därmed gör att det måste ske kommunikationen mellan de olika lagrena i appen som retarderar hela processen. Någon lösning till att förbättra processeringstiderna kom inte fram under projektets gång och är något som bör arbetas vidare med i framtiden.

5.0.6 Mobilspecifika problem

En av de mobiler som användes i testerna är en OnePlus 6T. Mobilen kan utan problem anta rollen som Sender, men kommunikationen lyckas inte när denna är Receiver. Anledningen till detta är oklar men en teori är att mobilen använder fel mikrofon för att spela in det inkommande ultraljudet och därmed drastiskt försämrar ljudkvalitén. Detta är ett känt problem för OnePlus 6 och 6T och påverkar vissa tredjepartsappar.[24]

Anledningen till att det inte fungerar på denna specifika mobil spelar dock inte så stor roll. Detta exempel lyftes fram för att demonstrera de stora skillnaderna som kan finnas mellan två Androidmobiler och vad som man måste ha i åtanke när man utvecklar för dom. Det är lättare att utveckla för iPhones som har någorlunda standardiserade specifikationer än för Android där det finns flera hundratals olika moderna märken och modeller. Speciellt i sådana här fall där hårdvaruspecifikationerna gällande inspelning och uppspelning av ultraljud är kritiska för funktionaliteten.

5.0.7 Etik och hållbarhet

5.0.7.1 Människopåverkan

En av projektets avgränsningar var att det inte skulle innefatta att undersöka om ultraljud påverkar människor i stora drag. Det testades enbart ifall det genererade ultraljudet är hörbart när det spelas upp eller inte. Men eftersom vi valde att utsätta yngre deltagare för hörseltester ville vi vara säkra på att det var säkert att göra det. I detta fall är ultraljuden som spelas upp lågfrekventa och utgör en ytterst liten skaderisk. Däremot, om ljudnivån eller SPL (Sound Pressure Level) blir tillräckligt hög så kan det bli det. WHO (World Health Organization) sade år 1982 att varken temporära eller permanenta hörselskador skulle kunna ske om man är utsatt för ultraljud vid SPL lägre än 120 dB[25]

Nuförtiden så tillverkas det tusentals olika mobiler med diverse specifikationer så det är omöjligt att ange ett exakt värde på maximal ljudnivå för dom alla. Man kan dock generellt säga att de flesta kan komma upp i ljudnivåer som är högre än 100 dB, och iPhones kan komma ända upp i 115 dB.[26] Som man kan se så ligger dessa ljudnivåer under den gräns som WHO nämner är skadlig. Med detta i åtanke kändes det acceptabelt att utföra testerna som planerat.

5.0.7.2 Integritet

När man skickar och tar emot ultraljud finns det en risk för att en tredje part kan vilja komma emellan och sabotera. Då länken är öppen och data som skickas inte är krypterad, är det lätt för motståndare att avlyssna och genomföra "man-in-the-middle"-attacker. För att motverka detta skulle vi vilja att det finns en check för säkerställa att båda mobilenheter som medverkar i kommunikationen är användare av Sleep Cycle. Man kan även se till att den data som skickas via ultraljud är krypterad för att förhindra att den kan tolkas eller emuleras.

5.0.7.3 Hållbarhet

Utifrån vårt perspektiv är den färdiga produkten i detta projekt hållbar främst på grund av det inte är en fysisk produkt som tar på jordens begränsade resurser. I och med att mobiltelefoner och mjukvara förbättras i framtiden, kan också applikationens kod uppdateras så att den uppehåller samma standard.

Man kan argumentera att ultraljudskommunikation drar mer mobilbatteri än vissa andra välanvända kommunikationstekniker. Däremot kan ultraljudskommunikationen ske parallellt med grundfunktionaliteten i Sleep Cycle-appen där den slutligen kommer att användas. Detta innebär att samtidigt som appen lyssnar på ljud från omgivningen för att analysera sömnfaser kan också samma inspelning användas för att ta emot ultraljud. På så sätt, och på säkert flera andra, kan man optimera funktionen så att slutprodukten inte behöver ta mer energi än vad som krävs.

5.0.8 Funktionalitet baserat på användaren

Man kan argumentera att det finns andra och mer tillförlitliga sätt att koppla ihop mobilenheter än den metod som undersöks i detta projekt. Dessa inkluderar bland annat Bluetooth och genom Wi-Fi. Från ett utvecklar- och rent praktiskt perspektiv så kan man tycka att man bör använda någon av dessa, eller flera i kombination ifall någon av dom skulle fallera.

Däremot, så måste man inse att det man utvecklar inte kommer att användas i ett vakuum och enbart i optimala miljöer, utan på en slutprodukt av en användare. Sleep Cycle-appen måste be om tillåtelse att få använda Bluetooth och att söka efter andra enheter på samma nätverk. Många användare blir misstänktsamma och väljer av säkerhetsskäl att inte tillåta dessa trots att sömnanalysens precision påverkas negativt av de valen.

Däremot behöver användaren tillåta appen att ha tillgång till mikrofon och högtalare på grund av appens grundfunktion. Ifall man kunde kombinera en tillåtelse från användaren för multipla funktioner, som med kommunikationen med ultraljud, så blir detta mindre påträngande på användaren. Självklart skulle användaren kunna välja ifall de ska tillåta kommunikation med andra enheter eller inte även via ultraljudskommunikation, men det skulle istället ske utan de påträngande tillåtelsenotiserna från applikationen. Detta är en av anledningarna till att vilja arbeta vidare med ultraljudskommunikation som det gjorts i detta projekt.

6

Slutsats

Under projekts gång har vi kommit till slutsatsen att mobiler har tillräckligt med prestanda för att kunna genomföra den signalbehandling som krävs för ultraljudskommunikation. Som det nämndes i diskussionskapitlet angående processeringstiden så kan en mobil generera och spela upp ultraljud utan större problem. Däremot så tar avkodningen av inspelningen mycket längre tid som också växer kvadratiskt beroende på dess längd. Detta är något som bör förbättras innan funktionen implementeras i en färdig produkt på marknaden.

Baserat på resultaten från funktionalitettesterna som utfördes kan man också dra slutsatsen att under vissa omständigheter så kan två mobiler i samma rum kommunicera med varandra samtidigt som de exkluderar mobiler utanför rummet. Dessa omständigheter för en lyckad kommunikation var mycket beroende på sovrummets storlek och akustik och slutproduktens funktionalitet är inte tillförlitlig i detta skede.

Ljudnivån som krävdes för en lyckad kommunikation var hög och räckte inte till i många fall. På grund av Androids upplösningsbegränsningar på WAV-filer så kan man dessutom inte generera en ultraljudsfil med högre ljudnivå. När medvetna störningar var introducerade i vissa testfall för att efterlikna en verklig miljö, till exempel prassel med täcket, så hade kommunikationen sämre resultat. Så trots slutsatsen från tidigare att mobiler har tillräckligt med processeringskraft att kunna ta emot och tolka ultraljud, så finns det andra begränsningar som förhindrar det till att fungera i praktiken.

Utöver det, så finns det risk att mobilens hårdvaran gällande mikrofon och högtalare inte räcker till. Dessa varierar beroende på enhetens modell och märke och gör att det är svårt att dra en avgörande slutsats kring detta.

Baserat på hörseltesterna som utfördes kan man också dra slutsatsen att människor inte kan höra det ultraljud som används under funktionalitetstesterna. Varken utvecklarna (22 och 24 år) eller de yngre hörseltestdeltagarna (4 och 6 år) kunde uppfatta ultraljudet som spelades upp. Dessa resultat visar att det finns möjligheter att sänka frekvensen och öka amplituden i hopp om att förbättra kommunikationens funktionalitet, samtidigt som det fortfarande är utanför människans hörselspann.

Referenser

[1] Android developer documentation, "AudioManager", developer.android.com https://developer.android.com/reference/android/media/AudioManager #PROPERTY_SUPPORT_MIC_NEAR_ULTRASOUND (accessed Feb. 9, 2021)

[2] IBM Corporation and Microsoft Corporation, "Multimedia Programming Interfaceand Data Specifications 1.0", www-mmsp.ece.mcgill.ca. http://www-mmsp.ece.mcgill.ca/Documents/AudioFormats/WAVE/Docs/riffmci.pdf (accessed Feb. 21, 2021)

[3] Sustainability of Digital Formats: Planning for Library of Congress Collections "Linear Pulse Code Modulated Audio (LPCM)", loc.gov, https://www.loc.gov/preservation/digital/formats/fdd/fdd000011.shtml (accessed Feb. 22, 2021)

[4] Samsung, "What is the difference between Blu-ray Disc, DVD & CD?", samsung.com, https://www.samsung.com/in/support/tv-audio-video/what-is-the-difference-between-blu-ray-disc-dvd-and-cd/ (accessed Mar. 3, 2021)

[5] S. W. Smith, "Digital Signal Processors" in *The Scientist and Engineer's Guide to Digital Signal Processing*, California Technical Publishing, USA, 2002, ISBN 0-9660176-3-3, [Online] Available: http://www.dspguide.com/ch28/4.htm

[6] NumPy, "What is NumPy?", numpy.org. https://numpy.org/doc/stable/user/whatisnumpy.html (accessed Feb. 9, 2021)

[7] SciPy, "Frequently asked questions", scipy.org. https://www.scipy.org/scipylib/faq.html (accessed Feb. 9, 2021)

[8] BLAS ,"BLAS (Basic Linear Algebra Subprograms)", netlib.org http://www.netlib.org/blas/ (accessed Feb. 21, 2021)

[9] Univ. of Tennessee; Univ. of California, Berkeley; Univ. of Colorado Denver, "LAPACK — Linear Algebra PACKage", performance.netlib.org, http://performance.netlib.org/lapack/ (accessed Feb. 21, 2021)

[10] python-for-android, "python-for-android", python-for-android.readthedocs.io, https://python-for-android.readthedocs.io/en/latest/# (accessed Sep. 28, 2020)

[11] python-for-android, "Recipes", python-for-android.readthedocs.io, https://python-for-android.readthedocs.io/en/latest/recipes/ (accessed Feb. 22, 2021)

[12] Chaquo Ltd, "Chaquopy - Python SDK for Android", chaquo.com, https://chaquo.com/chaquopy/ (accessed Oct 1. 2020)

[13] Android developer documentation, "Standalone toolchains (obsolete)", developer.android.com, https://developer.android.com/ndk/guides/standalone_toolchain (accessed Feb. 22, 2021)

[14] SciPy, "scscipy.io.wavfile.write", docs.scipy.org. https://docs.scipy.org/doc/scipy/reference/generated/scipy.io.wavfile.write.html (accessed Dec 1, 2020)

[15] Android developer documentation, "Supported media formats", developer.android.com https://developer.android.com/guide/topics/media/media-formats (accessed Feb. 9, 2021)

[16] Android developer documentation, "MediaPlayer", developer.android.com https://developer.android.com/reference/android/media/MediaPlayer (accessed Feb. 9, 2021)

[17] ExoPlayer, "ExoPlayer", exoplayer.dev https://exoplayer.dev (accessed Dec. 11, 2020)

[18] Android developer documentation, "AudioTrack", developer.android.com https://developer.android.com/reference/android/media/AudioTrack (accessed Dec. 11, 2020)

[19] Android developer documentation, "AudioRecord", developer.android.com https://developer.android.com/reference/android/media/AudioRecord (accessed Feb. 9, 2021)

[20] 'selimbousbih', "how to record .wav format file in android", stackoverflow.com (Jun. 2, 2020) https://stackoverflow.com/questions/5245497/how-to-record-wav-format-file-in-android/62154536#62154536 (accessed Feb. 9, 2021)

[21] Android developer documentation, "Introduction to Activities" developer.android.com, https://developer.android.com/guide/components/activities/intro-activities (accessed 2 Feb, 2021)

[22] Android developer documentation, "Fragments", developer.android.com https://developer.android.com/guide/fragments (accessed 2 Feb, 2021)

[23] Android Open Source Project, "High-Resolution Audio", source.android.com, https://source.android.com/devices/audio/highres-effects (accessed 12 Feb. 2021)

[24] R. Hager, "OnePlus 6 and 6T wrecking audio quality by using the wrong mic in some apps like Snapchat and WhatsApp", Android Police (Jan. 30, 2019) https://www.androidpolice.com/2019/01/30/oneplus-6-and-6t-wrecking-audio-quality-by-using-the-wrong-mic-in-some-apps-like-snapchat-and-whatsapp/ (accessed Jan. 18, 2021)

[25] M. Pawlaczyk-Łuszczyńska, A. Dudarewicz, "Impact of very high-frequency sound and low-frequency ultrasound on people - the current state of the art", Nofer Institute of Occupational Medicine, Łódź, Poland, 2020

[26] V7, "How Loud is Too Loud?" v7world.com, https://www.v7world.com/en/news-events/how_loud_is_too_loud (accessed Feb. 2, 2021)