Akustikk prosjekt

Gruppe 33

Mina Mangseth Svorkmo og Malin Rekdal

25. april 2022

Innhold

1 Innledning **4**

2 Verktøy for estimering av innfallsretning til en lydkilde **6**
2.1 Korrelasjon 6
2.2 Deteksjon av innfallsvinkel 7
2.2.1 Beregning av vinkel 8
2.2.2 Hvorfor minst tre mikrofoner? 10
2.3 ADC og RPi 10
2.4 Lavpassfilter 12
2.5 Utfordringer ved gjennomføring av eksperimentet og bygging av systemet 13

3 Design av målesystem **14**
3.1 Filter 15
3.2 ADC og RPi 16
3.2.1 Oppkobling 16
3.2.2 ADC sampler 19
3.2.3 Oppkobling for test av ADC-ene og RPi-en 19
3.2.4 Analysere dataen 20
3.3 Testoppsett av mikrofoner og lydkilde 20
3.4 Estimere vinkel til innkommende lyd 21

4 Resultater **23**
4.1 Resultat av lavpassfilteret 23
4.2 Resultat av ADC-ene og RPi-en 24
4.3 Resultat av estimering av vinkel 24

5 Diskusjon **29**

6 Konklusjon **31**

Referanser **32**

Appendiks **32**

A ADC sampler **32**

B Raspi analyze lab1 **37**

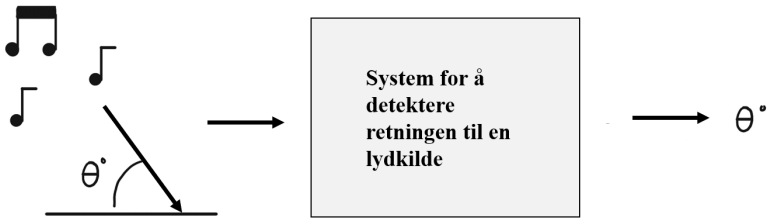
C Raspi analyze lab2 **39**

Sammendrag

Det er designet et system for estimering av innfallsretning til en plan lydbølge i to dimensjoner. Systemet oppfyller i stor grad sin hensikt med å estimere retningen til lydkilden, selv om det forekommer en del usikkerhet i målingene. Av 10 målinger var det 5 målinger som oppfyller at avstanden $\xi < 5°$, mens 7 målinger oppfyller at standardavviket $\sigma < 5°$.

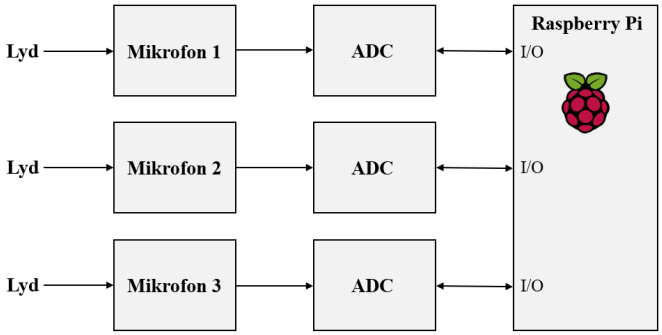
1 Innledning

Lokalisering av lydkilder blir brukt til mye i dagens samfunn. Et eksempel på dette er at anleggsarbeid har en maksgrense for hvor mye støy de kan produsere. I dette tilfellet kan det være vesentlig å kunne estimere hvilken retning lyden kommer fra. Videre vil vi se på en mulig måte å estimere dette, ved å lage et system som vist i figur 1.



Figur 1: Målet for systemet.

For å kunne estimere retningen til en lyd i to dimensjoner, trengs det tre mikrofoner. Målinger fra mikrofonene skal leses av en Raspberry Pi, men for at dette skal kunne skje må det analoge signalet fra hver av de tre mikrofonene sendes gjennom en analog til digital omvandler. Et slikt oppsett vises i figur 2.



Figur 2: Blokkskjema av systemet.

Det å designe et slikt system har en del forutsetninger. Dette blant annet at det er mulig å sende inn en lyd ved en bestemt vinkel for å teste systemet, og at mikrofonene som blir brukt er gode nok til å detektere en lydkilde som er på litt avstand. Dette er viktig fordi lydkilden må være en planbølge for at lyden skal ha samme retning inn til alle mikrofonene. Begrensinger som systemet kan ha er blant annet at det kan fungere dårlig dersom det er mye høyt støy i nærheten av mikrofonene, og at vi har for liten plass til å koble opp systemet slik at mikrofonene blir for nærme hverandre.

Ved å ta forutsetningene og begrensingene i betrakning ønsker vi å detektere et system som klarer å detektere vinkelen til et lydsignal med en feilmargin på treffsikkerheten på max 5°, og en maksbegrensing på tilfeldige feil som tilsvarer et standardavvik på maks 5°.

2 Verktøy for estimering av innfallsretning til en lydkilde

For å estimere retningen til en lydkilde kan man bruke et mikrofonarray og ta krysskorrelasjonen mellom signalene hver mikrofon mottar. På denne måten kan man finne tidsforsinkelsen av lydsignalet mellom hver mikrofon som senere kan benyttes til å beregne retningen til lyden. For å behandle de analoge signalene som mikrofonene mottar må signalene først konverteres til digitale signaler ved hjelp av ADC-er, før de kan mottas av en RPi. Et lavpassfilter kan være nyttig for å filtrere bort støy fra DC-signaler som kan forstyrre systemet.

I seksjon 2.1 til 2.4 går vi nærmere inn på disse temaene for å forklare teori som trengs for å designe de ulike delene i figur 2 gjennomgått. Forkortelser brukt der er vist i tabell 1.

Tabell 1: Forkortelser brukt videre.

Forkortelse	**Betydning**
ADC	Analog til digital omvandler
RPi	Raspberry Pi 3B
DMA	Direct Memory Access
GPIO	General-purpose input/output

2.1 Korrelasjon

Krysskorrelasjon er et mål på hvor like to signaler er. Krysskorrelasjonen mellom to diskrete signaler $x(n)$ og $y(n)$ er gitt ved

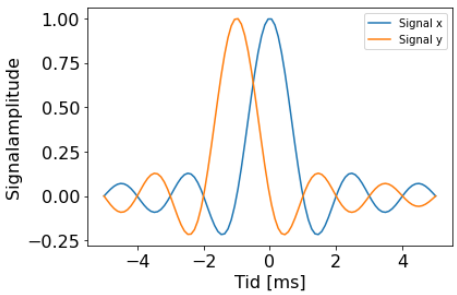
$$r_{xy}(l) = \sum_{n=-\infty}^{\infty} x(n)y(n+l) \tag{1}$$

der l står for det engelske ordet "lag". Man kan finne hvilken tidsforsinkelse, Δt, dette representerer ved hjelp av sammenhengen

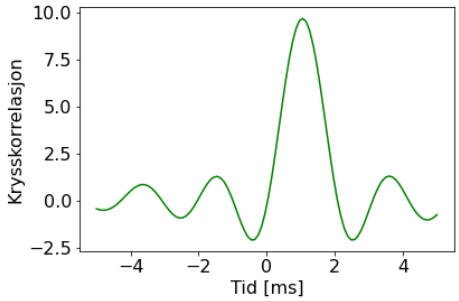
$$\Delta t = \frac{l}{f_s}, \tag{2}$$

der f_s er samplingsfrekvensen. Der korrelasjonsfunksjonen har høyest verdi, er de to signalene mest like, og der den er lav, har de to signalene et stokastisk forhold.

Figur 3a er et eksempel på to identiske signaler som er forskjøvet med 1ms. Denne forskyvingen kan enkelt ses som en topp i plottet av krysskorrelasjonen mellom dem, vist i figur 3b. I denne figuren ser vi at toppen er ved 1ms, som betyr at det er ved denne tidsforsinkelsen signalene er mest like, som stemmer med figur 3a.



(a) Plott av to identiske signaler x og y, der y er forskjøvet med et antall sampler tilsvarende 1ms



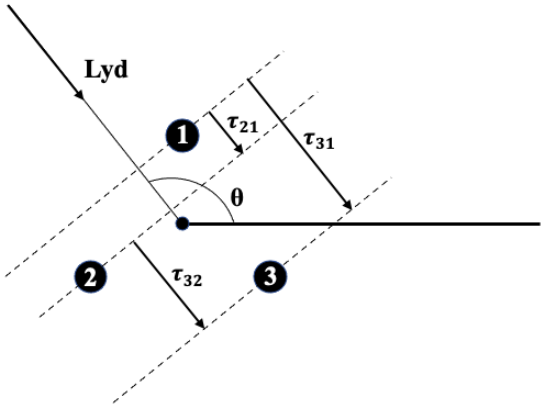
(b) Plott av krysskorrelasjonen mellom signal x og signal y i figur 3a, med en topp ved 1ms

Figur 3: To signaler og krysskorrelasjonen mellom dem.

Når man tar krysskorrelasjonen mellom to periodiske signaler, vil krysskorrelasjonen få flere topper, siden signalene er like ved flere verdier av l. Krysskorrelasjonen av for eksempel hvitt støy vil få en entydig topp ved den aktuelle tidsforsinkelsen og være lav ellers, da de to signalene ellers har et stokastisk forhold.

2.2 Deteksjon av innfallsvinkel

For å estimere hvor en lyd kommer fra i to dimensjoner trengs det minst tre mikrofoner. Med et oppsett med tre mikrofoner plassert i en likesidet trekant, som vist i figur 4, vil en innfallende plan lydbølge ankomme de tre mikrofonene ved forskjellige tidspunkt. Man kan da måle de tre tidsforskjellene τ_{21}, τ_{31} og τ_{32} ved å ta krysskorrelasjon mellom signalene fra mikrofonene, der τ_{ji} er tidsforskjellen fra mikrofon i til mikrofon j. Disse tidsforskjellene kan brukes til å beregne vinkelen til et gitt referansepunkt.



Figur 4: System for retningsbestemmelse av lyd med tre mikrofoner.

2.2.1 Beregning av vinkel

Av trigonometri har vi at

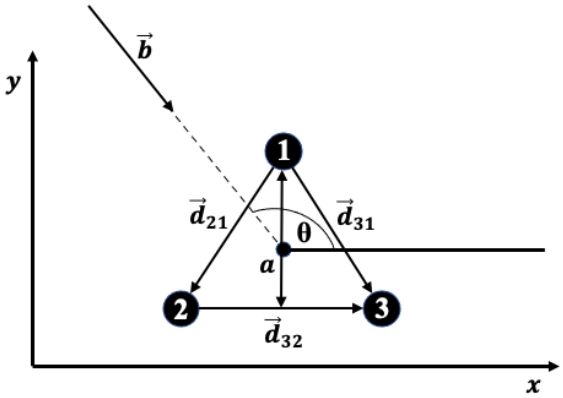
$$\cos\alpha_{ji} = \frac{-c\tau_{ji}}{|\vec{d_{ji}}|}, \qquad ji = 21, 31, 32 \tag{3}$$

der c er lydfarten, $\vec{d_{ji}}$ er avstandsvektorene mellom mikrofonene som vist i figur 5, og α_{ji} er vinkelen mellom $\vec{d_{ji}}$ og den innfallende bølgen $\vec{b}$. Dersom vi kombinerer denne formelen med formel for prikkprodukt mellom to vektorer gitt ved

$$\vec{d_{ji}} \cdot \vec{b} = |\vec{d_{ji}}||\vec{b}|\cos\alpha_{ji}, \tag{4}$$

får vi

$$\vec{d_{ji}} \cdot \vec{b} = |\vec{d_{ji}}||\vec{b}|\frac{-c\tau_{ji}}{|\vec{d_{ji}}|}. \tag{5}$$



Figur 5: Mikrofonoppsett i koordinatsystem med forklaring av størrelser.

Dersom vi velger å definere $|\vec{b}| = 1$ kan vi skrive om ligning 5 til

$$\vec{d_{ji}} \cdot \vec{b} + c\tau_{ji} = 0 \tag{6}$$

Deretter løser vi 6 ved hjelp av minste kvadraters metode og får feilen

$$\varepsilon = \sum_{ji}(\vec{d_{ji}} \cdot \vec{b} + c\tau_{ji})^2 = \sum_{ji}(x_{ji}x + y_{ji}y + c\tau_{ji})^2, \tag{7}$$

der $\vec{d_{ji}}$ dekomponeres til x_{ji} og y_{ji}, som er kjente verdier, og $\vec{b}$ dekomponeres til x og y, som er ukjente.

Videre ønsker vi å bestemme x og y som minimerer denne feilen. Dette kan gjøres ved å derivere ε med hensyn på x og y, og deretter sette uttrykkene lik null. Løsningene for x og y blir da

$$\begin{aligned} x &= \frac{c}{3\sqrt{3}a}(-\tau_{21} + \tau_{31} + 2\tau_{32}) \\ y &= \frac{c}{3a}(\tau_{21} + \tau_{31}). \end{aligned} \tag{8}$$

Videre har vi sammenhengen

$$\theta = \arctan \frac{y}{x}, \tag{9}$$

som gjør at vi enkelt kan finne vinkelen direkte fra x og y. Setter inn 8 og får

$$\theta = \arctan \left(\frac{\frac{c}{3a}(\tau_{21} + \tau_{31})}{\frac{c}{3\sqrt{3}a}(-\tau_{21} + \tau_{31} + 2\tau_{32})} \right) = \arctan \left(\sqrt{3} \frac{\tau_{21} + \tau_{31}}{\tau_{21} - \tau_{31} - 2\tau_{32}} \right) \tag{10}$$

I et slikt system hvor mikrofonene er plassert i en likesidet trekant med sidelengde d, er den maksimale tidsforsinkelsen mellom to mikrofoner gitt ved

$$\tau_{maks} = \frac{d}{c}. \tag{11}$$

Dersom samplingsfrekvensen er f_s, er det maksimale antallet sampler mellom to mikrofoner gitt ved

$$n_{maks} = \tau_{maks} \cdot f_s = \frac{df_s}{c} \tag{12}$$

Det maksimale antallet samplinger mellom to mikrofoner har mye å si for et slik system. Hvis for eksempel $n_{maks} = 2$, vil det være et svært begrenset antall kombinasjoner av de tre n-ene i formel 10, siden de må være heltall lavere eller lik 2. Dette vil dermed gi et svært begrenset antall vinkler θ, som fører til et unøyaktig system. n_{maks} bør derfor være stor.

Når krysskorrelasjon brukes til å finne tidsforsinkelser, vil τ_{ji} beregnes ut fra antall hele sampler n_{ji} gjennom sammenhengen

$$\tau_{ji} = \frac{n_{ji}}{f_s}. \tag{13}$$

Derfor kan man beregne vinkelen direkte ut fra antall samplinger ved å skrive om ligning 10 til

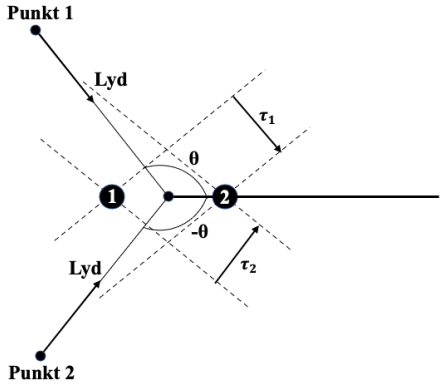
$$\theta = \arctan \left(\sqrt{3} \frac{n_{21} + n_{31}}{n_{21} - n_{31} - 2n_{32}} \right) \tag{14}$$

Siden $\arctan(1/1) = \arctan(-1/-1)$, betyr det at funksjonen arctan kun gir vinkler mellom $-90°$

og 90°. For å kunne beregne alle vinkler mellom 0° og 360° må man i tillegg se på x og y separat, ved å for eksempel si at dersom $x < 0$, så er $\theta = \arctan(x/y) + \pi$. [1]

2.2.2 Hvorfor minst tre mikrofoner?

Dersom man skal lage et system for å estimere hvor en lyd kommer fra ved hjelp av to mikrofoner, får vi systemet vist i figur 6, hvor de sorte sirklene markert med 1 og 2 representerer henholdsvis mikrofon 1 og mikrofon 2. I et slikt system oppstår problemet med at det er umulig å skille om lyden stammer fra punkt 1, plassert med vinkel θ fra referansepunktet, eller punkt 2 med vinkel $-\theta$.



Figur 6: System for retningsbestemmelse av lyd med to mikrofoner.

Om lyden stammer fra punkt 1, ser vi av figuren at tiden det tar fra lyden treffer mikrofon 1 til den treffer mikrofon 2 er τ_1, og for punkt 2 er denne tidsforsinkelsen τ_2. Siden punkt 2 har den samme vinkelen, bare negativ, ser vi at tidsforsinkelsen vil bli den samme uavhengig om lyden stammer fra punkt 1 eller punkt 2, altså er $\tau_1 = \tau_2$. Derfor er det nødvendig med minst tre mikrofoner for å estimere lydretning i to dimensjoner.

2.3 ADC og RPi

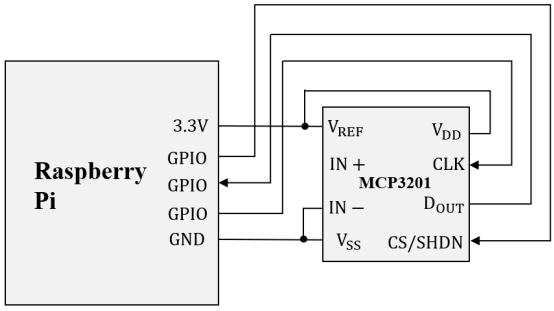
En Raspberry Pi 3B, videre omtalt som RPi, kan brukes gjennom en Pi Wedge tilkobling [2]. For å unngå problem med de logiske nivåene til GPIO pinnene bør man bruke 3.3V spenningsforskyning, og ikke 5V fra RPi-en.

For å kunne analysere signalet mikrofonene estimerer i RPi-en må man først konvertere de analoge signalene til digitale. Dette kan gjøres ved å bruke analoge til digitale omvandlerer, fremover omtalt som ADC-er, av type MCP3201 [3]. Hver av de har en kanal, og ved å kjøre flere slike i parallell kan man sample flere signal om gangen.

MCP3201 har 12 bits oppløsning, og den trenger minst 16 klokkesykluser for å ta en punktprøve og overføre den tilbake til RPi [3, s. 21]. 12 bits oppløsning fører også til at datasignalet blir representert

som 2^{12}=4096 verdier siden ADC en har 12 bits oppløsning. Dette medfører verdier innenfor 0 til 4095. Når man har en en driftspenning på V_{dd} = 3.3V vil ADC-ens oppløsning bli $\frac{V_{ref}}{4095}$ = 0.8mV. Av [3, s. 3] vet vi at pinnene tåler 0.6V til V_{DD} om V_{ss} er jord. De tåler altså et spenn fra V_{ss}-0.6V til V_{dd} + 0.6V.

For at ADC-ene skal fungere må de bli koblet til både forskyningsspenning og jord samt et Chip Select (CS) og Klokkesignal (CLK). Dette kan gjørest ved å bruke RPi med en kobling som vist i figur 7.



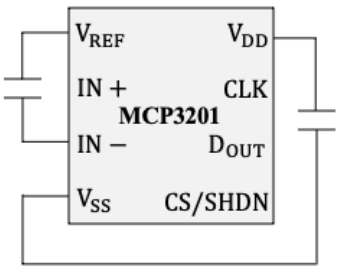
Figur 7: Kobling av RPi mot en ADC.

Her brukes 3.3V fra RPi-en som forskyningsspenning, og koblest til V_{DD} og V_{ref} på ADC-en. GND fra RPi-en koblest til både V_{ss} og IN- på ADC-en. Tre General-purpose input/output, videre omtalt som GPIO pinner fra RPi-en brukest til å ta inn data fra D_{out} fra ADC-en i tillegg til å sende ut CS og CLK signal til ADC-en. Med en slik oppkobling kan man da sende det analoge signalet fra en sensor på IN+ i ADC-en, og man får et digitalt signal ut fra D_{out} som sendest til RPi-en og kan analyserest videre. Denne koblingen er oppsummert i tabell 2.

Tabell 2: Samenkoblinger RPi og ADC.

RPi	**ADC**
3.3V	V_{ref} og V_{DD}
GPIO	CS
GPIO	CLK
GPIO	D_{out}
GND	V_{ss} og IN-

For å kommunisere mellom RPi og ADC er Direct Memory Access (DMA) veldig nyttig. Fordelen med DMA er at det avbelaster operativsystemet ved å fungere som en mellommann. Dette gjør at operativsystemet kan behandle data når det har tid, og det får derfor gjort meir. I systemet vårt



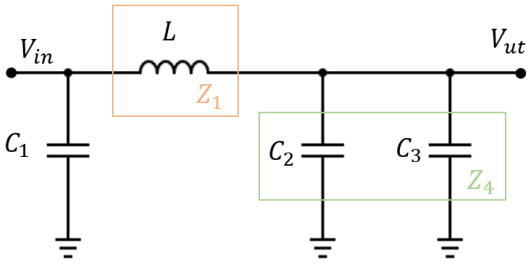
Figur 8: Frakoblingskondensatorer koblet til en ADC.

vil da DMA gjøre slik at vi kan lese GPIO-pinnene mye raskere siden ulike enheter av samme typer kan bli lest og skreve til samtidig.

Frakoblingskondensatorer koblet i paralell i ADC-en som vist i figur 8 er nyttig for å øke stabiliteten og skape mindre støy i forsyningslinjene. De kondensatorene havner i parallell når V_{REF} og V_{DD} kobles sammen, og V_{SS} og $IN+$ kobles sammen.

2.4 Lavpassfilter

Dersom man ønsker å fjerne alle variasjoner i et signal med frekvens over en knekkfrekvens f_c kan man bruke et lavpassfilter som vist i figur 9.



Figur 9: Lavpassfilter for forskyningspenningen.

Her er C_1 og C_2 store stabiliserings kondensatorer, L er en stor spole for undertrykking av høye frekvenser, og C_3 er en liten kondensator for å skape en vei for høye frekvenser.

Vi definerer $Z_1 = jwL$, $Z_2 = \frac{1}{jwC_2}$, $Z_3 = \frac{1}{jwC_3}$ og $Z_4 = Z_2||Z_3 = \frac{z_2 z_3}{z_2+z_3}$ og har av spenningsdeling at

$$V_{ut} = \frac{Z_4 \cdot V_{in}}{Z_4 + Z_1} \tag{15}$$

Da blir amplituderesponsen

$$|H(w)| = |\frac{V_{ut}}{V_{in}}| = |\frac{Z_4}{Z_4 + Z_1}| = |\frac{1}{1 - w^2 \cdot L(C_2 + C_3)}| \tag{16}$$

Vi vet at knekkfrekvensen til et lavpassfilter er etter en nedgang på -3dB. Dette tilsvarer en minking på $10^{-\frac{3}{20}} = \frac{1}{\sqrt{2}}$ i amplituderesponsen. Knekkfrekvensen er derfor frekvensen som gir

$$|H(w)| = |\frac{1}{1 - w_c^2 \cdot L(C_2 + C_3)}| = \frac{1}{\sqrt{2}} \tag{17}$$

Her finner man $w_c = 2 \cdot \pi f_c$, og derfor også knekkfrekvensen f_c ved å løyse ut.

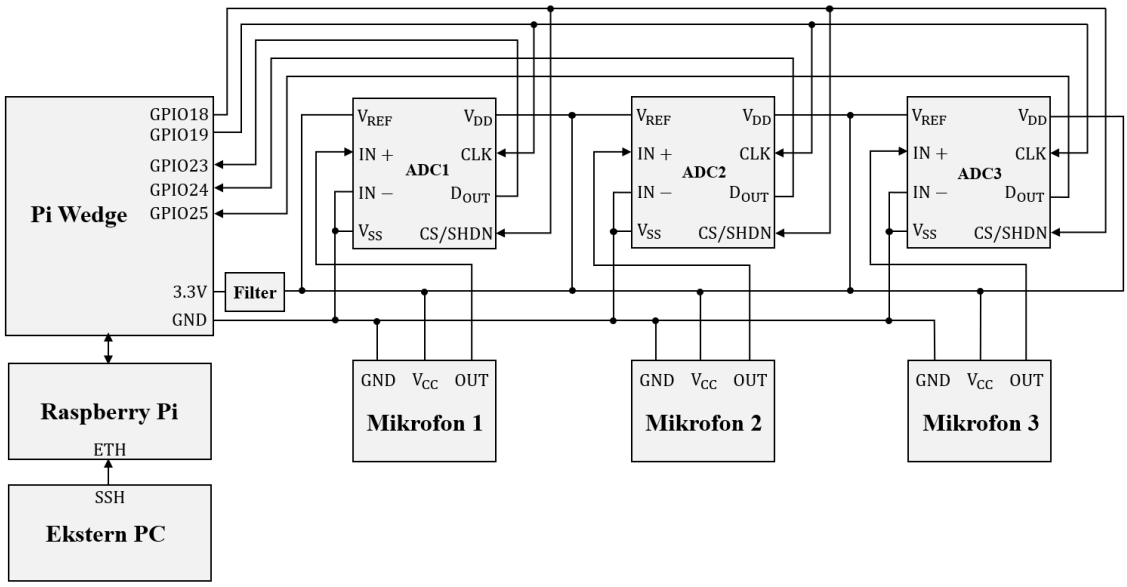
2.5 Utfordringer ved gjennomføring av eksperimentet og bygging av systemet

Ved implementering og testing av systemet er det flere mulige utfordringer som kan forekomme. Noen av desse er nemnt i seksjon 1. I tillegg til dette er det nemnt i seksjon 2.2.1 at en for lav verdi på n_{max} vil gjøre at vi ikke klarer å detektere så mange vinkler.

Uten om dette kan det også bli et problem at vi får dårlig plass på oppkoblingsbrettet som kan føre til mange kortslutninger og problem som er vanskelige å detektere. Liten plass kan også føre til at det blir for liten avstand mellom mikrofonene, som gjør at tidsforsinkelsene mellom dem blir svært små, og dermed mindre nøyaktige. Det kan også bli en utfordring under gjennomføringen av eksperimentet å ta målinger fra ulike vinkler da vi har en del andre komponenter rundt mikrofonene som kan komme i veie for målingene.

3 Design av målesystem

For å lage et system som kan estimere vinkelen til et innkommende lydsignal så bruker vi målesystemet vist i figur 10.



Figur 10: Blokkdiagram av hele systemet.

I dette målesystemet blir en ekstern PC brukt til å styre RPi-en gjennom SSH. Denne koblingen gjørest ved hjelp av en Ethernet kabel. Videre koblest RPi-en opp til en Pi Wedge som gjer at vi lett kan nå tak i pinnene på et oppkoblingsbrett. Fra Pi Wedgen blir 3.3V sendt gjennom et filter for å filtrere støy fra forskyningsspenningen. Dette blir forklart nærmere i seksjon 3.1.

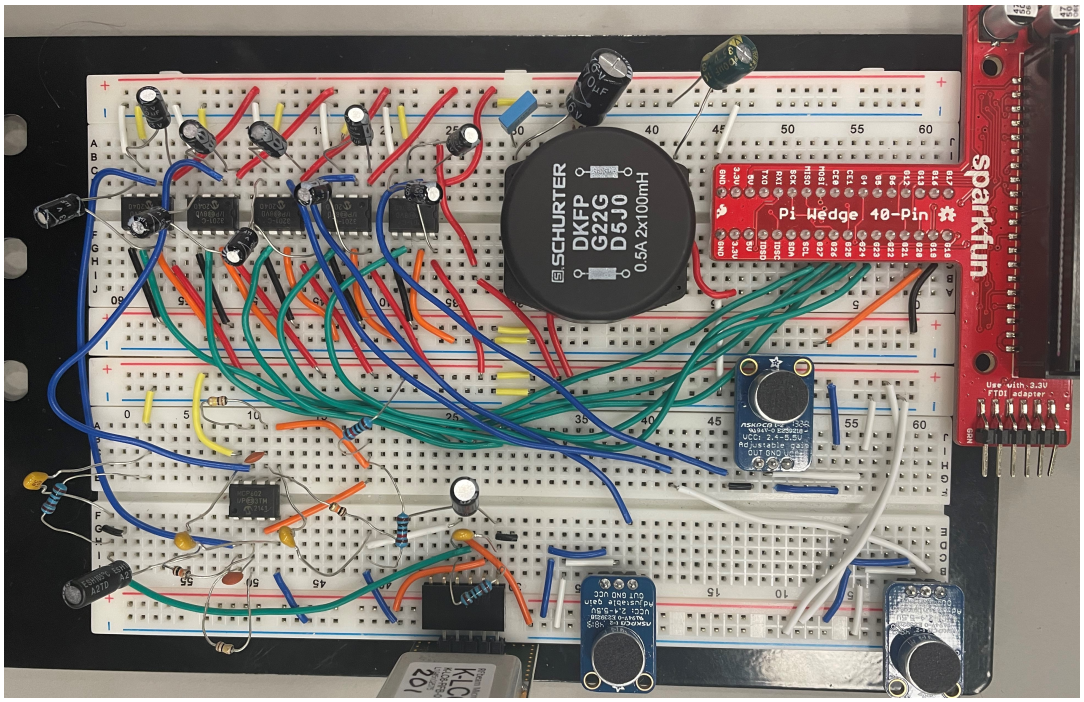
Videre blir kvar ADC koblet opp som omtalt i seksjon 2.3 og vist i figur 7. I dette spesifikke tilfellet blir GPIO18 koblet til CS, og GPIO19 koblet til CLK for alle ADC-ene. GPIO23-25 blir koblet til hver av de 3 ADC-ene sine D_{out}. Mer detaljer om koblingene av ADC-ene blir forklart i seksjon 3.2.

Videre er tre mikrofoner koblet til. Dette er nødvendig på bakgrunn av forklaringen i seksjon 2.2.2. V_{cc} er koblet til den filtrerte 3.3V spenningen, og GND er koblet til GND fra Pi Wedgen. Mikrofon 1 er koblet til IN+ på ADC1 og D_{out} fra denne går videre til GPIO23 på Pi Wedgen. Dette gjør at Mikrofon 1 sitt digitale signal blir omformet i ADC1 og sendt videre til input pinnen GPIO23. Tilsvarende er for Mikrofon 2 sitt OUT signal som går gjennom ADC2 til GPIO24 og Mikrofon 3 sitt OUT signal som går gjennom ADC3 og til GPIO25. Dette er oppsummert i tabell 3. Mer detaljer om oppkobling av mikrofonene blir forklart i seksjon 3.3.

Den fysiske oppkoblingen av dette systemet vises i figur 11.

Tabell 3: Samenkoblinger mellom RPi, ADC-er og mikrofonene.

Fra Mikrofon	Inn ADC	Ut ADC	Til Pi Wedge
Mikrofon 1: OUT	ADC1: IN+	ADC1: D_{out}	GPIO23
Mikrofon 2: OUT	ADC2: IN+	ADC2: D_{out}	GPIO24
Mikrofon 3: OUT	ADC3: IN+	ADC3: D_{out}	GPIO25



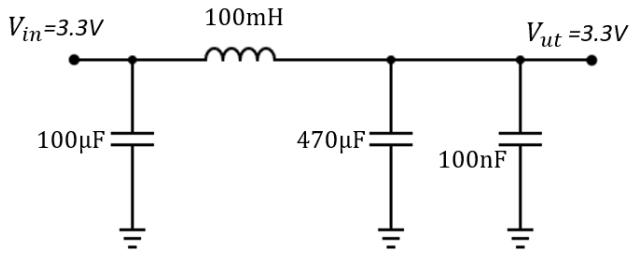
Figur 11: Fysisk oppkobling av system for å estimere vinkel på innkommende lyd.

Etter den fysiske koblingen er koblet opp trenger vi også et system for å hente og analysere data-en som kommer inn fra mikrofonene. Dette blir gjort med koden som omtalt i seksjon 3.2.2 og seksjon 3.4.

3.1 Filter

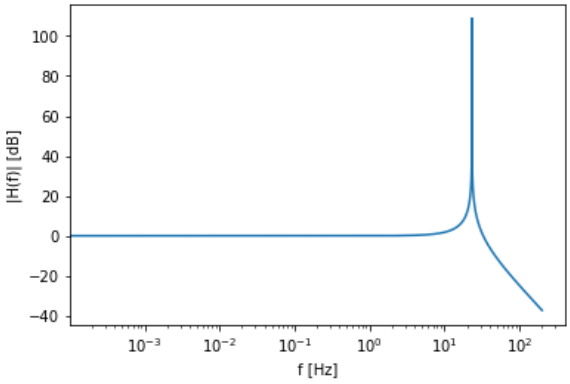
Som vist i figur 10 har vi et filter på utgangen av 3.3V fra RPi-en. Dette er et filter som omtalt i seksjon 2.4 som fjerner alle variasjoner i signalet over en knekkfrekvens f_c. Dette lavpassfilteret trengs fordi 3.3V spenningsforskyningne på RPi-en kan gi en del støy. Dette er hensiktsmessig å filtrere vekk da de analoge komponenter, som for eksempel mikrofonene ikke takler så mye støy i forsyningen.

Siden vi ønsker en DC spenning er det gunstig å ha et lavpassfilter med en lav knekkfrekvens. Vi ønsker derfor å designe et filter som vertfall filtrerer vekk frekvenser over 50 Hz. Får at $C_1 = 100$F, $C_2 = 470$F, $C_3 = 100$nF og $L = 100$mH gir knekkfrekvens $f_c = 36$Hz. Filteret i figur 12 vil derfor filtrere vekk støyet fra forskyningsspenningen på over 50Hz.



Figur 12: Lavpassfilteret med passende verdier.

Amplituderesponsen i ligning 16 er vist i figur 13.



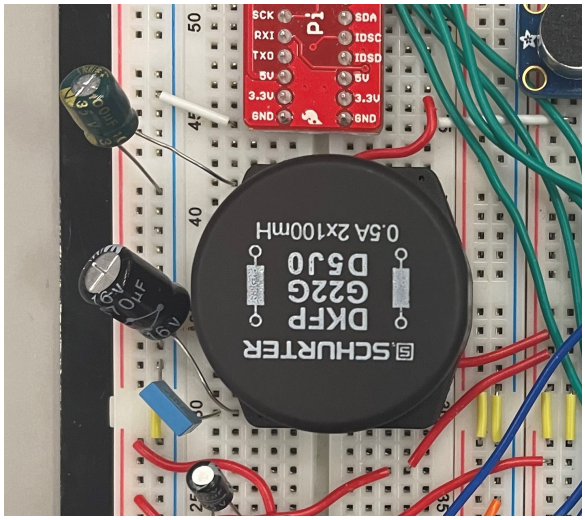
Figur 13: Amplituderesponsen for lavpassfilteret.

En fysisk oppkobling av lavpassfilteret er vist i figur 14.

3.2 ADC og RPi

3.2.1 Oppkobling

Vi koblet opp tre MCP3201 ADC-er på samme måte som koblingen omtalt i seksjon 3, bare uten mikrofonene koblet til IN+ inngangen. Her fungerer Pi Wedge, RPi, filteret og den eksterne pc-en på samme måte. I tillegg til dette ble to ADC-er til koblet opp på samme måte, og de bruker GPIO26 og GPIO27 for sine outputs.



Figur 14: Fysisk oppkobling av lavpassfilteret.

Alle fem har da et felles signal for klokke (CLK) og chip select (CS). I tillegg til dette har de fem seperate signal for å returnere data til RPi.

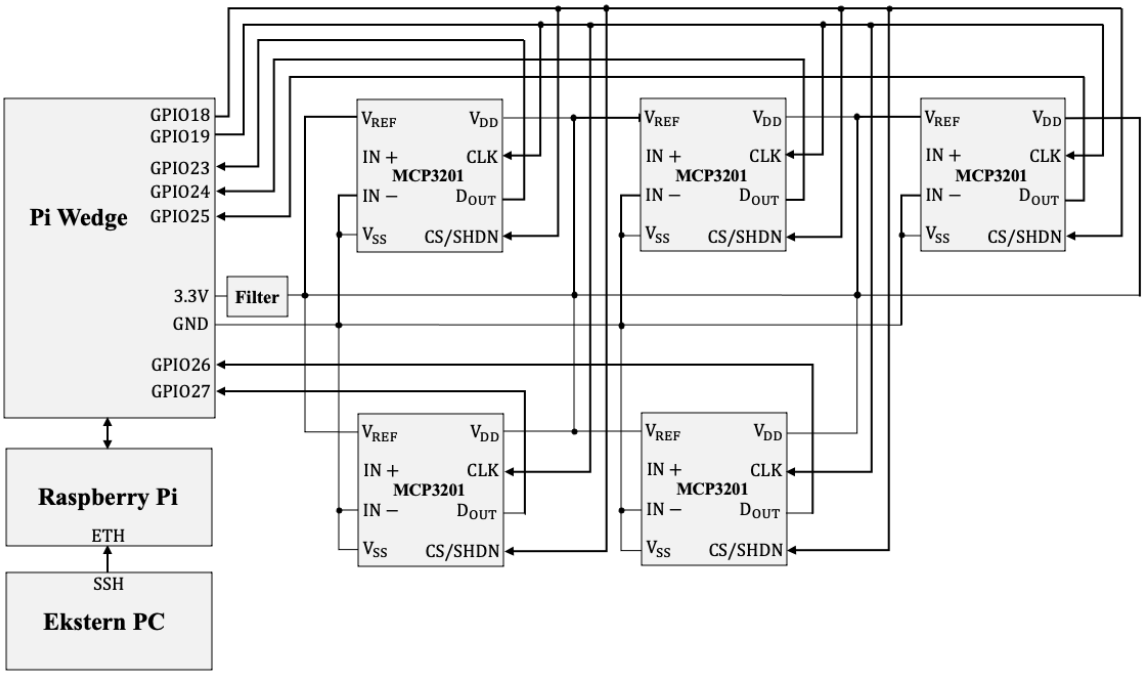
Denne oppkoblingen med 5 ADC-er er vist i figur 15.

Denne oppkoblingen gjør at systemet kan sample 5 kanaler samtidig. En oppsummering av koblingen er vist i tabell 4. I tillegg til å koble opp disse pinnene fysisk blir de definert i en kode som omtalt i seksjon 3.2.2.

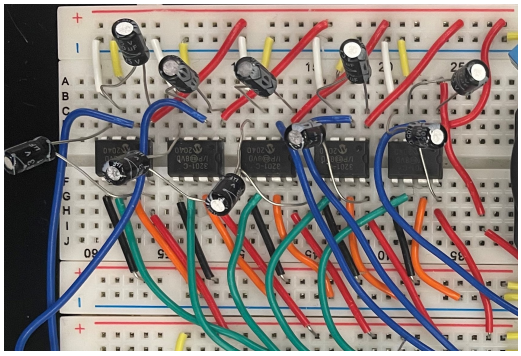
Tabell 4: Samenkoblinger mellom ADC og RPi.

RPi	**ADC**
GPIO 18	CS
GPIO 19	CLK
GPIO 23-27	D_{out} av ADC ene
GND	V_{ss} og IN-
3.3V	V_{DD} og V_{ref}

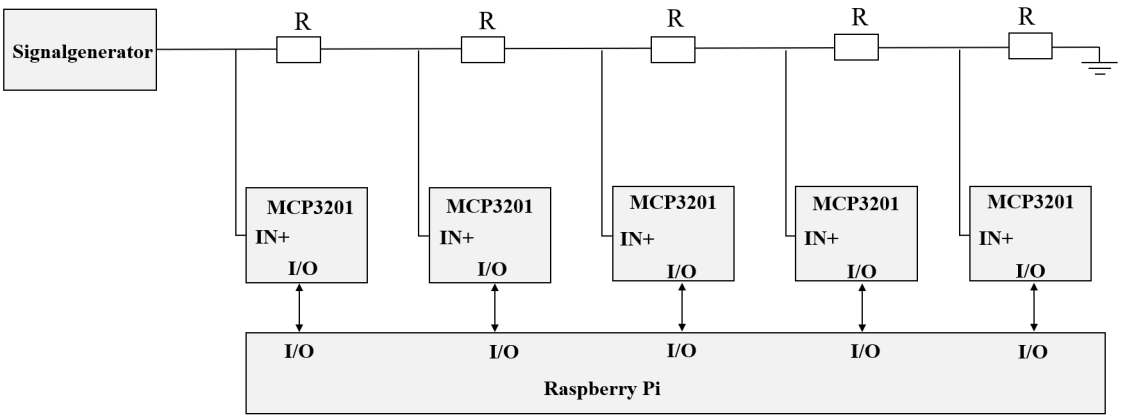
Som nemnt i seksjon 2.3 er frakoblingskondensatorer lurt å bruke. Vi bruker 1F kondensator mellom V_{dd} og V_{ss} og 10F kondensator mellom V_{ref} og IN- på ADC en. Dette blir så koblet opp for alle ADC-ene som vist i figur 8. Siden V_{ss} og IN- er koblet sammen, og V_{ref} og V_{DD} er koblet sammen i skjemaet i figur 15 vil de kondensatorene være i paralell. En fysisk oppkobling av frakoblingskondensatorene med ADC-ene er vist i figur 16.



Figur 15: Lavpassfilteret med passende verdier.



Figur 16: Fysisk oppkobling av frakoblingskondensatorene og ADC-ene.



Figur 17: Oppsett for test av ADC-er.

For å kommunisere mellom RPi og ADC-ene ved hjelp av DMA brukest pigpio [4]. Dette er et C-bibliotek som gjør at vi kan kommunisere ved hjelp av DMA.

3.2.2 ADC sampler

For å kunne studere verdiene samt bruke systemet som vist i figur 15 trenger vi et system som leser verdiene på D_{out} fra ADC-ene. For dette brukes koden som ligger i vedlegg A. Denne leser flere ADC-er i paralell og skriver dataen til en binærfil som videre kan brukest til analysering. Denne blir brukt til å ta inn testsignalene og mikrofonsignalene og analysere de som forklart i henholdsvis seksjon 3.2.4 og seksjon 3.4.

I denne koden blir pinsa vi bruker fra RPi-en definert slik at MISO1-5 er definert som henholdsvis verdiene 23-27 samt at CLK er definert som 19 og SPI_{SS} som tilsvarer CS er definert som 18. Dette er definisjoner som samsvarer med samenkoblingen mellom RPi og ADC-ene som omtalt i tabell 4.

Under kjøring tar denne inn antall samples og filnamnet vi vil lagre de binære verdiene i. I testingen brukte vi 312500 samples som input.

3.2.3 Oppkobling for test av ADC-ene og RPi-en

For å teste ADC-ene ønsker vi å sende inn signal på IN+ på de ulike ADC ene for figur 15 for så å studere resultatet vi får inn til RPi-en gjennom D_{out} pinnene. Vi ønsker ulike spenninger for å lettere se de ulike signalene i et felles plott. For å sette ulike spenninger på de ulike IN+ inngangene på ADC-ene sender vi inn et sinussignal med en signalgenerator og bruker spenningsdeling med motstander for å gi ulike amplituder inn på hver av de 5 ADC-ene. Et slikt oppstet er vist i figur 17. Her er I/O koblingene mellom RPi-en og ADC-ene lik som i figur 15.

Vi valte R = 470Ω til å være lik for alle motstandene for å få et likt forhold mellom signalene. Vi sendte så inn et sinussignal fra signalgeneratoren som varierte med frekvens på 1kHz mellom 0V og 2V. Spenningsdelingen med motstandene vil da fordele dette til ulike sinussignal med lik frekvens, men varierende amplitude.

3.2.4 Analysere dataen

For å faktisk teste systemet må ADC-sampler som forklart i seksjon 3.2.2 kjørest. Da blir systemet kjørt, og vi får verdiene av signalene lagret i en binærfil. Denne binærfilen må behandles og resultatene må plottes.

En viktig del av behandlingen er å gjøre om til analoge verdier igjen. Som omtalt i seksjon 2.3 er oppløsningen til ADC-en 0.8mV. For å finne den analoge verdien etter omgjeringa bruker vi derfor formelen

$$\text{analog data} = \text{digital data} \cdot 0.8 \cdot 10^{-3} V \tag{18}$$

I tillegg til å synliggjøre selve signalet ønsker vi å studere frekvensspekteret for å se at den frekvensen vi sender inn er den frekvensen systemet oppfatter. For å gjøre dette blir fft brukt på inn signalet som blir kalt data som vist i kodesnutten under.

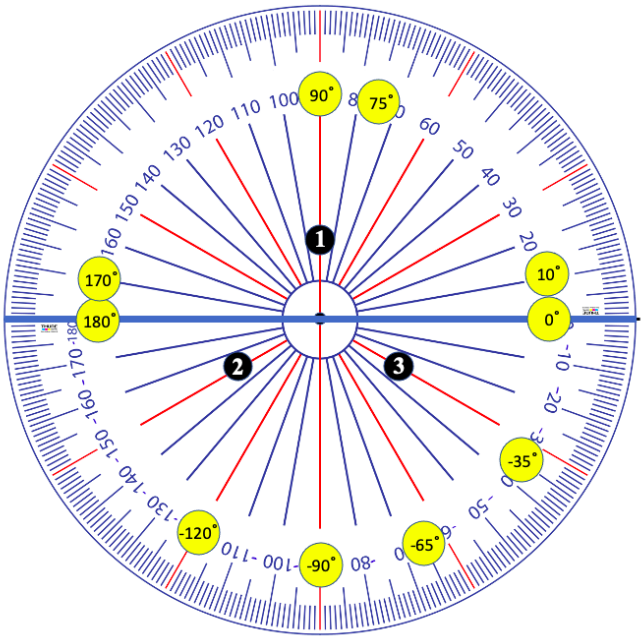
```
freq = np.fft.fftfreq(n=num_of_samples, d=sample_period) # Generate frequency axis
spectrum = np.fft.fft(data, axis=0)  # takes FFT of all channels
```

Den totale koden for denne analyseringen er vist i appendiks B.

3.3 Testoppsett av mikrofoner og lydkilde

Det er viktig at avstanden mellom mikrofonene ikke blir for liten, da dette fører til mindre tidsforskjeller, som kan resultere i et unøyaktig system som beskrevet i seksjon 2.2.1. Vi plasserte derfor mikrofonene i en likesidet trekant som vist i figur 18 med avstanden 5.6cm fra hverandre, som var det største vi fikk til på oppkoblingsbrettet. Vi gjorde målinger fra vinklene $-120°$, $-90°$, $-65°$, $-35°$, $0°$, $10°$, $75°$, $90°$, $170°$ og $180°$. De gule punktene i figur 18 representerer punktene vi sendte lyd fra, med gitt vinkel til referansepunktet. Lyden ble sendt omtrent 15 cm fra sentrum av mikrofontrekanten.

På grunn av teorien forklart i seksjon 2.1, bør ikke lyden vi tester systemet med være periodisk da det kan gjøre det vanskeligere å finne tidsforskjellen siden den ikke skiller seg ut som en enkel topp i krysskorrelasjonen. Vi velger derfor å teste systemet med hvitt støy som lydkilde.



Figur 18: Måleoppsett av vinklene til lydkilden.

3.4 Estimere vinkel til innkommende lyd

Koden kjøres og analyserest på samme måte som for å teste ADC-ene som omtalt i seksjon 3.2.4. I tillegg til å skalere enheten til analoge verdier må vi her også ta vekk DC komponenten i signalet. Dette gjørest ved å bruke koden under.

```
data = signal.detrend(data, axis=0)
```

Vi har også valgt og oppsample signalet med en faktor 4. Dette er på bakgrunn av at vi ser av ligning 12 i seksjon 2.2.1 at n_{maks} er direkte avhengig av samplingsfrekvensen f_s, og at en høyere f_c derfor gir en mer nøyaktig vinkel. For stor oppsampling skaper unødvendig mange punkt for signalet, og vi såg derfor med testing at en oppsamplingsfaktor på 4 ville fungere greit.

Etter oppsamplinga må vi lage en ny tidsakse samt tilpasse signalene med den nye oppsampla aksen. Tidsaksen blir lagd ved hjelp av numpy sin funkson linspace, mens numpy sin interp funksjon blir brukt for å øke antall sample punkt i signalet. Dette er vist i kodesnutten under for de tre signalene $s_i = s_1$, s_2 og s_3, som kommer fra de tre ulike mikrofonene.

```
t_upsampled = np.linspace(start=0, stop=num_of_samples*sample_period, num=
num_of_samples*upsampling_factor)
s_k = np.interp(t_upsampled, t, s_k)
```

Frekvensspekteret blir også funnet på samme måte som omtalt i seksjon 3.2.4. Målet er å finne

vinkelen til den innkommende lyden. For å gjøre dette må vi finne krysskorrelasjonen til signalene fra de ulike mikrofonene, og videre finne maksverdiene av de som tilsvarer en tidsforsinkelse mellom signalene. Dette gjørest med koden vist under for ij = 21, 31 og 32.

```
corr_ij = signal.correlate(si, sj, mode='same')
n_ij = corr_ax[np.where(corr_ij == max(abs(corr_ij)))[0][0]]
```

Her er corr_{ij} korrelasjonen mellom signalet til mikrofon i og signalet til mikrofon j, mens corr_{ax} er en akse sentrert rundt null med lik lengde som korrelasjonssignalene. Det blir brukt mode='same' som argument, som betyr at aksen til corr_{ij} blir av samme størrelse som s_i, men sentrert med hensyn til heile corr_{ij}. Dette går fint uten å miste noe informasjon fordi vi ønsker å se når to signaler er like, og vi trenger derfor ikke hele signalet uansett. n_{ij} er tidsforsinkelsen i antall samples. Som nemnt i ligning 13 i seksjon 2.2.1 er denne direkte proposjonal med tidsforsinkelsen τ_{ij}.

Når vi da har n_{21}, n_{31} og n_{32} kan vi bruke dette videre til å finne vinkelen. Dette gjørest med å bruke koden under.

```
theta = np.arctan2(np.sqrt(3)*(n_21+n_31),(n_31-2*n_32))
```

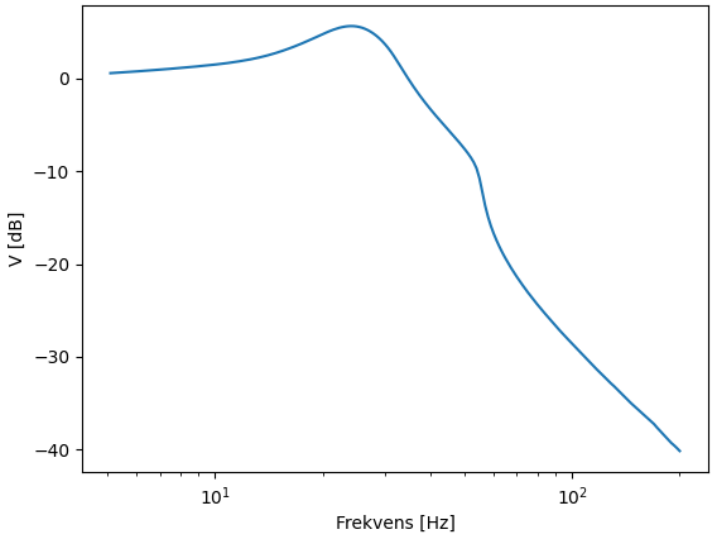
Her er formelen i ligning 10 fra seksjon 2.2.1 brukt. I seksjon 2.2.1 ble det omtalt at man må være påpasselig ved bruk av arctan for å få rett vinkel. Vi har derfor gjort et valg om å bruke arctan2 for å løyse dette problemet da denne velger rett kvadrant automatisk. På denne måten har man da den innfallende vinkelen systemet måler. I tillegg til å finne vinkelen ønsker vi å finne autokorrelasjonen til mikrofonsignalene for å se om makspunktet er i 0. Dette for å lett kunne se om systemet fungerer. Dette gjørest med samme korrelasjonsfunksjonen som definert tidligere, berre at vi tar inn samme signalet to gonger.

Den totale koden brukt for å finne den innkommende vinkelen er vist i appendiks C.

4 Resultater

4.1 Resultat av lavpassfilteret

Ideelt sett ønsket vi et lavpassfilter med amplituderespons som vist i figur 13. Dette har teoretisk knekkfrekvens $f_c = 36$Hz. Når vi realiserte filteret i figur 12 blir den realiserte amplituderesponsen som vist i figur 19.



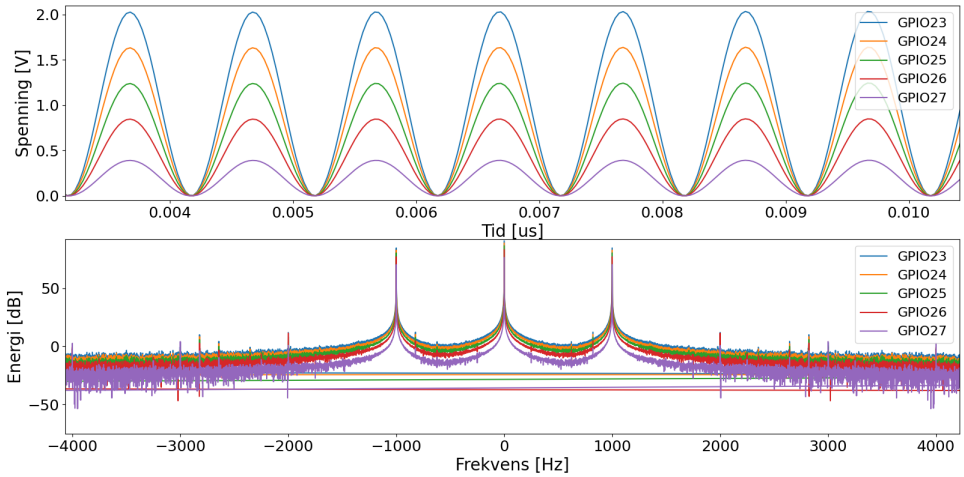
Figur 19: Amplituderesponsen til det realiserte filteret.

Dette har en knekkfrekvens på 39Hz, noe som er ganske nærme den ønska verdien. Det filtrerer greit vekk verdier rett over knekkfrekvensen, men spesielt ved frekvenser over 50Hz fungerer det bra. Dette betyr at det fungerer godt for vårt formål, som var å filtrere vekk støy fra forsyningsspenningen på 3.3V fra RPi en som hadde frekvenser over 50Hz.

Selv om det fungerer godt så ser vi at det er noen forskjeller mellom det teoretiske og det realiserte filteret. Den største forskjellen er at det teoretiske filteret har en topp ved omtrent 20 Hz. Denne er litt mindre i det realiserte filteret, og etter dette synker den fysiske amplituderesponsen ikke like jemnt som den teoretiske. Begge desse forskjellene skyldes fysiske begrensinger som ikke blir tatt hensyn til i teorien. Begge plottene går mot -40dB ved omlag 100Hz, og også før dette har de en omtrent lik demping. Oppsummert har de lik funksjonalitet, men har litt ulik form på grunn av realiseringen, og det realiserte filteret vil fungere godt til formålet vi har.

4.2 Resultat av ADC-ene og RPi-en

For å teste ADC-ene og RPi-en sin oppkobling blir systemet forklart i seksjon 3.2.3 brukt og informasjonen blir henta ut ved å bruke adc-sampler omtalt i seksjon 3.2.2. Dette gir signalet gitt i tidsdomenet og frekvensspekteret som vist i figur 20.



Figur 20: Resultat av sinussignal sendt inn i ADC ene vist som signal i tidsdomenet og frekvensspektrum.

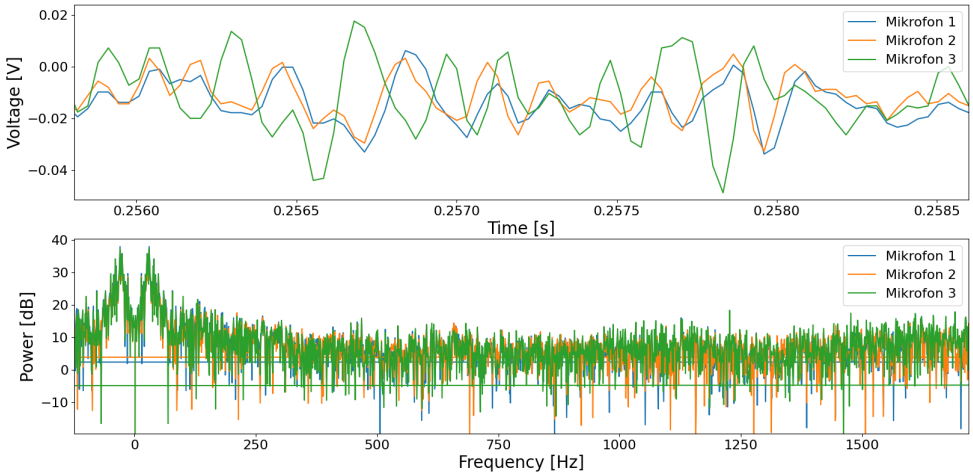
Signalet i tidsdomenet viser at vi får inn de ønsket signalene etter spenningsdelingen. Frekvensresponsen viser at vi har lite støy, samt 1kHz frekvens i alle signalene, som var det vi sendte inn. Systemet har derfor sendt gjennom signalet som ønsket.

4.3 Resultat av estimering av vinkel

Vi tester først systemet ved å sende hvit støy inn på mikrofonene fra en vinkel på -35°. Når vi har analysert dataen får vi signalene som vist i figur 21.

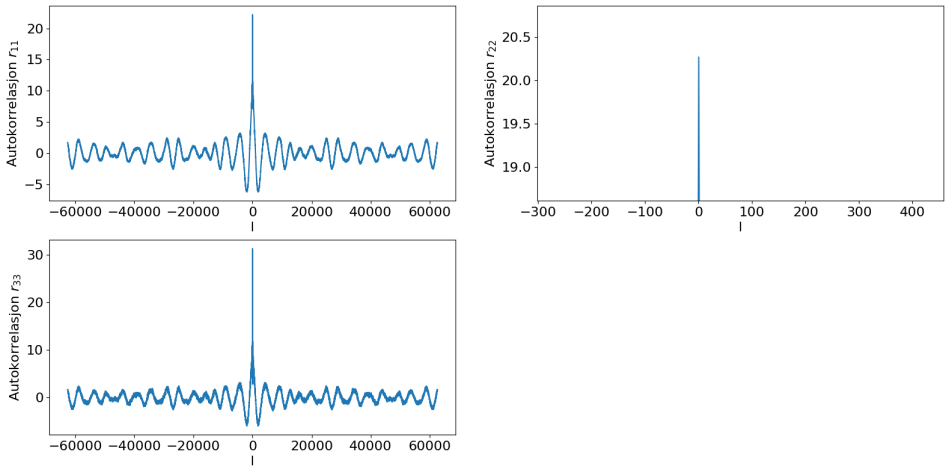
Her ser vi tydelig at lyden treffer mikrofon 3 først, så mikrofon 1 og 2 omtrent likt. Dette gir mening ut i fra at en vinkel på -35° blir sendt inn, når definisjonen av vinklene i forhold til mikrofonene er som gjort i figur 18. Vi ser også at signalet varierer rundt omtrent 0V, som betyr at fjerningen av DC spenningen med detrend som forklart i seksjon 3.4 fungerte som ønsket.

I frekvensspekteret ser vi at vi har mye bidrag av ulike frekvenser, som gir mening siden vi sendte inn et hvitt støy. Vi tar gjennomsnittet av støygulvet fra 500 til 1500 Hz, og gjennomsnittet av området rundt frekvensen med den høyeste toppen. Får da at signal til støyforholdet, SNR = 35dB - 6dB = 29dB. Dette betyr at signalet har et vesentlig større bidrag enn støyet.



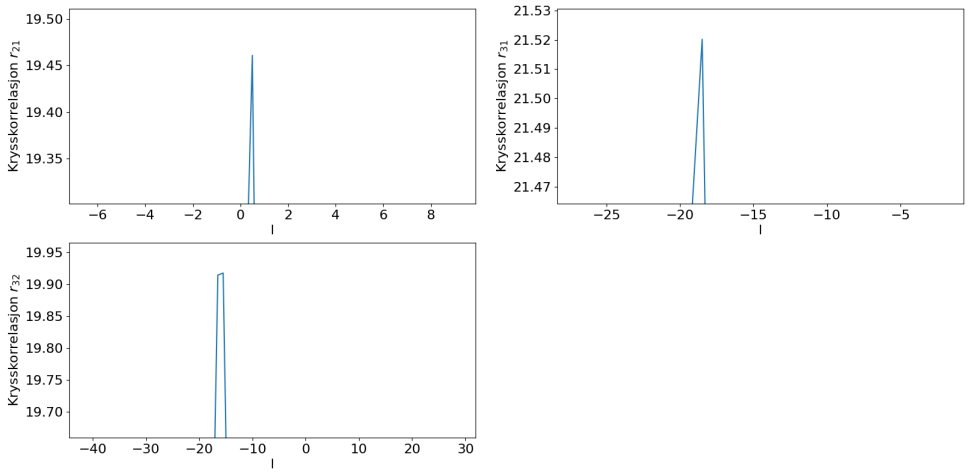
Figur 21: Signalene fra mikrofonene ved vinkel på -35°.

Ved å plotte autokorrelasjonen av de tre signalene ved -35° får vi resultatet i figur 22.



Figur 22: Autokorrelasjon mellom signalene fra mikrofonene ved vinkel på -35°.

Her er plottet for r_{11} fokusert nerme l = 0, som er for 0 lags. Vi ser da at den er sentrert rundt null. Dette betyr at korrelasjonsfunksjonen fungerer da dette er som forventa for en autokorrelasjon. Videre plotter vi korrelasjonen mellom de tre signalene og får r_{21}, r_{31} og r_{32} som vist i figur 23.



Figur 23: Krysskorrelasjon mellom signalene fra mikrofonene ved vinkel på -35° med fokus på maks-punktet.

Her har vi fokusert nærmere rundt den verdien for l som gir maksverdien til krysskorrelasjonen. Vi finner da at vi har n_{21} = 0.5 samples, n_{31}=-18.5 samples og n_{32} =-15.5 samples. Dette tilsvarer en tidsforsinkelse på τ_{21} = 4μs, τ_{31}=-148μs og τ_{32} =-124μs. Setter man dette inn i formel 10 får man at vinkelen er -31.94°. Dette er ganske nærme referansevinkelen på -35° som blir sendt inn. Resultatet fra korrelasjonsplottet stemmer derfor over ens med resultatet fra plottet av signalene. Vi får nemnleg også her at lyden treffer mikrofon 3 først siden tidsforsinkelsen fra mikrofon 3 til de to andre er ganske stor, og så treffer den mikrofon 1 og 2 omtrent likt siden tidstifferansen mellom de er veldig liten.

Vi tester videre for totalt 10 vinkler, med 5 målinger på hver vinkel. Resultatene av dette er vist i tabell 5. Her er Ref referansevinkelen vi fysisk har målt at vi sender inn lyd fra. m1 til m5 representerer de fem målingene vi har gjort med systemet vårt per vinkel, og μ er gjennomsnittet av disse fem målingene. ξ er forskjellen mellom gjennomsnittet og referansen, altså et mål på presisjon og systematisk feil, mens σ er standardavviket som er et mål på tilfeldig feil.

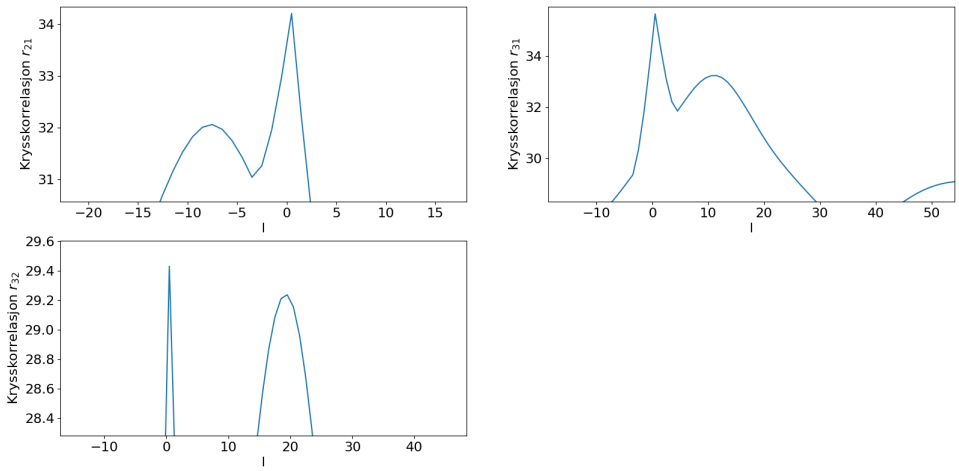
Det er interessant å se på både differansen ξ og standardavviket σ når man skal vurdere resultatene, siden disse forteller forskjellige ting. Et eksempel er at dersom vi kun ser på σ til vinkelen $-120°$, ser disse målingene veldig nøyaktige ut, men dersom vi i tillegg ser på ξ, ser vi at målingene ikke er helt presise sammenlignet med referansevinkelen. På en annen side kan det også godt tenkes at referansevinklene heller ikke er helt presise, da dette er fysiske målinger gjort med gradskive.

Dersom vi ser på referansevinkelen 170°, ser vi at det er et stort sprik mellom målingene. Det kan se ut som at det sannsynligvis har vært bakgrunnsstøy i omgivelsene plassert med vinkelen 120° til

Tabell 5: Resultat av målinger av 10 forskjellige vinkler.

Ref	**m1**	**m2**	**m3**	**m4**	**m5**	μ	ξ	σ
$-120°$	$-125.03°$	$-125.03°$	$-125.03°$	$-125.03°$	$-125.03°$	$-125.03°$	$5.03°$	$0°$
$-90°$	$-96.18°$	$-91.07°$	$-92.07°$	$-92.07°$	$-98.21°$	$-93.92°$	$3.92°$	$3.1°$
$-65°$	$-65.4°$	$-65.21°$	$-65.21°$	$-65.21°$	$-65.21°$	$-65.248°$	$0.25°$	$0.08°$
$-35°$	$-31.95°$	$-37.74°$	$-31.95°$	$-37.74°$	$-39.64°$	$-35.80°$	$0.80°$	$3.6°$
$0°$	$0°$	$-3.54°$	$-1.68°$	$-1.68°$	$-1.68°$	$-1.72°$	$1.72°$	$1.25°$
$10°$	$11.3°$	$11.3°$	$9.64°$	$-11.93°$	$-11.93°$	$1.68°$	$8.32°$	$12.44°$
$75°$	$87°$	$81.05°$	$87°$	$79.11°$	$87°$	$84.23°$	$9.23°$	$3.85°$
$90°$	$99.64°$	$91.95°$	$97.74°$	$95.82°$	$99.64°$	$96.96°$	$6.96°$	$3.22°$
$170°$	$120°$	$120°$	$177.46°$	$173.19°$	$174.79°$	$153.09°$	$16.91°$	$30.24°$
$180°$	$-178.37°$	$177.58°$	$-161.58°$	$180°$	$-178.43°$	$183.84°$	$3.84°$	$8.31°$

vårt system. Dette fører til høye verdier for ξ og σ. For å studere dette nøyere ser vi på figur 24 som viser en av målingene av vinkelen på 170° som viste en vinkel på 120°.

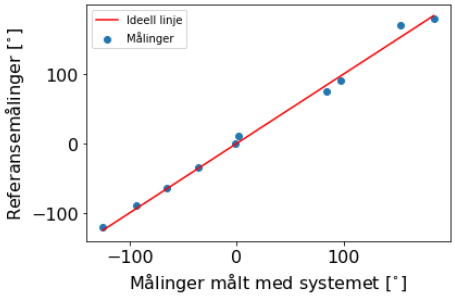


Figur 24: Krysskorrelasjon mellom signalene fra mikrofonene ved vinkel på 170° med fokus på maks-punktet for en måling som estimerte 120°.

Her ser vi at når vi studerer den høgaste verdien på lag nærme så ser vi at spesielt for r_{32} så er toppen ved $\mathrm{l} = 0$ akkurat høyere enn toppen ved $\mathrm{l} = 20$, men vi vet at ved 170° har vi en forsinkelse

mellom mikrofon 3 og mikrofon 2. Dette er derfor støy, da rett verdi her hadde vert at l = 20. For målingene for 170° som gir rett verdi er denne toppen høyere enn den på l = 0, og vi får da et helt annet resultat når vi finner maks verdien direkte med programmering. Her er det derfor også vesentlig å studere plottene for å finne den riktige verdien da det til tider kan være andre verdier for lag som er høyere en den faktisk ønskelige.

Figur 25 er et plott som sammenligner referansevinklene og gjennomsnittet av de fem målingene gjort per vinkel vist i tabell 5 med en ideell linje. Her ser vi at gjennomsnittet av målingene ikke er langt unna referansen, og vi har derfor ikke så mye systematisk feil. Denne figuren forteller derimot ingenting om standardavviket for hver referansevinkel.



Figur 25: Plott av målte vinkler og referansevinkler sammenlignet med ideell linje.

For en del vinkler får vi nøyaktig samme resultat på flere av målingene. Dette spesielt for målingene for 120°. Denne likheten kommer fra oppløsningen. Siden vi oppsamplet med en faktor 4, kan vi regne ut ved hjelp av ligning 12 den faktiske n_{maks} til å være

$$n_{maks} = 4\frac{df_s}{c} = 4\frac{5.6 \text{ cm} \cdot 31250 \text{ Hz}}{344 \text{ m/s}} = 20.35 = 21 \text{ samplinger.} \tag{19}$$

Dersom vi hadde oppsamplet signalet mer hadde vi økt n_{maks} som ville ført til en mer nøyaktig vinkel, som beskrevet i seksjon 2.2.1.

5 Diskusjon

Det vi ser av resultatdelen er at målingene våre er forholdsvis nøyaktige. For mange av verdiene i tabell 5 er vi delvis innenfor ønsket mål fra innledningen som var en feilmargin på treffsikkerheten på max 5°, og en maksbegrensing på tilfeldige feil som tilsvarer et standardavvik på maks 5°. Vi var innenfor målet for tilfeldige feil på 5 av 10 vinkler, og for målet for systematiske feil var vi innenfor på 7 av 10 målinger. Vi har derfor mer tilfeldige feil enn systematiske feil, noe som er vanskeligere å gjøre noe med. Det man derimot kan gjere for å forbedre det er å bruke en eller annen form for støyundertrykkelse for å forminske støy.

På grunn av liten plass på oppkoblingsbrettet, og ønske om å ha logiske plasseringer for ledningene for å lett kunne feilsøke ble ikke de digitale og analoge delene av kretsen separert så nøye. Dette kan ha ført til litt ekstra støy da de digitale signal linjene fører med seg en del støy. Dette er derfor et forbedringspotensial.

Siden målingene foregikk ved å holde lydkilden i hånda i en posisjon målt med gradskive, er det mange faktorer som kan føre til usikkerhet i referansevinklene. Det var svært vanskelig å få til å holde lydkilden i en nøyaktig målt posisjon som betyr at referansevinklene ikke nødvendigvis er de eksakte vinklene det måles fra. Små bevegelser av lydkilden under målinger kan også ha bidratt til unøyaktige referansevinkler. Det kunne vært fordelaktig å forbedre denne måten å måle referansevinkler på, til å bli mer nøyaktig.

Vi brukte frakoblingskondensatorer i parallell for å øke stabiliteten og skape mindre støy i forsyningslinjene. Dette har nok trolig minsket litt av støyet, og uten de kunne vi derfor fått enda dårligere resultat.

I designet av kretsen var det fokus på å holde ledninger så korte og organiserte som mulige. Dette både for å lettere kunne feilsøke, men også fordi vi jobbet med en del høye frekvenser, og lange ledninger kan da fungere som antenner for støyen inn og ut av kretsen. Vi hadde også fokus på å unngå jord eller spennings løkker da dette også kan være en kilde til støy. Dette kan derfor ha vert med på å redusere støyet litt.

Til tross for alle elementene som kunne tilføye støy fikk vi en SNR på 29dB for systemet, som betyr at signalet dominerer sterkt. Det er selvfølgelig forbedringspotensial, men for noen tilfeller er dette godt nok for et system som skal estimere en vinkel fra et lydsignal. Om dette er godt nok kommer veldig ann på bruksområdet, og må vurderest ut i fra hva systemet skal brukest til.

Dersom lydkilden ikke er plassert langt nok unna systemet, vil ikke lyden som treffer systemet være en planbølge. Dette fører til at hver mikrofon mottar lyd fra forskjellige vinkler, og beregning av vinkelen vil dermed bli svært unøyaktig. Siden vi målte fra omtrent 15 cm fra mikrofonene vil dette holde til et greit estimat av vinklene, men det kan diskuteres om de målte vinklene kunne vært mer presise om vi hadde økt denne avstanden. Dette kan ha vert en av feilkildene til at vi detekterte 120°

på noen av målingene for det som skulle være 170°. I nærmere analyse av de målingene såg vi også at det ofte kan være nyttig å studere krysskorrelasjonsplottene, og ikke berre stole på funksjonen som finner maks verdien. Dette fordi vi selv kan se at denne verdien ikke stemmer, men en funksjon ikke gjør dette.

Tar man resultatene i betraktning kan man konkludere med at systemet er egnet for å løse oppgaven. Dette berre dersom man tolererer at det blir noen feil estimeringer. Generelt sett fungerer systemet godt til å detektere innfallsvinkelen, men spesielt viss man legger sin egen tolkning til resultatet, eller om man vet den omtrentlige vinkelen og kan utelukke feil som skjedde med 170° målingene så kan man bruke systemet til å detektere vinkler forholdsvis nøyaktig.

Vi nevnte også i resultatdelen at en økning av samplingen ved hjelp av enda mer oppsampling kunne forbedret resultatet en del. Dette fordi vi ville at n_{max} skulle være stor. Dette hadde da påvirket hvor nøyaktig den klarer å tolke en vinkel. Dette er derfor et stort forbedringspotensial for systemet. Andre forbedringer som kunne blitt gjort er å ha større avstand mellom mikrofonene, og lagt til en form for støyundertrykking.

6 Konklusjon

Systemet oppfylte sin hensikt med å estimere vinkelen til en lydkilde. Av figur 25 ser man tydelig at de målte vinklene ikke viker mye fra referansevinklene. Av tabell 5 kan man derimot se at det er en del usikkerhet per måling. Dersom man sammenligner med kravene presentert i seksjon 1 ser man at 5/10 målinger oppfyller $\xi < 5°$, mens 7/10 målinger oppfyller $\sigma < 5°$.

Referanser

[1] TTT4280 Sensorer og instrumentering. Laboppgave 2 Akustikk: Mikrofonarray for å finne retning til en lydkilde. Trondheim: NTNU; 2022.

[2] SparkFun Electronics. SparkFun Pi Wedge. Colorado: SparkFun; 2022. Tilgjengelig fra: https://www.sparkfun.com/products/13717

[3] Microchip. MCP3201, 2.7V 12-Bit A/D Converter with SPI Serial Interface. Microchip; 2011. Tilgjengelig fra: http://ww1.microchip.com/downloads/en/DeviceDoc/21290F.pdf

[4] Pigpio. The pigpio library. pigpio; 2021. Tilgjengelig fra: http://abyz.me.uk/rpi/pigpio/

A ADC sampler

```
/*
adc_sampler.c
Public Domain
January 2018, Kristoffer Kj rnes & Asgeir Bj rgan
Based on example code from the pigpio library by Joan @ raspi forum and github
https://github.com/joan2937 | http://abyz.me.uk/rpi/pigpio/

Compile with:
gcc -Wall -lpthread -o adc_sampler adc_sampler.c -lpigpio -lm

Run with:
sudo ./adc_sampler

This code bit bangs SPI on several devices using DMA.

Using DMA to bit bang allows for two advantages
1) the time of the SPI transaction can be guaranteed to within a
   microsecond or so.

2) multiple devices of the same type can be read or written
   simultaneously.

This code reads several MCP3201 ADCs in parallel, and writes the data to a binary
    file.
Each MCP3201 shares the SPI clock and slave select lines but has
a unique MISO line. The MOSI line is not in use, since the MCP3201 is single
channel ADC without need for any input to initiate sampling.
*/

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
```

```
#include <pigpio.h>
#include <math.h>

/////// USER SHOULD MAKE SURE THESE DEFINES CORRESPOND TO THEIR SETUP ///////
#define ADCS 5      // Number of connected MCP3201.

#define OUTPUT_DATA argv[2] // path and filename to dump buffered ADC data

/* RPi PIN ASSIGNMENTS */
#define MISO1 23    // ADC 1 MISO (BCM 4 aka GPIO 4).
#define MISO2 24    //     2
#define MISO3 25    //     3
#define MISO4 26    //     4
#define MISO5 27    //     5

#define MOSI 10     // GPIO for SPI MOSI (BCM 10 aka GPIO 10 aka SPI_MOSI). MOSI not
    in use here due to single ch. ADCs, but must be defined anyway.
#define SPI_SS 18   // GPIO for slave select (BCM 8 aka GPIO 8 aka SPI_CE0).
#define CLK 19      // GPIO for SPI clock (BCM 11 aka GPIO 11 aka SPI_CLK).
/* END RPi PIN ASSIGNMENTS */

#define BITS 12            // Bits per sample.
#define BX 4               // Bit position of data bit B11. (3 first are t_sample +
    null bit)
#define B0 (BX + BITS - 1) // Bit position of data bit B0.

#define NUM_SAMPLES_IN_BUFFER 300 // Generally make this buffer as large as possible
    in order to cope with reschedule.

#define REPEAT_MICROS 32 // Reading every x microseconds. Must be no less than 2xB0
    defined above

#define DEFAULT_NUM_SAMPLES 31250 // Default number of samples for printing in the
    example. Should give 1sec of data at Tp=32us.

int MISO[ADCS]={MISO1, MISO2, MISO3, MISO4, MISO5}; // Must be updated if you change
    number of ADCs/MISOs above
/////// END USER SHOULD MAKE SURE THESE DEFINES CORRESPOND TO THEIR SETUP ///////

/**
 * This function extracts the MISO bits for each ADC and
 * collates them into a reading per ADC.
 *
 * \param adcs Number of attached ADCs
 * \param MISO The GPIO connected to the ADCs data out
 * \param bytes Bytes between readings
 * \param bits Bits per reading
```

```
 * \param buf Output buffer
*/
void getReading(int adcs, int *MISO, int OOL, int bytes, int bits, char *buf)
{
   int p = OOL;
   int i, a;

   for (i=0; i < bits; i++) {
      uint32_t level = rawWaveGetOut(p);
      for (a=0; a < adcs; a++) {
         putBitInBytes(i, buf+(bytes*a), level & (1<<MISO[a]));
      }
      p--;
   }
}


int main(int argc, char *argv[])
{
    // Parse command line arguments
    long num_samples = 0;
    if (argc <= 1) {
        fprintf(stderr, "Usage: %s NUM_SAMPLES\n\n"
                        "Example: %s %d\n", argv[0], argv[0], DEFAULT_NUM_SAMPLES);
        exit(1);
    }
    sscanf(argv[1], "%ld", &num_samples);

    // Array over sampled values, into which data will be saved
    uint16_t *val = (uint16_t*)malloc(sizeof(uint16_t)*num_samples*ADCS);

    // SPI transfer settings, time resolution 1us (1MHz system clock is used)
    rawSPI_t rawSPI =
    {
       .clk     =  CLK,  // Defined before
       .mosi    =  MOSI, // Defined before
       .ss_pol  =  1,   // Slave select resting level.
       .ss_us   =  1,   // Wait 1 micro after asserting slave select.
       .clk_pol =  0,   // Clock resting level.
       .clk_pha =  0,   // 0 sample on first edge, 1 sample on second edge.
       .clk_us  =  1,   // 2 clocks needed per bit so 500 kbps.
    };

    // Change timer to use PWM clock instead of PCM clock. Default is PCM
    // clock, but playing sound on the system (e.g. espeak at boot) will start
    // sound systems that will take over the PCM timer and make adc_sampler.c
    // sample at far lower samplerates than what we desire.
    // Changing to PWM should fix this problem.
```

```
gpioCfgClock(5, 0, 0);

// Initialize the pigpio library
if (gpioInitialise() < 0) {
   return 1;
}

// Set the selected CLK, MOSI and SPI_SS pins as output pins
gpioSetMode(rawSPI.clk,  PI_OUTPUT);
gpioSetMode(rawSPI.mosi, PI_OUTPUT);
gpioSetMode(SPI_SS,      PI_OUTPUT);

// Flush any old unused wave data.
gpioWaveAddNew();

// Construct bit-banged SPI reads. Each ADC reading is stored separatedly
// along a buffer of DMA commands (control blocks). When the DMA engine
// reaches the end of the buffer, it restarts on the start of the buffer
int offset = 0;
int i;
char buf[2];
for (i=0; i < NUM_SAMPLES_IN_BUFFER; i++) {
    buf[0] = 0xC0; // Start bit, single ended, channel 0.

    rawWaveAddSPI(&rawSPI, offset, SPI_SS, buf, 2, BX, B0, B0);
    offset += REPEAT_MICROS;
}

// Force the same delay after the last command in the buffer
gpioPulse_t final[2];
final[0].gpioOn = 0;
final[0].gpioOff = 0;
final[0].usDelay = offset;

final[1].gpioOn = 0; // Need a dummy to force the final delay.
final[1].gpioOff = 0;
final[1].usDelay = 0;

gpioWaveAddGeneric(2, final);

// Construct the wave from added data.
int wid = gpioWaveCreate();
if (wid < 0) {
    fprintf(stderr, "Can't create wave, buffer size %d too large?\n",
NUM_SAMPLES_IN_BUFFER);
    return 1;
}
```

```
// Obtain addresses for the top and bottom control blocks (CB) in the DMA
// output buffer. As the wave is being transmitted, the current CB will be
// between botCB and topCB inclusive.
rawWaveInfo_t rwi = rawWaveInfo(wid);
int botCB = rwi.botCB;
int topOOL = rwi.topOOL;
float cbs_per_reading = (float)rwi.numCB / (float)NUM_SAMPLES_IN_BUFFER;

float expected_sample_freq_khz = 1000.0/(1.0*REPEAT_MICROS);

printf("# Starting sampling: %ld samples (expected Tp = %d us, expected Fs = %.3f
 kHz).\n",
num_samples,REPEAT_MICROS,expected_sample_freq_khz);

// Start DMA engine and start sending ADC reading commands
gpioWaveTxSend(wid, PI_WAVE_MODE_REPEAT);

// Read back the samples
double start_time = time_time();
int reading = 0;
int sample = 0;

while (sample < num_samples) {
    // Get position along DMA control block buffer corresponding to the current
output command.
    int cb = rawWaveCB() - botCB;
    int now_reading = (float) cb / cbs_per_reading;

    while ((now_reading != reading) && (sample < num_samples)) {
        // Read samples from DMA input buffer up until the current output command

        // OOL are allocated from the top down. There are BITS bits for each ADC
        // reading and NUM_SAMPLES_IN_BUFFER ADC readings. The readings will be
        // stored in topOOL - 1 to topOOL - (BITS * NUM_SAMPLES_IN_BUFFER).
        // Position of each reading's OOL are calculated relative to the wave's
top
        // OOL.
        int reading_address = topOOL - ((reading % NUM_SAMPLES_IN_BUFFER)*BITS) -
 1;

        char rx[8];
        getReading(ADCS, MISO, reading_address, 2, BITS, rx);

        // Convert and save to output array
        for (i=0; i < ADCS; i++) {
            val[sample*ADCS+i] = (rx[i*2]<<4) + (rx[(i*2)+1]>>4);
        }
```

```
            ++sample;

            if (++reading >= NUM_SAMPLES_IN_BUFFER) {
                reading = 0;
            }
        }
        usleep(1000);
    }

    double end_time = time_time();

    double nominal_period_us = 1.0*(end_time-start_time)/(1.0*num_samples)*1.0e06;
    double nominal_sample_freq_khz = 1000.0/nominal_period_us;

    printf("# %ld samples in %.6f seconds (actual T_p = %f us, nominal Fs = %.2f kHz)
    .\n",
        num_samples, end_time-start_time, nominal_period_us, nominal_sample_freq_khz)
    ;

    double output_nominal_period_us = floor(nominal_period_us); //the clock is
    accurate only to us resolution

    // Path to your data directory/file from previous define
    const char *output_filename = OUTPUT_DATA;

    // Write sample period and data to file
    FILE *adc_data_file = fopen(output_filename, "wb+");
    if (adc_data_file == NULL) {
        fprintf(stderr, "# Couldn't open file for writing: %s (did you remember to
    change OUTPUT_DATA?)\n", output_filename);
  return 1;
    }

    fwrite(&output_nominal_period_us, sizeof(double), 1, adc_data_file);
    fwrite(val, sizeof(uint16_t), ADCS*num_samples, adc_data_file);
    fclose(adc_data_file);
    printf("# Data written to file. Program ended successfully.\n\n");

    gpioTerminate();
    free(val);

    return 0;
}
```

B Raspi analyze lab1

```
import numpy as np
```

```python
import matplotlib.pyplot as plt
import scipy.signal as signal


def raspi_import(path, channels=5):
    """
    Import data produced using adc_sampler.c.
    Returns sample period and ndarray with one column per channel.
    Sampled data for each channel, in dimensions NUM_SAMPLES x NUM_CHANNELS.
    """

    with open(path, 'r') as fid:
        sample_period = np.fromfile(fid, count=1, dtype=float)[0]
        data = np.fromfile(fid, dtype=np.uint16)
        data = data.reshape((-1, channels))
    return sample_period, data


sample_period, data = raspi_import('lab1_m linger') # Import data from bin file

sample_period *= 1e-6  # change unit to micro seconds

data = data*(0.8*10**(-3)) # Change to analog values

num_of_samples = data.shape[0]  # returns shape of matrix

t = np.linspace(start=0, stop=num_of_samples*sample_period, num=num_of_samples) #
    Generate time axis
freq = np.fft.fftfreq(n=num_of_samples, d=sample_period) # Generate frequency axis
spectrum = np.fft.fft(data, axis=0)  # takes FFT of all channels

# PLOTS: (Use data[:,n] to get channel n)
plt.rc('xtick', labelsize=18)
plt.rc('ytick', labelsize=18)
plt.tight_layout()

plt.subplot(2, 1, 1)
plt.xlabel("Tid [us]", fontsize = 20)
plt.ylabel("Spenning [V]", fontsize = 20)
plt.plot(t, data) # evt data[:,0]
plt.legend(['GPIO23', 'GPIO24', 'GPIO25', 'GPIO26', 'GPIO27' ],  fontsize = 16, loc =
     'upper right')

plt.subplot(2, 1, 2)
plt.xlabel("Frekvens [Hz]", fontsize = 20)
plt.ylabel("Energi [dB]", fontsize = 20)
plt.plot(freq, 20*np.log10(np.abs(spectrum))) # get the power spectrum
plt.legend(['GPIO23', 'GPIO24', 'GPIO25', 'GPIO26', 'GPIO27' ],  fontsize = 16, loc =
```

```
     'upper right')

plt.show()
```

C Raspi analyze lab2

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.signal as signal
import scipy.interpolate as interpolate

filename = 'm lingar/v170_m5'

upsampling_factor = 4

def raspi_import(path, channels=5):
    """
    Import data produced using adc_sampler.c.
    Returns sample period and ndarray with one column per channel.
    Sampled data for each channel, in dimensions NUM_SAMPLES x NUM_CHANNELS.
    """

    with open(path, 'r') as fid:
        sample_period = np.fromfile(fid, count=1, dtype=float)[0]
        data = np.fromfile(fid, dtype=np.uint16)
        data = data.reshape((-1, channels))
    return sample_period, data


def scale_and_remove_DC(data):
    """
    Scales data from digital to analog value, and removes DC component.
    Returnes the new data.
    Uses ADC_resolution that is defined, and average.
    """
    data = data*(0.8*10**(-3)) # Scales unit to analog values
    data = signal.detrend(data, axis=0)  # removes DC component for each channel
    return data


sample_period, data = raspi_import(filename) # Import data from bin file

sample_period *= 1e-6  # Change unit to micro seconds
fs = 1/sample_period

data = scale_and_remove_DC(data)
```

```
num_of_samples = data.shape[0]  # returns shape of matrix

t = np.linspace(start=0, stop=num_of_samples*sample_period, num=num_of_samples) #
    Generate time axis
freq = np.fft.fftfreq(n=num_of_samples, d=sample_period) # Generate frequency axis

spectrum = np.fft.fft(data, axis=0)  # takes FFT of all channels


# PLOT: (Use data[:,n] to get channel n)
plt.rc('xtick', labelsize=16)
plt.rc('ytick', labelsize=16)
plt.tight_layout()

plt.subplot(2, 1, 1)
plt.xlabel("Time [s]", fontsize = 20)
plt.ylabel("Voltage [V]", fontsize = 20)
plt.plot(t, data[:,0:3]) # evt data[:,0]
plt.legend(['Mikrofon 1', 'Mikrofon 2', 'Mikrofon 3'],  fontsize = 16, loc = 'upper
    right')

plt.subplot(2, 1, 2)
plt.xlabel("Frequency [Hz]", fontsize = 20)
plt.ylabel("Power [dB]", fontsize = 20)
plt.plot(freq, 20*np.log10(np.abs(spectrum[:,0:3]))) # get the power spectrum
plt.legend(['Mikrofon 1', 'Mikrofon 2', 'Mikrofon 3'],  fontsize = 16, loc = 'upper
    right')

plt.show()


"""
# Signals for testing
f = 100
fs = 5000
l = 50
l2 = 70
t = np.arange(-100, 100)

s1 = np.sinc(2*np.pi*f/fs*t)
s2 = np.roll(s1, l)
s3 = np.roll(s1, l2)
"""

s1 = data[:,0]
s2 = data[:,1]
s3 = data[:,2]
```

```
t_upsampled = np.linspace(start=0, stop=num_of_samples*sample_period, num=
    num_of_samples*upsampling_factor)

# Adjust signals to new upsampled axis.
s1 = np.interp(t_upsampled, t, s1)
s2 = np.interp(t_upsampled, t, s2)
s3 = np.interp(t_upsampled, t, s3)

# Find correlation
corr_21 = signal.correlate(s2, s1, mode='same')
corr_31 = signal.correlate(s3, s1, mode='same')
corr_32 = signal.correlate(s3, s2, mode='same')

# Find autocorrelation
autocorr1 = signal.correlate(s1, s1, mode='same')
autocorr2 = signal.correlate(s2, s2, mode='same')
autocorr3 = signal.correlate(s3, s3, mode='same')

corr_ax = np.linspace(-len(corr_21)//2, len(corr_21)//2, len(s1))

# Find the sample and time where the correlation has a maximum
n_21 = corr_ax[np.where(corr_21 == max(abs(corr_21)))[0][0]]
print("n21:", n_21, "samples")
print("tau21:", (n_21/(fs*upsampling_factor))*(10**6), "mikro sekund")

n_31 = corr_ax[int(np.where(corr_31 == max(abs(corr_31)))[0][0])]
print("n31:", n_31, "samples")
print("tau31:", (n_31/(fs*upsampling_factor))*(10**6), "mikro sekund")

n_32 = corr_ax[int(np.where(corr_32 == max(abs(corr_32)))[0][0])]
print("n32:", n_32, "samples")
print("tau32:", (n_32/(fs*upsampling_factor))*(10**6), "mikro sekund")

# Uses arctan2(numerator, denumerator) because we want to get the right quadrant
    directly
theta = np.arctan2(np.sqrt(3)*(n_21+n_31),(n_21-n_31-2*n_32))

print("\n")
print("Theta i radianer: ", theta)
print("Theta i grader:", np.degrees(theta))

# Correlation plots:
plt.rc('xtick', labelsize=16)
plt.rc('ytick', labelsize=16)
plt.tight_layout()
```

```
plt.subplot(2, 2, 1)
plt.xlabel("l", fontsize = 16)
plt.ylabel("Krysskorrelasjon $r_{21}$", fontsize = 16)
plt.plot(corr_ax, corr_21)

plt.subplot(2, 2, 2)
plt.xlabel("l", fontsize = 16)
plt.ylabel("Krysskorrelasjon $r_{31}$", fontsize = 16)
plt.plot(corr_ax,corr_31)

plt.subplot(2, 2, 3)
plt.xlabel("l", fontsize = 16)
plt.ylabel("Krysskorrelasjon $r_{32}$", fontsize = 16)
plt.plot(corr_ax, corr_32)
plt.show()

# Autocorrelation plots:
plt.rc('xtick', labelsize=16)
plt.rc('ytick', labelsize=16)
plt.tight_layout()

plt.subplot(2, 2, 1)
plt.xlabel("l", fontsize = 16)
plt.ylabel("Autokorrelasjon $r_{11}$", fontsize = 16)
plt.plot(corr_ax, autocorr1)

plt.subplot(2, 2, 2)
plt.xlabel("l", fontsize = 16)
plt.ylabel("Autokorrelasjon $r_{22}$", fontsize = 16)
plt.plot(corr_ax,autocorr2)

plt.subplot(2, 2, 3)
plt.xlabel("l", fontsize = 16)
plt.ylabel("Autokorrelasjon $r_{33}$", fontsize = 16)
plt.plot(corr_ax, autocorr3)
plt.show()
```