Bacheloroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Det humanistiske fakultet
Institutt for musikk

Hallvard Jensen

Oversettelse fra lyd til MIDI ved hjelp av spektralanalyse

Bacheloroppgave i Musikkteknologi
Veileder: Øyvind Brandtsegg
Mai 2021

NTNU
Kunnskap for en bedre verden

Hallvard Jensen

Oversettelse fra lyd til MIDI ved hjelp av spektralanalyse

Bacheloroppgave i Musikkteknologi
Veileder: Øyvind Brandtsegg
Mai 2021

Norges teknisk-naturvitenskapelige universitet
Det humanistiske fakultet
Institutt for musikk

NTNU
Kunnskap for en bedre verden

Sammendrag

MIDI brukes mye i moderne musikkproduksjon, men spilles som oftest inn ved hjelp av keyboard. Denne oppgaven ser nærmere på hvordan man kan bruke andre instrumenter til dette formålet ved å konvertere lydsignaler til MIDI. For å gjøre dette benyttes FFT-algoritmen, og vi ser nærmere på hvilke metoder som brukes for å behandle dataen som algoritmen produserer. I forbindelse med oppgaven ble applikasjonen *anyMidi* utviklet for å prøve ut teknikker fra teorien i praksis. Denne applikasjonen var ved prosjektets slutt delvis funksjonell, men hadde fremdeles mye forbedringspotensial. Det diskuteres hvilke algoritmer som ble implementert og i hvilken grad de fungerer, samt planer for videre utvikling av algoritmer.

Abstract

MIDI is used to a great extent in modern music production, but is recorded with a keyboard most of the time. This thesis takes a closer look at how other instruments can be used for this purpose, by converting audio signals to MIDI. The FFT algorithm is used to achieve this, and we examine which methods are used to process the data produced by the algorithm. In conjunction with this thesis the application *anyMidi* was developed to test theory in practice. At the projects end, the appliaction was partially functional, but still had potential for improvement. The implemented algorithms and their functionalities are discussed, along with potential plans for further development.

Forord

Musikkteknologi er et fagfelt hvor det er vanskelig å finne gode norske begreper som beskriver teknologien. Ofte bruker man engelske ord for å mer presist forklare hva man snakker om. Ettersom norske oversettelser av engelske musikkteknologiske uttrykk ofte blir unaturlige og upresise, velger jeg i denne oppgaven å unngå bruken av dem. Istedet vil jeg bruke de engelske uttrykkene direkte hvor det er nødvendig. Når jeg bruker engelske musikkteknologiske ord vil de være markert med *kursiv skrift* for å indikere at dette ikke er norsk.

Innhold

1 Introduksjon

Dagens musikkproduksjon foregår hovedsakelig digitalt. Innspilling av musikk, miksing og produksjon gjøres stort sett i en DAW (*digital audio workstation*). Her kan produsenten manipulere lyd i det digitale domenet, før man igjen konverterer det til lyd mennesker kan høre. Selv komposisjonsprosessen og innspillingsprosessen kan gjøres heldigitalt ved hjelp av å spille inn eller plotte MIDI. Fra sin barndom på 80-tallet hvor instrumenter var koblet sammen med kabler, har MIDI-protokollen blitt skilt ut og kan nå manipuleres direkte ved hjelp av en datamaskin. Likevel spiller man fortsatt inn MIDI ved hjelp av fysiske kontrollere. Keyboardet er nok den mest populære kontrolleren og er standardinventar i et musikkstudio. Samtidig er pad-baserte kontrollere etter hvert brukt i større grad, ettersom det er enklere for å programmere trommer eller sekvensere sampler.

Keyboardets store representasjon innen MIDI-innspilling gjør at andre analoge instrumenter har delvis falt utenfor. Disse instrumentene har dog ikke forsvunnet fra prosessen, selv om store deler av produksjonen skjer digitalt. De fleste musikere og komponister er kyndige med flere instrumenter, og ofte er visse instrumenter bedre egnet til noen formål enn andre. Dermed kan det være interessant å bruke disse grensesnittene for å spille inn digitalt. Den mest universelle måten å gjøre dette på er å analysere lyden instrumentet lager, og deretter bruke dette som grunnlag for å produsere MIDI-data. Ekspressive nyanser i utøverens spillestil blir på den måten brakt med i MIDI-representasjonen. Krysningspunktet mellom den analoge og digitale verden er et godt utforsket område, men det har en del nyanser som er viktige å være bevisst på. Ikke bare er det viktig å vite om fallgruvene som kan føre til feil, men det er også viktig å bevare musikaliteten i transformasjonen.

Dermed vil denne oppgaven dreie seg om hvilke valg som tas i det man vil bringe analog lyd inn i det digitale domenet og videre til MIDI. Som en del av utforskningen har jeg utviklet en applikasjon som skal konvertere et monofonisk gitarsignal til MIDI ved hjelp av FFT-analyse. Spørsmålsstillingen for denne oppgaven vil da være:

> *Hvordan går man fram for å konvertere fra spektraldomenet til MIDI, ved hjelp av FFT—på en musikalsk måte?*

Applikasjonen som jeg har utviklet er programmert i C++ ved hjelp av rammeverket JUCE. Grunnen til dette er at C++ gir stor kontroll over detaljnivået i utviklingen, og vil dermed kunne gi meg en større forståelse for konseptene i teorien. JUCE på sin side er et mye bruk rammeverk i industrien (JUCE, 2021a), og gjør prosessen enklere. I tillegg gir dette meg erfaring med potensielt yrkesrelevante verktøy, som er et motivasjonsmoment for meg.

1.1 Kontekst

Lyd-til-MIDI-applikasjoner har ikke like stor markedsandel som en rekke andre musikkprogramvarer, likevel er idéen i seg selv langt fra unik. Et enkelt nettsøk avslører at det finnes utallige webapplikasjoner som kan konvertere en lydfil om til en MIDI-fil—både som åpen og lukket kildekode. Også i DAW-en Ableton Live eksisterer flere funksjoner som lar brukeren konvertere et innspilt lydklipp om til MIDI-noter, direkte i arbeidsmiljøet. Blant annet er det mulig å ekstrahere harmonier, melodier og rytmikk fra et hvilket som helst lydklipp, og produsere MIDI basert på denne informasjonen (Ableton, 2021).

Både de mange nettbaserte tjenestene og den integrerte funksjonen i Ableton Live jobber med ferdig innspilt lyd. Selv om teknologien ligner, skiller det seg fra denne oppgavens mål som dreier seg om å konvertere i sanntid. Når prosesseringen skjer i sanntid vil det være høyere krav til å redusere forsinkelser, ettersom timing er sentralt for å bevare musikalitet—et av premissene for denne oppgaven.

En programvare som i stor grad minner om konseptet i denne oppgaven er MIDI Guitar 2, utviklet av Jam Origin. Denne applikasjonen konverterer et gitarsignal til MIDI i sanntid med lav forsinkelse i tillegg til at den kan analysere polyfoniske signaler (Jam Origin, 2021b). Denne applikasjonen har rikelig med tilleggsfunksjoner som også går utover det som er relatert til MIDI. MIDI Guitar 2 kan brukes både frittstående, men kan også brukes som en VST i en DAW (Jam Origin, 2021a). Denne programvaren er proprietær, og dermed er ikke kildekoden tilgjengelig. Av den grunn kan ikke applikasjonen fungere som noe mer enn inspirasjon til denne oppgavens elementer, og alle algoritmer vil måtte hentes fra andre kilder.

Som man ser er teknologien allerede utprøvd og applikasjoner som MIDI Guitar

2 fungerer veldig godt til formålet. Det betyr at denne oppgaven kun vil se på én mulig implementasjon, og omhandler undersøkelse av hvordan disse teknikkene fungerer gjennom praktisk utprøvning. Spektralanalyse og *pitch detection* er omfattende temaer og denne oppgavens omfang tillater ikke en fullstendig gjennomgang av alle de tekniske momentene vedrørende disse teknikkene. Likevel er det interessant å belyse områder som er relevante for å gjøre konvertering av lyd til MIDI på en god måte.

2 Teori

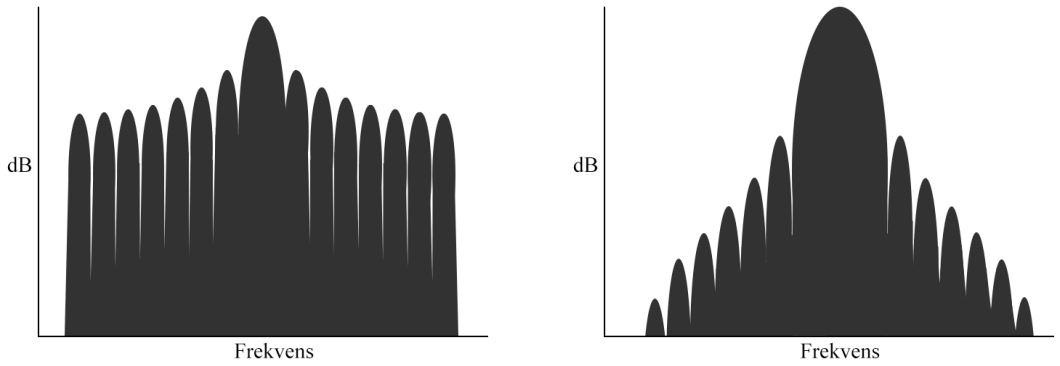
2.1 FFT

Fast fourier transform (FFT) er en algoritme som effektivt implementerer *discrete fourier transform* (DFT). Fouriertransformasjon er et matematisk konsept som forenklet sagt tilnærmer funksjoner ved å summere sammen sinus- og cosinusfunksjoner. Dette er spesielt nyttig hvis man opererer med ikke-kontinuerlige funksjoner slik som en firkantbølge, og ønsker å tilnærme denne i en kontinuerlig form. Ettersom kontinuerlige fouriertransformasjoner tar integralet over et intervall, ville dette skapt uendelig antall verdier som ville vært umulig for en datamaskin å behandle. I stedet bruker man DFT, siden man her ender opp med et diskret (dvs. tellbart og endelig) antall verdier (Steiglitz, 1996).

Vi vil ikke gå inn i detaljene i FFT-algoritmen her, men kort sagt gjør det at det er like gjennomførbart å programmere i frekvensdomenet som i tidsdomenet (Rockmore, 2000). Det vi heller skal se på er hvordan transformasjonene er representert i en FFT. Algoritmen tar inn et signal og deler frekvensspekteret opp i *bins*. Dette er et gitt antall frekvensbånd som er fordelt lineært utover intervallet 0 Hz opp til Nyquist-frekvensen, som er halvparten av samplingsfrekvensen (Steiglitz, 1996). Hver *bin* inneholder en amplitudeverdi for frekvensen som *bin*-en representer. Matematisk vil frekvensen til hver *bin* være gitt ved:

$$f = \frac{k}{N} f_s \tag{1}$$

hvor $k \in [0, N]$, N er FFT-størrelsen og f_s er samplingsfrekvensen. Eksempelvis kan man ha en FFT-størrelse på 2048 og en samplingsfrekvens på 48 kHz. Som nevnt jobber FFT opptil Nyquist-frekvensen, som i dette tilfellet vil være 24 kHz. Dermed vil man ha 1025 *bins* med første *bin* på 0 Hz og siste *bin* på 24 kHz, hvor hver *bin* er separert med et intervall på 23.4 Hz. Et viktig moment å bemerke seg her er at man ikke får representert frekvenser som faller mellom intervallene, og dette skaper utfordringer som vil bli diskutert nærmere.



(a) Rektangulært vindu (b) Hamming-vindu

Figur 1: Effekten av vindusfunksjoner

2.2 Vinduer

Når man gjør DFT (og i forlengelse, FFT) på et avkortet signal, dvs. at en sampel av en bølgeform avsluttes før amplituden har returnert til 0, kan det oppstå 'lekkasjer' av frekvenser (Girgis & Ham, 1980; Steiglitz, 1996). Figur 1a illustrerer hvordan frekvenser 'lekker' fra den midterste lappen (frekvensen vi faktisk analyserer) over til sidelappene (nærliggende frekvenser). Ettersom FFT-ens hastighet sjeldent sammenfaller nøyaktig med perioden til alle analyserte bølgeformer, vil slike lekkasjer ofte oppstå. Dataen blir dermed tilgriset med feil informasjon, og det er ønskelig å begrense dette.

En løsning er å bruke vindusfunksjoner. I FFT analyseres samplene fra et signal i segmenter som kalles vinduer. Om ingen annen konfigurasjon er satt heter det at vinduet er rektangulært—dvs. at man har en uvektet analyse av samplene (se Figur 1a). Målet til vindusfunksjoner er å gradvis tvinge en bølge ned mot 0 før vinduet avsluttes, slik at unaturlige, brå overganger ikke skjer. For eksempel er Hamming-vinduet én enkelt vektet cosinus-syklus som multipliseres med den samplede bølgen. Resultatet av multiplikasjonen er nettopp at signalet får en jevnere opp- og nedstigning fra og til 0, og dermed reduserer lekkasjefrekvenser betraktelig. Dette illustreres i Figur 1b hvor utslaget til sidelappene har blitt betraktelig redusert. En god analogi er å se på DFT gjort på et signal multiplisert med en vindusfunksjon i tidsdomenet, som det samme som en filtrering ville vært i frekvensdomenet (Steiglitz, 1996).

Samtidig er bruk av vindusfunksjoner et kompromiss. Som man ser i Figur 1 blir

midterste lapp bredere når man benytter en vindusfunksjon på signalet (merk at disse figurene ikke er matematisk nøyaktige, og er kun ment å illustrere problemene). Når man endrer måten man transformerer samplene gjennom en FFT på, får det konsekvenser for resultatet. Dette er et kompromiss mellom oppløsning på den midterste lappen og frekvenslekkasje til sidelappene (Steiglitz, 1996). Av denne grunn har det blitt utviklet en rekke ulike variasjoner av vindusfunksjoner, hvor hver av dem vekter relasjonen mellom oppløsning og lekkasjer ulikt.

2.3 Utfordringer med FFT

Når man jobber med FFT er det mange faktorer som kan skape problemer for representasjonen av frekvensspekteret som man ønsker å analysere. Dermed er det viktig å være bevisst valgene man gjør i prosessen og hvordan man tolker dataen. Man vil neppe få gode resultater om man bruker FFT blindt, uten å vite noe om hva dataen betyr og hvilke problemer som kan oppstå. Girgis og Ham (1980) snakker om hovedsakelig tre fallgruver: *aliasing*, *picket fencing* og frekvenslekkasje. Ettersom vi allerede har diskutert frekvenslekkasjer i forbindelse med vinduer, ser vi kun nærmere på de to første punktene.

Den første fallgruven er et kjent problem som handler om at lave frekvenser vil oppstå i analysen av et signal, hvis høyeste samplet frekvens er høyere enn halvparten av samplingsfrekvensen (Nyquist-frekvensen). Årsaken til dette er at perioden til høye frekvenser vil være for kort til at samplene klarer å gjengi svingninger i bølgen. Samplene vil derimot representere disse frekvensene som en mye lavere frekvens enn den faktisk er. Derfor er det som nevnt viktig å ha en samplingrate høyere enn det dobbelte av den høyeste frekvensen man forventer å sample.

Picket fencing kommer av at man alltid vil ha lavere antall *bins* enn antall mulige frekvenser i en FFT. *Bins* er diskrete verdier, mens frekvensspektrumet er kontinuerlig og inneholder uendelig antall verdier. Resultatet er at de aller fleste frekvenskomponenter i et analysert signal havner et sted mellom to *bins*. Dette fører igjen til at det oppstår 'frekvenslekkasjer' som påvirker de to *bin*-ene som frekvensen ligger mellom. Dvs. at alle samplede frekvenser som faller et sted mellom to *bins* ikke vil bli korrekt representert, men heller 'lekke' over til sine naboer. Amplituden til den faktiske fre-

kvensen vil fordele seg på de nærliggende *bin*-ene og forholdet bestemmes av hvilken *bin* som er nærmest.

2.4 Representasjon av lyd som MIDI

MIDI er for de fleste som jobber med musikk et veldig kjent konsept. Det er en protokoll som gjør det mulig å arbeide med musikk i digitale systemer, og har sine røtter tilbake til 1980-tallet (Huber, 2020). Moore (1988) sammenligner musikk med språk, og her poengteres det at språk ikke bare er verbalt, men at intonasjon, uttrykk o.l. også spiller enormt mye inn på hva ordene faktisk betyr. De samme prinsippene gjelder for musikk. Det finnes ikke bare informasjon i tonene, men også i klangen og intonasjonen. Videre forklarer Moore (1988) hvilke begrensninger MIDI har for å representere musikk. Siden musikalsk informasjon er kompleks, er det vanskelig å reprodusere intimiteten og ekspressiviteten av et tradisjonelt musikkinstrument i en datamaskin. Toner med små variasjoner kan oppfattes av hørselen vår som ulike både i tonehøyde og klangfarge. En datamaskin som analyserer tonehøyde, vil derimot kunne anse disse to ulike tonene som identiske. På den måten vil ikke den samme musikalske opplevelsen kunne produseres digitalt.

Med gitar som eksempel ser man mange uoverensstemmelser mellom de fysiske vibrasjonene i gitaren og hva MIDI er i stand til å representere. Gitarens tonehøyde vil fluktuere en god del ved anslaget av en streng. Dette gjør at det som høres som én enkelt tone fra gitaren i realiteten er en oscillerende tonehøyde som flater ut til én enkelt tone. I tillegg vil strengenes tykkelse påvirke hvor mye tonehøyden fluktuerer. Konstruksjon av strenger varierer også, der mørke strenger som oftest er viklet, mens de lyse ikke er det. Alle disse momentene, sammen med hvilken eksitasjonsteknikk som brukes—plekter eller fingre—har stor betydning for hvordan lyden arter seg (Carral & Paset, 2008). Dette er forhold som MIDI ikke er i stand til å håndtere, og vil dermed gjøre at analog informasjon kan feiltolkes eller forsvinne hvis det skal konverteres til MIDI.

Det betyr derimot ikke at MIDI umulig kan brukes til å uttrykke musikk på en meningsfull måte, men det krever at man er bevisst de valgene man tar. Hvis man skal bruke MIDI som en representasjon for analog lyd er det sentralt at man definerer

hva oppgaven til MIDI er. For å bruke eksemplet med gitaren kan man spørre seg: Er det ønskelig at MIDI skal emulere gitarens svingende tonehøyde, eller er vil man representere den tonehøyden musikeren prøver å spille? Ved å gjøre slike refleksjoner vet man hvilke grep som må gjøres for å f.eks. forberede et signal for analyse. På den måten oppnås det ønskede resultatet mye lettere enn om man gikk ut fra at MIDI-protokollen oppfører seg på samme måte som analog lyd.

3 Prosess

I tilknytning til denne oppgaven ble applikasjonen *anyMidi* utviklet som en måte å prøve ut teorien som diskuteres her. Prosessen hadde en relativt lang startfase hvor jeg hovedsaklig lærte meg JUCE-rammeverket og innhentet teori om FFT og spektralanalyse. Gradvis startet utviklingsstadiet til applikasjonen, som opptok en ganske stor del av semesteret denne oppgaven ble skrevet. Prosessen foregikk trinnvis, hvor jeg jobbet med en bestemt del av applikasjonens funksjonalitet frem til at jeg nådde et nivå der jeg ikke hadde nok kunnskap. Deretter måtte jeg tilegne meg den nye kunnskapen som skulle til for å legge til eller raffinere funksjonalitet i applikasjonen, før jeg kunne sette i gang med neste trinn.

I oppstartsfasen av utviklingen var planen at programmet skulle være en VST (*virtual studio technology*) som kunne importeres som *plug-in* til en DAW. Før utviklingsprosessen var kommet seg skikkelig i gang bestemte jeg meg derimot for at applikasjonen skulle være frittstående og fungere sammen med en tredjeparts programvare som rutet MIDI-signalet inn til en DAW. Grunnen til dette var et ønske om at applikasjonen skulle få det analoge instrumentet til å oppføre seg som et MIDI-instrument. Vanligvis blir MIDI-instrumenter koblet til en datamaskin med for eksempel USB, og en driver sørger for at DAW-en kan finne disse instrumentene og benytte seg av dem. Ved å gjøre *anyMidi* frittstående slipper man å importere programmet som en *plug-in*, men istedet bruke det som et hvilket som helst MIDI-grensesnitt. Samtidig slipper man å rute lyd fra en lydkanal til en MIDI-kanal i hvert DAW-prosjekt, slik man måtte gjort i for eksempel Ableton Live.

3.1 Lyd til MIDI

I en undersøkelse gjort i New Zealand i 2000 ble det undersøkt hvilke positive og negative konsekvenser MIDI har for musikkomposisjon. Testsubjektene var musikkstudenter på ulike fordypningsnivåer. Airy og Parr (2001) fant ut at MIDI i hovedsak gjør det enklere å lage musikk. Det er med på å gjøre musikk mer tilgjengelig ved at musikere kan jobbe mer uavhengig med egne prosjekter, og ikke er like begrenset av tilgjengelighet på instrumenter eller egne evner. På den andre siden er keyboard

det dominerende instrumentet for å spille inn MIDI, noe flere studenter hadde dårlige erfaringer med.

> The MIDI controller used by students for this study was the keyboard. The use of this controller was a central issue for many students. [A student] expressed doubt that they would be able to realise their musical ideas on a MIDI controller that was not their first instrument of choice. Only three students considered they were conversant with keyboard instruments and over half the students recognised a lack of keyboard skills to be their major barrier to successful composition. Therefore, the ability for most students to utilise the keyboard as a controller for either keyboard music or to access other sonorities remained largely unrealised. (Airy & Parr, 2001, s. 45)

Et problem ser altså ut til å være at MIDI som oftest blir innspilt ved hjelp av keyboard, som ikke er alles foretrukne kontroller. Det vil være med på å begrense kreativiteten hos utøveren og dermed være et hinder i den kompositoriske prosessen. Ved å tilgjengeliggjøre MIDI-innspilling ved hjelp av andre instrumenter, kan man også bevare kreativiteten og uttrykket som ligger hos komponisten.

anyMidi som brukes som casestudie for denne bacheloroppgaven har som mål å løse denne problemstillingen, og dermed tilgjengeliggjøre MIDI utenfor domenet til keyboardet. Applikasjonen i seg selv er instrumentuavhengig, men analysemetoder og valg i utviklingsprosessen har vært orientert rundt å optimalisere resultater for gitar. Utviklingsprosessen og valgene vedrørende *anyMidi* fungerer som katalysatorer for diskusjonen rundt konseptene i denne oppgaven, og applikasjonen i seg selv brukes som en praktisk utprøvning av den teorien som blir diskutert.

3.2 Valg av C++ og JUCE

Implementasjonen av *anyMidi* er gjort i programmeringsspråket C++ ved hjelp av rammeverket JUCE. Valget å utvikle applikasjonen på denne måten har gjort prosessen mye mer tidkrevende og kompleks enn om jeg hadde benyttet meg av modulbaserte språk slik som Csound eller MaxMSP. Kodesnutter eksisterer allerede og bruken av dem kunne gjort utviklingsprosessen mindre tidkrevende i tillegg til å produsere bedre resultater. I mitt tilfelle har jeg valgt å skrive mye kode fra bunnen av til tross for dette. Fra et programvareutviklingsstandpunkt er det dårlig praksis å for det første lage noe som allerede eksisterer, og for det andre å ikke benytte seg av de bibliotekene og den koden som man har mulighet til.

Poenget med en slik utviklingsprosess har vært å få en dypere forståelse av konseptene som ligger til grunne for algoritmene som brukes. Målet med denne oppgaven er ikke å lage et nyskapende produkt, men å forstå noen aspekter ved spektralanalyse og FFT. Særlig er poenget å ha høy kontroll over hvilke valg som gjøres i konverteringen fra lyd til MIDI for å forstå hvorfor man tar disse valgene, og da er det viktig å kunne jobbe på et lavere programmeringsnivå.

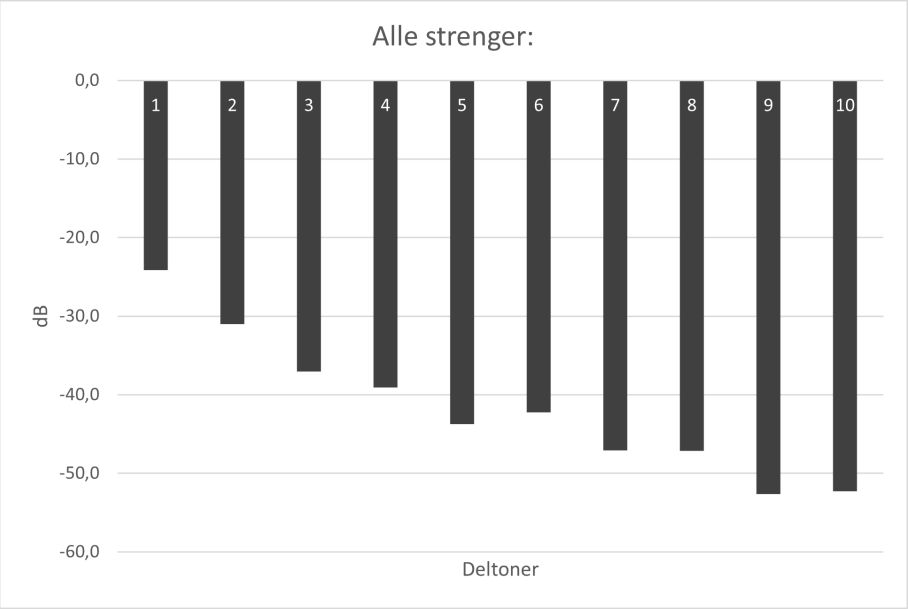
En annen viktig grunn til valget av programmeringsspråk er hvilken rolle C++ og JUCE har. Disse verktøyene er mye brukt innen programvareindustrien for musikk og lyd. Det gjør at jeg har stor egeninteresse av å bli komfortabel med disse teknologiene, siden det vil gi meg erfaringer og kunnskaper som både er faglig relevante og yrkesrelevante.

3.3 Testing

Under utviklingen av *anyMidi* benyttet jeg en Gibson SG Standard el-gitar for å teste om programvaren oppførte seg slik jeg forventet. Dermed ble også parametre satt og justeringer gjort for å tilpasses akkurat dette instrumentet. Dette påvirket både kodebasen, men også prosessen i seg selv, ettesom alle valg i spektralanalysen ble tatt utfra resultater fra denne el-gitaren. For å finne tonehøyden til et signal må man finne den fundamentale frekvensen i signalet (Kumaraswamy & Poonacha, 2019). Det finnes ulike måter å gjøre dette på, men i *anyMidi* blir dette gjort ved å analysere overtonespektrumet til instrumentet.

Ettersom alle fysiske instrumenter er ulike i konstruksjon, må man kartlegge overtonespektrumet på noen måte. Vanlige bølgeformer har ofte kjente overtonespektrum, men analoge instrumenter har ikke nødvendigvis like bestemte spektrum. Måten problemstillingen ble løst på var å bruke en *plug-in* i Ableton Live som viser en grafisk fremstilling av overtonespektrumet. Dette gjør det mulig å kartlegge instrumentets frekvensinnhold.

Det ble gjort en systematisk datainnsamling ved å spille alle strenger—én etter én—i bånd 5, 10 og 17 på el-gitaren. Datagrunnlaget ble dermed 180 enkelttester. Dette ga god spredning i tonehøyde i tillegg til at både de mørke strengene, som er viklet, og de lyse strengene, som ikke er det, ble representert. For hver tone ble utslaget



Figur 2: Gjennomsnittlig deltone-utslag fra alle forsøk

i negative desibel til de ti første deltonene i spektrogrammet notert. Gjentakende mønster og hvordan de ulike strengene og båndene påvirket dem, kom på den måten til syne. All data fra forsøkene er å finne i Appendix A.

Forsøket viste at det var en del variasjon i overtonespektrumet ut fra hvilken strengtykkelse som ble spilt. Enda større variasjon var det om tonen ble spilt i høye eller lave bånd på gitaren. Figur 2 viser det gjennomsnittlige utslaget til de ti første deltonene basert på hele datagrunnlaget. Målsetningen var å plukke ut de n første deltonene i et signal ved å hente ut frekvensene med størst utslag. Ved å la $n = 5$ vil man utfra Figur 2 ende opp med å plukke ut deltoner 1, 2, 3, 4 og 6, siden deltone 6 er sterkere enn 5. Dette må man ta høyde for, og i tillegg er resultatene fra disse forsøkene ikke nødvendigvis dekkende nok. Samme problem oppstår ved å velge $n = 7$ eller $n = 9$. For å prøve å unngå slike problemer kan man velge $n = 6$, ettersom forkskjellen mellom alle deltoner 1-6 og 7 er relativt stor. Denne kunnskapen ga grunnlag for å lage en algoritme som analyserer overtonespektret til gitaren.

4 Resultater

Etter utviklingsprosessen fremstår *anyMidi* som et *proof of concept*, som til en viss grad klarer å gjennomføre den oppgaven den er designet for. Applikasjonen har derimot sine utfordringer, og forbedringspotensial på flere punkter. I denne delen gjennomgås de viktigste funksjonalitetene applikasjoner har og hvordan de relaterer til teorien.

4.1 Preprosessering

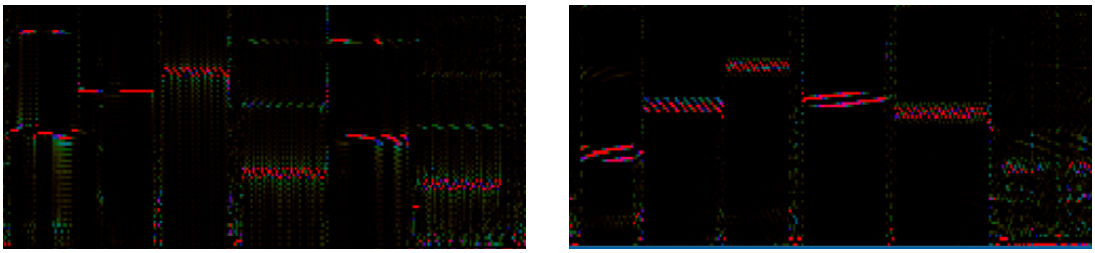
Det er viktig å ha gode rådata når man skal gjøre enhver form for analyse. Det gjør dataen enklere å håndtere og fører til færre problemer lengre ut i prosessen. Gitar er et analogt instrument, og dette innfører unøyaktigheter i signalet. I tillegg skal lyden fra gitaren gjennom elektronikk som påvirker signalet ytterligere. Dermed benyttes to sentrale verktøy for å minimere støy: filter og gate.

Det første som skjer med signalet, er at det blir sendt gjennom et IIR-filter. Filteret er satt til å filtrere ut alle frekvenser under 75 Hz. Grunnen til akkurat denne verdien er at en standardstemt gitar har sin laveste mulige frekvens på om lag 82 Hz, altså tonen E_2 (Suits, 2021). Filteret gir dermed litt slingringsmonn på mørkeste frekvens, men passer på at ingen dype frekvenser kan tilgrise signalet fra gitaren. Dette er særdeles viktig ettersom de mørkeste MIDI-notene er godt under denne frekvensen—faktisk helt ned til 8.18 Hz. Det er ikke ønskelig at gitaren skal kunne produsere disse MIDI-notene.

Etter å ha filtrert ut de mørkeste frekvensene går signalet gjennom en *noise gate*. I *anyMidi* skjer dette etter at FFT-prosessoren har begynt å arbeide med signalet. Når signalet er transformert til *bins*, blir alle frekvenser under en bestemt terskel, nullet ut. Denne terskelverdien er vilkårlig bestemt og er et resultat av testing under utviklingen. Dermed vil disse *bin*-ene, som mest sannsynlig kun inneholder støy, ikke påvirke den påventende analysen.

4.2 Vindusfunksjon

Som forklart tidligere er vindusfunksjoner sentrale for å unngå frekvenslekkasjer i en FFT. I *anyMidi* benyttes Hamming-vinduet på signalet. Grunnen til dette er at



(a) Uten bruk av vindusfunksjon (b) Med bruk av vindusfunksjon

Figur 3: Vindusfunksjon i praksis

Hamming-vinduet gir god oppløsning, mens det fremdeles gir rimelig god dynamisk rekkevidde (JUCE, 2021b). Det vil si at bredden til den midterste lappen i frekvensspekteret forblir relativt smal, noe som er viktig i dette tilfellet, hvor vi ønsker å presist analysere enkeltfrekvenser. Hadde den midterste lappen vært for bred ville det blitt svært vanskelig å bestemme hvilken frekvens man faktisk observerte. Figur 3 viser to spektrogrambilder av singaler fra el-gitar. *Bins* er her fordelt med lave frekvenser mot bunnen av bildet, og høye frekvenser mot toppen. Rød farge tilsvarer sterkere amplitude, mens sort tilsvarer ingen amplitudeutslag. Man ser at signalet i Figur 3a har mye mer 'frekvenslekkasjer' enn signalet i Figur 3b, hvor Hamming-vinduet er benyttet. Dette henger sammen med Figur 1b fra Seksjon 2, og viser effekten av vindusfunksjoner i praksis.

Siden vindusfunksjoner jevner ut signalet og dermed minsker utslaget til de ulike *bin*-ene, må man kompensere for dette for å representere amplitudeverdien korrekt. I *anyMidi* gjøres dette etter at FFT er gjort på signalbufferen. Siden Hamming-vinduet brukes, må amplitudene til alle *bin*-ene multipliseres med 1,85 (Siemens, 2019). Slik sikres det at amplitudeverdiene ikke blir for lave i forhold til hva signalet faktisk er.

4.3 Behandling av FFT-data

FFT-størrelsen i *anyMidi* er satt til 2048, som gir 1024 *bins*. Høyere FFT-størrelse gir bedre oppløsning av frekvenser, så det er ønskelig å ha denne så høy som mulig. I tillegg må FFT-størrelsen være en potens av 2 (Serov et al., 2021). Utprøving av applikasjonen har vist at denne FFT-størrelsen ikke introduserer merkbart mer forsinkelser i prosesseringen enn en lavere FFT-størrelse. Når signalbufferen er fylt opp

gjøres FFT på signalet, og dataen vil være klar til å bli behandlet.

Når hovedprosessen i *anyMidi* gjør et kall for å få ut analysen av FFT-en, kjøres to funksjoner først. Metoden `cleanUpBins` i *anyMidi* har som oppgave å gjøre FFT-dataen lettere å behandle. Det er i denne metoden *noise gate* er implementert. Ettersom Hamming-vinduet har gjort midtre lapp til hver frekvens bredere, prøver `cleanUpBins` å redusere bredden på lappen til å kun være én *bin*. Grunnen til at dette er viktig er at om én enkelt frekvens har lekket over til sin nabo-*bin*, kan det gjøre at denne *bin*-en får ganske høy amplitude. Videre er det mulig at begge disse *bin*-ene blir oppfattet som deltoner senere i prosessen. Det vil gjøre det vanskelig å bestemme den faktiske tonen, ettersom to nærliggende frekvenser kjemper om samme rolle.

Måten `cleanUpBins` løser dette på er å legge til alle *bins* med amplitude større enn 0 i en liste. Normalt sett vil påfølgende *bins* først stige i amplitude, for så å synke, og på den måte forme en lapp. Når en *bin* med amplitude 0 blir oppdaget etter en lapp, signaliserer dette at hele lappen ligger i listen. Deretter summeres alle amplitudeveridene sammen og skrives til den midterste av disse *bin*-ene. Amplitudene til resten av *bin*-ene i listen settes til 0. På den måten bevares amplitudeverdien til frekvensen som ble samplet, samtidig som at man har samlet lappen til kun én bin. I lave frekvenser vil denne gjennomsnittsregningen antagelig være nokså unøyaktig ettersom oppløsningen er lav. I de høye frekvensene derimot, vil dette ofte gi en ganske presis gjetning på den reelle frekvensverdien.

Etter at *bin*-ene i FFT-en er gjort litt enklere å behandle, kjøres funksjonen `mapBinsToNotes`. Funksjonen itererer gjennom alle *bin*-ene og regner ut frekvensen ved bruk av Likning (1) fra side 4. Viderer brukes en funksjon, `findNearestNote`, til å finne den korresponderende MIDI-noten til frekvensen. `findNearestNote` tar inn en liste, `noteFreq`, hvor indeksene representerer notetall på samme måte som MIDI-protokollen, mens verdiene er notens korresponderende frekvensverdi. En variant av binærsøk-algoritmen brukes for å søke etter frekvensen som er nærmest frekvensen til *bin*-en. Grunnen til at binærsøk benyttes, er fordi et ordinært søk ved å iterere over listen av noter vil kunne ta veldig lang tid. Innen algoritmeteori heter det at binærsøk har en kompleksitet på $\Theta(lg(n))$ (Cormen et al., 2009, s. 799–800). Et ordinært søk har kompleksitet på $\Theta(n)$ som er betraktelig høyere. I en applikasjon som denne, som skal

reprodusere musikalsk informasjon i sanntid, er det kritisk å unngå så mye forsinkelser som mulig.

Når frekvensen har blitt tilegnet et notetall, legges *bin*-ens amplitudeverdi til i en liste. Denne listen er bygget opp på samme måte som `noteFreq`, hvor indeksen tilsvarer notetallet. Listen benyttes i neste funksjon, `determineHarmonics`, som bestemmer hvilke *bins* som skal brukes i overtoneanalysen. I korte trekk henter denne funksjonen ut de n sterkeste *bin*-ene i FFT-en, hvor n er bestemt av parameteren `numPartials`.

Målet er at etter all foregående behandling av FFT-dataen, og med en nøye utvalgt verdi for `numPartials`, skal de sterkeste *bin*-ene i FFT-en representere deltonene til signalet fra gitaren.

4.4 Spektralanalyse

Som tidligere nevnt ble analyse av overtoner valgt som metode for å bestemme grunnfrekvensen. *Bins* i en FFT er lineært fordelt, men frekvensspektrumet normalt sett er representert på en logaritmisk skala. Det kommer av at hørselen vår opplever tonehøyde som proporsjonal med logaritmen av frekvens (Pigeon, 2021). Resultatet er dermed at FFT-en i praksis har mye lavere oppløsning i de lavere frekvensene enn i de høye. Det gjør at det er mer presist å analysere det øvre sjiktet av frekvensspektrumet til et signal. Ved å utnytte overtonespektrumet til et signal, kan man bestemme grunnfrekvensen uten å måtte belage seg fullstedig på å analysere *bins* med lav oppløsning.

Algoritmen som analyserer tone ut fra harmonisk spekter i applikasjonen, kan sees i Kode 1. `analyzeHarmonics` ser på deltonene et signal produserer, og gjør en vurdering basert på deres frekvenser. Som kjent kan man redusere *picket fencing* ved å utnytte de høye frekvensene av signalet. Ettersom overtoner er heltallsmultiplum av grunnfrekvensen kan man dividere frekvensene til overtonene med multiplumsverdien for å finne grunnfrekvensen. Dette forutsetter at deltonens nummer er kjent. Fra forsøket som ble gjort viste det seg at de seks første deltonene i gjennomsnitt var sterkere enn de resterende. Dermed kan disse seks deltonene hentes ut og itereres gjennom i stigende rekkefølge. For hver av dem divideres frekvensen deres med multiplumet (som i Kode 1 tilsvarer `i + 1`) for å finne grunnfrekvensen.

Kode 1 begynner, i linjer 3-4, med å hente ut de seks sterkeste deltonene fra

FFT-en (metoden er beskrevet tidligere). Videre itereres det gjennom hver deltone og frekvensverdien hentes ut i linje 12. I linje 14 divideres frekvensen med nummeret på deltonen (altså `i + 1`) for å finne grunnfrekvensen til deltonen. Deretter brukes en annen algoritme i linje 15 til å finne notetallet til frekvensen; dette notetallet korresponderer med notetall i MIDI. Notetallet gis i linje 17 en score med vekting utfra den binære logaritmen av frekvensverdien. Som diskutert opplever hørselen frekvenser som logaritmisk proporsjonale. Dermed gir denne verdien en god indikasjon på hvor god oppløsning den aktuelle frekvensen har. For å korrekt representere amplituden til signalet, legges alle deltonenes amplituder sammen til en total amplitude i linje 19. Videre i linjer 22-31 hentes notetallet med høyest score ut, og blir bestemt som den korrekt analyserte noten. Til sist returneres den analyserte tonen sammen med amplituden for å videre produsere en MIDI-tone.

Kode 1: Analyse av deltoner

```
1:  std::pair<int, double> MainComponent::analyzeHarmonics()
2:  {
3:      constexpr int numHarm{ 6 };
4:      auto harmonics = fft.getHarmonics(numHarm, noteFrequencies);
5:
6:      int fundamental{ 0 };
7:      double maxAmp{ 0.0 };
8:      std::map<int, double> scores;
9:      double totalAmp{ 0.0 };
10:     for (int i = 0; i < numHarm; ++i)
11:     {
12:         double freq = noteFrequencies[harmonics[i].first];
13:
14:         double fundamental = freq / (i + 1);
15:         int note = anyMidi::findNearestNote(fundamental, noteFrequencies);
16:
17:         scores[note] += 1.0 * log2(fundamental);
18:
19:         totalAmp += harmonics[i].second;
20:     }
21:
22:     int correctNote{ 0 };
23:     double maxScore{ 0.0 };
24:     for (std::pair<int, double> s : scores)
25:     {
26:         if (s.second > maxScore)
27:         {
28:             correctNote = s.first;
29:             maxScore = s.second;
30:         }
31:     }
32:
33:     std::pair<int, double> analyzedNote = std::make_pair(correctNote,
        totalAmp);
34:     return analyzedNote;
35: }
```

4.5 MIDI

MIDI-behandling i *anyMidi* er relativt enkel og består av å generere *note on*-meldinger basert på analyse av FFT-dataen, i tillegg til å generere *note off*-meldinger når en algoritme bestemmer at tonen er over. Avgjørelsen av notetallet gjøres på grunnlag av hva Kode 1 har avdekket, og notenummeret og amplitudeverdien herfra brukes til å lage en MIDI-note. Ettersom `analyzeHarmonics` ikke er begrenset av MIDI-protokollens maksimale noteverdi på 127, vil det kunne oppstå tilfeller hvor man får noteverdier utenfor MIDI-rekkevidde. I disse tilfellene vil MIDI-prosessoren håndtere avviket ved å la være å generere en MIDI-note.

Algoritmen som er utviklet for å bestemme hvorvidt en *note on* eller *note off* skal produseres heter `determineNoteValue` i kildekoden. Den baserer seg på å analysere amplitudeverdien til den nåværende tonen, samt å sammenligne denne med foregående tone. For å gjøre dette defineres et *attack threshold* og et *release threshold*. I tillegg vil foregående *note on* lagres i minnet, slik at den kan brukes i sammenligning med neste kandidat. Forholdene som må ligge til grunne for at en ny MIDI-note skal bli generert er én av følgende:

1. Hvis ingen note spiller for øyeblikket, behøver kun amplituden til gjeldende tone å være over *attack threshold* for å generere en ny *note on*.

2. Hvis en *note on* ikke er avløst av en *note off* i MIDI-prosessoren må en av disse betingelsene stemme for å generere en ny *note on*:

   (a) Det nye notetallet må være ulikt den foregående notetallet, i tillegg til å være over *attack threshold*, eller

   (b) Gjeldende amplitude må være en gitt størrelsesorden større enn amplitudeverdien til noten som allerede spiller (i *anyMidi* må den være 2 ganger større).

3. Hvis gjeldende notetall er den samme som forrige og amplituden har falt under *release threshold*, genereres en *note off* med dette notetallet.

Når en ny MIDI *note on* blir generert vil den benytte seg av amplitudeverdien

gitt av `analyzeHarmonics`. Denne verdien ligger i intervallet $[0, 1]$ og multipliseres med 127, som brukes til å bestemme MIDI-notens *velocity*.

4.6 Problemer

Som nevnt i begynnelsen av denne seksjonen, presterer *anyMidi* relativt bra. Den er særlig god til å korrekt analysere toner fra de lyse strengene. Likevel lider den av flere problemer. Det mest alvorlige problemet er at frekvensgjenkjenningen på gitarens mørke strenger er svært dårlig. Applikasjonen klarer ikke å analysere en enkelt tonehøyde, og ender opp med å produsere en strøm av MIDI-noter med svært varierende tonehøyde. Problemet blir verre desto mørkere streng og desto lavere bånd man spiller i. Strengtykkelsen synes å være den avgjørende faktoren, som betyr at de lyse strengene, som ikke er viklet, virker upåvirket.

Et annet problem er at applikasjonen kan slite med å finne riktig oktav. Den relative noten kan være riktig, men MIDI-notene som blir produsert kan raskt hoppe mellom oktaver over og under den faktiske tonen. Dette problemet er ikke like lett å teste på tykkere strenger ettersom analysen der allerede er veldig problematisk. Likvel er ikke oktavhoppene like prevalente på lyse strenger, noe som gjør at applikasjonen får vist fram funksjonaliteten sin i dette området.

Videre er MIDI-notenes tonehøyde noen ganger én halvnote feil i forhold til den spilte tonen. Dette er ikke et veldig fremtredende problem i de områdene *anyMidi* allerede presterer ganske bra. Ingen tydelig sammenheng er funnet mellom de frekvensområdene som opplever denne type feil. Likevel ser det ut til at tonene som blir feilanalysert er konsekvente i å bli feilanalysert.

Det siste sentrale problemet omhandler forsinkelser. Applikasjonen som den fremstår nå har såpass lav forsinkelse at den kan brukes i sanntid, men forsinkelsen er likevel merkbar. Som tidligere nevnt ser det ikke ut til at FFT-størrelsen har noen spesiell innvirkning på dette. Årsaken ligger dermed antagelig i en av algoritmene som benyttes i løpet av prosesseringen.

5 Drøfting

At *anyMidi* er et tildels fungerende program tyder på at implementasjon av teori til en viss grad har vært vellykket. Likevel opplever applikasjonen en rekke problemer som gjør at funksjonaliteten ikke er tilstrekkelig på alle punkter. Vi skal nå drøfte implementasjonsvalgene og deres begrunnelser, samt se på forbedringspotensialer og mulige feilgrep som er gjort under utviklingen.

5.1 Opprydding i frekvenser

Preprosesseringen som gjøres med singalet ble implementert for å redusere støy som kan problematisere analysen av FFT-en på et senere tidspunkt. IIR-filteret filterer ut frekvenser *under* den laveste frekvens gitaren produserer, men filtrerer ingenting *over* den høyeste frekvens gitaren produserer. Når spektralanalysen kun henter ut grunnfrekvensen og de fem første overtonene, er det ingen poeng i å ta med de resterende høye frekvensene. Disse frekvensene kan være en av årsakene til oktavhopp i MIDI-notene, og burde filtreres ut hvis de ikke skal benyttes i spektralanalyse i fremtiden.

Bruken av *noise gate* er også et grep som ble gjort for å unngå uønskede frekvenser i FFT-analysen. Samtidig muliggjør implementasjonen av *noise gate* funksjonaliteten til `cleanUpBins`-metoden, som er avhengig av at det eksisterer *bins* med amplitudeverdi 0. Den bruker disse *bin*-ene til å bestemme hvor frekvenslapper starter og slutter. Likevel kunne metoden vært implementert med et annet *threshold* større enn 0. Dette hadde på en annen side vært ekvivalent med å bruke *noise gate* på FFT-dataen.

Eksistensen til metoden `cleanUpBins` er en direkte konsekvens av at Hamming-vinduet brukes på signalet. Som allerede diskutert, reduserer Hamming-vinduet amplituden til 'lekkasjefrekvenser' (som opptrer som sidelapper i frekvensdomenet), men det øker også bredden på midtlappen. Denne brede midtlappen er nemlig det `cleanUpBins` prøver å komprimere. Hvis midtlappen blir for bred kan andre *bins* bli påvirket og dermed gjøre at flere nærliggende *bins* tolkes som deltoner i spektralanalysen. `cleanUpBins` slår alle *bin*-ene som er påvirket av midtlappen sammen til én *bin* og komprimerer dermed lappen til én frekvens.

Dette stiller derimot spørsmålet: er det nødvendig å bruke vindusfunksjon, når

en annen algoritme fjerner både sidelapper og midtlapper likevel. Svaret er antagelig at det er nødvendig, ettersom reduksjon av 'lekkasjefrekvenser' mest sannsynlig har et stort helhetlig utslag. Likevel kunne det vært interessant å se på hvor godt `cleanUpBins` hadde klart seg på egenhånd, og om Hamming-vinduet gjør prosessen vanskeligere for metoden. Videre kunne man eksperimentert med andre typer vindusfunksjoner som reduserer sidelappene i mindre grad, men som til gjengjeld har høy oppløsning. I kombinasjon med en *noise gate* med høyere *threshold*—som muligens kan fjerne noen av lekkasjefrekvensene—kunne det kanskje gitt bedre resultater.

5.2 Overtoneanalyse

Bruken av overtoneanalyse som en måte å bestemme grunnfrekvensen er et godt utgangspunk. Ved å utnytte FFT-ens høye oppløsning i høye frekvenser, arbeider man direkte med å unngå *picket fencing*, som er definert som et sentralt problem ved bruk av FFT. De lave frekvensene til et vilkårlig singal vil sjeldent treffe nøyaktig i frekvensen til en *bin*, og vil dermed ofte bli feilrepresentert i en FFT. Kunnskapen om overtonespektrumet til signalkilden gir et stort fortrinn ettersom man utfra andre frekvenser i spektrumet kan si noe om grunnfrekvensen.

Måten overtoneanalyse er implementert har dog forbedringspotensial. Slik overtoner bestemmes nå er ved å hente ut *bin*-ene med størst amplitude. Grunnen til at dette til en viss grad fungerer i denne applikasjonen er fordi tonegjenkjenningen er monofonisk. Det harmoniske spektrumet er mye mindre komplekst enn om signalet var polyfonisk, og det er dermed enklere å analysere. En alternativ måte å bestemme deltoner på kunne vært ved å først gjøre en gjetning på grunnfrekvensen—f.eks. slik *anyMidi* gjør nå. Deretter kunne man sjekket frekvenser hvor det er forventet å finne overtoner, for så å finjustere grunnfrekvensen utfra denne informasjonen. En slik metode med flere iterasjoner kunne gitt mer presise resultater. Likevel er det en fare for at dette ville introdusert for mye forsinkelse i prosesseringen til at det hadde vært akseptabelt.

Antall deltoner som blir analysert er på dette tidspunktet seks. Grunnlaget for dette ligger i data fra et uformelt forsøk gjort med 180 tester. Selv om applikasjonen klarer å prestere i sin oppgave til en viss grad, behøver det ikke bety at spektralana-

lysen er gjort på en optimal måte.

I Seksjon 3 diskuteres muligheten for å velge ut enda flere overtoner fra signalet. Dette ville tillatt utnyttelsen av enda høyere frekvenser—igjen, med enda bedre oppløsning. Høy oppløsning gjør det mulig å mer nøyaktig kunne bestemme frekvensene til deltonene. Forsøket viste derimot at det kunne være vanskelig å bestemme hvilke multiplum av grunnfrekvensen de høye deltonene var. Dette kan gjøre analysen upresis. Likevel brukes allerede flere deltoner til å bestemme grunnfrekvensen, som et grep for å ikke sette all tillit til én av deltonene i et signal. Dette er et prinsipp som fremdeles gjelder selv om analysen skulle behandle flere deltoner.

5.3 Musikalitet

En av målsetningene for denne oppgaven var å utforske hvordan man bevarer musikaliteten av en opptreden når man konverterer denne til MIDI. Denne problemstillingen besvares ikke av enkeltavgjørelser, men heller av summen av alle valg. Siden poenget i denne oppgaven er å produsere MIDI, er ikke klanginformasjonen fra instrumentet interessant. Hovedoppgaven blir dermed å finne tonehøyden til signalet, som gjøres av den allerede diskuterte spektralanalysen.

Den nest viktigste parameteren er anslaget til tonen. I MIDI er dette kjent som *velocity* og i FFT er det kjent som amplitude. Som vi har sett gjøres det flere grep for å bevare denne informasjonen fra instrumentet. I FFT-en gjøres det kompensasjon på alle *bins* for å motvirke amplitudereduksjonen Hamming-vinduet skaper. Under spektralanalysen legges alle deltonenes amplituder sammen for å ivareta det opplevde anlsaget fra det fysiske instrumentet. Amplitudeinformasjonen gis deretter videre til MIDI-prosessoren, som omgjør dette til *velocity*.

Likevel er det noen momenter som ikke er i tråd med en musikalsk konvertering. Forsinkelser introdusert av prosessering gjør opplevelsen mindre musikalsk. Det er mulig å bruke applikasjonen til å spille inn MIDI i sanntid, men det forutsetter bruk av f.eks. kvantiseringshjelpemidler i en DAW. Oktavfeil og andre artefakter som oppstår under konverteringen gjør også at *anyMidi* ikke er klar for praktisk bruk. På en annen side viser applikasjonens funksjonalitet på dette tidspunktet at musikalsk representasjon er en mulighet i fremtiden.

5.4 Alternative algoritmer

I en eventuell videreutvikling av *anyMidi*, kan det være interessant å se på alternative løsninger og algoritmer. Det finnes utallige måter å gjøre *pitch detection* på, og løsningene presentert i denne oppgaven er bare et knippe forslag, utarbeidet over en kort periode. FFT i seg selv behøver heller ikke være den optimale måten å løse oppgavens mål på, og det kan være gunstig å refaktorere hele kodebasen.

De La Cuadra et al. (2001) presenterer flere ulike algoritmer for å finne tonehøyde av et signal. Blant annet er det interessant å se på ACF-algoritmen (*autocorrelation function*). Denne jobber i tidsdomenet og ikke i frekvensdomenet, slik som *anyMidi* gjør. ACF måler hvor mye et signal korrelerer med en tidsforskjøvet versjon av seg selv. Periodiske signaler korrelerer sterkt med seg selv hvis den er tidsforskjøvet med den fundamentale perioden til signalet (De La Cuadra et al., 2001, s. 3). Denne algoritmen diskuteres også av Kumaraswamy og Poonacha (2019). Her presenteres en metode som reduserer oktavfeil i gjenkjenning av tonehøyde. Vi har allerede diskutert at oktavfeil er et problem i *anyMidi*, noe som gjør denne løsningen interessant. Det underbygger spørsmålet om hvorvidt valget av metode for denne oppgaven ikke er optimalt. Muligens har spektralanalyse av en FFT begrensede muligheter, og vil komme til kort uansett hvordan man velger å analysere dataen.

Et annet eksempel er Klapuri (2006) som bruker Fouriertransformasjoner i sin analyse. Her baserer analysen seg på informasjon fra overtoner—slik som denne oppgaven. Det diskuteres måter å optimalisere vektleggingen av deltoner, hvor maskinlæring trekkes inn som en måte å bestemme denne vektingen på. Denne innfallsvinkelen har noe til felles med metodene som brukes i denne oppgaven, hvor det er samlet inn data om overtonespektrumet til gitaren. Ved å ha et enda større datagrunnlag og formalisere tolkningen av dataen, kunne algoritmene fra denne oppgaven muligens fremdeles benyttes, med noen modifikasjoner.

6 Konklusjon

I denne oppgaven har det blitt sett nærmere på hvordan man kan konvertere fra spektraldomenet til MIDI ved hjelp av FFT-algoritmen. Bakgrunnen for dette har vært å utforske alternative måter å produsere MIDI på, som tillater bruken av andre instrumenter enn keyboard til å gjøre oppgaven. Målet har vært å utforske hvilke grep som må tas for å bevare musikalitet i konverteringen mellom lyd og MIDI, samt undersøke teknikker for å korrigere og minske feil i prosesseringen.

Blant korreksjonsteknikker har det blitt sett på preprosessering i form av *noise gate* og filter. I tillegg har vi gått nøye inn på vindusfunksjoner og hvordan det er et kompromiss mellom lekkasjefrekvenser og frekvensoppløsning. Det mest sentrale momentet i oppgaven har vært utførelsen av spektralanalyse og hvordan man kan omgå FFT-ens begrensninger i frekvensoppløsning ved å bruke et instruments overtonespektrum.

Sammen forsøker disse teknikkene å ivareta musikalitet. En av måtene dette gjøres på er ved å kompensere for amplitudereduksjon. Spektralanalysen på sin side prøver å finne kun tonehøyden til signalet, og ser bort ifra instrumentets klangfarge. Til sist er valget av FFT og andre algoritmer med lav kompleksitet et forsøk på å minske forsinkelser i prosesseringen, som kan være forstyrrende i en musikalsk opptreden.

Som en måte å teste ut disse aspektene i praksis ble en applikasjonen *anyMidi* utviklet for å utføre oppgaven å konvertere signalet fra en el-gitar til MIDI. Ved prosjektets slutt klarer denne applikasjonen å utføre noen av oppgavene som var målsatt. Samtidig har *anyMidi* flere problemer som gjør at den ikke presterer optimalt. Den opplever oktavfeil, feiltolker i noen tilfeller tonehøyden til signalet og kan ikke brukes på de mørke strengene. I tillegg opplever applikasjonen forsinkelser i prosesseringen.

Det har blitt diskutert hvilke forbedringer som kan gjøres for å optimalisere algoritmene i denne oppgaven. Muligens finnes det også andre metoder enn FFT som bedre kan oppnå oppgavens målsetning. Isåfall må problemet få en helt ny innfallsvinkel, og ny teori må trekkes inn. Likevel viser det seg at det fremdeles kan være gunstig å bruke FFT. Uavhengig av hvilken retning som måtte velges, er det et klart behov for videre utvikling av algoritmer og videre utforskning av teori rundt temaet.

Referanser

Ableton. (2021). https://www.ableton.com/en/manual/converting-audio-to-midi/

Airy, S. & Parr, J. M. (2001). MIDI, music and me: students' perspective on composing with MIDI. *Musical Education Research*, *3*(1), 41–49.

Carral, S. & Paset, M. (2008). The influence of plectrum thickness on the radiated sound of the guitar. *The Journal of the Acoustical Society of America*, *123*(5), 5647–5652.

Cormen, T. H., Leiserson, C. E., Rivest, R. L. & Stein, C. (2009). *Introduction to algorithms* (3. utg.). The MIT Press.

De La Cuadra, P., Master, A. S. & Sapp, C. (2001). Efficient pitch detection techniques for interactive music. *ICMC*.

Girgis, A. A. & Ham, F. M. (1980). A quantitative study of pitfalls in the FFT. *IEEE Transactions on Aerospace and Electronic Systems*, *AES-16*(4), 434–439.

Huber, D. M. (2020). *The MIDI Manual: A Practical Guide to MIDI within Modern Music Production*. Taylor & Francis.

Jam Origin. (2021a). MIDI Bass & MIDI Guitar Features. https://www.jamorigin.com/docs/features/

Jam Origin. (2021b). Technology. https://www.jamorigin.com/technology/

JUCE. (2021a). Made with JUCE. https://juce.com/discover/made-with-juce

JUCE. (2021b). Tutorial: Visualise the frequencies of a signal in real time. https://docs.juce.com/master/tutorial_spectrum_analyser.html

Klapuri, A. (2006). Multiple fundamental frequency estimation by summing harmonic amplitudes. *ISMIR 2006 - 7th International Conference on Music Information Retrieval*, 216–221.

Kumaraswamy, B. & Poonacha, P. G. (2019). Octave Error Reduction in Pitch Detection Algorithms Using Fourier Series Approximation Method. *IETE Technical Review*, *36*(3), 293–302. https://doi.org/10.1080/02564602.2018.1465859

Moore, F. R. (1988). The dysfunctions of MIDI. *Computer Music Journal*, *12*(1), 19–29.

Pigeon, S. (2021). The non-linearities of the Human Ear. https://www.audiocheck.net/soundtests_nonlinear.php

Rockmore, D. N. (2000). The FFT: an algorithm the whole family can use. *Computing in Science & Engineering*, *2*(1), 60–64.

Serov, A. N., Chumachenko, D. A. & Shatokhin, A. A. (2021). Method of Reducing the Amplitude and Phase Spectrum Measurement Error of the Power Quality Measuring Instruments. *2021 3rd International Youth Conference on Radio Electronics, Electrical and Power Engineering (REEPE)*, 1–8.

Siemens. (2019). Window Correction Factors. https://community.sw.siemens.com/s/article/window-correction-factors

Steiglitz, K. (1996). *A digital signal processing primer: with application to digital audio and computer music*. Addison-Wesley Publishing Company.

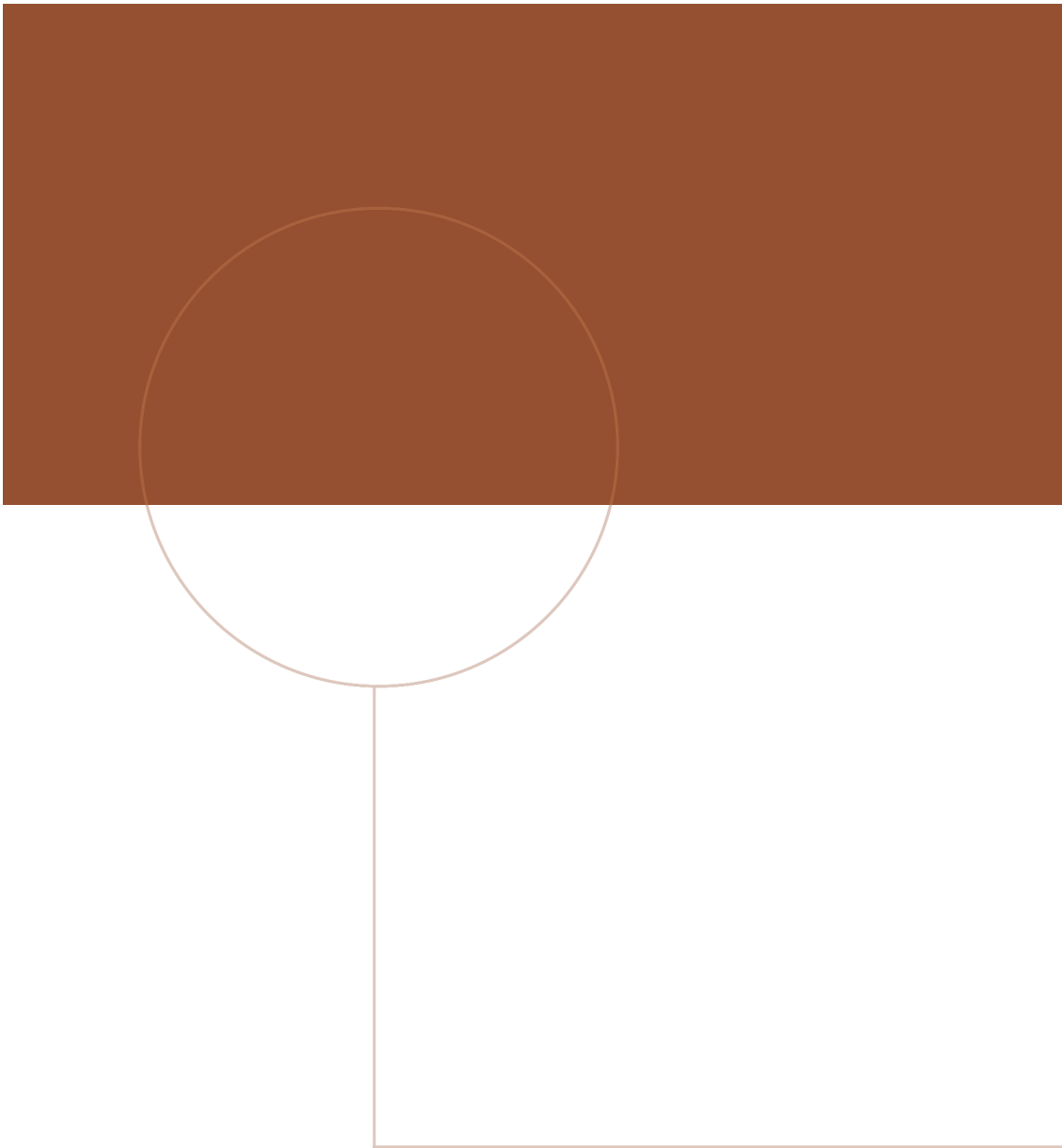
Suits, B. H. (2021). Tuning. https://pages.mtu.edu/~suits/notefreqs.html

Vedlegg A Kartlegging av deltoner i gitarsignal

Note	Streng	Bånd	1. deltone	2. deltone	3. deltone	4. deltone	5. deltone	6. deltone	7. deltone	8. deltone	9. deltone	10. deltone
A4	1	5	-27,0	-25,0	-28,2	-40,4	-47,9	-46,3	-55,1	-42,7	-43,1	-53,5
E4	2	5	-26,0	-26,3	-36,1	-49,8	-44,7	-45,5	-57,0	-46,4	-45,8	-47,7
C4	3	5	-23,3	-22,0	-31,6	-44,0	-43,2	-43,8	-52,1	-42,4	-38,9	-48,0
G3	4	5	-28,7	-24,0	-31,5	-44,8	-48,4	-46,1	-52,9	-40,3	-41,7	-45,1
D3	5	5	-27,7	-23,6	-30,5	-46,6	-46,8	-47,1	-55,1	-41,3	-37,2	-46,8
A2	6	5	-23,4	-21,0	-27,7	-43,9	-41,1	-41,7	-49,0	-39,3	-38,4	-40,9
	Lave bånd:		**-26,0**	**-23,7**	**-30,9**	**-44,9**	**-45,4**	**-45,1**	**-53,5**	**-42,1**	**-40,9**	**-47,0**
D5	1	10	-25,0	-29,5	-43,3	-46,2	-57,6	-43,1	-40,8	-53,7	-66	-56,2
A4	2	10	-23,2	-26,9	-42,3	-37,1	-49,5	-38,9	-37,4	-48,9	-52,2	-52,2
F4	3	10	-21,2	-26,1	-43,8	-33,7	-45,9	-33,9	-33,1	-43,5	-45,1	-48,6
C4	4	10	-26,3	-28,3	-45,3	-39,5	-51,3	-39,4	-37,7	-43,8	-46,4	-43,0
G3	5	10	-33,0	-33,4	-41,8	-44,7	-41,8	-41,3	-45,8	-43,2	-46,9	-52,6
D3	6	10	-23,5	-24,6	-42,1	-30,7	-42,1	-34,0	-29,7	-44,6	-38,3	-40,5
	Midtre bånd:		**-25,4**	**-28,1**	**-43,1**	**-38,7**	**-48,0**	**-38,4**	**-37,4**	**-46,3**	**-49,2**	**-48,9**
A5	1	17	-21,3	-38,4	-43,2	-38,5	-45,0	-56,7	-60,1	-61,2	-77,3	-69,5
E5	2	17	-20,6	-40,6	-38,2	-34,1	-41,5	-44,9	-53,4	-57,5	-75,9	-67,9
C5	3	17	-19,5	-41,7	-32,5	-32,9	-37,6	-41,4	-52,9	-56,5	-75,4	-71,2
G4	4	17	-22,0	-43,6	-43,3	-34,3	-38,6	-42,2	-48,4	-53,7	-70,8	-57,2
D4	5	17	-23,5	-48,0	-34,0	-31,4	-34,0	-40,7	-45,1	-46,5	-57,9	-49,9
A3	6	17	-19,0	-35,1	-31,0	-30,7	-30,4	-33,3	-41,7	-43,2	-50,5	-50,0
	Høye bånd:		**-21,0**	**-41,2**	**-37,0**	**-33,7**	**-37,9**	**-43,2**	**-50,3**	**-53,1**	**-68,0**	**-61,0**
	Lyse strenger:		-23,0	-30,7	-37,7	-39,6	-45,9	-43,8	-49,1	-50,3	-57,7	-57,2
	Mørke strenger:		-25,2	-31,3	-36,4	-38,5	-41,6	-40,6	-45,0	-44,0	-47,6	-47,3
	Alle strenger:		-24,1	-31,0	-37,0	-39,1	-43,7	-42,2	-47,1	-47,2	-52,7	-52,3

Vedlegg B Kildekode for anyMidi

Kildekoden til applikasjonen *anyMidi* er lagt ved som en separat zip-fil til denne oppgaven. Siste versjon av applikasjonen er også å finne på Github under denne lenken: https://github.com/EwanMe/anyMidi.



NTNU

Kunnskap for en bedre verden