VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

SOFTWAROVÝ KYTAROVÝ LOOPER S DÁLKOVÝM OVLÁDÁNÍM
SOFTWARE GUITAR LOOPER WITH REMOTE CONTROL

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Jan Kalník**
AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. Jiří Schimmel, Ph.D.**
SUPERVISOR

BRNO 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Audio inženýrství**
Ústav telekomunikací

Student: Bc. Jan Kalník ***ID:*** 154758
Ročník: 2 ***Akademický rok:*** 2017/18

NÁZEV TÉMATU:

Softwarový kytarový looper s dálkovým ovládáním

POKYNY PRO VYPRACOVÁNÍ:

V prostředí Pure Data navrhněte a realizujte softwarový kytarový looper s rozpoznáváním začátků hudebních událostí. K tomu využijte některý z algoritmů tzv. „onset“ detektorů, který samostatně realizujete jako modul (tzv. „external“) prostředí Pure Data. Dále realizujte hardwarový dálkový ovladač pro řízení základních funkcí looperu pomocí protokolu MIDI.

DOPORUČENÁ LITERATURA:

[1] KAVAN, Jan. Pure Data: platforma pro tvorbu interaktivního díla. Brno: Janáčkova akademie múzických umění v Brně, 2013. ISBN 8074600335, 234 stran.

[2] CHUNG. B. Multimedia Programming with Pure Data. Packt Publishing Ltd., 2013. ISBN> 978-1-78216-464-7

Termín zadání: 5.2.2018 ***Termín odevzdání:*** 21.5.2018

Vedoucí práce: doc. Ing. Jiří Schimmel, Ph.D.
Konzultant:

prof. Ing. Jiří Mišurec, CSc.
předseda oborové rady

ABSTRAKT

Tato práce se zabývá vytvořením softwarového kytarového looperu v graficém programovacím jazyce Pure Data a následným sestrojením jeho dálkového ovladače. Tento ovladač bude realizován ve formě MIDI kontroleru. Součástí programu bude zároveň zarovnávací algoritmus, který se postará o přesné zarovnání vytvořené smyčky na dobu tak, aby nevznikaly nežádoucí rytmické či tempově závislé nedokonalosti.

KLÍČOVÁ SLOVA

Looper, kytara, algoritmus, kontroler, MIDI, Pure Data

ABSTRACT

This thesis deals with creating a software guitar looper in a visual programming language called Pure Data and also building a specific remote controller for it. The remote controller will be based on a MIDI protocol. As a part of the software, there will be a special aligning algorithm, that will take care of precise alignment of the created loop on beat in the way that no other intrusive rhythmical or tempo related elements are present.

KEYWORDS

Looper, guitar, algorithm, controller, MIDI, Pure Data

KALNÍK, Jan. *Softwarový kytarový looper s dálkovým ovládáním*. Brno, 2017, 40 s. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: doc. Ing. Jiří Schimmel, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Softwarový kytarový looper s dálkovým ovládáním“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu doc. Ing. Jiřímu Schimmelovi, Ph.D. a zároveň také panu MgA. Janu Kavanovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno ....

podpis autora

OBSAH

SEZNAM OBRÁZKŮ

ÚVOD

Muzika se již od pradávna zakládá na repetici. To, že se některé části, jako například refrény skladeb, opakují, nám dává určitý pocit jistoty, ale také v nás probouzí pocit radosti z toho, že určitou pasáž rozpoznáváme. Můžeme si toho všimnou i ve skladbách slavných autorů, jako jsou například *J. S. Bach*, *W. A. Mozart*, *J. Pachelbel*, aj. Speciální typ repetice v hudbě se nazývá kánon, který můžeme slyšet například ve skladbě *Canon in D* od výše zmíněného *J. Pachelbela*.

V dnešní době se těchto pricipů využívá například pomocí tzv. „*looperů*“. Jedná se o jedno až několika pedálové krabičkové efekty, které můžou být použity s různými nástroji. Nejčastěji je však vidíme ve spojení s kytarami nebo mikrofony. Pomocí těchto efektů můžeme nahrát až libovolný počet vrstev v jedné nezávislé smyčce. Délka smyčky, počet nezávislých smyček a také počet nástrojových nebo mikrofonních vstupů se ale různí podle výrobce.

Takovýto looper však ztrácí modularitu. Proto jsme se rozhodli jít cestou softwarovou, která skýtá nespočet možností, a vytvořit tak looper v programu Pure Data (dále také Pd). Takto vytvořený looper nebude nijak omezen vstupy ani výstupy a bude záležet pouze na audio rozhraní uživatele. Zároveň bude také modulární podle potřeb uživatele. Pd jsme vybrali proto, že jde o volně dostupný, multi-platformní software a je tak přístupný každému uživateli.

První část práce bude zaměřena na teoretické znalosti, obecné seznámení s problémem, přehled možných algoritmů, jejich porovnání a následný výběr nejvhodnějšího kandidáta. V první kapitole se dozvíme bližší informace o looperech, jak se používají a na jakém principu fungují. Zde si také vysvětlíme celkový proces vytváření smyčky a jejího zpracovávání.

Kapitola druhá se bude zabývat grafickým programovacím jazykem Pure Data. Objasníme si, jak se v tomto jazyku pracuje a jakým způsobem jej lze využít k vytváření vlastních programů. Zároveň se blíže podíváme na to, jak vytvořit externí knihovnu (tzv. *external*) a objasníme si rozdíl mezi abstrakcemi a subpatchi.

Ve třetí kapitole představíme problém zarovnávání na dobu. Zde bude snahou navrhnout speciální algoritmus, který bude pracovat na principu detekce počátku dob a bude zajišťovat to, aby byla smyčka synchronní a bez jakýchkoliv skoků nebo rytmických niancí mezi koncem a opakovaným začátkem smyčky.

Čtvrtá kapitola nám přiblíží možné algoritmy pro zarovnání fráze na dobu a také zde proběhne porovnání určitých algoritmů. Na konci části bude vybrán algoritmus, který posléze budeme implementovat do prostředí Pure Data.

Druhá část práce bude zaměřená na implementaci algoritmu do Pure Data, realizaci

samotného looperu, jeho popis a seznámení s s grafickým návrhem v Pd. V neposlední řadě bude cílem této práce vyrobit jednoduchý hardwarový ovladač pro tento looper, který bude komunikovat se softwarem skrz protokol MIDI (Komunikační protokol – Musical Instrument Digital Interface).

1 TEORETICKÁ ČÁST STUDENTSKÉ PRÁCE

1.1 Looper

Jak již bylo zmíněno výše, looper, neboli česky smyčkovač, je efekt, díky kterému můžeme nahrávat krátké fráze, které se poté opakují, dokud je nezastavíme. Na dané fráze můžeme také vrstvit další fráze, které se poté přehrávají společně s předešlými frázemi. Na trhu existují různé typy looperů. Rozdělit je můžeme do dvou základních kategorií:

- jednopedálové
- vícepedálové

Jednopedálové loopery

Tento typ looperů je specifický tím, že má pouze jeden pedál a tím jsou ovládány všechny jeho funkce. Má tím pádem jednu stopu, do které se dá nahrávat. Mezi jeho přednosti patří kompaktnost, nízká hmotnost, jednoduchost a příznivá pořizovací cena. Tento looper je vhodný pro začátečníky nebo pro muzikanty, kteří jej chtějí používat hlavně na trénování a zdokonalení se. Dá se však použít i jako profesionální nástroj na pódiu.

Vícepedálové loopery

U tohoto typu je naopak výrazně horší kompaktnost, jelikož je looper složen ze dvou (i více) pedálů. S tím souvisí větší hmotnost. Existují ale i speciální typy, které mají místo pedálů pouze menší spínací tlačítka a jsou tak o něco lehčí a kompaktnější. Tyto loopery jsou ale používány výhradně pro práci s mikrofonem, jelikož je třeba tato tlačítka spínat ručně a nikoliv nohou. Mezi přednosti vícepedálových looperů patří jejich sofistikovanější funkce. Díky více pedálům mají totiž více na sobě nezávislých stop, do kterých je možno nahrávat. Tato funkce je velmi zásadní pro muzikanty, kteří chtějí zařadit například *live looping*[1] do svého repertoáru.

Oba tyto loopery mají však jednu zásadní vadu a tou je jejich konektivita. Žádný z nich totiž nedisponuje více než pěti vstupy a pokud ano, tak jsou to například tyto vstupy: 1 mikrofonní (XLR), 2 nástrojové (JACK) a 1 stereo pomocný vstup (AUX)

[1] Forma loopingu, kde je umělec většinou sám na pódiu a používá looper pro tvorbu představení v reálném čase. Ke tvorbě smyček se mohou použít různé nástroje, samply, apod.

[1]. Příklady jednopedálových i vícepedálových looperů můžeme vidět na obrázku 1.3.

Obr. 1.1: Jednopedálový looper Boss RC-3

Obr. 1.2: Vícepedálový looper Boss RC-30

Obr. 1.3: Příklad jednopedálových a vícepedálových looperů

1.1.1 Ovládání looperu

Loopery jsou většinou krabičkové efekty, vzhledově podobné kytarovým efektům jako jsou například distortion, reverb, compressor, aj. Znamená to tedy, že se ovládají stiskem pedálu nohou. Existuje ale i varianta pouze se spínacími tlačítky [3], které se ovládají ručně.

První smyčka se tvoří vždy stejně a to tak, že se sešlápne pedál (nebo jakákoliv jiná alternativa, např. v podobě tlačítka) a v ten moment začne looper nahrávat veškerý signál, co do něj posíláme. Konec smyčky vytvoříme opětovným stisknutím téhož pedálu. Dále existují dvě hlavní varianty následného vrstvení smyček.

Record/Play/Overdub

Tato varianta je vhodná pro začátečníky. Jedná se o to, že při druhém stisku pedálu se za prvé ukončí první smyčka a za druhé se začne tato smyčka přehrávat od začátku, pořád dokola. Umělec tak může hrát do smyčky, aniž by si ji jakkoliv přehrál. Pokud se rozhodneme, že chceme k této smyčce přidat další vrstvu, zmáčkneme pedál potřetí a v daném místě začneme vrstvit další smyčku na smyčku původní. Opětovným stisknutím pedálu se dostáváme zpět do módu přehrávání a tento postup se analogicky opakuje, dokud se nerozhodneme smyčku zastavit, nebo jakkoliv jinak přerušit.

Record/Overdub/Play

Varianta vhodnější pro pokročilejší uživatele. Funguje podobně jako varianta první pouze s tím rozdílem, že při druhém stisku se dostáváme místo do módu přehrávání

rovnou do módu vrstvení. Poté funguje tato varianta stejně jako varianta první. Tato funkce výrazně zrychlí tvorbu smyček v reálném čase a proto se používá hlavně při live loopingu. Umělec totiž neztrácí čas přehráváním smyčky a rovnou tvoří další vrstvu. Tento princip je u live loopingu velmi důležitý, jelikož jde o to, abyste posluchače nenudili a tím pádem se zbytečně nezdržovali přehráváním smyčky, ale rovnou mohli vrstvit.

Při zhotovování našeho looperu budeme využívat této varianty, jelikož se bude jednat o looper specializovaný hlavně na live looping.

Nyní máme smyčku připravenou, víme jak vytvářet další vrstvy a co nám zbývá je tedy práce a úprava smyčky.

Některé loopery, většinou ty vícepedálové, mají kromě více nezávislých smyček na sobě také jednu „výhodu“. Mají totiž vestavěné efekty jako je například reverb, reverse, modulátor a jiné a tyto efekty můžeme na vytvořenou smyčku aplikovat. Nebo je také můžeme použít při vrstvení a frázi obohacenou efektem tak přímo do smyčky nahrát. Důvod, proč jsem slovo výhoda dal do uvozovek je ten, že mně osobně tyto věci nepřijdou jako výhoda. Je totiž pravdou, že spousta umělců by se bez těchto efektů obešla, tím pádem se díky nim uměle a zbytečně navyšuje cena produktu. Pokud by umělec přece jen efekt potřeboval, může si jej dle mého názoru dokoupit zvlášť. Jde totiž o to, že takto vybavený looper postrádá jednoduchost ovládání a je naopak velmi snadné udělat nějakou chybu při vystoupení. Samozřejmě existují ale i umělci, kteří tyto efekty uvítají přímo v looperu nehledě na přehlednost a jednoduchost, kterou tím dle mého názoru ztrácí. Proto si myslím, že jít cestou softwarovou je cesta menšího odporu, větší modularity, přehlednosti a jednoduchosti. V dnešní době totiž kvalita a rychlost výpočetní techniky exponenciálně stoupá a my se tak můžeme mnohem snadněji přiblížit latenci[2] analogových efektů.

V neposlední řadě je třeba smyčku nějak zastavit. To probírá buď rychlým dvojstiskem pedálu (u jednopedálových looperů) nebo jedním stiskem příslušného pedálu na zastavení smyčky (u vícepedálových looperů).

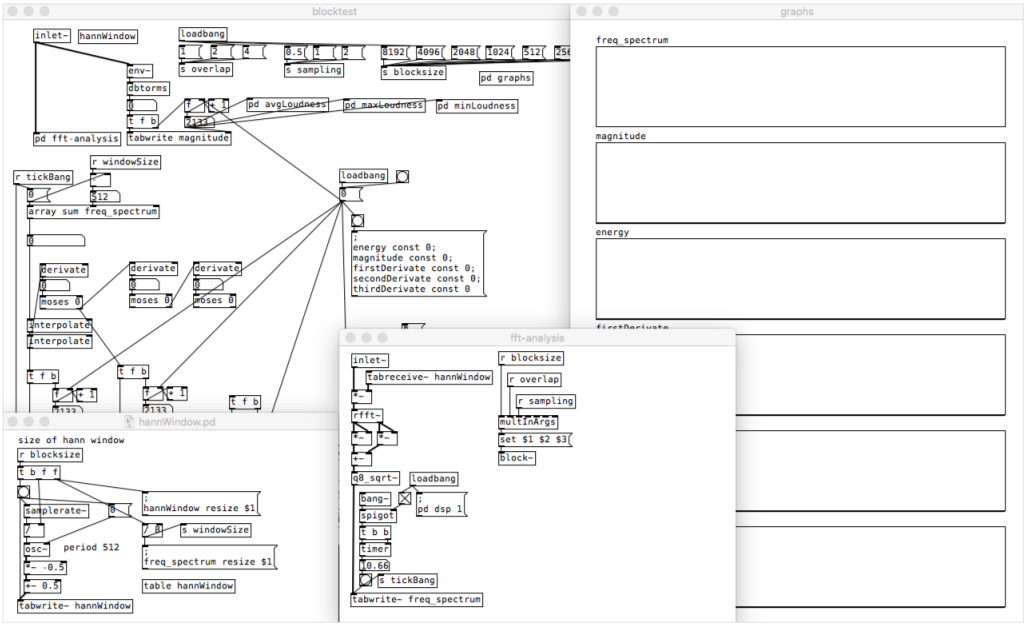
Smazání smyčky docílíme rychlým dvojstiskem a přídržením pedálu po delší dobu (cca 2 sekundy) nebo opět dedikovaným tlačítkem na smazání smyčky. U vícepedálových looperů mohou být tyto funkce u každé z nezávislých stop a poté pro výslednou smyčku jako celek. Příkladem je Boss RC-300 [1].

[2]Doba uplynulá mezi akcí a reakcí pozorovaného systému, [5].

1.2 Pure Data

V této kapitole se seznámíne s programovacím jazykem Pure Data, který bude využíván pro tvorbu softwarové části diplomové práce.

Jedná se o grafický programovací jazyk, který se dá využít pro tvorbu interaktivní počítačívé hudby, zejména různých audio či video projektů [9], [10]. To je oblast, kde Pd opravdu září. Můžeme jej ale použít i na tvorbu jiných interaktivních programů. Představivosti se meze nekladou.



Obr. 1.4: Otevřený projekt s několika patchi v Pure Data.

1.2.1 Historie

V 90. letech minulého století se Miller Puckette rozhodl, že vyvine program, který bude dostupný všem a díky kterému budou moci uživatelé vytvářet různé interaktivní, multimediální projekty. A tak vznikl program Pure Data. Můžete jej vidět, s několika otevřenými patchi, na obrázku 1.4. I přesto, že je jako zakladatel považován Miller Puckette, jedná se o open source software [8]. Znamená to tedy, že se může kdokoliv podílet na vývoji tohoto programu. Zároveň je zde velmi široká komunita lidí, kteří vytvářejí externí knihovy. O těch bude ale řeč v kapitole 1.2.4.

Puckette byl zastáncem toho, že software má být přístupný pro lidi zdarma. To také přispělo k tomu, že se rozhodl po hádce odstoupit od týmu vývojářů programu Max/MSP [6]. Tento program je velice podobný Pd, dokonce se dá říci, že sdílí stejný princip a také jádro programu. Max/MSP je ovšem placená verze. To se Puckettovi nelíbilo a tak se od projektu trhl. Další z rozdílů mezi Max/MSP a Pd je to, že Pd

má velmi rychlý audio engine, kdežto Max/MSP tuto věc zanedbal na úkor video enginu, který má ale oproti Pd o dost vyspělejší. Záleží tedy, jakou máte představu o vašem budoucím projektu, finanční rozpočet a podle těchto parametrů tedy zvolit příslušný produkt.

Důležitý podnět pro Pucketta bylo to, aby i „(pod)průměrně technicky založení“ umělci byli schopni vytvořit svůj vlastní projekt v Pd, aniž by museli mít za sebou napsány stovky řádků kódu v jiném programovacím jazyce. Proto je Pd uživatelsky velmi přívětivé a jednoduché na naučení.

Pd je možné spustit téměř na jakémkoliv zařízení. Zajišťuje to jeho multiplatformita a má tak široké využití od osobních počítačů přes embedded zařízení až po chytré telefony.

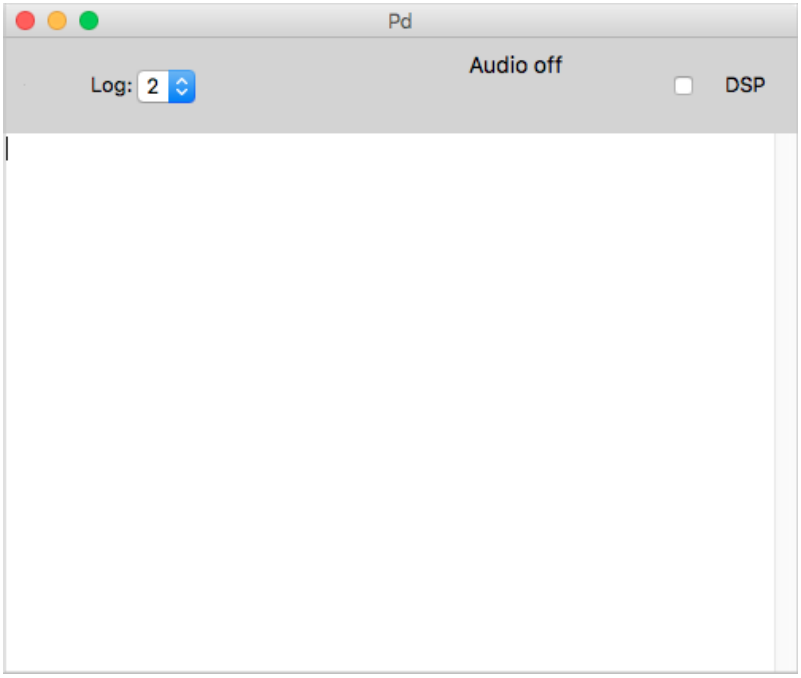
1.2.2 Práce s Pd

Jak již bylo výše zmíněno, Pure Data je grafický programovací jazyk. Znamená to tedy, že pro vytváření programů je třeba GUI (Grafické rozhraní – Graphical User Interface). Při otevření programu se zobrazí konzole, na obrázku 1.5, kde se vypisují data podle potřeby uživatele, chybové hlášky (podle nastavení logu) a další informace (např. načtené externí knihovny). Můžeme zde zároveň vypnout nebo zapnout DSP (Digitální Signálový Procesor). Při práci s Pd je praktické se naučit pár klávesových zkratek, které nám usnadní práci v tomto programu. Díky nim můžeme například vytvářet objekty, zprávy a nebo zapínat a vypínat zmíněný DSP. Dále je nutno podotknout, že pro editaci programů vytvořených v Pd je třeba mít zaplý editační mód (klávesová zkratka `Ctrl(command)+E`). Pokud tento mód bude vyplý, tak nebudete moci patch upravovat, což je ideální pro živé vystoupení. Mohli byste si totiž omylem odstranit nebo přesunout důležité části programu a vaše vystoupení by tak bylo znehodnoceno.

Patch

První věc, kterou si budeme muset vytvořit je takzvaný patch, nebo také česky plátno. Je to univerzální prostor, ve kterém se tvoří Pd soubory nebo abstrakce a vkládáme do něj prgramovací prvky. Některé z těchto prvků můžeme nastavovat v menu *properties* skrz kliknutí pravého tlačítka myši. Mezi základní objekty, které můžeme v patchi použít, patří:

- object box - nejpoužívanější objekt v Pd, existuje několik druhů, tyto objekty vykonávají určité procesy a jsou ekvivalentem funkcí v negrafickém programovacím jazyce; v našem případě budeme například používat objekt `[rfft~]` pro výpočet reálné fourierovy transformace nebo objekty `[adc~]` a `[dac~]` pro audio vstupy a výstupy



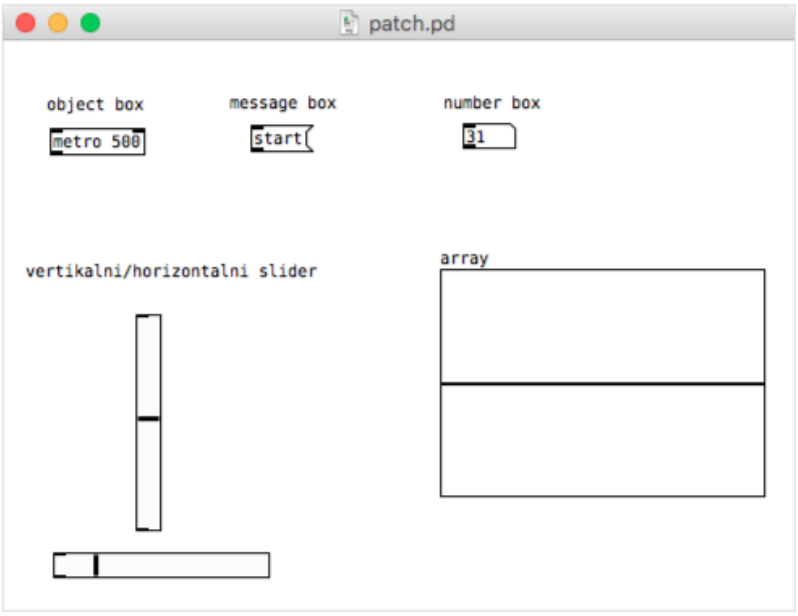
Obr. 1.5: Konzole při startu Pure Data.

- message box - využívá se pro posílání speciálních instrukčních zpráv do object boxů
- number box - objekt, který je vysloveně specializován na zobrazování a redistribuci číselných hodnot, pomocí kliknutí levého tlačítka myši a táhnutí je možno měnit hodnotu ve stanoveném rozmezí
- vertical/horizontal slider - jedná se o objekt slideru neboli posuvníku, kterým můžeme libovoně posouvat a na jehož výstupu jsou pak hodnoty čísel, které náleží nastavenému intervalu hodnot
- array - grafický objekt pro ukládání číselných hodnot

Grafické znázornění objektů v patchi můžeme sledovat na obrázku 1.6. Tyto objekty se poté spojují pomyslnými kabely tak, že výstup jednoho objektu je propojen se vstupem jiného objektu. U některých objektů je ale nutno posílat zprávy takzvaně „vzduchem“, jelikož nemají žádný fyzický vstup. Zároveň je důležité rozlišovat normální, číslicový input od signálového. Nemůžeme totiž zapojit signálový input do číslicového a obráceně. Object box, který využívá právě signálového inputu se značí tak, že za názvem objektu je takzvaná *tilda*, neboli ~. Bližší informace, jak pracovat s objekty v Pd nalezneme v tutoriálech [11], [12].

1.2.3 Abstrakce a subpatche

Abstrakce i subpatche jsou vlastně jedna a tatáž věc, ale každá je brána za jiný konec. Jsou to v podstatě vnořené patche do našich původních patchů. Rozdíl mezi



Obr. 1.6: Ukázka objektů Pd v patchi.

abstrakcemi a subpatchi je takový, že abstrakci stačí vytvořit pouze jednou a můžeme ji používat pod stejným názvem několikrát v tom samém ale i v jiném patchi. Kdežto subpach slouží spíše na takové to „uklizení kódu“. Funguje podobně jako abstrakce, má vstupy i výstupy, avšak jej můžeme pod jedním názvem v jednom patchi použít pouze jednou. Zároveň je subpatch součástí svého mateřského patche, takže k němu nemáme přístup bez toho, aniž bychom otevřeli mateřský patch. Vytvoření subpatche je jednoduché, stačí do object boxu napsat `pd` mezera a název subpatche. Vytvoření abstrakce je ještě jednodušší, jelikož jakýkoliv projekt, který uložíte s příponou `.pd` je vlastně brán zároveň jako abstrakce a je možno jej vložit do jiných projektů tak, že vytvoříme object box a napíšeme do něj jméno projektu (abstrakce), který chceme použít. Podmínkou korektní funkcionality je, že musí být soubory vnořených projektů (abstrakcí) ve stejné složce.

1.2.4 Externí knihovny

Externí knihovny, nebo taktéž externaly, jsou knihovny, na jejichž vytvoření se podílí široká komunita okolo Pure Data. Jedná se o objekty, naprogramované v jazyce `C`. Tyto externaly se instalují přes rozhraní Pd nebo je možno external stáhnout a uložit na přesně dané místo, požadované softwarem, do počítače. Rozdíl mezi externalem a abstrakcí je například takový, že si uživatel, pokud má zájem, může abstrakci otevřít a libovolně ji upravovat. U externích knihoven je tomu bohužel tak, že jsou tyto knihovny zkompilovány přímo pro Pd a nemůžeme tedy nijak měnit jejich funkcionalitu bez znalosti zdrojového kódu externalu. Vynikající návod, jak

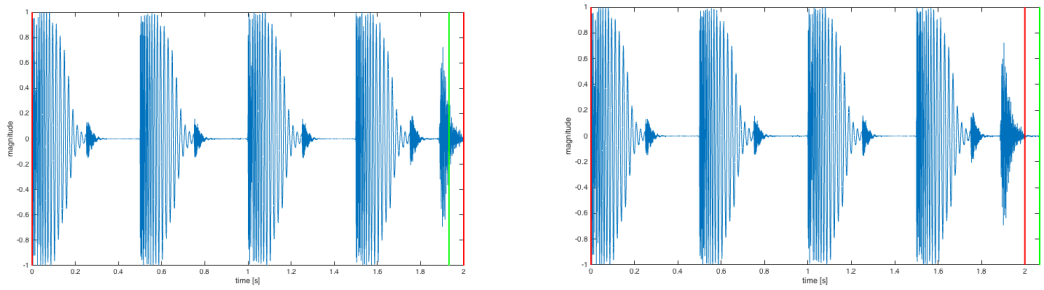
naprogramovat takovýto external najdeme v [4].

Jelikož jsou ale i všechny objekty v Pd naprogramovány v jazyce `C`, tak se nemusíme obávat o nějakou velkou ztrátu výpočetní rychlosti i při tvorbě svých vlastních „externalů“ pouze z abstrakcí. Navíc jsou tyto externaly modifikovatelné uživatelem a jednodušeji se debugují. Nevýhoda může však spočívat v nedostatečné optimalizaci kódu přímo na daný projekt, jelikož mohou být při tvorbě externalů z abstrakcí použity objekty provádějící určité výpočty, které v našem projektu nepotřebujeme a my tyto vlastnosti nemůžeme upravit bez znalosti zdrojového kódu objektu.

1.3 Seznámení s problémem

Nikdo z nás není dokonalý a vyrovnat se strojové přesnosti metronomu je místy velmi obtížné. A to platí dvojnásob u tvoření muziky v reálném čase. U live loopingu je však velmi důležitá přesnost hráče, se kterou sešlápne jednotlivé pedály. Tento úkol je pro začátečníky velmi obtížný až místy nemožný. Při tvorbě smyčky tak vznikají různé ruchové elementy, skoky, glitche a podobné věci, kterým se chceme vyvarovat.

Příčinou těchto zvukových niancí je nepřesná synchronizace sešláplnutí pedálu s následující dobou taktu. Na obrázcích 1.7a a 1.7b jsou vyobrazeny zvukové soubory a můžeme zde vidět, že byl pedál sešlápnut buď moc pozdě nebo zase moc brzo. Zelená vertikální čára symbolizuje moment sešlápnutí pedálu a červená vertikální čára zase reálnou první dobu smyčky. Aby byla smyčka správně sesynchronizovaná a nevznikaly další nepřirozené artefakty, musely by se tyto čáry překrývat.



(a) Sešlápnutí pedálu brzy.

(b) Sešlápnutí pedálu pozdě.

Obr. 1.7: Typy problémů, které mohou nastat.

Naším úkolem tedy bude navrhnout algoritmus, který se postará o tyto nepřirozené artefakty a zarovná tak konec vytvořené smyčky přesně na dobu. Pokud se bavíme o frázi zarovnat na dobu, myslíme tím zarovnat smyčku tak, aby byla první doba zarovnána na začátek smyčky a také, aby konec smyčky korespondoval s nadcházející první dobou znovu opakující se smyčky.

1.3.1 Začátek smyčky

V této diplomové práci se budeme zabývat pouze zarovnáním konce smyčky na dobu, zarovnáním začátku však nikoliv. Proto je třeba vybrat jednu z níže uvedených variant, aby výsledný algoritmus správně fungoval.

Na výběr máme ze dvou možností začátku nahrávání smyčky. Buď budeme předpokládat, že při stisku pedálu je hráč přesný a tedy stisk pedálu koresponduje s první dobou smyčky, nebo začneme nahrávat až poté, co hlasitost překročí určitý práh. Možnost druhá nám zajistí, že stisk pomyslného pedálu bude vždy na dobu, avšak

při nastavení moc vysokého nebo moc nízkého prahu nemusí vstupní signál vůbec tento práh přesáhnout a nespustí tak nahrávání nebo naopak bude velmi citlivý a spustí se při sebemenším zvukovém vzruchu. U první možnosti máme zase jistotu přesného sešlápnutí a spuštění nahrávání, ale to nemusí nutně korespondovat s první dobou nahrávané fráze. Z obou možností nám ale plyne, že pokud budou splněny výše zmíněné podmíky, tak bude začátek smyčky roven první době nahrávané fráze.

Zároveň si ale musíme uvědomit, že řešení výše uvedeného problému má několik možných variant. Také navíc nebudou dostupná žádná vstupní data o cílovém tempu, počtu taktů, druhu taktu ani počtu dob. Možné varianty si popíšeme v následujících kapitolách.

1.3.2 Zarovnání v reálném čase

Představme si, že uživatel sešlápne pedál pro ukončení smyčky moc brzo. Algoritmus by v průběhu nahrávání smyčky analyzoval doby a průběžně by vypočítával tempo dané fráze. Pokud by zjistil, že sešlápnutí je moc brzy a nekoresponduje s pravděpodobnou nastávající dobou v daném tempu tak by vyčkal a ukončil nahrávání přesně na dobu.

Tato varianta je jistě výhodná, co se časové úspory týče, ale má hned několik nedostatků. Prvním nedostatkem je, že je tato varianta aplikovatelná pouze na situaci, kdy uživatel sešlápne pedál moc brzy. Pokud bychom sešlápli pedál později, algoritmus není schopen v reálném čase toto vyhodnotit a ukončit tak nahrávání ještě než sešlápneme pedál se zpožděním. Je však teoreticky možné frázi ihned po ukončení nahrávání zkrátit na požadovanou délku a tím tento problém obejít. Druhým a hlavním problémem ale je, že tím, jak bude algoritmus čekat s ukončením nahrávání přesně na dobu, ale my už jsme fyzicky pedál sešlápli, tak psychologicky očekáváme, že se při našem fyzickém sešlápnutí pedálu začne přehrávat smyčka od začátku a my máme rovnou možnost vrstvit. Jakékoliv zpoždění, které by bylo tímto způsobeno nás může jako interpreta velmi rozhodit a udělat tak více škody než užitku.

1.3.3 Zarovnání nezávislé na čase

Opakem předchozí varianty je varianta nezávislá na čase. Znamená to, že výslednou smyčku po sešlápnutí pedálu uložíme a provedeme na ni požadované výpočty pro zarovnání fráze na dobu. Je to ekvivalent detekce tempa jakéhokoliv audio souboru. Tato varianta je pro nás ovšem nepřípustná, jelikož musíme frázi stále opakovat a nemůžeme si ji dovolit nejprve nahrát, poté ji pozastavit, provést příslušné algoritmy a poté opět spustit, už však zarovnanou na doby.

Klady této varianty jsou mimo jiné přesná detekce tempa a z toho vyplývající přesné zarovnání fráze. Můžeme totiž použít sofistikovanějších algoritmů, které jsou časově náročnější, ale o to přesnější výsledky dávají. Zároveň je úspěšně podchycena podmínka, že musí být algoritmus použitelný pro oba problémy popsané výše.

1.3.4 Zarovnání ve skoro-reálném čase

Takovým kompromisem mezi oběma předchozími variantami je varianta poslední a to takzvané *zarovnání ve skoro-reálném čase*. Jedná se o variantu, kde druhým sešlápnutím ukončíme nahrávání fráze a ta se nám začne přehrávat od začátku. Zatím neřešíme, jestli jsme přesně na dobu, či nikoliv. Jakmile je tedy smyčka ukončena, začínáme ji automaticky přehrávat a máme čas do konce první repetice na potřebné výpočty a zkrácení nebo prodloužení výsledné fráze.

Problém u této varianty nastane při optimalizaci algoritmu. Budeme chtít použít převážně složitější algoritmy pro přesnější zarovnání, ale zároveň ne moc časově náročné. Vhodným využitím správných algoritmů ale získáme optimální variantu pro základ detekčního algoritmu, která je aplikovatelná na oba problémy. Zároveň budeme schopni zarovnat smyčku do konce první repetice a tím pádem si posluchač nebude moci všimnout žádných nechtěných zvukových artefaktů již po první repetici.

V nadcházející kapitole se budeme zabývat algoritmy, které by nám mohly pomoci výše zmíněné problémy vyřešit.

1.4 Porovnání algoritmů

Existuje mnoho algoritmů, které se dají využít pro detekci takzvaných *onsetů*[3]. Patří mezi ně algoritmy založené na detekci onsetů v časové doméně - pomocí výpočtů rozdílné energie součinu oken[4] a audio signálu, [13], dále jsou zde algoritmy pro detekci onsetů ve frekvenční doméně - využití podobného principu jako předešlý algoritmus [14], můžeme také použít algoritmy, které hojně využívají neuronových sítí, [15], a nebo různých kombinací a zdokonalení předešlých algoritmů.

Celý algoritmus budeme programovat v Pd jako abstrakci, kterou budeme moci použít i v jiných projektech. Důvod pro rozhodnutí naprogramovat algoritmus jako abstrakci a ne jako external v `C` je ten, že při tvorbě abstrakce je možno lépe vidět do nitra „kódu“ a budoucí uživatel tak může názorně vidět kudy mu signál vede a

[3] Ekvivalent k začátku doby nebo taktéž náběžné hrany audio obálky.

[4] Může být Hannovo, Hammingovo, obdélníkové; využívá se při rozdělení dlouhého audio signálu na menší bloky, které jsou poté lépe zpracovatelné.

jak se zpracovává. Pokud bude navíc Pd spuštěno bez grafického rozhraní (naprogramovat patch bez grafického rozhraní problém je, ale spustit jej, pokud je navíc ovládán nějakým kontrolerem, problém není), dostáváme se k totožným rychlostem výpočtu, jako kdybychom použili externí knihovnu naprogramovanou v `C`. Zároveň tak otevíráme cestu modularitě, o které byla řeč na začátku práce, a uživatel si tak může algoritmus upravit podle svých potřeb.

1.4.1 Volba algoritmu pro onset detection funkci

Pojďme si nyní přiblížit dva algoritmy, které byly zmíněny výše a to algoritmus pracující pouze v časové doméně a poté algoritmus pracující v doméně frekvenční. Z těchto dvou algoritmů poté vybereme ten, který nám bude nejvíce vyhovovat pro řešení našeho problému.

Algoritmus v časové doméně

První algoritmus, popsaný v [13], nejdříve rozdělí audio data do H bloků po N vzorcích. Toho dosáhneme roznásobením daného audio signálu Hannovým oknem o délce N. Pokud není celková délka signálu N-násobkem délky okna tak je třeba prodloužit signál o daný počet nul tak, aby byl výsledný signál o délce l dělitelný velikostí okna. Dále se vypočítá energie těchto bloků pomocí vzorce

$$E(j) = \sum_{i=0}^{N-1} x_i^2, \qquad j \in 0, 1, \ldots, H-1, \tag{1.1}$$

kde E je energie bloku, N je délka Hannova okna (pro následující výpočty budeme brát v potaz, že $N = 1024$), $H = \frac{l}{N}$ je počet bloků a x_i je příslušný vzorek bloku v rozmezí 0 až $N-1$. Zároveň je třeba předpokládat, že signál, který takto analyzujeme je mono signál. Pokud by byl signál stereo tak musíme analogicky upravit předešlou rovnici na

$$E(j) = \sum_{i=0}^{N-1} x[l]_i^2 + x[r]_i^2, \tag{1.2}$$

kde $x[l]_i$ a $x[r]_i$ jsou příslušné vzorky levé a pravé strany signálu ve stereu.

Nyní si vypočítáme rozdíly hodnot energie daného okna s energií předešlého okna, ale budeme brát v potaz pouze kladné rozdíly

$$\begin{aligned} D(j) = E(j) - E(j-1), \quad &\text{pokud} \quad E(j) - E(j-1) \geq 0 \\ D(j) = 0, \quad &\text{pokud} \quad E(j) - E(j-1) < 0, \end{aligned} \tag{1.3}$$

kde D je hodnota rozdílů energií a $j \in 0, 1, \ldots, H-1$.

Dále je potřeba vypočítat klouzavý průměr těchto hodnot. Ten vypočítáme jako

$$A(j) = \frac{1}{21} \sum_{i=j-10}^{j+10} D(i), \qquad j \in 11, 12, \ldots, H-12, \tag{1.4}$$

kde A je kouzavý průměr. Pojďme si ale blíže vysvětlit tento vzorec. Důvod proč je součet rozdílů energií vydělen číslem 21 je, že chceme, aby měl klouzavý průměr okolí výpočtu cca $0.5s$. A pokud máme délku okna $N = 1024$ a vzorkovací frekvenci $f_{vz} = 44100Hz$, tak $0.5s \approx 21$ vzorků. Zároveň uvažujme, že pro jednu hodnotu kouzavého průměru vezmeme takové rozmezí hodnot, že daný index umístíme vprostřed a odečteme, resp. přičteme k němu hodnotu 10. Tím pádem nám vyjde $2 \cdot 10 + 1 = 21$ hodnot. Musíme také předpokládat, že hodnota průměru pro $j \in \langle 0, 10 \rangle$ a $j \in \langle H-11, H-1 \rangle$ bude jedna konstanta, jelikož nemáme dostatek dat před nebo za středovým číslem. U těchto konstant budeme počítat

$$\begin{aligned} A(0 \to 10) &= \frac{1}{21} \sum_{i=0}^{20} D(i), \quad \text{resp.} \\ A(H-11 \to H-1) &= \frac{1}{21} \sum_{i=H-21}^{H-1} D(i) \end{aligned} \tag{1.5}$$

Pokud bychom chtěli, aby měl klouzavý průměr větší okolí výpočtu, tak musíme tento vzorec analogicky upravit.

Jako další krok je třeba propustit skrz práh klouzavého průměru pouze takové hodnoty rozdílů energií, které daný práh, vynásobený konstantou C, překračují.

$$\begin{aligned} D(j) = D(j) - CA(j), \quad &\text{pokud} \quad D(j) > 0 \\ D(j) = 0, \quad &\text{pokud} \quad D(j) \leq 0, \end{aligned} \tag{1.6}$$

kde opět $j \in 0, 1, \ldots, H-1$. Konstantu C volíme v rozmezí 1.3 až 1.7 pro dobrou detekci onsetů. Doba totiž musí stanovený práh překročit výrazně, jinak bychom dostávali spoustu falešných detekcí. Hodnoty, které tento práh překročí vyhlásíme jako onsety a analogicky tedy i doby.

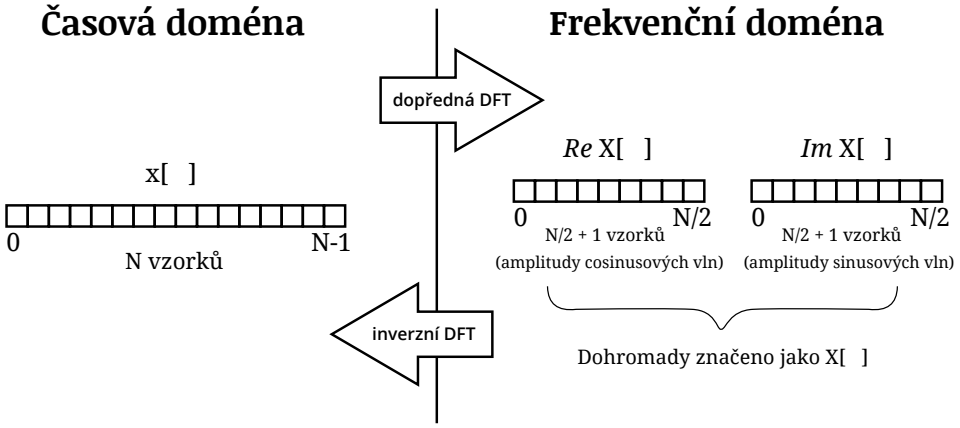
Tento algoritmus je velmi rychlý a tím pádem i nenáročný na výpočetní výkon, ale tím, že signál analyzujeme pouze v doméně časové, tak nedostáváme tak přesné hodnoty a navíc i přes všechnu optimalizaci velké množství falešných detekcí.

Algoritmus ve frekvenční doméně

Druhý algoritmus, který byl navrhnut, je algoritmus ve frekvenční doméně. Tento algoritmus je velmi podobný předešlému algorimu, avšak má jednu hlavní (ne)výhodu.

Nejprve je třeba audio signál převést z časové domény do domény frekvenční. Toho docílíme pomocí DFT (Discrete Fourier Transfor – Diskrétní Fourierova transformace). Důvod proč jsme dali předložku *ne* do závorky je ten, že sice je tento algoritmus přesnější, ale také náročnější na výpočet.

Reálná diskrétní Fourierova transformace Jak je napsáno v [16], výsledkem reálné diskrétní Fourierovy transformace je $\frac{N}{2} + 1$ reálných i imaginárních hodnot, kde N je délka signálu. Tu volíme tak, aby byla mocninou čísla 2. Jsou dva důvody proč takto činíme. Prvním je fakt, že výpočetní technika pracuje s daty ve dvojkové soustavě a díky tomu je mocnina čísla 2 vždy přirozenou délkou signálu. Za druhé je to z důvodu využití algoritmu FFT (Fast Fourier Transform – Rychlá Fourierova transformace), který je nejefektivnějším algoritmem pro výpočet DFT, [17], a ten funguje nejlépe (má největší časovou úsporu), pokud je délka signálu právě mocninou čísla 2. Na obrázku 1.8 můžeme vidět schéma procesu DFT. Signál v časové doméně standardně značíme malými písmeny a analogicky k tomu signál ve frekvenční doméně písmeny velkými.



Obr. 1.8: Blokové schéma reálné DFT. Inspirováno [16]

Reálnou a imaginární část DFT pro dané jedno okno poté spočítáme jako

$$\begin{aligned} Re\ X(k) &= \sum_{i=0}^{N-1} x(i) cos\Big(\frac{2\pi ki}{N}\Big) \\ Im\ X(k) &= -\sum_{i=0}^{N-1} x(i) sin\Big(\frac{2\pi ki}{N}\Big), \end{aligned} \tag{1.7}$$

kde $k \in 0, 1, \ldots, \frac{N}{2}$, $Re\ X$ je reálná část DFT a $Im\ X$ imaginární část. Výsledný tvar je poté $X(k) = Re\ X(k) + j \cdot Im\ X(k)$.

Jelikož náš vstupní signál bude velmi dlouhý a tento algoritmus by nebyl schopen takovýto objem dat zpracovat, budeme muset využít algoritmu STFT (Short Time Fourier Transform – Krátkodobá Fourierova Transformace). Jak již název vypovídá, tento algoritmus rozdělí vstupní signál na menší časové úseky, na které poté aplikuje algoritmus FFT[5]. Délka daného okna a možnost překrývání těchto oken poté koresponduje s rozlišením výsledného signálu ve frekvenční doméně. Čím větší okno je, tím jemnější rozlišení má daný spektrogram[6], ale zároveň se tím navyšuje náročnost výpočtů. Pro náš problém budeme volit okno o velikosti $N = 1024$, což je takový kompromis mezi jemností rozlišení a výpočetní náročností a tato okna se nebudou překrývat. Počet těchto oken závisí na délce signálu. Analogicky k prvnímu algoritmu budeme počítat, že je délka vstupního signálu buď násobkem čísla 2 nebo bude doplněna tak, aby tuto podmínku splňovala. Můžeme tedy signál rozdělit na $H = \frac{l}{N}$ bloků, kde l je výsledná, upravená délka vstupního signálu a N je velikost Hannova okna.

Daný vstupní signál tedy roznásobíme Hannovým oknem o délce N a jednotlivé bloky pomocí algoritmu reálné FFT převedeme do frekvenční domény. Jelikož jsou ale výstupem reálné FFT hodnoty amplitud funkcí sinus a kosinus, můžeme tato data nějak srozumitelně zpracovat, aby se v nich dalo dobře vyznat a byly tak lépe reprezentovatelné. To zajistíme převedením do tzv. *polárních souřadnic*. Pro přepočet jednoho okna existuje vztah

$$\begin{aligned} Mag\ X(k) &= \sqrt{Re\ X(k)^2 + Im\ X(k)^2} \\ Phs\ X(k) &= arctan\frac{Im\ X(k)}{Re\ X(k)}, \end{aligned} \tag{1.8}$$

kde $Mag\ X$ je amplituda dané bázové frekvence k a $Phs\ X$ je fázový posun. Rozdíl mezi značením v původních pravoúhlých souřadnicích oproti novým polárním si můžeme představit takto.

V pravoúhlých souřadnicích nám reálná FFT rozdělí vstupní signál na $\frac{N}{2} + 1$ vln funkce sinus a kosinus s určitými amplitudami, kdežto v polárních souřadnicích nám jej rozdělí na $\frac{N}{2} + 1$ vln funkce kosinus o dané amplitudě ($Mag\ X$) a fázovém posunu ($Phs\ X$). Výše byl ale zmíněn pojem *bázová frekvence*, tak si jej nyní pojďme objasnit.

Bázová frekvence Nejlépe si tento pojem můžeme vysvětlit na příkladu. Převádíme signál z domény časové do domény frekvenční. Tento signál je delší, takže

[5]FFT je upravenou a zrychlenou verzí algoritmu DFT, kde použitím Cooley-Tukeyho algoritmu jsme schopni zredukovat počet aritmetických operací z původních $O(N^2)$ na $O(NlogN)$, [17].

[6]Obrazová reprezentace spektra frekvencí daného vstupního signálu.

použijeme algoritmus STFT. Délku Hannova okna zvolíme $N = 1024$. Vzorkovací frekvence je $f_{vz} = 44100Hz$. Pokud tedy jedno okno převedeme do frekvenční domény, dostáváme $\frac{N}{2} + 1$ hodnot, v našem případě $\frac{1024}{2} + 1 = 512 + 1 = 513$. Maximální frekvence, kterou může reálná FFT reprezentovat je podle Nyquistova teorému, [18], polovina vzorkovací frekvence, tedy $22050Hz$. Pro první bázovou frekvenci platí $\frac{22050}{513} \cdot \frac{1}{2} \approx 21,5Hz$. Druhou bázovou frekvenci vypočítáme jako $21,5 + \frac{22050}{513} \cdot 1 \approx 64,5Hz$, třetí jako $21,5 + \frac{22050}{513} \cdot 2 \approx 107,5Hz$, a tak dále. Tento výpočet můžeme zobecnit na

$$f_B = f_{B(1)} + \left(\frac{\frac{f_{vz}}{2}}{\frac{N}{2} + 1} \right) \cdot (B - 1), \quad B \in 2, 3, \ldots, \frac{N}{2}, \tag{1.9}$$

kde f_{vz} je vzorkovací frekvence a $f_{B(1)}$ vypočítáme jako

$$f_{B(1)} = \left(\frac{\frac{f_{vz}}{2}}{\frac{N}{2} + 1} \right) \cdot \frac{1}{2}, \quad B = 1. \tag{1.10}$$

U hodnoty poslední $B = \frac{N}{2} + 1$ se pak $f_B = \frac{f_{vz}}{2}$.

Pokud se analyzovaná frekvence nerovná frekvenci bázové, tak je tato frekvence vyjádřena jako daný poměr nejbližších bázových frekvencí.

Už tedy víme, jak převést signál z časové domény do domény frekvenční a je tedy potřeba objasnit jakými výpočty bude algoritmus pokračovat.

V nadcházejícím kroku budeme pracovat pouze s hodnotami aplitud $Mag\ X$. Tento krok se nazývá v angličtině *spectral flux*, což můžeme volně do češtiny přeložit jako *spektrální rozdíl*. Ačkoliv se může zdát, že díky názvu se bude jednat o složitý výpočet, opak je pravdou. Jedná se totiž o pouhý součet rozdílů všech hodnot amplitud daného okna s oknem předchozím. Tento krok můžeme zapsat jako

$$SF(j) = \sum_{k=0}^{\frac{N}{2}} Mag\ X(j,k) - Mag\ X(j-1,k), \quad j \in 0, 1, \ldots, H-1, \tag{1.11}$$

kde $SF(j)$ je spektrální rozdíl bloku j s předešlým a N je délka bloku. Pro každý blok tedy dostaneme pouze jednu hodnotu, se kterou budeme dále pracovat.

Nyní je třeba provést tzv. *jednocestné usměrnění*. Znamená to, že hodnoty spektrálních rozdílů, které jsou menší než 0, dáme rovny 0.

$$SF(j) = 0, \quad \text{pokud} \quad SF(j) < 0. \tag{1.12}$$

Dále vypočítáme klouzavý průměr. Jeho popis můžeme najít v kapitole 1.4.1. Stačí pouze analogicky upravit rovnice 1.4 a 1.5 pro náš případ.

Nakonec vyhodnotíme, které hodnoty spektrálních rozdílů přesáhnou příslušné hodnoty klouzavých průměrů vynásobených konstantou C, kterou volíme v rozmezí od 1.3 až 1.7, a tyto hodnoty jsou naše požadované onsety.

Důvod, proč je výhodnější využít tento algoritmus oproti algoritmu v časové doméně je zejména ten, že díky převodu do frekvenční domény můžeme omezit vsupní signál pouze na takový frekvenční rozsah, který nám umožní zpřesnění detekce daných dob. Například použitím dolní a horní propusti.

Máme tedy výslednou onset detection funkci a teď nám pouze zbývá detekovat tempo skladby a podle něj danou frázi zarovnat.

1.4.2 Detekce tempa z onset detection funkce

Jako hlavní algoritmus pro výpočet onset detection funkce jsme tedy vybrali algoritmus ve frekvenční doméně, neboli *spectral flux* algoritmus. Díky němu jsme dostali výsledné hodnoty onsetů a nyní je třeba z těchto onsetů zjistit tempo dané fráze.

Opět se na řešení tohoto problému můžeme dívat ze dvou stran. Jedna možnost je využití statistiky a poměru vzdáleností mezi jednotlivými onsety. Poté bychom mohli pomocí společného násobku těchto hodnot odhadnout tempo. Pokud budeme mít pravidelný rytmus, tak může tato varianta dávat velmi spolehlivé výsledky. Pokud jsou ale doby ve frázi nepravidelně rozprostřeny, tak nám tato varianta stačit nebude. Navíc se mohou v osnet detection funkci vyskytovat ne jenom přesné onsety, ale je možné, že se skrz práh klouzavého průměru dostanou i další hodnoty, které však doby nebudou. Pokud bychom tyto hodnoty zohlednili ve výpočtu, tak bychom díky nim zvyšovali nepřesnousttohot výpočtu.

Proto se blíže podívejme na druhou možnost a tím jsou hřebenové filtry. Ano, i takovýto nevalný jev jako je hřebenový filtr nám může výrazně ulehčit výpočet výsledného tempa fráze. Představme si, že máme tedy onset detection funkci, která nám říká, na kterých pozicích se nacházejí doby. Poté provedeme konvoluci této funkce s hřebenovým filtrem o frekvenci, která odpovídá určité hodnotě BPM (Beats Per Minute – označení tempa, neboli počtu dob za minutu), a následně všechny tyto hodnoty sečteme. Největší z těchto sečtených hodnot poté odpovídá tempu, u kterého se nejvíce shodují jednotlivé onsety s hřebeny filtru. U těchto hřebenových filtrů je dán určitý rozsah temp (v našem případě 80 BPM až 280 BPM) a tyto filtry mají mezi sebou vždy 1 BPM rozdíl. Dále je třeba připomenout, že se zde násobí hodnoty detekční funkce od 0 do $H - 1$ a analogicky je třeba, aby stejnou délku měly i jednotlivé hřebenové filtry. Zároveň je podchycen problém, že né všechny hodnoty onset detection funkce musí být doby. Tyto „falešné doby“ mají většinou velmi malou hodnotu, takže konvoluci nijak zásadně neovlivní.

Pokud známe tempo dané fráze, tak nám zbývá už jen poslední krok a to zarovnání dané fráze.

1.4.3 Zarovnání fráze

Tento krok, ačkoliv by se čekalo, že nebude nijak složitý, je ale dosti zákeřný a hlavně je to nejdůležitější část celého algoritmu. Korektní zarovnání bude rozhodovat o tom, jestli uslyšíme na konci fráze nežádoucí zvukové artefakty či nikoliv.

Nastává nám však dilema, jak tento algoritmus pozná, jestli má danou frázi zkrátit či prodloužit? To můžeme vyřešit tím, že budeme uvažovat délku fráze buď jako 4/4 takt nebo 3/4 takt a jejich poměrné kombinace. Z toho můžeme pomocí společného násobku s výsledným tempem získat délku fráze. Toto ale není univerzální řešení. Co když bude například skladba v 5/4 nebo 7/4 taktu[7]? Daný algoritmus by tak podle toho, co jsme popsali výše, buď jednu dobu ubral, nebo o jednu dobu prodloužil. Musíme tedy vymyslet, jak by se daná fráze zarovnala na dobu přesně.

Uvažovat o zarovnání na nejbližší nadcházející dobu také nejde. Pokud bychom totiž sešlápli pedál moc pozdě, tak by nám to danou smyčku ještě více prodloužilo. Řešením je ale zarovnat frázi na nejbližší možnou dvojnásobnou dobu.

To zajistíme tak, že výslednou hodnotu BPM vynásobíme číslem 2, poté vydělíme číslem 60 a tak dostaneme hodnotu dvojnásobného tempa za jednu sekundu (analogicky BPS - Beats Per Second). Pokud výsledek vynásobíme délkou zatím neupravené fráze v sekundách, dostaneme výsledný počet dob (beatů, hřebenů). Ten je však třeba zaokrouhlit na celé číslo, čímž provedeme zarovnání. Pokud bude hodnota nižší než $X.5$, kde X je celé číslo počtu hřebenů, tak se zaokrouhlí dolů na X a tím pádem tedy i zkrátí fráze. Naopak při hodnotě vyšší nebo rovné $X.5$ se zaokrouhlí nahoru na $X + 1$ a dojde k prodloužení fráze. Zároveň nám z tohoto vyplývá jedna důležitá podmínka a to, že největší možná chyba, které se může interpret dopustit je polovina vzdálenosti jedné doby BPM od druhé. Pokud se interpret zmýlí o více než je jedna doba dvojnásobného tempa, tak bude algoritmus brát v potaz tuto dobu jako další a zarovná až na ni.

Toto řešení není sice úplně přesné, ale zato univerzální a můžeme si tedy dovolit nahrát frázi v jakémkoliv taktu s jakýmkoliv počtem dob. Musí ale platit podmínka zmíněná výše a to, že se nesmíme zmýlit v sešlápnutí pedálu více jak o jednu dobu dvojnásobného tempa oproti detekovanému tempu.

Teoretický základ pro detekci tempa máme tedy za sebou a nyní je třeba tyto znalosti aplikovat při tvorbě looperu a implementovat tento algoritmus do Pure Data.

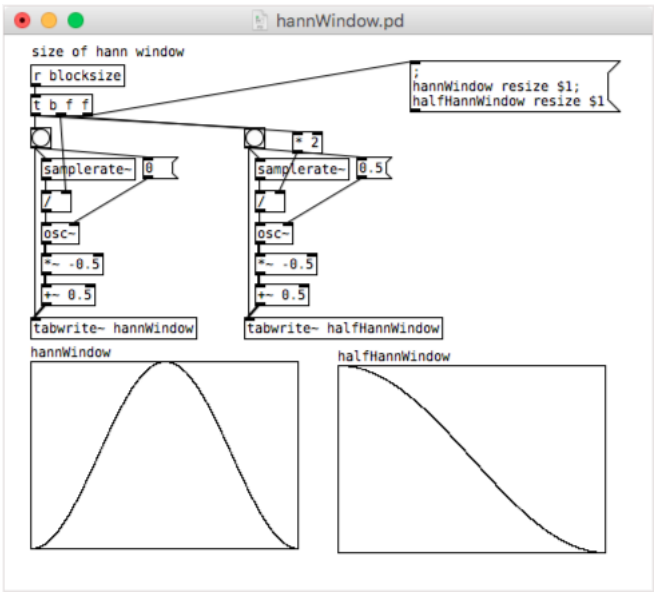
[7]Příkladem písně v 7/4 taktu je například *Money* od kapely *Pink Floyd*.

2 PRAKTICKÁ ČÁST STUDENTSKÉ PRÁCE

V této kapitole se budeme zabývat grafickým návrhem looperu, implementací předešlého algoritmu do Pure Data a v neposlední řadě návrhem a výrobou vlastního harwarového ovladače pro looper, který bude se softwarem komunikovat skrz protokol MIDI.

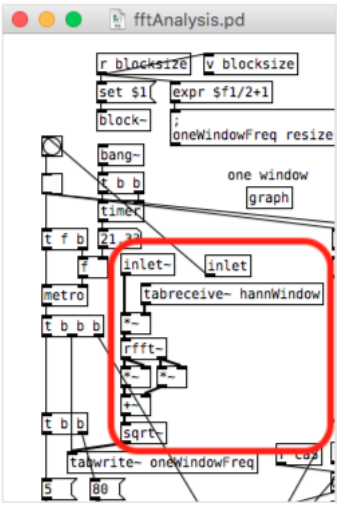
2.1 Realizace algoritmu v Pd

Nejdříve bude třeba zvolit délku Hannova okna a zároveň také velikost okna, se kterým bude počítat algoritmus FFT. Délky těchto dvou oken se schodují, což bude vysvětleno níže. Velikost okna analýzy zvolíme pomocí objektu `[block~]`. Poté vygenerujeme Hannovo okno pomocí objektu `[osc~]`. Tento objekt nám generuje harmonický signál funkce kosinus o dané frekvenci. Pokud chceme získat jedno Hannovo okno, musíme tento signál nejdříve omezit na danou délku, což můžeme udělat tak, že budeme vpisovat tento signál do arraye. Pokud jí nastavíme určitou velikost, tak nám tato array pojme data pouze o tomto rozsahu, čímž máme podchycenou první podmínku. Poté je třeba vyškálovat funkci na polovinu původního rozsahu, přesněji z $\langle -1, 1\rangle$ na $\langle 0, 1\rangle$. Všechny hodnoty tedy vydělíme 2 a přičteme ke každé 0.5. Nakonec je třeba tyto hodnoty buď vynásobit -1 nebo posunout fázi o 0.5. Tento krok je důležitý pro správnou interpretaci Hannova okna, jelikož potřebujeme z funkce kosinus dostat funkci sinus. Celkový proces výpočtu můžeme vidět na obrázku 2.1.



Obr. 2.1: Ukázka výpočtu Hannova okna (vlevo) a poloviny Hannova okna (vpravo) v Pd. Obě arraye mají stejnou velikost.

Dále je třeba provést Fourierovu transformaci. K tomu použijeme objekt `[rfft~]`. Tento objekt bere na vstupu signálová data a na jeho výstupech jsou dané reálné (levý výstup) a imaginární (pravý výstup) části reálné Fourierovy transformace. Pd počítá se signály vždy v blocích nastavených parametrem objektu `[block~]`. Příklad výpočtu Fourierovy transformace vidíme na obrázku 2.2.

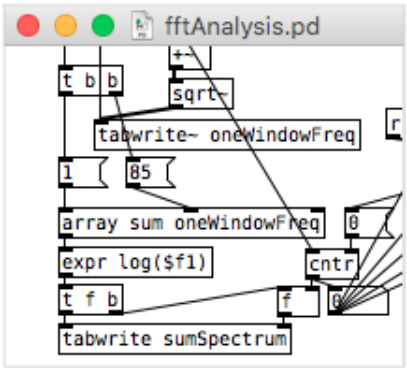


Obr. 2.2: Výpočet reálné Fourierovy transformace.

Důvodem stejné velikosti Hannova okna a objektu `[block~]` je to, že ještě před samotnou Fourierovou transformací je žádoucí konvolovat jeden blok dat právě s Hannovým oknem, čímž docílíme hladšího průběhu FFT. Výpočet FFT provádíme každých $\frac{N}{f_{vz}} \cdot 1000ms$, aby nám okna navazovala na sebe. Je třeba připomenout, že nepoužíváme žádný překryv oken.

Můžeme si všimnout, že hned po výpočtu FFT je signál převeden do polárních souřadnic a to pomocí rovnice 1.8 a následně jsou tyto hodnoty zapsány do arraye. Dále jsou hodnoty amplitud sečteny, ale pouze v rozsahu od 1 do 85 vzorku, místo od 0 do 512. To koresponduje s rozsahem bázových frekvencí, při volbě $N = 1024$ a $f_{vz} = 48000Hz$, přibližně $70Hz - 4000Hz$, vyobrazeno na 2.3. Je třeba zdůraznit, že ještě předtím, než zapíšeme výslednou hodnotu součtu do následující arraye, je tato hodnota zlogaritmována přirozeným logaritmem. Děláme tak z důvodu přirozeného vnímání sluchového ústrojí, podrobněji se můžeme dočíst v [19].

Následuje spektrální rozdíl, neboli *spectral flux*, vždy dané hodnoty s hodnotou předchozí podle rovnice 1.11. V tomto výpočtu zároveň zohledníme podmínku, že když je rozdíl záporný, tak místo něj zapíšeme 0. Dále je třeba vypočítat klouzavý průměr. Bereme v potaz, že tento klouzavý průměr bude mít okolí výpočtu cca $0.5s$. Při vzorkovací frekvenci $f_{vz} = 44100Hz$ a délce okna $N = 1024$ z toho vyplývá zhruba 21 hodnot spektrálních rozdílů. Abychom zahlásili daný spektrální rozdíl jako dobu,



Obr. 2.3: Příklad součtu amplitud v daném rozsahu.

je třeba hodnotu klouzavého průměru překročit o násobek určité konstanty, v našem případě $C = 1.6$.

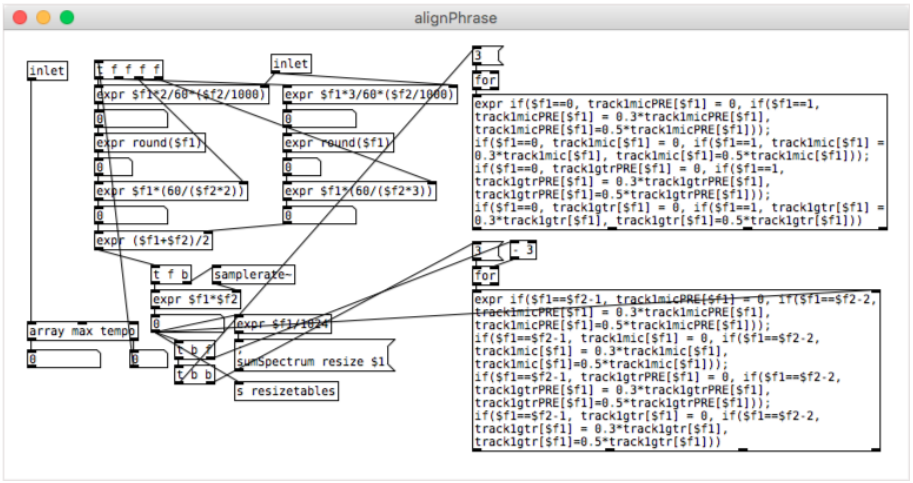
Výsledná onset detection funkce měla ale místy falešné onsety, kterých bylo třeba se zbavit. Po prostudování problému bylo zjištěno, že pokud jsou některé doby výrazně hlasitější, tak se skrz práh může dostat více dob ze sousedních bloků. Tento problém jsme zredukovali tím, že danou hodnotu vyhlásíme jako výslednou dobu pouze pokud jsou její sousední hodnoty a zároveň sousední hodnoty těchto sousedních hodnot menší než daná hodnota. Jedná se o upravený *peak picking* algoritmus popsaný v [20].

V neposlední řadě je třeba získat výsledné tempo fráze z onset detection funkce a to využitím již dříve zmíněných hřebenových filtrů v kapitole 1.4.3. Musíme ale podotknout, že tyto filtry je nutno generovat již při startu programu, jelikož vygenerování těchto filtrů trvá přibližně $15s$. Po tomto inicializačním čase je možno začít looper plně používat. Zároveň je třeba si ujasnit, že hřebenové filtry jsou generovány takové délky, aby bylo možno zanalyzovat frázi o maximální délce zhruba $20s$, což by mělo být dostatek k určení tempa fráze. Výslednou hodnotu po konvoluci a následném sečtení všech hodnot zapíšeme do arraye. Po konvoluci onset detection funkce se všemi hřebenovými filtry je třeba z arraye temp vybrat tu největší a podle pozice, na které se nachází, identifikujeme výsledné tempo fráze.

Bohužel se u určitých druhů nahraných frází setkáváme s nepřesnými hodnotami detekce. Je to dáno tím, když je ve frázi obsaženo hodně synkopických[1] nebo nepravidelných dob. Po výsledné konvoluci s hřebenovými filtry poté vychází jiná tempa. Tím nám tedy vzniká problém při zarovnání fráze. Algoritmus si totiž myslí, že daná fráze je nahraná jako 3/4 takt, i když se ve skutečnosti jedná jen o synkopické doby 4/4 taktu.

Proto jsme museli upravit původní tvrzení z kapitoly 1.4.3. V této kapitole se píše, že doba na kterou se bude zarovnávat bude nejbližší možná doba dvojnásob-

[1]Předražená doba před očekávanou přízvučnou dobou.



Obr. 2.4: Abstrakce pro zarovnání fráze na dobu.

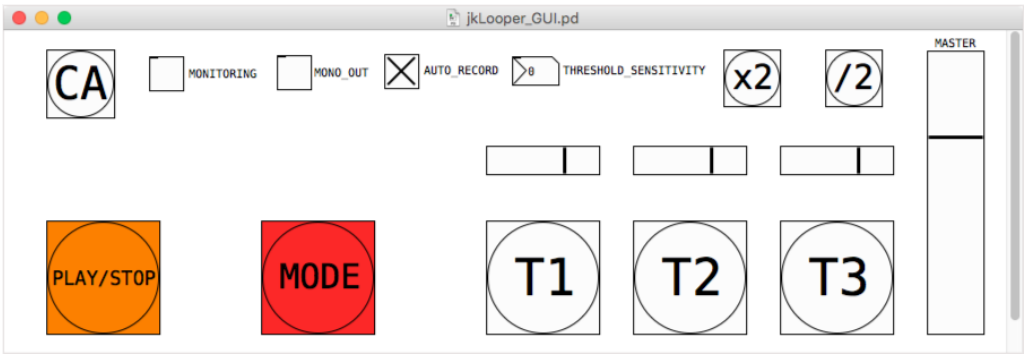
ného tempa. Toto by však platilo pouze v případě, že algoritmus korektně odhadne tempo fráze. Řekli jsme, že to se ve spoustě případů nestane, obzvláště při složitějších tempech, a tak je třeba přijít na jiné řešení. Výsledkem je výpočet, který vydělí detekované tempo 60 pro získání hodnoty za jednu sekundu a následně je vynásobeno zároveň čtyřmi a šesti. Poté je výsledná hodnota vynásobena délkou analyzované fráze v sekudách. Hodnota vynásobená čtyřmi je analogická k taktu 4/4 typu a hodnota vynásobená šesti zase k taktu 3/4 typu. Danou výslednou hodnotu obou výpočtů zaokrouhlíme a zpětně převedeme. Vydělíme tedy hodnotu čtyřmi nebo šesti, vynásobíme 60 a vydělíme detekovaným tempem. Výsledné hodnoty sečteme a uděláme z nich průměr. Nebude se tedy jednat o přesné zarovnání, ale ve většině případů se právě zaokrouhlené zpětně převedené hodnoty rovnají a je navíc vyřešen problém nepřesné detekce tempa. Navíc se jedná a zarovnání sluchem skoro nepostřehnutelné. To hlavní, o co nám jde, je omezit nechtěné zvukové artefakty při skoku z konce smyčky na začátek, na minimum. Na obrázku 2.4 můžeme vidět abstrakci, která se postará o zarovnání.

Jako úplně poslední krok detekce tempa budeme chtít pouze daný konec a začátek ošetřit proti nechtěným praskáním. To se děje neplynulým navázáním vzorků na sebe. Proto vynásobíme první 3 hodnoty fráze postupně 0, 0.3 a 0.5. Analogicky třetí hodnotu od konce vynásobíme 0.5, druhou 0.3 a poslední 0. Tímto vytvoříme velmi krátký fade in a fade out.

Všechny tyto výpočty, které jsou popsány výše najdeme v akstrakcích `BPMdetect.pd` a `fftAnalysis.pd`.

2.2 Grafický návrh looperu

Zde si představíme grafický návrh looperu v prostředí Pd. Ukážeme si základní ovládací prvky a také jednotlivé funkce looperu. Pro spuštění grafického rozhraní looperu otevřete v Pd soubor `jkLooper_GUI.pd`.



Obr. 2.5: Návrh GUI pro looper.

2.2.1 Popis ovládání looperu

Ovládací tlačítka Na výše zmíněném obrázku můžeme vidět, že se looper skládá z několika částí. Úplně vlevo jsou tlačítka pro spuštění nebo zastavení celé smyčky (PLAY/STOP), tlačítko pro smazání celé smyčky (CA) a také tlačítko pro změnu módu (MODE).

Druhá část looperu se skládá ze tří tlačítek (T1, T2, T3) a jejich ovládacích prvků. Těmi jsou slidery pro samostatnou hlasitost každé z frází. Zároveň v právem horním rohu můžeme vidět tlačítka X2 a /2. Tyto tlačítka slouží pro případné prodloužení nebo zkrácení fráze v další stopě.

Třetí část looperu disponuje hlavním sliderem pro celkovou hlasitost smyčky.

Inicializace a funkce looperu Při startu looperu jsou generovány všechny hřebenové filtry. Tento proces nám zabere zhruba 15s při délce okna jednoho hřebenového filtru $N = 1000$ vzorků. Délka tohoto okna už je upravena pro konvoluci s onset detection funkcí, která je 1024 krát menší než délka okna fráze, jelikož na ní byla provedena Fourierova analýza o délce okna 1024 vzorků. Tím pádem jsme schopni analyzovat tempo z fráze až 20s dlouhé. Samotná fráze je však omezena maximální délkou jedné minuty.

Hlavní ovladací prvky looperu jsou tlačítka T1, T2 a T3. Tyto tlačítka jsou používány pro nahrávání do jednotlivých stop 1, 2 a 3. Grafické rozhraní looperu vidíme na obrázku 2.5. Další důležité tlačítko je tlačítko MODE, které nám přepíná z

nahrávacího módu do módu přehrávacího. Přepínáním mezi módy tak můžeme ušetřit místo na pedály. Looper je automaticky při startu v módu nahrávacím (barva tlačítka červená) a není jej možné přepnout do přehrávacího módu (barva tlačítka zelená), dokud není nahrána fráze aspoň v jedné stopě. V nahrávacím módu se tlačítka jednotlivých stop chovají tak, že se při prvním stisknutí začne nahrávat, po druhém vrstvit a po třetím přehrávat. V přehrávacím módu se poté chovají tak, že při stisknutí se daná fráze vypne nebo naopak zapne. Je nutno podotknout, že takovéto vypnutí fráze je realizováno pouze jako tlačítko mute, tedy neprovedeme úplné zastavení smyčky, pouze ji ztišíme na minimum. Dále je velmi důležité zdůraznit, že je třeba dodržet pořadí nahrávaných stop. Nejdříve se nahrává do stopy první (T1), poté do stopy druhé (T2) a nakonec do stopy třetí. Žádná jiná kombinace zatím není možná.

Zároveň je nutno podotknout, že máme na výběr ze dvou možných začátků smyčky. První a přednastavená možnost je automatické spuštění nahrávání po překročení hlasitostního prahu, který si můžeme také nastavit. Je dobré s tímto parametrem chvíli experimentovat a zajistit tak ideální spuštění přímo pro dané audio rozhraní. Pokud sešlápneme pedál a tlačítko na obrazovce zeleně bliká, znamená to, že looper čeká na dostatečně silný vstupní signál, aby mohl začít nahrávat. Druhou možností je začít nahrávání přímo se sešlápnutím pedálu, což je ideální pro pokročilejší hráče.

U ostatních stop je důležité sešápnout pedál před ukončením předešlé stopy, abychom tak inicializovali, že chceme začít nahrávat do stopy druhé, resp. třetí. Tlačítko bude blikat zeleně a jakmile začne fráze znovu od začátku, tak se automaticky začne nahrávat do vybrané stopy. Zároveň je třeba zdůraznit, že nahrávaná stopa sama přejde z nahrávání do vrstvení, není možné nahrávání zastavit dříve.

Představme si, že jsme v situaci, kdy chceme nahrát například rytmický základ, basovou linku a progresi akordů a to vše do separátních stop. Abychom nemuseli jako první nahrávat dlouhý rytmický podklad, který se stále opakuje a má délku basové linky nebo progrese akordů, můžeme nahrát pouze jeden či dva takty tohoto základu a poté délky ostatních stop vynásobit nebo vydělit mocninou čísla 2. K tomu slouží dvě dedikovaná tlačítka x2 a /2. Tyto tlačítka je třeba stistknout ještě před stisknutím tlačítka pro spuštění nahrávání další stopy.

Vše co jsme výše popsali může být místy poměrně matoucí, tak si pojďme ovládání looperu názorně ukázat na jednom příkladu.

Ukázka ovládání looperu

Praktickou ukázku rozdělíme pro přehlednost do kroků, které by měl uživatel zreprodukovat pro shodný výsledek.

1. Nejprve je třeba nastavit zda-li chceme automatické nahrávání či nikoliv. Pokud ano, je třeba nastavit práh hlasitosti na takovou úroveň, kdy začne při testovacím úderu nebo zahrané notě problikávat zelená barva v pozadí čísla.
2. Poté zvolíme jestli chceme mono výstup. Pokud ne, každý nástroj (jeho nahraná suma smyček) půjde do svého vlastního audio výstupu - momentální konfigurace dovoluje dva vstupy a dva výstupy.
3. Dále si zvolíme, jestli chceme zpětný monitoring našeho signálu nebo ne. Původní hodnota je nastavena na vypnuto.
4. Stiskneme tlačítko T1 pro nahrávání fráze a začneme nahrávat.
5. Po ukončení smyčky stiskneme tlačítko znovu pro ukončení nahrávání - fráze se sama zarovná na nejbližší možnou dobu při dodržení podmínek z kapitoly 1.4.3.
6. Poté můžeme vrstvit nebo stiskneme tlačítko znovu a tím umožníme nahrávání do další stopy.
7. Stiskneme tlačítko pro nahrávání do druhé stopy.
8. Vyčkáme, než předešlá fráze skončí a poté se začne nahrávat fráze do další stopy (změna z blikajícího zeleného tlačítka na červené).
9. Po ukončení nahrávání této stopy se ujistíme, že obě tlačítka, T1 i T2 jsou zelená a že tlačítko MODE je ve stavu nahrávání.
10. Stiskneme tlačítko /2 a poté tlačítko pro spuštění nahrávání do třetí stopy. Výsledná nahraná fráze bude dvakrát kratší a sesynchronizovaná s ostatními frázemi.
11. Pokud všechna tlačítka svítí zeleně, je možno přepnout z narhávacího módu do módu přehrávacího.
12. Zde můžeme stopy různě vypínat a zapínat, popřípadě využít tlačítko pro vypnutí a zapnutí všech nahraných stop PLAY/STOP.
13. Na konci stiskneme tlačítko CA pro uvedení looperu do původního stavu.

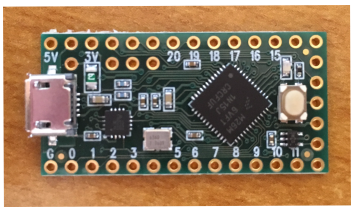
2.3 MIDI kontroler

V této kapitole si popíšeme konstrukci MIDI kontroleru se základními funkcemi pro looper. Realizace MIDI kontroleru proběhla pomocí těchto součástek:

- 1 mikrokontroler *Teensy LC*
- 6 tlačítkových spínačů

Pomocí těchto spínačů bude možno ovládat všechny stopy (T1, T2, T3), tlačítko MODE, PLAY/STOP a CA. Ukázku použitých spínačů a mikrokontroleru vidíme na obrázku 2.8.

Využili jsme mikrokontroleru Teensy LC, jelikož má tu vlastnost, že na každém digitálním vstupu již má takzvaný pull-up rezistor. Proto je velmi jednoduché se-



Obr. 2.6: Mikrokontroler Teensy LC.

Obr. 2.7: Tlačítkové spínače.

Obr. 2.8: Mikrokontroler (vlevo) a tlačítkové spínače (vpravo) použity pro kontrukci ovladače.

strojit ovladač na tomto mikrokontroleru. Pro prosté tlačítkové spínače není třeba dalších součástek, rezitrorů, kapacitorů, apod. Zároveň podporuje *MIDI over USB*, což je pro nás také výhoda. MIDI kontroler bude fungovat poté jako USB MIDI a celý mikrokontroler se tak bude zároveň napájet a posílat MIDI data skrz jeden USB kabel. To je obrovská výhoda oproti jiným konkurenčním mikrokontrolerům jako je například Arduino, které MIDI over USB nepodporuje.

Další výhodou tohoto mikrokontroleru je, že jej můžeme programovat pomocí Arduino IDE (Integrated Development Enviroment - Integrované vývojové prostředí) ve spojení s vestavěným modulem *Teensyduino*.

Výsledný kód tedy není nijak složitý díky tomuto jednoduchému rozhraní. Je důležité si uvědomit, že pokud je spínač seplý, tak je vyvořen zkrat mezi pinem a zemí a daný pin vyčítá hodnotu *LOW*. Pokud spínač rozepneme, daný pin vyčítá hodnotu *HIGH* , jelikož je příslušný pull-up rezistor propojen s napětím $5V$ uvnitř čipu. Poté nám stačí vytvořit podmínku pro tyto dva stavy a pro každou z nich poslat určitou hodnotu MIDI, tedy *note* a *velocity* (MIDI hodnota noty a její velikost/hlasitost).

Zdrojový kód pro tento MIDI kontroler je na přiloženém CD.

2.4 Možná vylepšení

Z výše zmíněných výsledků můžeme usoudit, že by se dalo pár věcí upravit, aby nám algoritmus zarovnával přesněji. První věc je výpočet onset detection funkce pro více frekvenčních pásem. V každém pásmu může být totiž dominantní jiný nástroj, který může zrovna určovat tempo. Například u bicích nástrojů je možné velmi spolehlivě zjistit tempo pomocí hi-hat činelů, které jsou však ve vyšších frekvenčních pásmech a jsou maskovány například silnějšími údery velkého a malého bubnu.

Jako další vylepšení můžeme analyzovaný signál předzpracovávat dynamickými efekty jako je kompresor, expander nebo šumová brána. Při správném nastavení parametrů bychom mohli dostat sjednocenější a vyrovnanější signál, který by byl

jednodušší na zpracování a dával tak přesnější výsledky detekce onsetů, tempa a následného zarovnání fráze.

Dalším vylepšením by mohlo být naprogramování celého detekčního algoritmu jako external pro Pd v jazyce `C` a výrazně tak optimalizovat určité procesy. Z toho by nám vyplývala časová úspora a tak možné zvolení složitějších algoritmů.

U hardwarové části by mohlo být určitým vylepšením přídání potenciometrů pro ovládání hlasitosti jak celkové tak jednotlivých stop a přidání dalších tlačítkových spínačů pro tlačítka x2 a /2.

3 ZÁVĚR

Nejdříve jsme se v této práci seznámili s pojmem *looper*, co to je za zařízení a jak funguje.

Dále jsme popsali prostředí Pure Data, vysvětlili si práci v něm a seznámili se se základními pojmy potřebnými pro pochopení této práce.

Také jsme si představili daný problém zarovnání na dobu a jeho možná řešení díky onset detection algoritmům. Blíže jsme si popsali, jak jednotlivé algoritmy fungují a který z nich byl nejvhodnější pro náš problém.

Setkali jsme se ovšem s problémy, kdy nám algoritmus detekoval špatné tempo z výsledné kovoluce s hřebenovými filtry. Bylo zjištěno, že nekorektní tempo bylo detekováno při použití synkopických dob. Řešení tohoto problému by mohlo být rozdělení analyzovaného signálu do více frekvenčních pásem a zároveň využít kombinaci jak konvoluce detekční funkce s hřebenovými filtry, tak statistickým výpočtem vzdáleností jednotlivých onsetů mezi sebou. Je však dosti pravděpodobné, že by tyto algoritmy byly velmi časově náročné a tak by bylo třeba implementovat výše zmíněné řešení jako external naprogramovaný v `C`.

Nakonec nebyl z časových a finančních důvodů zhotoven kontroler úplně podle představ autora, ale jelikož ho autor hodlá používat pravidelně pro svou vlastní potřebu, tak se tomuto designovému a funkčnímu nedostatku bude věnovat i po odevzdání práce.

LITERATURA

[1] *BOSS RC - 300 Specifications* [online]. [cit. 2017-12-14]. Dostupné z: `http://ca.boss.info/products/rc-300/specifications/`

[2] BELLO, Juan Pablo, Laurent DAUDET, Samer ABDALLAH, Chris DUXBURY, Mike DAVIES a Mark B. SANDLER. A Tutorial on Onset Detection in Music Signals. *IEEE TRANSACTIONS ON SPEECH AND AUDIO PROCESSING*. 2005, (5), 13.

[3] *BOSS RC - 505* [online]. [cit. 2017-12-14]. Dostupné z: `http://ca.boss.info/products/rc-505/`

[4] ZMÖLNIG, IOhannes m. *HOWTO write an External for Pure Data*. Graz: Institute of electronic music and acoustic, 2014.

[5] Latence. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation [cit. 2017-12-14]. Dostupné z: `https://cs.wikipedia.org/wiki/Latence`

[6] *Max* [online]. [cit. 2017-12-14]. Dostupné z: `https://cycling74.com/`

[7] DIXON, Simon. ONSET DETECTION REVISITED. *Proc. of the 9th Int. Conference on Digital Audio Effects (DAFx-06)*. Austrian Research Institute for Artificial Intelligence, 2006, , 5.

[8] Open Source Software. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation [cit. 2017-12-14]. Dostupné z: `https://en.wikipedia.org/wiki/Open-source_software`

[9] *Pure Data* [online]. [cit. 2017-12-14]. Dostupné z: `http://puredata.info/`

[10] Pure Data. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation [cit. 2017-12-14]. Dostupné z: `https://en.wikipedia.org/wiki/Pure_Data`

[11] *Programming electronic music in Pd* [online]. [cit. 2018-05-21]. Dostupné z: http://www.pd-tutorial.com/english/index.html

[12] *Pd Tutorials and HOWTOs?* [online]. [cit. 2018-05-21]. Dostupné z: https://puredata.info/docs/tutorials/

[13] *Beat Detection Algorithms (Part 1)* [online]. [cit. 2018-05-21]. Dostupné z: http://mziccard.me/2015/05/28/beats-detection-algorithms-1/

[14] *Beat This: A Beat Synchronization Project* [online]. [cit. 2018-05-21]. Dostupné z: https://www.clear.rice.edu/elec301/Projects01/beat_sync/beatalgo.html

[15] BÖCK, Sebastian, Andreas ARZT, Florian KREBS a Markus SCHEDL. *ONLINE REAL-TIME ONSET DETECTION WITH RECURRENT NEURAL NETWORKS*. Department of Computational Perception Johannes Kepler University, Linz, Austria: Conference on Digital Audio Effects (DAFx-12), 2012, , 4.

[16] STEVEN W. SMITH. *The scientist and engineer's guide to digital signal processing*. 2nd ed. San Diego, Calif: California Technical Pub, 1999. ISBN 09-660-1767-6.

[17] *Rychlá Fourierova transformace* [online]. [cit. 2018-05-21]. Dostupné z: https://cs.wikipedia.org/wiki/Rychlá_Fourierova_transformace

[18] *Nyquist—Shannon sampling theorem* [online]. [cit. 2018-05-21]. Dostupné z: https://en.wikipedia.org/wiki/Nyquist—Shannon_sampling_theorem

[19] KLAPURI, Anssi. *SOUND ONSET DETECTION BY APPLYING PSYCHOACOUSTIC KNOWLEDGE* [online]. Signal Processing Laboratory, Tampere University of Technology [cit. 2018-05-21].

[20] ZECHNER, Mario. *Onset Detection Part 7: Thresholding & Peak picking* [online]. [cit. 2018-05-21]. Dostupné z: https://www.badlogicgames.com/wordpress/?p=187

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

BPM	Beats Per Minute – označení tempa, neboli počtu dob za minutu
DFT	Discrete Fourier Transfor – Diskrétní Fourierova transformace
DSP	Digitální Signálový Procesor
FFT	Fast Fourier Transform – Rychlá Fourierova transformace
GUI	Grafické rozhraní – Graphical User Interface
IDE	Integrated Development Enviroment - Integrované vývojové prostředí
Max/MSP	Grafický programovací jazyk
MIDI	Komunikační protokol – Musical Instrument Digital Interface
Pd	Pure Data – grafický programovací jazyk
STFT	Short Time Fourier Transform – Krátkodobá Fourierova Transformace