МАТЕМАТИЧКА ГИМНАЗИЈА

МАТУРСКИ РАД

из предмета

рачунарство и информатика

на тему

Алати за дигиталну продукцију звука

Ученик

Ђорђе Васиљевић, IV$_e$

Ментори

проф. Нина Алимпић
Математичка гимназија

проф. Марко Перић
Музичка школа „Даворин Јенко“

Београд, јун 2017.

Садржај

1. Увод

Прва грамофонска плоча за чије снимање је коришћен микрофон и електронско појачавање звука је издата 1920. године [1]. Тиме је у продукцији звука отпочела ера технологије засноване на аналогном електричном сигналу. Наредних деценија продукција звука се развија заједно са успоном музичке индустрије, радија, телевизије и филма. Дигитална технологија у продукцији звука почиње да се појављује 1970-тих година, а 1982. године је на тржиште изашао компакт диск (*CD*) као дигитални носач звука који је почео да замењује грамофонске плоче. Током 1990-тих година дигитална технологија почиње да заузима централно место у студијима за продукцију звука.

Овај рад описује алате за дигиталну продукцију звука укључујући питања дигиталне обраде сигнала на којој се базира већина алата, као и питања аутоматизације послова који се обављају коришћењем алата за дигиталну продукцију звука.

У поглављу након уводног је дат упоредни приказ аналогног и дигиталног продукцијског окружења.

У трећем поглављу је детаљно описана функционалност дигиталне аудио радне станице са посебним осврtом на дигиталну аудио радну станицу *Reaper*.

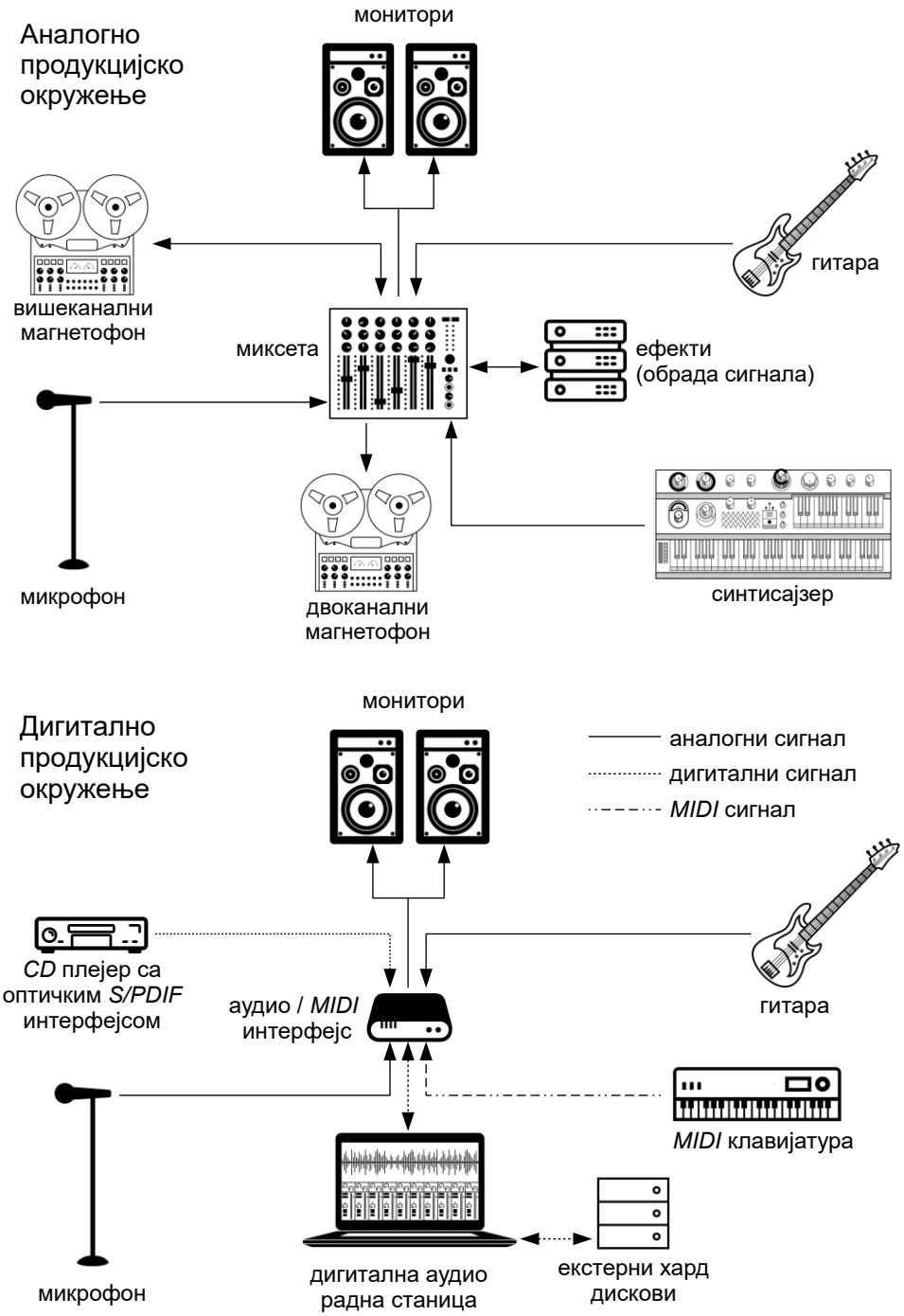
Четврто поглавље је посвећено дигиталном запису и техникама дигиталне обраде звучног сигнала. На почетку поглавља се објашњава математички модел дигиталног сигнала, након чега је дат једноставан пример дигиталне обраде звучног сигнала. Други део поглавља је посвећен Фуријеовом развоју и дискретној Фуријеовој трансформацији, почевши од теоријских основа и завршавајући са примером програма за спектралну анализу у програмском језику *Python*.

Пето поглавље је посвећено проблему временске корекције одсвираних тонова у снимку и софтверском модулу који аутоматизује једну фазу рада на временским корекцијама. Софтверски модул имплементира додатну активност у дигиталној аудио радној станици *Reaper*, а у имплементацији се такође користи програмски језик *Python*. Изворни код софтверског модула је дат у прилогу 1.

На крају желим да изразим захвалност менторима на уложеном времену, помоћи и пренесеном знању током израде матурског рада.

2. Од аналогне миксете до дигиталне аудио радне станице

На слици 1. шематски је приказано типично аналогно продукцијско окружење, као и дигитално продукцијско окружење засновано на дигиталној аудио радној станици.



Слика 1. Аналогно и дигитално продукцијско окружење[1]

[1] Коришћене су илустрације из *Noun* пројекта следећих аутора: Richard Nixon (звучник), Aaron K. Kim (екстерни интерфејс), Daniela Baptista (микрофон), Francisco Javier Diaz Montejano (клавијатура), Alexander Panasovsky

Са појавом првих дигиталних уређаја, продукцијско окружење је заправо постало хибридно, што значи да се користе и аналогне и дигиталне компоненте. Укуси, навике и уверења у вези аналогне и дигиталне технологије се разликују међу продукцијским професионалцима и музичарима, тако да не изгледа сваки модеран студио као што је приказано у доњем делу слике 1. Неки професионалци и даље користе аналогну миксету у комбинацији са дигиталном радном станицом, а ни потпуно аналогна продукција у којој се ради са магнетофонским тракама није престала да постоји.

2.1. Аналогно продукцијско окружење

У доба аналогне технологије, продукција звучних записа користила је вишеканалне студијске магнетофоне (на пример 24 канала) за радне материјале и двоканални (стерео) магнетофон за финални, такозвани мастер снимак. Мастер снимак је даље умножаван на грамофонским плочама и аудио касетама.

У средишту аналогног продукцијског окружења налази се миксета (engl. *mixing console*). На слици 2. је фотографија једне аналогне миксете.

Слика 2. Аналогна миксета STUDER 900

Главна функција миксете је да више улазних аудио канала комбинује у један излазни, тј. два канала у стерео продукцији. При томе су на појединим каналима обично могућа разна подешавања као што су:

- Јачина звука, за шта се обично користе велики клизни потенциометри у дну миксете (види слику 2)
- Балансирање улаза на леви и десни стерео канал (пановање), чиме се постиже да слушалац стекне утисак где се извор звука налази у простору
- Могућност избора како ће бити повезани канали, групе канала, додатни улази и излази и слично, за шта се користи систем прекидача

(електрична гитара), Iconic (лаптоп рачунар, вишеканални магнетофон), Ralf Schmitzer (*CD* плејер), Lee Hills (миксер) , Lucas Rod (синтисајзер), Creative Stall (ефекти), Richard Schumann (екстерни хард диск)

- Баланс између различитих фреквенцијских опсега звука (функција еквилајзера) чиме мењамо боју звука за шта се обично користи неколико ротационих потенциометара
- Корекција динамике, што значи да се додатно појачају или утишају деонице које су посебно гласне или посебно тихе

Подешавања за један улазни канал су груписана у вертикалне траке, а у траци се може налазити и мерач нивоа сигнала. Обично се и прикључци за каблове налазе у правцу одговарајуће траке са задње стране. Посебне траке могу постојати за групе канала и на крају имамо две траке за леви и десни канал главног излаза.

На миксету је могуће повезати различите ефекте у облику одвојених модула, а неке миксете имају и уграђене ефекте.

2.2. Дигитално продукцијско окружење

У дигиталном продукцијском окружењу се уместо магнетофона користе рачунарски хард дискови (укључујући *SSD* дискове). И даље можемо срести да се у неким фазама продукције користе магнетофонске траке, али се дигитални део продукције ради са дигиталним звучним записом у фајловима.

У потпуно дигитализованој продукцији, као оној у доњем делу слике 1, аналогни уређаји се могу срести у зони пре аналогно дигиталне (АД) конверзије као и у зони након дигитално аналогне (ДА) конверзије, према звучницима. У пракси се у многим студијима комбинује употреба аналогне и дигиталне обраде и то се ради на различите начине, па чак и да се у процесу продукције више пута конвертује између аналогног и дигиталног сигнала. У овом раду нећемо даље улазити у детаље могућег комбиновања аналогне и дигиталне технологије и претпоставићемо да имамо потпуно дигитализовану продукцију.

У окружењу приказаном у доњем делу слике 1. централно место заузима дигитална аудио радна станица што је заправо рачунар са аудио интерфејсом и одговарајућим софтвером. Дигитална аудио радна станица врши функцију миксете, синтисајзера и уређаја за ефекте, при чему се уместо магнетофонских трака користе фајлови на хард диску.

Када посматрамо како то физички изгледа, уместо синтисајзера видимо клавијатуру која је *MIDI*[2] контролер, тј. само кориснички улазни уређај (попут обичне тастатуре), док се синтеза звука на основу команди са клавијатуре дешава у оквиру дигиталне аудио радне станице под контролом централног процесора рачунара и софтверске имплементације синтезе звука.

На сличан начин је могуће да се и класичне физичке команде миксете (клизни и ротациони потенциометри, тастери и други прекидачи) имплементирају као *MIDI* контролер. У том случају и миксета подржава класичне физичке команде, а да је то заправо само кориснички улазни уређај за дигиталну аудио радну станицу. Такви уређаји се зову контролне табле за дигиталне аудио радне станице (слика 3).

[2] MIDI је industrijski standard za povezinanje muzičkih uređaja, https://www.midi.org/

Слика 3. Avid Control Surface-S3 Controller

3. Дигиталне аудио радне станице

Дигиталне аудио радне станице су уређаји или апликативни софтвер за снимање, обраду и продукцију песама, говора, аудио ефеката и звука генерално. Модерне радне станице су апликације за персоналне рачунаре (*PC* или *Mac*), а постоје и у форми апликација за таблете и паметне телефоне. Користе се за продукцију музике, радија, телевизије, мултимедије за интернет и још много тога. Радне станице у форми специјализованог уређаја још увек се могу срести у ТВ и радио станицама, али се и ту замењују са апликацијама за персоналне рачунаре.

Поред тога што замењује функције бројних аналогних уређаја, дигитална аудио радна станица доноси и суштинске новине, као што су

1. нелинеарна монтажа (енгл. *non-linear editing*),
2. широка понуда виртуелних музичких инструмената, ефеката и индикатора у виду софтверских компоненти,
3. код неких радних станица постоји могућност додатне аутоматизације скриптима и екстензијама.

Нелинеарна монтажа значи да када, на пример, на почетне снимке применимо неке ефекте, исецамо делове снимка и комбинујемо их у нови снимак и сл., ми не мењамо аудио фајлове у којима су ти снимци сачувани, већ радна станица посебно памти све трансформације које смо применили на почетне снимке и кориснику презентује крајњи резултат. Сваки пут када радна станица пушта монтирани звук (команда *play*) примењују се „у лету“ све трансформације. То значи да у сваком тренутку можемо променити параметар за ефекат који смо користили у првој фази и да одмах пустимо како то на крају звучи, уместо да морамо да поново монтирамо наредне фазе, као што бисмо то морали у класичној линеарној монтажи.

Виртуални музички инструменти су софтверске компоненте којима генеришемо звук са широком понудом инструмената независних аутора. Слична ствар је и са ефектима, који су у аналогној технологији посебни уређаји, док у дигиталној технологији могу да представљају софтверске компоненте. На крају и разни индикатори који анализирају и визуелно приказују параметре звука на одређеном каналу такође могу да се имплементирају као софтверске компоненте.

Софтверске компоненте за виртуелне музичке инструменте, ефекте и индикаторе се у свету дигиталних радних станица називају аудио плагиновима [1] (енгл. *audio plug-in*). Посебно значајна ствар је да постоје стандарди за аудио плагинове које подржавају разне дигиталне аудио радне станице. Аутор плагина може да се определи за неки од шире прихваћених стандарда и на тај начин подржи разне типове радних станица. Захваљујући стандардизацији је формирана велика понуда плагинова који могу да се користе у разним радним станицама. Најпознатији стандард тог типа је *Virtual Studio Technologies* (*VST*) који је развила компанија *Steinberg* [2]. Стандард *VST* није подржан од стране неког признатог стандардизационог тела нити индустријског конзоцијума, али је просто заживео у пракси.

Поред плагинова, поједине дигиталне аудио радне станице подржавају разне врсте проширења којима се аутоматизују поједине активности и имплементирају нове функционалности. Таква проширења, за разлику од плагинова, нису стандардизована између различитих дигиталних аудио радних станица.

У оквиру овог матурског рада биће приказана и имплементација софтверског модула који представља једно такво проширење (поглавље 1).

Први покушаји дигиталне обраде звука у седамдесетим и осамдесетим годинама двадесетог века суочили су се са проблемом веома скупе меморије, релативно спорим процесорима и малом брзином хард дискова.

Једна од првих масовније коришћених дигиталних аудио радних станица је *Soundstream* која се појавила на тржишту 1978. године и која је користила најмодерније компјутерске компоненте тог времена. Састојао се од централног рачунар (*Digital Editing System*) DEC PDP-11/60 који је покретао софтвер зван *Digital Audio Processor,* четрнаестоинчних хард дискова, осцилоскопа за анализу звучних таласа снимљених на дискове и рачунарског терминала.

Слика 4. DEC PDP-11/60

У другој половини осамдесетих кућни и персонални рачунари као што су *Apple Macintosh*, *Atari ST* и *Commodore Amiga* били су довољно моћни за дигиталну обраду звука. У истом периоду се појављује *MIDI* интерфејс и клавијатуре које се могу повезати са рачунаром. Појавом радне станице *Pro Tools* компаније *Digidesign* 1991. велики музички студији су са традиционалне аналогне почели да прелазе на дигиталну обраду.

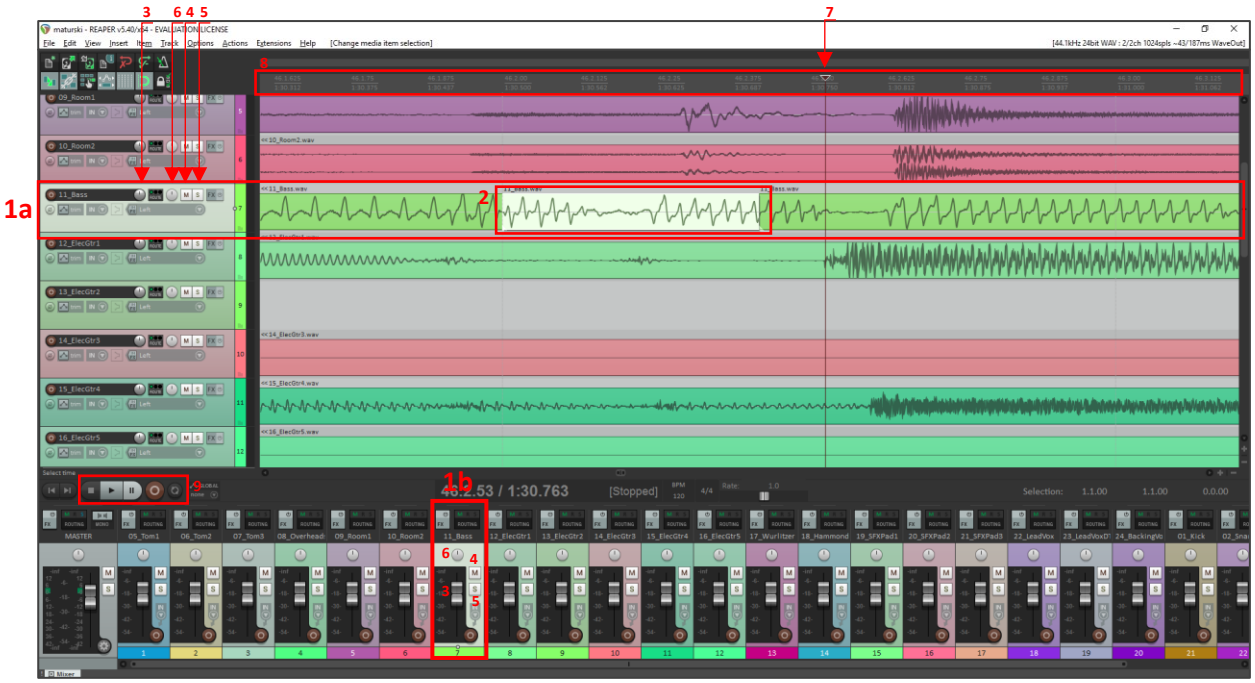
Концепт дигиталне продукције звука се није суштински променио протеклих 25 година (погледај слику 5), а поред *Pro Tools*-а популарне аудио радне станице су *Cubase Pro*, *Logic Pro, Studio One*, *Ableton Live, FL Studio, Reason* и друге.

Слика 5. Pro Tools V1.0 (1991) и Pro Tools V12 (2015)

Последњих пар година нагло расте популарност радне станице *Reaper* како међу професионалним студијима тако и аматерима. Ствари које издвајају *Reaper* су приступачна цена, брзина и пре свега модуларност. *Reaper* представља први аудио радну станицу која кориснику оставља могућност да у потпуности прилагоди кориснички интерфејс својим потребама и

навикама. Такође постоји могућност писања скрипти којима је могуће аутоматизовати готово све послове као и коришћење екстензија независних аутора. Велика флексбилност има и своју цену, многи професионалци нису спремни да потроше сате на подешавања, на тражење нових скрипти и екстензија које ће им на крају пружити ефикаснији рад.

Централни део корисничког интерфејса (слика 6) сваке дигиталне аудио радне станице садржи дводимензиони приказ аудио канала: по хоризонтали је временска оса, а по вертикали су један испод другог поређани различити канали, тј. траке.



Слика 6. Изглед корисничког интерфејса од Reaper DAW

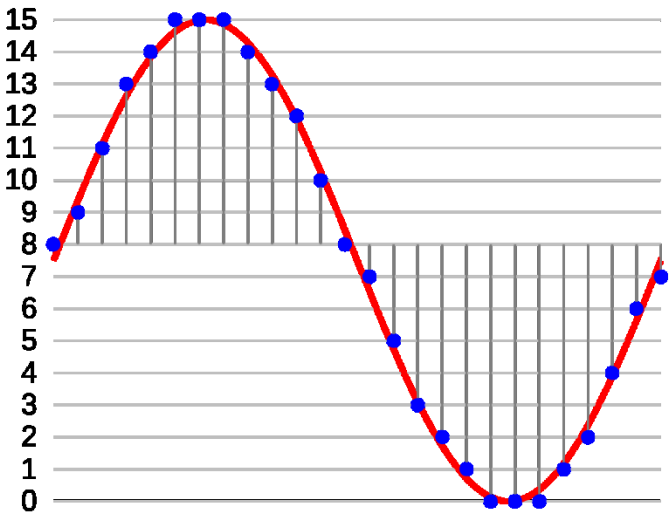
1a	канал у хоризонталном призаку
1b	канал у миксер приказу
2	исечак (*media item*)
3	потенциометар за јачину звука
4	дугме за утишавање канала (mute)
5	дугме за соло режим
6	поменциометар за пановање
7	курсор
8	временска линија
9	заустави/пусти/паузирај/снимај, врти у круг дугмићи

Можемо рећи да кориснички интерфејс имитира концепт канала на миксетама, при чему се додатно графички приказује и таласни облик сигнала. С обзиром да је практичније да временска оса буде хоризонтална, траке су хоризонталне, распоређене једна испод друге. На слици 6 у доњем делу прозора видимо да *Reaper* омогућава и додатан приказ распореда команди као на миксети.

4. Дигитални запис и обрада аудио сигнала

Математички модел дигиталног записа аудио сигнала потиче од *PCM* (*Pulse-code modulation*) метода дигиталног записивања аудио сигнала [3].

PCM метод ћемо илустровати на примеру[3].



Слика 7. Узорковање и квантификовање сигнала у 4-битном LPCM методу

На слици 7. је приказан синусоидни звучни талас (црвена крива) који је узоркован и квантификован *PCM* методом. Талас је узоркован у правилним временским интервалима који су приказани вертикалним линијама. На тај начин добијамо дискретну репрезентацију улазног сигнала (плаве тачке) која се даље дигитално записује за потребе чувања, преноса и/или обраде. Када су нивои квантификовања линеарни (код електричног аналогног сигнала то значи да дупло већи број означава приближно дупло већи напон) метод зовемо и *Linear Pulse-Code Modulation* (*LPCM*), мада се префикс *Linear* (линеарни) често изоставља јер је то најчешћи случај примене *PCM* метода.

За синусоидни талас на слици 7. добијамо серију очитаних вредности нивоа сигнала:

8, 9, 11, 13, 14, 15, 15, 15, 14,…

А иста серија у бинарном бројном систему изгледа:

1001, 1011, 1101, 1110, 1111, 1111, 1111, 1110, …

Када бинарне записе бројева надовежемо један на други добијамо дигиталан сигнал у *PCM* формату. Аудио CD-ови, на пример, за сваки од два стерео канала користе 16-битни *PCM* запис са учесталошћу узорковања од 44.1 KHz [4].

Постоје бројни други формати аудио записа, али сви који су у широј употреби полазе од истог математичког модела *PCM* серије.

---

[3] Пример и пратећа слика су презети из чланка „*Pulse-code modulation*“ Википедије, https://en.wikipedia.org/wiki/Pulse-code_modulation

У дигиталној обради се аудио сигнал обично представља као *PCM* серија бројева, тј. неколико серија уколико имамо више канала.

PCM серија која улази у дигиталну обраду може потицати од:

1. аналогно-дигиталне (АД) конверзије,
2. дигиталног улазног интерфејса,
3. аудио снимка сачуваног у дигиталном формату који је декодиран у просту *PCM* серију или
4. резултата претходне фазе дигиталне обраде.

PCM серија која је резултат дигиталне обраде може даље бити:

1. конвертована у аналогни сигнал помоћу дигитално-аналогоне (ДА) конверзије,
2. прослеђена на дигитални излазни интерфејс,
3. енкодирана у аудио снимак који се чува у дигиталном формату или
4. улаз у наредну фазу дигиталне обраде.

Електронска кола која конвертују аналогни аудио сигнал у дигитални називају се аналогно-дигитални (АД) конвертери, а електронска кола која дигитални сигнал конвертују у аналогни називају се дигитално-аналогни (ДА) конвертери.

На слици 1. се АД и ДА конвертери, као и дигитални улазни и излазни интерфејси налазе у аудио интерфејсу прикљученом на рачунар. Дигиталну обраду сигнала обавља централни процесор рачунара под контролом рачунарске апликације за дигиталну аудио радну станицу заједно са инсталираним плагиновима и проширења. Аудио снимци у дигиталном формату се чувају као фајлови на хард диску прикљученом на рачунар (интерном или екстерном).

Бројеви у *PCM* серији могу бити:

1. неозначени цели бројеви у опсегу од 0 до $2^d - 1$, где нулти ниво има вредност 2^{d-1} (као у примеру са слике 7, где је $d = 4$),
2. означени цели бројеви у опсегу од -2^{d-1} до $2^{d-1} - 1$ или
3. бројеви у покретном зарезу у опсегу од -1.0 до 1.0, при чему се у међурезултатима између фаза дигиталне обраде може дозволити да бројеви изађу из тог опсега.

За правилну интерпретацију сигнала треба нам још учесталост узорковања, а у случају целих бројева и вредност d која представља број битова.

4.1.1. Пример дигиталне обраде сигнала

Претпоставимо да имамо k дигиталних аудио канала које је потребно комбиновати у један моно канал (класично „миксовање“). Сваком каналу може да се подешава улазна јачина звука, за шта замишљамо да се користе потенциометри у опсегу -60dB до 30dB. Позиција потенциометра на 0dB значи да се сигнал нити појачава нити утишава.

Користићемо следеће ознаке за улазне и излазне податке:

d – број битова за улазну и излазну *PCM* серију

k – број улазних канала

$s_{j,i}$ – *PCM* серија неозначених целих бројева за j-ти улазни канал (0 до $2^d - 1$)

p_j – положај потенциометра за j-ти улазни канал (-60dB до 30dB)

o_i – *PCM* серија неозначених целих бројева за излазни канал (0 до $2^d - 1$)

Прво ћемо вредност улазне серије нормализовати на интервал $[-1,1)$, тј. конвертовати у *PCM* серију бројева у покретном зарезу

$$s_{j,i}^* = \frac{s_{j,i} - 2^{d-1}}{2^{d-1}}$$

Приметимо да $s_{j,i}^*$ постиже вредност -1 када је $s_{j,i} = 0$, али не може да достигне вредност 1. Највећа могућа вредност од $s_{j,i}^*$ се постиже када је $s_{j,i} = 2^d - 1$, а то је

$$\frac{2^d - 1 - 2^{d-1}}{2^{d-1}} = 1 - \frac{1}{2^{d-1}}$$

Затим треба да одредимо коефицијент појачања c_j који одговара вредности у децибелима. За то користимо формулу

$$c_j = \sqrt{10^{\frac{p_j}{10}}} = 10^{\frac{p_j}{20}}$$

Вредност испод корена представља формулу за конверзију према дефиницији децибела. Дефиниција децибела је заснована на релативном односу снага звучних сигнала. Са друге стране АД конверзија мери напон, а напон на излазу микрофона не одговара снази звука, већ звучном притиску. Како је снага звука сразмерна квадрату амплитуде звучног притиска, а бројеви у *PCM* серији су сразмерни звучном притиску, коефицијент појачања/утишања који се примењује на бројеве из *PCM* серије треба да је корен коефицијента који би се применио на снагу звука.

Нормализован излазни сигнал ће бити

$$o_i^* = \sum_{j=0}^{k-1} c_j \cdot s_{j,i}^*$$

Остало је да нормализовани излазни сигнал вратимо на целобројни

$$o_i = \left\lfloor (\min(\max(o_i^*, -1 + \varepsilon), 1 - \frac{1}{2^{d-1}} + \varepsilon) + 1) \cdot 2^{d-1} \right\rfloor$$

где је ε мали позитиван број који штити од непрецизности аритметике у покретном зарезу. Таква непрецизност би код целобројног одсецања које се примењује на крају могла да направи резултат за један мањи од жељеног. Било која вредност од ε која је између 0 и $\frac{1}{2^{d-1}}$ даје исти резултат, тако да за ε можемо узети $\frac{1}{2^d}$.

У обради дигиталног сигнала неки пут је згодније радити са нормализованим вредностима у покретном зарезу, а за неке алгоритме је ефикасније да се примењују директно на целобројне вредности. Приметимо да у овом примеру бројеви у покретном зарезу у међурезултатима могу да изађу из интервала $[-1,1)$ и нису ограничени на вредности које се могу добити конверзијом из целобројних. Због тога смо код конверзије назад у целе бројеве прво ограничили вредности на интервал $[-1,1)$ а на крају смо одсекли цео део.

4.1.2. Фуријеов развој и дискретна Фуријеова трансформација

Претпоставимо да имамо периодични звучни талас. Акустички то значи да звук не мења ни висину ни боју ни јачину, а математички то значи да је функција f која описује звучни талас

периодична. Домен функције f је на временској оси, а вредност функције је ниво сигнала, што у физичком смислу може представљати звучни притисак или напон у електричном сигналу. Функција f је периодична са периодом T уколико је $f(x+T)=f(x)$ за свако x за које и x и $x+T$ припадају домену функције f. Фреквенција звука је при томе $\nu=\frac{1}{T}$.

Идеја Фуријеовог развоја [5] је да периодичну функцију покушамо апроксимирати тригонометријском сумом

$$s_N(x)=\frac{A_0}{2}+\sum_{n=1}^{N}A_n\sin\left(\frac{2n\pi}{T}x+\phi_n\right)$$

Приметимо да функција $\sin\left(\frac{2n\pi}{T}x+\phi_n\right)$ има периоду $\frac{T}{n}$, што значи да представља синусоидни звучни талас фреквенције $n\nu$. У акустици је то фреквенција n-тог хармоника. Значи да ће Фуријеов развој представљати математичку интерпретацију разлагања звука на хармонике, при чему је A_n интензитет n-тог хармоника, а ϕ_n је фазни померај n-тог хармоника. $\frac{A_0}{2}$ представља нулти положај у звучној осцилацији.

Ако применимо адитивну формулу за $\sin$, добијамо:

$$s_N(x)=\frac{a_0}{2}+\sum_{n=1}^{N}a_n\cos\left(\frac{2n\pi}{T}x\right)+\sum_{n=1}^{N}b_n\sin\left(\frac{2n\pi}{T}x\right)$$

Где је $a_0=A_0$, $a_n=A_n\sin(\phi_n)$, $b_n=A_n\cos(\phi_n)$. У Фуријеовом развоју се вредности коефицијената $a_0,a_1,b_1,\dots,a_n,b_n$ одређују према следећим формулама:

$$a_n=\frac{2}{T}\int_{x_0}^{x_0+T}f(x)\cos\left(\frac{2n\pi}{T}x\right)dx$$

$$b_n=\frac{2}{T}\int_{x_0}^{x_0+T}f(x)\sin\left(\frac{2n\pi}{T}x\right)dx$$

Ако су испуњени одређени услови за функцију f, s_N са овако дефинисаним коефицијентима конвергира ка f.

Фуријеов развој такође може да се представи у комплексној форми:

$$s_N(x)=\sum_{n=-N}^{N}c_n e^{i\frac{2n\pi}{T}x}$$

При чему је

$$c_n = \begin{cases} \frac{1}{2}a_0, & n = 0 \\ \frac{1}{2}(a_n - ib_n), & n > 0 \\ \frac{1}{2}(a_{-n} - ib_{-n}), & k < 0 \end{cases}$$

Ако усвојимо да је $b_0 = 0$, можемо краће записати $c_n = \frac{1}{2}\left(a_{|n|} - ib_{|n|}\right)$. Такође важи

$$c_n = \frac{1}{T}\int_{x_0}^{x_0+T} f(x)e^{-i\frac{2n\pi}{T}x}dx$$

Приметимо и да је $|c_n| = |c_{-n}| = A_{|n|}$, што значи да на основу коефицијената c_n непосредно одређујемо интензитет хармоника.

Када су познате вредности функције f у одређеном броју тачака (као код PCM серије) поставља се питање како да приближно одредимо вредности коефицијената из Фуријеовог развоја.

Претпоставимо да на периоди $[0, T)$ знамо вредности функције f у N тачака

$$y_0 = f(x_0), y_1 = f(x_1), \dots, y_{N-1} = f(x_{N-1})$$

где је

$$x_k = k\frac{T}{N}$$

Тада интеграл

$$c_n = \frac{1}{T}\int_0^T f(x)e^{-i\frac{2n\pi}{T}x}dx$$

Можемо апроксимирати са

$$c_n \approx \bar{c}_n = \frac{1}{T}\frac{T}{N}\sum_{k=0}^{N-1} y_k e^{-i\frac{2n\pi}{T}x_k} = \frac{1}{N}\sum_{k=0}^{N-1} y_k e^{-i\frac{2nk\pi}{N}}$$

Пресликавање $(y_0, y_1, \dots, y_{N-1}) \mapsto (Y_0, Y_1, \dots, Y_{N-1})$ одређено формулом

$$Y_n = \sum_{k=0}^{N-1} y_k e^{-i\frac{2nk\pi}{N}}$$

назива се (по дефиницији) дискретном Фуријеовом трансформацијом [6].

Ако би се применио алгоритам за рачунање дискретне Фуријеова трансформације који непосредно имплементира формулу, ефикасност тог алгоритма би очито била $O(N^2)$.

Постоји ефикаснији алгоритам за рачунање дискретне Фуријеове трансформације који се назива брза Фуријеова трансформација [6] (енгл. *Fast Fourier Transform*, *FFT*) и који има ефикасност $O(N\log(N))$. Када је потребна анализа хармоника у дигиталној обради сигнала, обично се користи библиотека која има имплементиран *FFT* алгоритам.

Све у свему, анализу хармоника радимо тако што прво из *PCM* серије издвојимо N узастопних вредности из временског периода $T = \frac{N}{\nu_u}$, где је са ν_u означена фреквенција узорковања. Затим урадимо брзу Фуријеову трансофрмацију и добијемо $Y_0, Y_1, \dots, Y_{N-1}$. Хармоник са фреквенцијом $\frac{n}{T} = n\frac{\nu_u}{N}$ има интензитет $\frac{|Y_n|}{N}$.

Уколико желимо да на основу декомпозиције на хармонике одредимо тачне параметре синусоидних сигнала, приближно одређујемо c_n као $\frac{Y_n}{N}$.

Када желимо да синтетишемо звучни сигнал приближан изворном, примењујемо инверзну дискретну Фуријеову трансформацију:

$$y_n = \sum_{k=0}^{N-1} Y_k e^{i\frac{2nk\pi}{N}}$$

Каже се да дискретна Фуријеова трансформација преводи из временског у фреквентни домен, а инверзна дискретна Фуријеова трансформација преводи назад из фреквентног у временски домен. Пре превођења назад у временски домен се може направити измена на нивоу фреквентног домена (на пример променити интензитет одређених фреквенција), на чему се базирају многе технике дигиталне обраде сигнала.

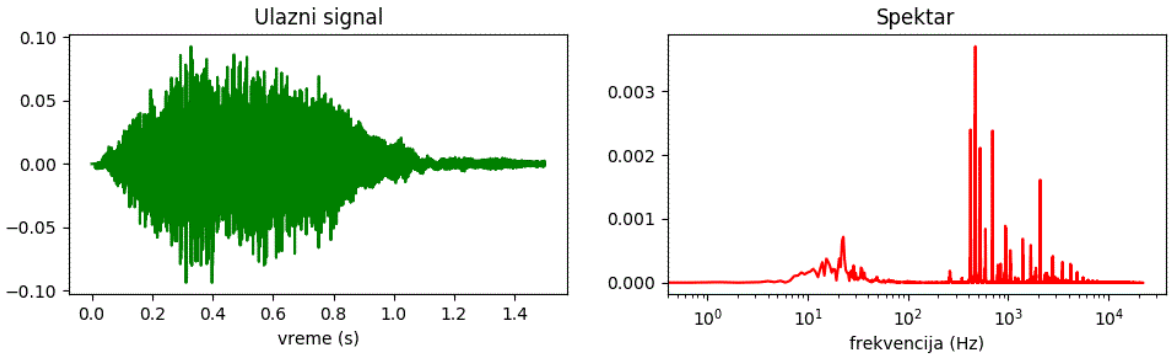
У пракси се често узима велики период на коме се примењује дискретна Фуријеова трансформација, већи од највеће таласне дужине чулног звука, па се детектују различите фреквенције звука као хармоници фиктивног звучног таласа који је на врло ниској фреквенцији. То што смо описали је један од метода спектралне анализе звука.

На слици 8 је приказана имплементација спектралне анализе у програмском језику *Python*.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import wavfile
# čitamo snimak
frek_uz, data = wavfile.read('violina.wav')
# uzimamo početnih 1.5 sekundi levog kanala i normalizujemo 16-bitne brojeve
pcm = data.T[0][0:1.5*frek_uz] / 2**16
N = len(pcm)
# odredimo Furijeove koeficijente za prvu polovinu rezultata FFT
c = np.fft.fft(pcm)[0:N/2] / N
# intenzitete u spektru određujemo kao apsolutne vrednosti kompl. brojeva
spektar = abs(c)
# crtamo grafikone
plt.subplot(1, 2, 1)
plt.title('Ulazni signal')
plt.xlabel('vreme (s)')
vremenska_osa = np.linspace(0.0, N/frek_uz, N)
plt.plot(vremenska_osa, pcm, 'green')
plt.subplot(1, 2, 2)
plt.title('Spektar')
plt.xscale('log')
plt.xlabel('frekvencija (Hz)')
frek_osa = np.linspace(0.0, len(spektar)*frek_uz/N, len(spektar))
plt.plot(frek_osa, spektar, 'red')
plt.show()
```

Слика 8. Имплементација спектралне анализе у програмском језику Python

Резултат извршавања програма са слике 8 је приказан слици 9.



Слика 9. Резултат извршавања програма са слике 8

Програм користи модуле *NumPy*, *SciPy* и *matplotlib* који нису део основне дистрибуције *Python*-а, али су у широкој употреби у области инжењерских и научних израчунавања [7].

Функција *wavfile.read* враћа фреквенцију узорковања и објекат који садржи једну или више *PCM* серија (за случај да аудио фајл има више канала). При томе се користе *NumPy* низови за које су скаларне математичке операције додефинисане тако да се извршавају над сваким појединачним елементом низа и дају нови низ. То нам је омогућило једноставнији код на

неколико места (коришћено је за дељење елемената низа скаларом и рачунање апсолутних вредности елемената низа).

У програму смо користили само прву половину резултата *FFT* што је уобичајена пракса.

Приметимо да улазни сигнал на слици 9 није чист периодичан сигнал. Са слике 9 видимо да свакако мења амплитуду. Фуријеов ред смо теоријски објаснили на бази чистог периодичног сигнала, али се у пракси дискретна Фуријеова трансформација може користити и када сигнал није такав. Интерпретација тако примењене Фуријеове трансформасије је комбинација искуственог и теоријског знања. У конкретном примеру је у питању снимак акорда на виолини, тако да се у спектралној анализи препознају хармоници, као и чињеница да звук долази са више жица.

5. Софтверски модул за поравнавање маркера за *Reaper*

Приликом снимања у студију, обично се користи посебан канал за сваки инструмент. При томе се за акустичне инструменте користе микрофони који су пажљиво постављени и усмерени да сниме звук одређеног инструмента.

Даље ћемо се усредсредити на снимке удараљки, пре свега бубњева, а слични проблеми се могу појавити и на сличан начин решавати и код других врста инструмената.

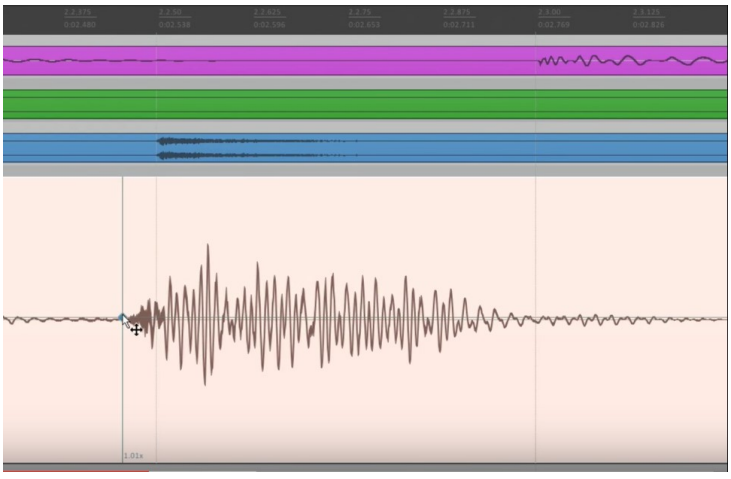
Звук бубња доминантно одређује ритам и због тога ударци који нису временски прецизни доста умањују квалитет извођења. За временску корекцију удараца користе се елементарне технике аудио монтаже, као што је сечење и померање исечака или растезање и сабијање делова звучног записа тако да ударац дође на своје место. Једини проблем је што се у оквиру извођења дешава пуно удараца (више стотинина), па временске корекције удараца могу бити заморан и дуготрајан посао приликом монтаже звука. Због тога дигиталне аудио радне станице настоје да са могућностима које нуде олакшају тај посао.

Временска корекција удараца се грубо може поделити у две фазе:

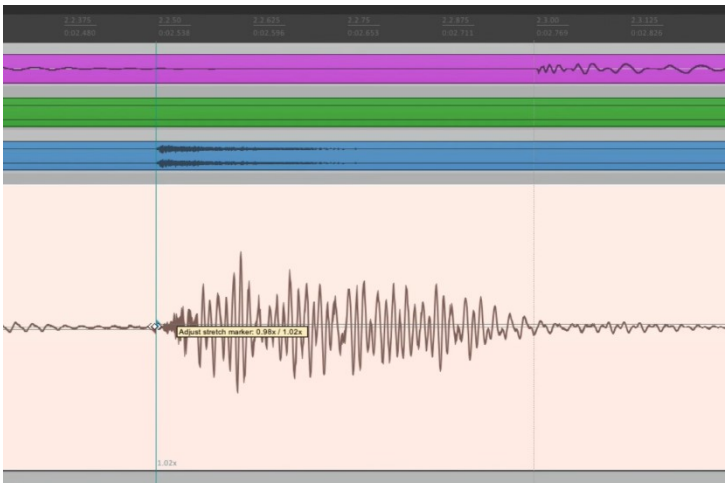
1. Маркирање удараца
2. Извођење трансформација тако да се маркиране тачке у снимку помере на задато место

У случају да користимо технику растезања и сабијања звучног записа (енгл. *stretch and shrink*) тада у првој фази постављамо маркере за растезање (слика 10), а у другој фази померамо маркере за растезање тако да ударац буде на месту где треба (слика 11). Због померања маркера у десно, део снимка лево од маркера (од претходног маркера или од почетка) постаје мало развучен, а део снимка десно од маркера (до следећег маркера или до краја) постаје мало сабијен.

Просто растезање и сабијање сигнала би променило фреквенцију, тј. висину, звука јер се исти број звучних осцилација дешава у дужем (код растезања) или краћем (код сабијања) времену. Због тога се примењују посебни алгоритми за временско скалирање звука који не мењају висину звука [8].



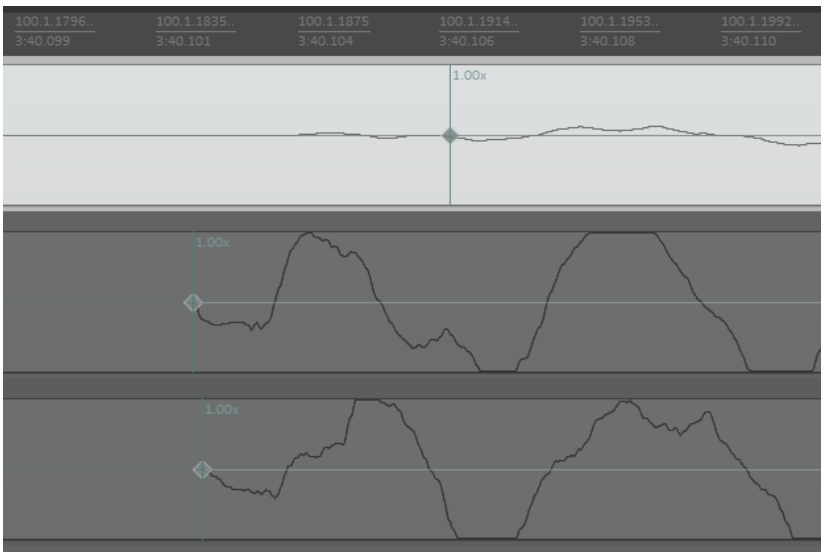
Слика 10. Постављање маркера за растезање на почетак ударца (притиском на Shift-W)



Слика 11. Померањем макрера за растезање помера се почетак ударца

Да не бисмо морали ручно да подешавамо маркер по маркер, *Reaper* омогућава аутоматско маркирање удараца кроз *Dynamic Split* акцију. *Dynamic Split* акција нуди опцију да умеће маркере за растезање уместо да сече звучни клип. *Dynamic Split* има могућност подешавања бројних параметара и најчешће се може постићи задовољавајуће препознавање удараца.

Проблем који треба да решимо настаје када имамо више трака у којима желимо да поставимо маркере. У случају када су ударци из разних трака довољно добро међусобно синхронизовани или је чак у питању снимак истог ударца са два микрофона, мала је вероватноћа да ће *Dynamic Split* поставити маркере на потпуној истој позицији, разликоваће се за неколико милисекунди или за део милисекунде (слика 12).



Слика 12. Акцијом Dynamic Split маркери у три суседне траке су постављени на блиске позиције

Акција *Dynamic Split* нема опцију да поставља маркере узимајући у обзир више трака истовремено. Једино што нуди је опција да маркере постави у све траке из групе на исте позиције, али само на основу садржаја једне траке.

Нама је потребно да постигнемо да се идентификују ударци у неколико изабраних трака, да се на крају у свим тракама поставе маркери на истим позицијама, али тако да се мале разлике у позицијама поравнају.

5.1. Функционалност софтверског модула

Идеја је да користимо постојећу функционалност *Dynamic Split* акције и да имплементирамо нову акцију за поравнавање маркера коју ћемо извршавати након постављања маркера помоћу *Dynamic Split*.

Нова акција „Поравнај маркере“ би требало да се користи тако што се прво селектују аудио исечци (енгл. *audio clip*, а *Reaper* користи термин *media item*) у којима се већ налазе маркери за растезање, а затим се покрене акција.

Када се појаве маркери на временски блиским позицијама (праг толеранције за одређивање шта је временски блиско корисник задаје на почетку) акција ће узети у обзир само приоритетнији маркер, при чему се приоритет гледа на следећи начин:

- ако су маркери из различитих трака, приоритетнији је маркер из оне траке која је изнад
- ако су маркери из исте траке, приоритетнији је маркер који је временски ранији

Након што се на тај начин занемаре маркери који су временски блиски приоритетнијим маркерима, од преосталих маркера из свих аудио исечака се формира коначан скуп позиција маркера.

На крају у сваком аудио исечку треба да остану уписани маркери на позицијама из коначног скупа које упадају у временски опсег датог исечка. Другим речима, на свакој позицији из коначног скупа се у свакој траци појављује по један маркер, осим када у траци дата позиција није покривена ни једним од селектованих аудио исечака.

5.2. Техничка реализација софтверског модула

Reaper нуди неколико могућности за израду софтверског модула који имплементира нову акцију.

У оквиру *ReaScript* могућности подржана су три програмска језика: *EEL2* (језик који је посебно дизајниран у оквиру *Reaper*-а), *Lua* и *Python*. Поред тога је подржана израда проширења у *C++* програмском језику, када нам је на располагању и шири скуп ствари које можемо извести, као што је додавање опција у менију при инсталацији проширења, слобода у изради корисничког интерфејса, ефикаснији рад, итд. Са друге стране, израда проширења у *C++* програмском језику је знатно сложенија.

Идеја је била да у оквиру матурског рада израдим прототип користећи *ReaScript* могућност, а евентуална израду проширења у *C++* програмском језику је остављена за будући рад.

Имплементације програмских језика *EEL2* и *Lua* уграђене су у *Reaper*, док *Python* треба посебно инсталирати, али је након тога интеграција са *Python*-ном једнако добра као са друга два програмска језика. *EEL2* је недовољно документован и нема подршку независних едитора. *Lua* се као уграђени програмски језик доста користи у развоју рачунарских игара и све више у софтверу за *IoT* (*internet of things*), али има сведене могућности програмског језика и стандардне библиотеке, тако да тражи мало више напора и навикавања када дође до имплементације нетривијалних алгоритама и структура података. На крају се показало да је *Python* најудобнији за израду овог софтверског модула, а додатан напор да се инсталира и конфигурише *Python* је минималан. Наравно, уколико би се софтверски модул дистрибуирао већем броју корисника, потреба инсталације *Python*-а би могла бити проблем, али бисмо се у том случају све једно определили за израду проширења у *C++* програмском језику.

У имплементацији софтверског модула морали смо да решимо серију техничких и пројектантских питања:

1. Како одређене ствари постићи кроз позиве функција из *ReaScript API*, а посебно:
    a. Како очитати све потребне податке из *Reaper* објеката и како на крају ажурирати *Reaper* објекте
    b. Како додатно комуницирати са корисником и како запамтити подешавања које је корисник изабрао
    c. Како обезбедити да се приказ у *Reaper*-у уредно освежи након завршетка акције и да се акција правилно приказује код *undo* и *redo* командама
2. Како пројектовати интерну структуру података у коју ће да се смести очитане вредности и на основу које ће да се формирати пречишћен скуп позиција
3. Како алгоритам за проналажење блиских маркера да буде у зони задовољавајуће ефикасности

5.2.1. Коришћење функција из *ReaScript API*

Очитавање података из *Reaper* објеката имплементирано је у функцији *ocitajPodatkeIZReapera* (прилог 1) и ту користимо позиве функција из *ReaScript API*:

1. Позивом функције *RPR_CountSelectedMediaItems* добијамо број селектованих исечака (променљива *brojMI*) у одређеном пројекту, где параметар нула означава текући пројекат (*Reaper* дозвољава да у више табова отворимо више пројеката).
2. Позивом *RPR_GetSelectedMediaItem* добијамо одређен исечак, тј. *Python* објекат преко кога даље приступамо одговарајућем *Reaper* објекту (променљива *mediaItem*).
3. За дати исечак узимамо активни тејк (променљива *activeTake*) позивом *RPR_GetActiveTake*. Један исечак може да има више тејкова, што у пракси најчешће представља више покушаја снимања.
4. Из исечка и тејка се очитава:
    a. почетна позиција исечка на временској скали пројекта (променљива *itemOffset*) позивом *RPR_GetMediaItemInfo_Value(mediaItem, "D_POSITION")*
    b. дижина исечка (променљива *itemLength*) позивом функције *RPR_GetMediaItemInfo_Value(mediaItem, "D_LENGTH")*
    c. позицију у односу на почетак снимка из аудио фајла (променљива *srcOffset*) позивом *RPR_GetMediaItemTakeInfo_Value(activeTake, "D_STARTOFFS")*
5. До траке (променљива *track*) се долази преко исечка позивом функције *RPR_GetMediaItem_Track(mediaItem)*.
6. За траку се очитава јединствени идентификатор (променљива *guid*) који се користи да знамо да ли смо први пут дошли до те траке или смо преко неког другог исечка већ приступали тој траци, као и назив траке (променљива *tname*) позивима функција *RPR_GetTrackGUID* и *RPR_GetTrackName*
7. Даље се из тејка очитавају маркери тако што се прво очита колико има маркера позивом. *RPR_GetTakeNumStretchMarkers*, а затим се очитава позиција сваког од маркера позивом *RPR_GetTakeStretchMarker*.

Код ажурирања *Reaper* објеката користи са прво функција *RPR_DeleteTakeStretchMarkers* којом бришемо један или више маркера за растезање из тејка (у овом случају бришемо све маркере) и затим функција *RPR_SetTakeStretchMarker* којом уписујемо маркер на позицију у тејку. Код уписивања је потребно навести и одговарајућу позицију у изворном снимку. Неопходно је тачно израчунати позицију у изворном снимку, јер ће у супротном *Reaper* растегнути снимак тако да

задата позиција у снимку дође на позицију маркера. Овде смо наишли на незгодну ситуацију да *ReaScript API* предвиђа да је позиција у изворном снимку опциона и тада се аутоматски одређује, али у варијанти за *Python* одговарајући параметар просто није стављен као опциони. Зато смо морали да решимо како да сами израчунамо позицију у изворном снимку, али нам је то помогло да додатно разумемо како ствари функционишу.

Комуникација са корисником и памћење шта је корисник изабрао дешава се у функцији *ucitajParametre*. *ReaScript API* има ограничене могућности за комуникацију са корисником: функција *RPR_GetUserInputs* подиже прозор у коме се од корисника тражи да унесе један или неколико параметара, док функција *RPR_ShowMessageBox* кориснику приказује поруку.

Учитани параметри могу да се запамте између два покретања акције позивањем *RPR_SetExtState*, а добијају се назад позивом *RPR_GetExtState*. Пре тога се позивом *RPR_HasExtState* може поставити питање да ли то што хоћемо да тражимо постоји.

Да би се ажурирана стања *Reaper* објеката одмах по завршетку акције приказала у корисничком интерфејсу, потребно је позвати *RPR_UpdateArrange* пре краја акције. Да би се акција уредно приказивала код *undo* и *redo* команди, потребно је позвати *RPR_Undo_BeginBlock* на почетку акције и позвати *RPR_Undo_EndBlock* на крају акције.

5.2.2. Интерна структура података

У интерној структури података користимо класе *MediaItemData* за очитане податке у вези са исечком и тејком, *МаркерData* за податке у вези са маркером, *TrackData* за податке у вези са траком и *RadniSkup* за корени објекат интерне структуре података.

Објекат класе *RadniSkup* садржи колекцију трака (објеката класе *TrackData*) коју истовремено чува као речник ради ефикасног приступа на основу јединственог идентификатора и као листу ради приступа у оригиналном редоследу.

Објекат класе *TrackData* садржи и колекцију исечака (objekata klase *MediaItemData*) и колекцију маркера (објеката класе *МаркерData*). Колекција маркера је у интерној структури података издигнута на ниво траке, јер позиције свих маркера у траци желимо да посматрамо заједно. Колекција маркера је сортирана по позицијама маркера, због чега је у класи *MarkerData* дефинисана функција *__lt__* која имплементира операцију мање.

5.2.3. Алгоритам за проналажење блиских маркера

Најједноставнији алгоритам за проналажење блиских маркера је поређење сваког са сваким и тај алгоритам има квадратну зависност од броја маркера. Пошто број маркера може да пређе 1000, унутрашња петља алгоритма може да пређе 1.000.000 пролаза, што значи да алгоритам треба оптимизовати уколико је могуће.

Уобичајено решење за овај тип проблема је сортирана колекција маркера и бинарна претрага, чиме се ефикасност повећава на $O(n \log(n))$ gde je n број маркера. Ми смо то урадили на нивоу сваке траке, тако да је остала квадратна зависност по броју трака, што већ није критично.

Додатна оптимизације није неопходна, а усложнила би и структуру и алгоритам. Наиме, морали бисмо додатно да правимо јединствену колекцију свих маркера. Генерално, приликом развоја софтвера треба рационално одмерити до које мере инсистирати на оптимизацији.

6. Закључак

Овај матурски рад није одговарао на питање да ли је боља аналогна или дигитална технологија. Квалитет звука не можемо до краја описати параметрима који би нам за исто извођење снимљено и обрађено на два различита начина дали одговор шта је боље. Квалитет звука је субјективан. Дигитална технологија нам омогућава да снимак „испегламо“ тако да звучи савршено прецизно и ритмички и мелодијски али то може звучати стерилно и неприродно. Нека одступања од математички идеалног извођења нису грешке већ део уметничког изражавања извођача и циљ нам је да то задржимо. Ситна одступања и кашњења такође чине звук природнијим и често пријатнијим. Исто као што извођач поред технике којом влада даје и свој уметнички допринос делу које изводи тако и посао продуцента има и техничког и уметничког.

Алати за дигиталну продукцију звука треба да омогуће продуценту ефикасно обављање техничког дела посла али и да могу да испрате стил рада продуцента.

Софтверски модул који сам имплементирао и који је описан у завршном поглављу управо одговара на такве потребе и подржава да се мала временска одступања не исправљају.

У даљем раду ми је циљ да направим функционално заокружену екстензију која ће допунити могућности *Reaper*-а у области временске корекције одсвираних тонова. Једна од ствари коју ће екстензија подржати је очување грува. Грув је одступање од идеалног извођења нота које представља карактеристичан израз извођача или стила, а не грешку. Уколико буде било потребно имплементираћу и прилагођен алгоритам дигиталне обраде звучног сигнала за одређивање тренутка ударца.

Улога алгоритама у дигиталној продукцији све је значајнија. Са једне стране алгоритми све боље опонашају класичне аналогне компоненте са свим својим варијацијама и одступањима, а са друге стране алгоритми доносе нови квалитет. Оно што у једној верзији софтвера продуцент мора да уради у већем броју корака у наредној верзији ради алгоритам. У дизајну таквих алгоритама потребно је познавати продукцију звука, акустику, математику и програмирање. Израда матурског рада ми је помогла да уђем у ову мултидисциплинарну област користећи знање које сам стекао у Математичкој гимназији и Музичкој школи „Даворин Јенко“.

Литература

[1] A. Millard, *America on record: a history of recorded sound*, Cambridge University Press, 2005.

[2] M. A. Collins, *Professional Guide to Audio Plug-ins and Virtual Instruments*, Burlington, MA: Focal Press, 2003.

[3] Steinberg Media Technologies, *VST API Documentation*, 2017. https://www.steinberg.net/en/company/developers.html [приступљено 17. 05. 2017.]

[4] W. M. Waggener, *Pulse code modulation techniques: with applications in communications and data recording*, New York: Van Nostrand Reinhold, 1995.

[5] *Philips Compact Disc, Philips Historical Products*, Philips, 2013. http://www.philips-historische-producten.nl/cd-uk.html [приступљено 17. 05. 2017.]

[6] R. N. Bracewell, *The Fourier transform and its applications*, New York: McGraw-Hill, 1986.

[7] E. O. Brigham, *The Fast Fourier Transform and Its Applications*, New Jersey: Prentice-Hall, 1988.

[8] *Scientific Computing Tools for Python*. https://www.scipy.org/about.html [приступљено 17. 05. 2017.]

[9] B. G. Crockett, *High quality time-scaling and pitch-scaling of audio signals*. U.S. Patent 7,610,205, 27. 10. 2009.

Прилог 1. Изворни код модула за поравнавање маркера

```
from reaper_python import *

EXT_SECTION = "PoravnajMarkere"
EXT_PARAMS = "parametri"

def lowerBound(a, x, less):
    """Binarna pretraga.

    Argumenti:
        a - sortirana lista
        x - trazena vrednost
        less - fukncija koja poredi vrednost iz liste i trazenu vrednost,
            a pri tome odgovara redosledu sortiranja

    Rezultat koji se vraca:
        indeks koji predstavlja "lower bound" za trazenu vrednost, tj. ako
        trazena vrednost postoji, to je indeks prvog pojavljivanja trazene
        vrednosti, a ako trazena vrendost ne postoji, to je indeks na koji bi
        trebalo da se umetne po redosledu sortiranja.
    """
    def lowerBoundRec(a, b, e):
        if b == e:
            return b
        m = (b + e) // 2
        if less(a[m], x):
            return lowerBoundRec(a, m + 1, e)
        return lowerBoundRec(a, b, m)
    return lowerBoundRec(a, 0, len(a))

class MediaItemData:
    """Sadrzi podatke o Reaper-ovom MediaItem objektu."""
    def __init__(self, mediaItem, activeTake, itemOffset,
                 length, srcOffset, countOfMarkers):
        self.mediaItem = mediaItem
        self.activeTake = activeTake
        self.itemOffset = itemOffset
        self.length = length
        self.srcOffset = srcOffset
        self.countOfMarkers = countOfMarkers

class MarkerData:
    """Sadrzi podatke o Reaper-ovom 'streach marker'-u."""
    def __init__(self, position, itemOffset):
        self.position = position
        self.itemOffset = itemOffset
```

```
        self.globalPosition = position + itemOffset
        self.deleted = False

    def __lt__(self, other):
        return (self.globalPosition, self.itemOffset) < (other.globalPosition,
other.itemOffset)

    def rastojanje(self, other):
        return abs(self.globalPosition - other.globalPosition)

def less_MarkerData_GlobalPosition(marker, globalPosition):
    return marker.globalPosition < globalPosition

class TrackData:
    """Sadrzi podatke o Reaper-ovom MediaTrack objektu."""
    def __init__(self, guid, name):
        self.guid = guid
        self.name = name
        self.mediaItemDataList = []
        self.markerDataList = []

    def obrisiBliske(self, prioritetnijaTraka, prag):
        for marker in prioritetnijaTraka.markerDataList:
            if marker.deleted:
                continue
            i = lowerBound(self.markerDataList, marker.globalPosition - prag,
                           less_MarkerData_GlobalPosition)
            n = len(self.markerDataList)
            while i < n and marker.rastojanje(self.markerDataList[i]) <= prag:
                self.markerDataList[i].deleted = True
                i += 1

class RadniSkup:
    """Krovna klasa za podatke iz Reaper-ovih objekata."""
    def __init__(self):
        self.trackDataDict = {}
        self.trackDataList = []

    def getOrCreateTrackData(self, guid, name):
        if guid in self.trackDataDict:
            return self.trackDataDict[guid]
        track = TrackData(guid, name)
        self.trackDataDict[guid] = track
        self.trackDataList.append(track)
        return track

    def obrisiSveBliske(self, prag):
        for i in range(len(self.trackDataList)):
            for j in range(i + 1, len(self.trackDataList)):
```

```
                self.trackDataList[j].obrisiBliske(self.trackDataList[i], prag)

def ucitajParametre():
    """Ucitava od korisnika parametar za prag tolerancije.

    Vrednost koja se vraca:
    prag - prag tolerancije u sekundama u okviru koga se smatra
        da su dva markera vremenski bliska
    """
    if RPR_HasExtState(EXT_SECTION, EXT_PARAMS):
        uinput = RPR_GetExtState(EXT_SECTION, EXT_PARAMS)
    else:
        uinput = "5"

    while True:
        isOK, _, _, _, uinput, _ = RPR_GetUserInputs(
            "Parametri za poravnavanje markera", 1,
             "Prag u milisekundama", uinput, 100)

        if not isOK:
            return False, 0.0
        else:
            try:
                prag = float(uinput) / 1000
                RPR_SetExtState(EXT_SECTION, EXT_PARAMS, uinput, True)
                return True, prag
            except ValueError:
                porukaOGresci = "Neispravno unet broj: {}".format(uinput)
        RPR_ShowMessageBox(porukaOGresci, "Pogresan unos", 0)
        return False, 0.0

def ocitajPodatkeIZReapera(prag):
    """Iz Reaper-ovih objekata ocitava podatke u internu strukturu podataka.

    Argumenti:
        prag - prag tolerancije u sekundama

    Vraca se par:
        prva vrednost - da li je ocitavanje uspelo
        druga vrednost - objekat klase ``RadniSkup`` koji sadrzi internu
            strukturu podataka ocitanih iz Reaper-ovih objekata
    """
    brojMI = RPR_CountSelectedMediaItems(0)
    if brojMI == 0:
              RPR_ShowMessageBox(
             "Morate izabrati jednu ili vise stavki (media items)", "Greska", 0)
        return False, []

    RPR_ShowConsoleMsg("brojMI = {}\n".format(brojMI))
```

```
    radniSkup = RadniSkup()
    for i in range(brojMI):
        mediaItem = RPR_GetSelectedMediaItem(0, i)
        activeTake = RPR_GetActiveTake(mediaItem)
        if activeTake is None:
            continue
        itemOffset = RPR_GetMediaItemInfo_Value(mediaItem, "D_POSITION")
        itemLength = RPR_GetMediaItemInfo_Value(mediaItem, "D_LENGTH")
        srcOffset = RPR_GetMediaItemTakeInfo_Value(activeTake, "D_STARTOFFS")
        track = RPR_GetMediaItem_Track(mediaItem)
        guid = RPR_GetTrackGUID(track)
        _, _, tname, _ = RPR_GetTrackName(track, "", 100)
        trackData = radniSkup.getOrCreateTrackData(guid, tname)
        brojMarkera = RPR_GetTakeNumStretchMarkers(activeTake)
        mediaItemData = MediaItemData(
            mediaItem, activeTake, itemOffset, itemLength, srcOffset, brojMarkera)
        trackData.mediaItemDataList.append(mediaItemData)
        for idx in range(brojMarkera):
            _, _, _, pozicija, _ = RPR_GetTakeStretchMarker(
                                        activeTake, idx, 0.0, 0.0)
            trackData.markerDataList.append(MarkerData(pozicija, itemOffset))
    for trackData in radniSkup.trackDataList:
        trackData.markerDataList.sort()
        p = None
        for marker in trackData.markerDataList:
            if p is not None and abs(marker.globalPosition - p) < prag:
                marker.deleted = True
            else:
                p = marker.globalPosition
    return True, radniSkup

def upisiPromeneUReaper(radniSkup, prag):
    """Na osnovu interne strukture podataka se upsuju promene u Reaper-ove objekte.

    Argumenti:
        radniSkup - objekat klase ``RadniSkup`` u kome je upisana interna
            struktura podataka
        prag - prag tolerancije u minisekundama
    """
    poruka = ""
    finalMDL = []
    for trackData in radniSkup.trackDataList:
        izbrisanih = 0
        for marker in trackData.markerDataList:
            if marker.deleted:
                izbrisanih += 1
            else:
                finalMDL.append(marker)
        poruka += 'Traka: "{}", izbrisano/bilo markera: {}/{}\n'.format(
```

```
            trackData.name, izbrisanih, len(trackData.markerDataList))

    finalMDL.sort()

    for trackData in radniSkup.trackDataList:
        for mediaItemData in trackData.mediaItemDataList:
            take = mediaItemData.activeTake
            RPR_DeleteTakeStretchMarkers(take, 0, mediaItemData.countOfMarkers)
            levaPozicija = mediaItemData.itemOffset - prag
            desnaPozicija = mediaItemData.itemOffset + mediaItemData.length + prag
            i = lowerBound(finalMDL, levaPozicija, less_MarkerData_GlobalPosition)

            while i < len(finalMDL) and finalMDL[i].globalPosition <= desnaPozicija:
                markerData = finalMDL[i]
                razlikaPozicija = markerData.itemOffset - mediaItemData.itemOffset
                pozicija = markerData.position + razlikaPozicija
                srcPozicija = pozicija + mediaItemData.srcOffset
                RPR_SetTakeStretchMarker(take, -1, pozicija, srcPozicija)
                i += 1
    return poruka

def poravnajMarkere():
    """Glavna funkcija u izvrsavanju skripta."""
    isOK, prag = ucitajParametre()
    if not isOK:
        return
    isOK, radniSkup = ocitajPodatkeIZReapera(prag)
    if not isOK:
        return
    radniSkup.obrisiSveBliske(prag)
    poruka = upisiPromeneUReaper(radniSkup, prag)
    if poruka is not None:
        RPR_ShowMessageBox(poruka, "Poravnanje markera izvrseno", 0)

RPR_Undo_BeginBlock()
RPR_ClearConsole()
poravnajMarkere()
RPR_UpdateArrange()
RPR_Undo_EndBlock("Poravnavanje markera", -1)
```