FELADATKIÍRÁS

A feladatkiírást a tanszéki adminisztrációban lehet átvenni, és a leadott munkába eredeti, tanszéki pecséttel ellátott és a tanszékvezető által aláírt lapot kell belefűzni (ezen oldal *helyett*, ez az oldal csak útmutatás). Az elektronikusan feltöltött dolgozatban már nem kell beleszerkeszteni ezt a feladatkiírást.

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Méréstechnika és Információs Rendszerek Tanszék

Hangoló megvalósítása Android platformon

Szakdolgozat

Készítette
Pituk Dávid

Konzulens
Dr. Bank Balázs

2016. május 27.

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott *Pituk Dávid*, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, 2016. május 27.

Pituk Dávid
hallgató

Kivonat

Napjainkban az okostelefonok teljesen életünk részévé váltak. Ezen eszközök megjelenése legalább akkora változást hozott világunkba, mint az első személyi számítógépek elterjedése az 1990-es években, és a trendet figyelve megállapíthatjuk, hogy telefonjaink egyre több szerepet vesznek át számítógépeinktől és egyéb eszközeinktől (GPS, játékkonzolok, kamerák stb.). Az alapvető telefonfunkciókon kívül rengeteg más területen is használjuk őket, például böngészésre, e-mailezésre, navigációra vagy multimédiás célokra. A technológia fejlődésével ezen funkciók is egyre használhatóbbak lettek, így ma már egyes készülékek profi fényképészek, videószerkesztők vagy akár zenészek eszközeivé váltak.

Dolgozatomban a zenei jelfeldolgozás, azon belül is a zenei hangolás területével foglalkoztam. Célom az volt, hogy kifejlesszek egy hangoló szoftvert az egyik legnépszerűbb okostelefonos platformra, Androidra. Az alkalmazás két fő hangoló funkcióval rendelkezik. Az egyik egy kromatikus hangmagasság-felismerő megoldás, mellyel a készülék viszonylag nagy pontossággal tudja meghatározni a mikrofonba játszott hang frekvenciáját. A másik funkció egy polifonikus detektáló, mely kifejezetten a gitár hangolását segíti.

Az alkalmazás megvalósításához először ismereteket gyűjtöttem a zenei jelek elméletével kapcsolatban. MATLAB felhasználásával megvizsgáltam napjaink legnépszerűbb idő- és frekvenciatartománybeli hangoló algoritmusait a pontosság és gyorsaság tekintetében, majd az eredményeknek megfelelően kiválasztottam a két legalkalmasabbnak talált módszert a kromatikus, illetve a polifonikus hangoló megvalósításához.

Emellett utánajártam, hogy mennyire alkalmas az Android platform alapvető zenei jelfeldolgozási feladatok ellátására.

Az alkalmazás fejlesztése után annak funkcióit különböző hangszerek által keltett hangmintákkal teszteltem, továbbá összehasonlítást végeztem a jelenleg piacon lévő (Google Play-ben fellelhető) megoldásokkal is.

Végül megállapítottam, hogy a kromatikus modul kissé érzékeny a zajra, míg a polifonikus hangoló esetében a legmélyebb húr hangjának detektálása még nem elég pontos. Ezek kijavítása után, illetve plusz funkciók (pl.: referencia A hang frekvenciájának szabad megválasztása, polifonikus esetben capo beállítás stb.) hozzáadása után akár a Google alkalmazásboltba is fel lehetne tölteni a szoftvert.

Abstract

Smartphones has become an important part of our everyday life. The appearance of these devices made as huge impact in our world as the spreading of PC-s in the 1990's, and if we look at the trend, we can clearly see that our phones are taking over tasks from personal computers and other devices, such as GPS, video game consoles and cameras. We use smartphones for browsing, e-mailing, navigation or for multimedia. As technology evolves, these functions are getting better and better, and smartphones have become the tools of professional photographers, video makers or musicians.

The topic of this thesis is about instrument tuning using audio signal processing techniques. My goal was developing a tuner-software on Android, which is one of the most popular smartphone operating systems, besides IOS and Windows Phone. The application includes two main functions. One is a chromatic tuner, which detects the frequency of a sound recorded by the microphone. The other is a polyphonic tuner which helps guitar players to make a quick string check.

To create this application, I had to make a literature search about the theory of musical signals. I used MATLAB for testing the most often used timedomain and frequencydomain based algorithms regarding on accuracy and speed. Based on my findings, I have chosen the algorithms best fitted to chromatic and polyphonic tuning.

Furthermore, I gathered information about the Android system and Java programming language regarding their capability of audio signal processing.

After development, I tested the software with different instruments and frequencies, and compared it with other tuner applications available in Google Play.

After testing and comparing, I found out that the chromatic module is a bit sensitive for noise, and the polyphonic tuner is not accurate enough for detecting the pitch of the *E2* string. After fixing these issues, and adding some extra features like changeable reference *A* frequency or fret offset settings for the polyphonic tuner, I could upload the software to the Google Play Store.

Bevezető

> Sorry for the tune up between time, but what the hell, cowboys are the only ones who stay in tune, anyway...
>
> *James Marshall Hendrix*

A zene körülvesz valamennyiünket. Halljuk a rádióból, televízióból, számítógépünkből vagy az MP3 lejátszónkból. Átlagos zenehallgatóként nincs szükségünk arra, hogy komolyabb ismereteink legyenek a zene hátterével kapcsolatban. Azonban a zenészek gyakran évtizedeket töltenek zenetanulással. Komoly tudásra tesznek szert a zeneelmélet minden területéről, közben napi több órát töltenek hangszeres képességeik tökéletesítésével. Az évek során hallásuk annyira kifinomulttá válik, hogy könnyedén képesek különbséget tenni olyan hangok között is, amelyeket az átlagember egyformának hall. Napjainkra ezek az apró különbségek elektronikus eszközökkel kimutathatók, így erre építve kialakult a hangolástechnika, mely fontos szerepet tölt be a profi zenei eszközök fejlesztésében.

Már évszázadokkal ezelőtt léteztek kezdetleges hangolási módszerek (pl. hangvilla segítségével történő hallás utáni hangolás), mára azonban az elektronika fejlődésével számtalan pontosabbnál pontosabb megoldás látott napvilágot.

Természetesen manapság a legjobb megoldásokat a többnyire DSP-re (*digital signal processor*) épülő, különálló hangoló készülékek jelentik, de számos egyéb, szélesebb körben elérhető, számítógép- vagy okostelefon-alapú megoldás is megjelent.

Szakdolgozatomban az okostelefon-alapú hangolás megvalósításával foglalkozom. Megvizsgáltam, hogy az Android operációs rendszert futtató készülékek hardveresen és szoftveresen alkalmasak-e kromatikus és polifonikus hangolás megvalósítására. A kromatikus hangoló a használt hangszertől függetlenül visszajelzést ad a lejátszott hang pontos frekvenciájáról, míg a polifonikus hangoló a gitárosokat segíti oly módon, hogy az egyszerre megpengetett üres gitárhúrok behangoltságáról ad visszajelzést, ezáltal időt spórolhat a felhasználónak. Ehhez először információt kellett gyűjtenem a zenei jelfeldolgozással és az Android platformmal kapcsolatban.

A dolgozat első fejezetében a hangolással kapcsolatos alapvető ismereteket tárgyalom, majd a második fejezetben röviden bemutatom az elektromos hangolók fejlesztésének történetét.

A harmadik fejezetben bemutatom, és a MATLAB segítségével tesztelem a szakirodalomban található legelterjedtebb idő- és frekvenciatartománybeli algoritmusokat, majd a tesztek eredményei alapján kiválasztom a két hangoló eljáráshoz használható legalkalma-

sabb módszert, és a negyedik fejezetben részletesen kifejtem azok működését.

Az Android operációs rendszerről és a fejlesztői környezetről az ötödik fejezetben írok, továbbá itt ismertetem a hangolóalkalmazás fejlesztésének fontosabb részeit.

A hatodik fejezetben a megvalósított alkalmazást összehasonlítom a Google alkalmazásboltjából ingyenesen letölthető kromatikus és polifonikus hangolószoftverekkel, és ez alapján értékelem a implementált hangolómodulokat.

Az utolsó fejezetben összefoglalom a munkám, majd kitérek arra, hogy milyen továbbfejlesztési megoldásokat tartogat még a téma.

1. fejezet

Elméleti alapok

1.1. A hang

Fizikai értelemben a hang alatt egy rugalmas közeg (esetünkben levegő) mechanikus rezgését értjük, mely hullámként tovaterjed. Mivel a zenei hangok esetében a légnyomás kváziperiodikus változásáról van szó, így a zenei jel rendelkezik a szinuszos jelek két fontos paraméterével, amplitúdóval és frekvenciával. A hanghullámokat alkotó frekvenciák széles skálán változnak, azonban ebben az írásban az emberi fül által érzékelhető frekvenciatartománnyal, azon belül is a zenei hangoknál előforduló 30 Hz és 3000 Hz közötti frekvenciaértékekkel foglalkozom.

Amennyiben a hang csupán egyetlen egy szinuszosan változó jel, a hangmagasság megegyezik a jel frekvenciájával:

$$y = A\cos(2\pi ft + \varphi) \tag{1.1}$$

A valóságban azonban szimplán szinuszos hangok nem túl gyakoriak. Ennél sokkal valószínűbb, hogy a hanghullám több különböző frekvenciájú hullám összességeként áll elő. Például egy gitáron megpengetett húr is több felharmonikust tartalmaz, mely felharmonikusok együtt határozzák meg a húr hangszínét. Mindezek ellenére egyértelműen különbséget tudunk tenni az akár több száz felharmonikust tartalmazó hangok magassága közt is.

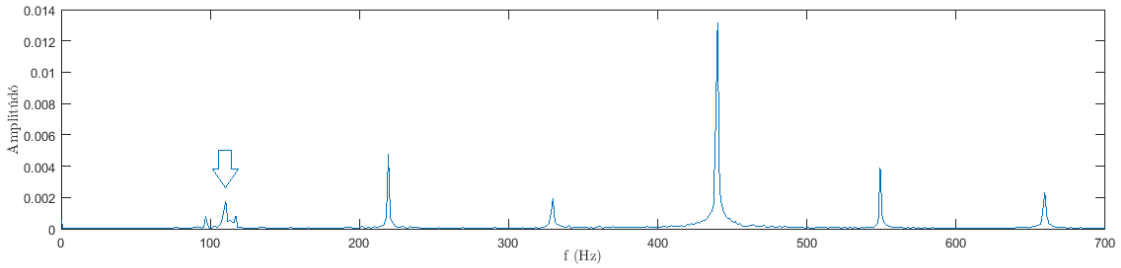
1.2. A "pitch", azaz hangmagasság

A hangmagasság (angolul „pitch”) egy adott hang szubjektív, érzékelt tulajdonsága. Azt biztosan állíthatjuk, hogy a hangmagasság és a hangot alkotó frekvenciák közt szoros kapcsolat van. Ha ezek a frekvenciák nagyobbak, a hangmagasságot is nagyobbnak érzékeljük. Jogosnak tűnik a feltételezés, hogy egyszerűen az alapharmonikust tekintjük hangmagasságnak.

Ez sok esetben valóban így van, azonban az is előfordulhat, hogy az alapharmonikus nincs jelen[1], vagy amplitúdója nem dominál a többi frekvenciakomponensekhez képest.

[1] Ezt a jelenséget virtuális hangmagasságnak hívjuk[11].

Az utóbbira egy példa az elektromos gitár 110 Hz-es „A” húrja által keltett hang spektruma (1.1 ábra).



1.1. ábra. *A 110 Hz-es "A" gitárhang spektruma*

Az 1.1 ábrán jól látható, hogy a második és a negyedik felharmonikus is nagyobb amplitúdóval rendelkezik, mint az alapharmonikus. A legelterjedtebb értelmezés szerint a hang magassága az a frekvenciaérték, amelyen lejátszva egy szimpla szinuszos hangjel ugyanazt a hangmagasságérzetet kelti, mint az eredetileg érzékelt hang.

1.3. Zenei hangrendszer és a MIDI szabvány

Mára a zeneelméletnek úgynevezett zenei hangrendszerbe sorolja a különböző magasságú hangokat[2]. A hangrendszer a zenei hangokat hét kitüntetett hangmagasságú törzshangra osztja. Az angolszász országokban az ábécé betűivel (C, D, E, F, G, A, B) nevezik és jelölik őket. Ezek közül az „A” hangok frekvenciáját 1955-ben standardizálták (55 Hz, 110 Hz, 220 Hz, 440 Hz stb.).

A hangrendszeren belül használható hangok összességét nevezzük a hangrendszer hangkészletének. A hangrendszeren belül a törzshangokból származtatunk minden más zenei hangot, ezeket módosított hangoknak hívjuk.

A kétszeres frekvenciakülönbségű hangok neve azonos, de tényleges hangmagasságuk eltér. Ennek jelzésére a teljes hallható hangtartományt szintén kétszeres frekvenciakülönbségű kisebb tartományokra osztjuk, „C” hangtól „C” hangig, ezeket fekvéseknek (vagy oktávoknak) nevezzük [16, 14].

[2]Az első kísérletek a zenei hangok rendszerezésére még jóval a papír feltalálása előttre tehetők. A legrégebbi leletet a mai Irak területén találták meg. A kőtáblába vésett kezdetleges kotta több mint 4000 éves és egy háromszólamú dalt tartalmaz.

A módosított hangokkal együtt egy fekvésben összesen 12 hang található, melyek úgynevezett „félhang” távolságra követik egymást. Ezen hangokat és a hozzájuk tartozó frekvenciaértékeket szemlélteti az 1.2 ábra.

Hang	Kontra	Nagy	Kis	Egyvonalas	Kétvonalas	Háromvonalas
A	55,00	110,00	220,00	440,00	880,00	1760,00
A♯/B♭	58,27	116,54	233,08	466,16	932,33	1864,66
B	61,74	123,47	246,94	493,88	987,77	1975,53
C	65,41	130,81	261,63	523,25	1046,50	2093,00
C♯/D♭	69,30	138,59	277,18	554,37	1108,73	2217,46
D	73,42	146,83	293,66	587,33	1174,66	2349,32
D♯/E♭	77,78	155,56	311,13	622,25	1244,51	2151,62
E	82,41	164,81	329,63	659,26	1318,51	2692,17
F	87,31	174,61	349,23	698,46	1396,91	2793,83
F♯/G♭	92,50	185,00	369,99	739,99	1479,98	2959,96
G	98,00	196,00	392,00	783,99	1567,99	3135,96
G♯/A♭	103,83	207,65	415,30	830,61	1661,22	3322,44

1.2. ábra. *Hangok és frekvenciáik Hz-ben*

Mivel a zenei hangok frekvenciáinak kapcsolata exponenciális, ezért hamar igény született egy lineáris kapcsolatot leíró rendszer megalkotására, mely könnyen felhasználható digitális zenei jelfeldolgozás során.

Erre a célra nagyon jól használható a MIDI (Musical Instrument Digital Interface)[3] szabvány, mely minden hangkészletben szereplő hanghoz hozzárendel egy egész számot (1.3 ábra). Például a 440 Hz-es „A” hang MIDI kódja a 69, míg a két félhanggal alacsonyabban fekvő „G”-hez a 67-es kód tartozik [17].

Egy adott f frekvenciájú hang MIDI kódját a következőképpen kaphatjuk meg:

$$n = 69 + 12 \log_2 \left(\frac{f}{440} \right) \tag{1.2}$$

Két egymást követő egészszámú MIDI kód közötti tartományt századpontosságú egységekre, úgynevezett centekre osztjuk, így a hangolók pontosságát tipikusan centekben (vagy tized centekben) adják meg.

Míg átlagosan az emberek 20 cent pontossággal képesek megkülönböztetni hangmagasságokat, addig képzett zenészeknél ez a távolság 10 cent vagy kevesebb is lehet. Ennek megfelelően a digitális elven működő hangolóberendezések is általában 10 centnél nagyobb elhangolódást jeleznek hamis hangnak.

[3] A MIDI egy szintetizátorok és stúdióeszközök összekötésére alkalmas szabvány. Fizikailag egy aszinkron soros vonali kommunikáció, melynek során csak a hangmagasságot, ritmust és a lejátszandó hangszer kódja kerül tobábbításra, így a hang kiadása már a MIDI hangeszköz dolga, az MP3-mal és WAV-val ellentétben, ahol a fájl magát az egész zenét tartalmazza.

Dolgozatomban hangolással foglalkozom, így ezentúl ezt a rendszert használom a detektált hangmagasság jelzésére.

Hang	Kontra	Nagy	Kis	Egyvonalas	Kétvonalas	Háromvonalas
A	33	45	57	69	81	93
A♯/B♭	34	46	58	70	82	94
B	35	47	59	71	83	95
C	36	48	60	72	84	96
C♯/D♭	37	49	61	73	85	97
D	38	50	62	74	86	98
D♯/E♭	39	51	63	75	87	99
E	40	52	64	76	88	100
F	41	53	65	77	89	101
F♯/G♭	42	54	66	78	90	102
G	43	55	67	79	91	103
G♯/A♭	44	56	68	80	92	104

1.3. ábra. *Hangok és MIDI értékeik*

2. fejezet

Hangolás

A hangolás technikája egyidős az első hangszerek megjelenésével. Társas zenélés során vagy több hangképzővel rendelkező hangszerek (zongora, gitár, hegedű stb.) esetén elengedhetetlen fontosságú a harmónia megteremtése.

A legkézenfekvőbb megoldás a hallás utáni hangolás, melyet mai napig alkalmaznak a zenészek. Habár mindenki számára elérhető és egyszerű módszer, megbízhatósága a hangolást végző személy zenei hallásának pontosságától függ, így a profi zenészek körében hamar megszületett az igény olyan mérőberendezésekre, amelyek minden körülmények közt nagy pontossággal képesek visszajelzést adni a játszott hangmagasságokról.

2.1. Elektronikus hangolóeszközök

A történelem során több mechanikai alapú megoldás is született, azonban kellően precíz, professzionális célokra is használható készülékekre az 1900-as évek közepéig, az elektronika aranykoráig kellett várni. 1936-ban jelent meg a C.G. Conn nevű cég gondozásában az első stroboszkópos hangoló, ami a Stroboconn névre hallgatott, és közel negyven évig gyártották. Igazán népszerű modellé a Peterson Electro-Musical Products 1967-ben piacra dobott készüléke, a Model 400 vált. A 400-ast és annak változatait olyan híres előadók használták, mint a The Who, a Pink Floyd, Frank Zappa, Jimi Hendrix és Neil Young.

2.1. ábra. *Peterson Model 400*

Az elektromechanikai elven működő stroboszkópos hangoló két fő részből áll: egy fényforrásból és az előtte forgó lemezből. A fényforrás (többnyire LED) a játszott hanggal megyező frekvenciával villog. Közben a lemez – melybe különböző átlátszó mintákat vágtak – az előre beállított frekvenciával forog. Például ha a gitár 110 Hz-es *A* húrját szeretnénk behangolni, akkor beállítjuk, hogy a lemez is 110 Hz-en forogjon. Amennyiben a játszott hang (így a LED-ek villogásának) és a lemez pörgésének frekvenciája meegyezik, a lemez mintáit (az emberi látás tulajdonságai miatt) közel statikusnak érzékeljük. Azonban ha valamennyi eltérés van, akkor a minták mozogni kezdenek [13].

A stroboszkópos hangolók felbukkanása hatalmas fejlődés volt a hangolástechnikában. Habár egyszerre csak egy, előre megadott hang pontosságáról volt képes visszajelzést adni, precizitása[1] mai napig kimagaslónak számít, így egészen mostanáig jellenek meg a stroboszkópos hangolók kisebb méretű, analóg-digitális hibrid változatai [9, 10].

Mint minden más területen, a hangolóeszközök világában is nagy változásokat hozott a digitális technika és az integrált áramkörök megjelenése. Az eszközök egyre olcsóbbá váltak, és a különböző szenzoroknak köszönhetően rengeteg eltérő megoldás született detektálás módjának szempontjából.

2.2. ábra. *Az egyszerre hangolóként és capo-ként funkcionáló TAPO, közvetlenül a húrok rezgését érzékeli*

Piacra kerültek az első digitális kromatikus hangolók, amelyek előzetes beállítások nélkül képesek megállapítani a játszott hang magasságát és elhangolódásának mértékét. Ezek többnyire mikrofon mellett jack bemenettel is rendelkeznek, így zavaró hangok nélkül képesek az elektromos hangszerek hangolását segíteni.

A fejlett DSP processzorokkal ellátott eszközök gyorsan és könnyedén megbirkóztak a frekvenciatartománybeli számításokkal, így a nagyobb cégek piacra dobták saját polifonikus gitárhangolójukat. Ezek az eszközök az egyszerre lepengetett húrok közül egyből képesek kimutatni a hamisakat, ezáltal gyorsítva a hangolás menetét.

[1] Például a Sonic Research napjainkban is gyártott stroboszkóp alapon működő ST-300-as modelljének hirdetéseiben 0.02 centes pontosságot garantál.

2.3. ábra. *A TC Electronic polifonikus és a KORG népszerű kromatikus hangolója*

A 2000-es évek második felében megjelentek az első érintőkijelzős okostelefonok, és mára a világ lakosságának nagy része rendszeresen használ valamilyen IOS-es vagy Androidos készüléket. Ezen telefonok lelkét adó ARM chipek az évek során jelentős fejlődésen mentek keresztül, így számukra ma már nem jelent akadályt összetettebb zenei szoftverek futtatása sem. Ennek köszönhetően számtalan kromatikus hangolóprogram lelhető fel az online alkalmazásboltokban, amelyek olcsóbb digitális társaikhoz hasonlóan 1-2 centes pontossággal képesek működni.

2.4. ábra. *Az Androidra és IOS-re is elérhető PitchLab hangoló alkalmazás*

3. fejezet

Hangmagasság-detektálási módszerek

A hangok szenzoros érzékelése és mintavételezése után periodikus adatfolyamokként jelennek meg a különböző rendszerekben. Mint minden jel, úgy a hangokat reprezentáló adatsorozatok vizsgálata is alapvetően idő- és frekvenciatartományban történhet.

A fejezet első felében a szakirodalomban található fontosabb időalapú módszereket járom körbe, míg utánuk a frekvenciatartománybeli eljárásokat vizsgálom meg. A vizsgálataim fő célja, hogy pontos és gyors megoldást találjak az elkészítendő androidos alkalmazás kromatikus és polifonikus funkcióihoz. A különböző algoritmusok implementálásához és elemzéséhez a MATLAB programrendszer R2015a verzióját használtam.

A vizsgálandó hanganyagokat telefonom mikrofonjával rögzítettem, és .WAV formátumba mentettem. Ezután a MATLAB beépített *audioread* függvényével ezeket a hangfájlokat tömbökbe olvastam[1]. A hanganyagok rögzítésekor több különböző hangszert is felhasználtam, mint fuvola, klasszikus és elektromos gitár, szaxofon, trombita vagy zongora.

3.1. Időtartománybeli eljárások

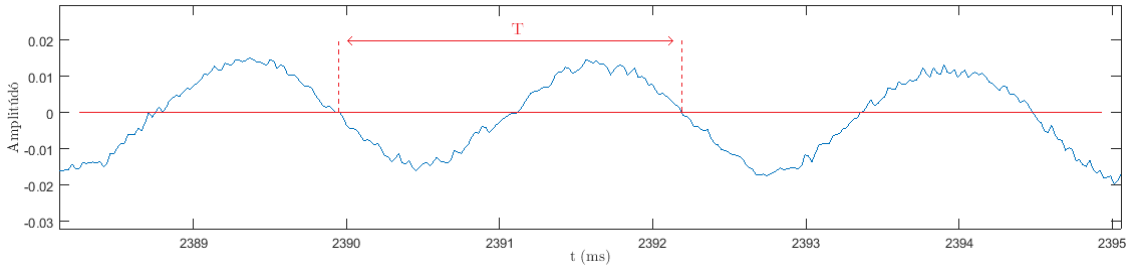
Az időtartománybeli analízis egyik legnagyobb előnye a gyorsaság, mivel a hang 1-2 periódusa már elegendő a pontos frekvenciameghatározáshoz. Ebből adódóan a ma kapható kromatikus hangolók nagy része is ezt részesíti előnyben a frekvenciatartománybeli vizsgálattal szemben.

Időtartományban a pontosság a vizsgálandó hangmagasság növekedésével romlik. Az androidos készülékek által használt 44,1 kHz-es mintavételi frekvencia 22,7 mikroszekundumos felbontást jelent. Ez egy 55 Hz-es kontra A hangnál *worst case* esetben $\pm 2,1$ centes, míg a 880 Hz-es kétvonalas A esetében már $\pm 34,2$ centes eltérést okoz. Természetesen a pontosság interpolációval, átlagolással és egyéb eljárásokkal nagymértékben javítható.

[1] Az *audioread* függvény a hangfájl bájtjait -1 és 1 közé normált *double* értékekké alakítja

3.1.1. Zero-Crossing Rate (ZCR)

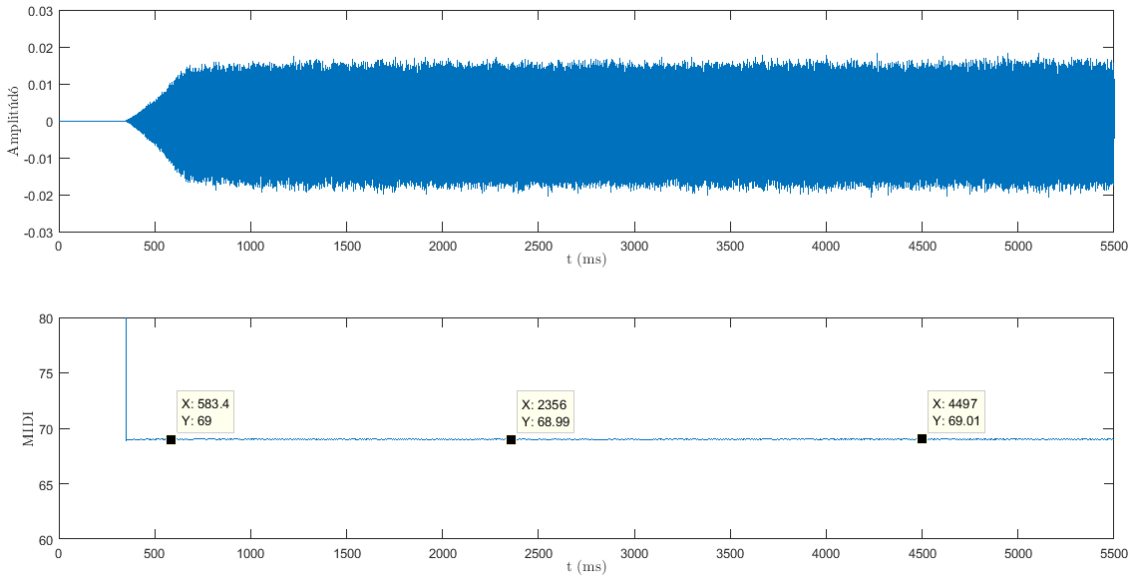
Periodikus jelek frekvenciájának meghatározására a legkézenfekvőbb megoldás a nullátmenetek keresése (angolul Zero-Crossing Rate [ZCR]). Lényege, hogy megkeressük azokat a pontokat, ahol a jel metszi az y = 0 egyenest. Minden második ilyen metszéspont közt eltelt idő megegyezik a jel periódusidejével, amiből egyértelműen számolható a frekvencia [11].



3.1. ábra. *ZCR szemléltetése egy függvénygenerátorral keltett zenei hangon*

A hangfájlt beolvassuk, utána aluláteresztő szűrővel szűrjük, majd a minden második zérusátmenetből számolt frekvenciát tömbbe mentjük. Ezeket a frekvenciaértékeket MIDI kóddá alakítva az idő függvényében ábrázoljuk.

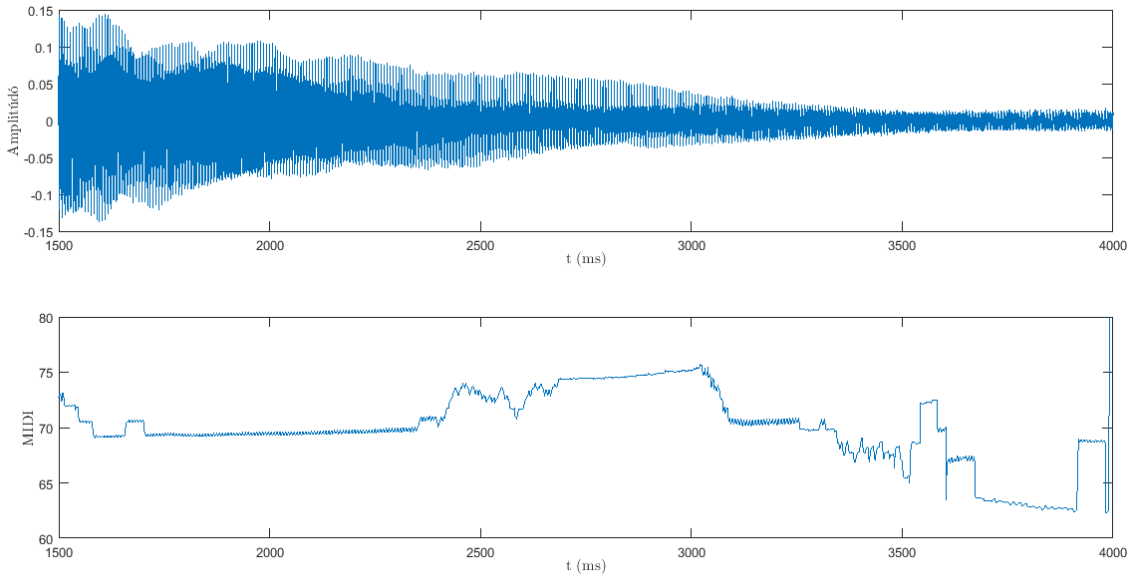
Egy egyszerű, felharmonikusoktól mentes 440 Hz-es (69-es MIDI kódú egyvonalas A) szinuszjelre számolt értékeket a 3.2 ábra mutatja:



3.2. ábra. *A ZCR által detektált hangmagasság*

Mérés során azt tapasztaltam, hogy egyszerű jeleknél problémamentesen tudja az algoritmus meghatározni az értékeket, azonban ahogy arra számítani lehet, egy konkrét hangszer hangjának magasságát már nem képes ilyen pontossággal megtalálni.

A 3.3 ábrán egy elektromos gitár D húrjának (146,83 Hz, MIDI: 50) ZCR által számolt hangmagasságai találhatóak.



3.3. ábra. *A ZCR által detektált hangmagasság*

A 3.3 ábra jól mutatja, hogy a kimeneti értékek még csak meg sem közelítik az elvárt 50-es MIDI szintet. Ez annak köszönhető, hogy a húros hangszerek hangja jellemzően rengeteg felharmonikust tartalmaz, így ez a primitív eljárás nem alkalmas a hangmagasság pontos meghatározására.

Persze lehet még az algoritmuson különböző szűrési eljárásokkal javítani, azonban ez már jelentősen növelné a számítási teljesítményt, tehát elmondhatjuk, hogy bár a Zero-Crossing eljárás elképesztően gyors, összetett audiojelek vizsgálatára alkalmatlan.

3.1.2. Autocorrelation Function (ACF)

A hangmagasság megállapításának egyik gyakran használt módszere a különböző korrelációalapú algoritmusok alkalmazása. Ennek a legáltalánosabb típusa az autokorreláció (angolul Autocorrelation Function [ACF]), amelynek lényege, hogy a jelet eltolja és összeszorozza eredeti önmagával, így a periodicitás és ismétlődő minták megtalálására nagyon jól használható.

Az általános diszkrétidejű autokorrelációs függvény a következőként írható le:

$$y(\tau) = \sum_{i=-\infty}^{\infty} x_i x_{i+\tau} \tag{3.1}$$

Ha a jel T szerint periodikus, akkor x_i megegyezik x_{i+T}-vel, így az autokorrelációnak maximuma lesz T egész számú többszöröseinél.

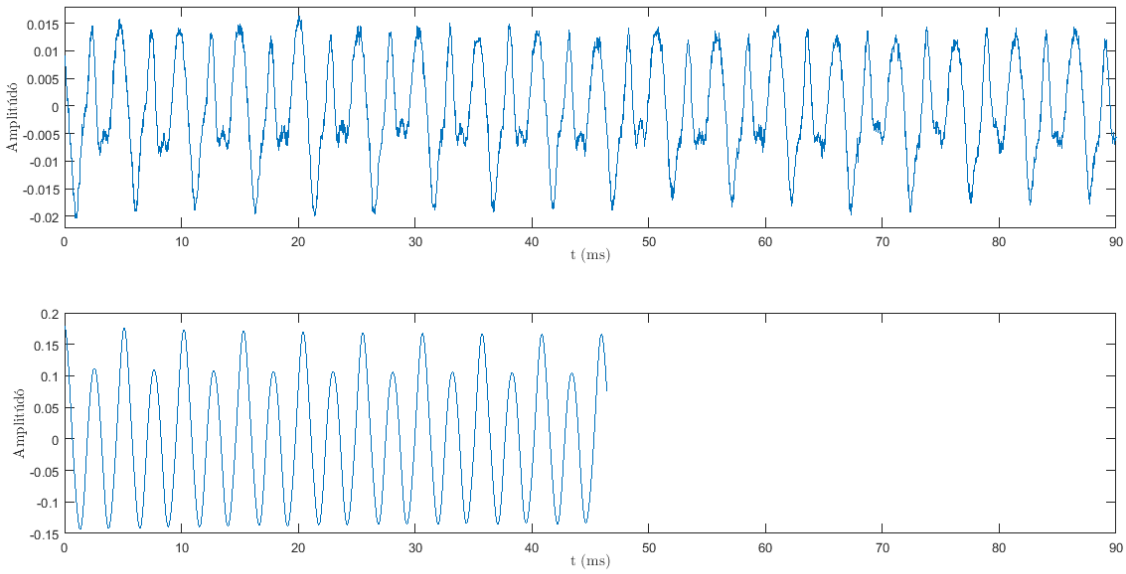
A gyakorlatban, valósidejű jelfeldolgozáskor nem az egész adathalmazzal dolgozunk, hanem annak egy részletével, úgynevezett ablakkal. Az ablakméret általában kicsi, tehát a

frekvencia az ablakon belül közelítőleg állandó. Azonban nagyon gyors frekvenciaváltozások esetén a pontosság csökken[2]. Mindenesetre a korreláció helyes működéséhez célszerű olyan ablakméretet választani, ami magába foglal legalább két periódusnyi jelet. Mivel ez megkövetelné a detektálandó frekvencia ismeretét, ezért az ablakhosszt az előforduló legkisebb frekvenciához[3] kell méretezni.

Az L hosszú ablak autokorrelációs függvénye a következő:

$$y(\tau) = \sum_{i=0}^{L/2-1} x_i x_{i+\tau}, \qquad 0 \leq \tau \leq \mathrm{L}/2 \tag{3.2}$$

Ennek a hátránya, hogy az ablak fele úgymond kárba vész, azonban a maximumok közel állandó értéket vesznek fel, így a számunkra hasznos csúcsok könnyebben felismerhetők. Ezt szemlélteti a 3.4 ábra, amelyen a klasszikus gitár G húrjának időfüggvénye, és annak autokorrelációs időfüggvénye látható.



3.4. ábra. *ACF előtt és után*

Jól látható, hogy a korreláció elvégzése után a felharmonikusok kevésbé dominálnak az eredeti jelhez képest.

A detektálás során tehát meg kell találni az alapharmonikusok (maximális csúcsok) helyét, majd azok különbségéből meghatározni a periódusidőt. Mivel a továbbiakban bemutatásra kerülő módszerek is mind a korrelációra alapulnak, ezért azok esetében is hasonlóan történik a periódusidő kiszámítása.

A feladat legnagyobb nehézsége az alapharmonikusok egzakt szétválasztása a felharmo-

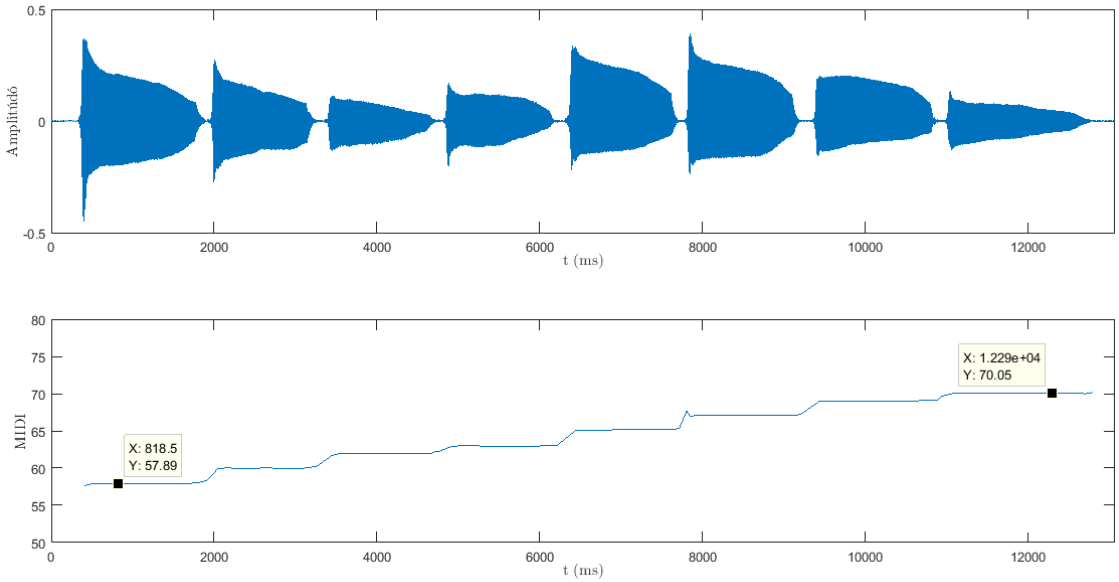
[2]Szerencsére hangolás során nem kell számítani arra, hogy a játszott hang frekvenciája rövid idő alatt sokat változna, így a korrealáción alapuló módszerek ilyen szempontból megfelelőek.

[3]Ezt 41,2 Hz-nek választottam, ami a szubkontra E hang frekvenciája és a négyhúros basszusgitár legmélyebb húrjának hangmagassága

nikusoktól. Ez úgy történik, hogy először megkeressük az összes csúcsot, majd a legnagyobb csúcs amplitúdójának k-szorosánál (0<k<1) nagyobb amplitúdójú csúcsokat tekintünk domináns csúcsoknak. Ezen csúcsok közt eltelt időből számítjuk a frekvenciát. Persze felmerül a kérdés, hogy mi alapján választjuk meg úgy k értékét, hogy minden alapharmonikushoz tartozó csúcs figyelembe legyen véve, ugyanakkor ne kerüljenek be felharmonikusok a számításba, amelyek oktávhibát okozhatnak.

Végül tapasztalati úton ezt a konstanst 0,9-nek választottam, ami a tesztek után is jó értéknek bizonyult.

A 3.5 képen egy szaxofon által lejátszott hangsorozat (B dúr törzshangjai kis B-től egyvonalas B-ig, MIDI: 58 - 70) ACF által detektált hangmagasságai láthatóak.



3.5. ábra. *Az ACF által detektált hangmagasságok*

Bár az ábrából nem látható, de az algoritmus valóban képes meghatározni 1-2 cent pontossággal a hangmagasságokat, továbbá a számítási idő is kedvező, mivel a 4000 minta hosszú ablakokat átlagosan 8 század másodperc alatt dolgozza fel a MATLAB (Intel i5-2520m, 8 GB DDR3)[4].

3.1.3. Square Difference Function (SDF)

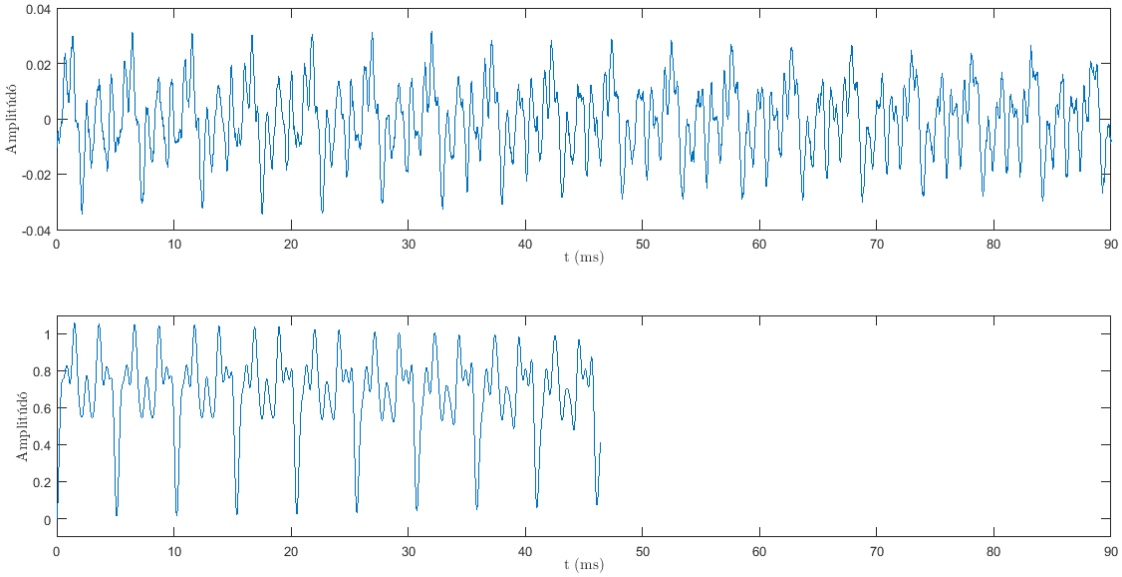
A négyzetesküllönbség-függvény (angolul Square Difference Function [SDF]) is a korrelációs eljárások családjába tartozik.

Egy L hosszú ablak esetén a következőként számítható:

$$y(\tau) = \sum_{i=0}^{L/2-1} (x_i - x_{i+\tau})^2, \qquad 0 \leq \tau \leq L/2 \tag{3.3}$$

[4]A MATLAB által meghatározott számítási idők az Androidra való implementáció után csökkennek, köszönhetően az ARM processzorok erőteljes adatcsatornás (pipelining) feldolgozásának[7].

A képletből jól látszik, hogy az előzőleg bemutatott autokorrelációs függvényhez képest itt a periodikusan ismétlődő csúcsoknál lokális minimumok lesznek. Habár a beiktatott elemenkénti új művelet (kivonás) miatt a számítási idő kicsivel nagyobb lesz, a különbség négyzetre emelésével az alapharmonikusok jobban el fognak különülni a felharmonikusoktól. Ezt a 3.4 ábra mutatja be, amelyen az elektromos gitár G húrjának időfüggvénye, és az azon végrehajtott SDF utáni időfüggvény látható.

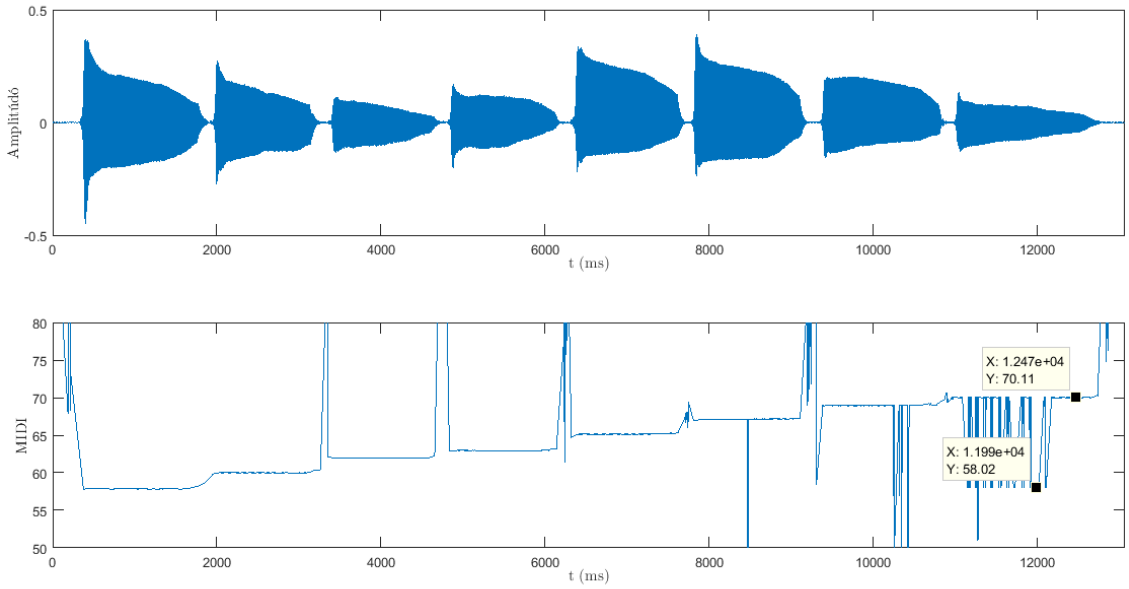


3.6. ábra. *SDF előtt és után*

A csúcsok detektálásához minimumkeresést kell alkalmazni. Mivel az SDF értékkészletének alsó korlátja 0, így a korábban bemutatott ACF-fel ellentétben itt nem lehet zérusátmeneten alapuló eljárással egyszerűsíteni a csúcsdetektálást. Továbbá, a minimumhelyek értékei folyamatosan változnak[5], így itt is szükséges egy k konstans alkalmazása (k>1), ami megszabja, mely minimumokat vegyünk be a számításba. Ezt az értéket tapasztalati úton 9-nek választottam. Tehát ha egy alsó csúcs értéke alacsonyabb az SDF ablak alsó csúcsértékének kilecszeresénél, akkor az domináns csúcsnak vehető.

A 3.7 ábra a korábban használt szakszofonos hangminta négyzeteskülönbség-függvény által számolt hangmagasságait szemlélteti.

[5]Ez a hang amplitúdójának változásából következik, ami hangszerenként eltérő. Például egy húros hangszer esetében folyamatosan csökken, ahogy a húr veszít az energiájából, míg egy fúvós hangszer esetében növekedhet és csökkenhet is.



3.7. ábra. *Az SDF által detektált hangmagasságok*

Látható, hogy a magasabb hangoknál itt már komolyabb alsó oktávhibák is előfordulnak, ami a k konstans helytelen megválasztásának köszönhető. A jó érték megválasztása az SDF esetén különösen nehéz feladat, mert a csúcsok ingadozásának mértéke nem csupán hangszer- hanem erősen frekvenciafüggő is. Ez azt jelenti, hogy választható egy nagyobb konstans, ami a magasabb frekvenciáknál jó működést biztosítana, azonban az alacsonyabb frekvenciáknál felső oktávhibát okozna.

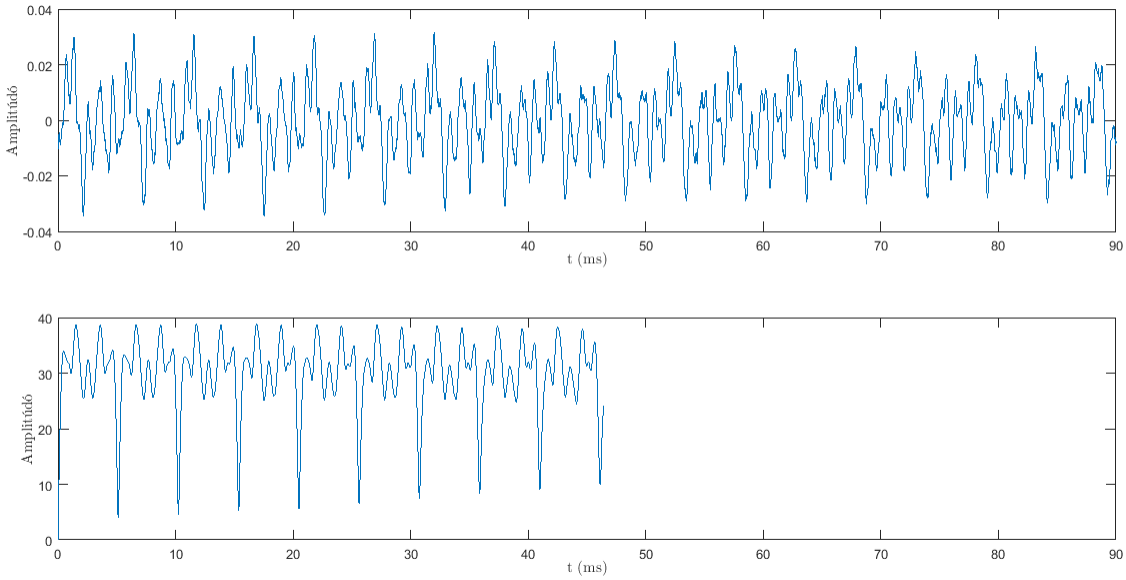
3.1.4. Average Magnitude Difference Function (AMDF)

Az átlagos különbségfüggvény (angolul Average Magnitude Difference Function [AMDF]) nagyon hasonlít az előzőekben bemutatott SDF-re.

Az AMDF-et a következőképpen definiáljuk:

$$y(\tau) = \sum_{i=0}^{L/2-1} |x_i - x_{i+\tau}|, \qquad 0 \leq \tau \leq \mathrm{L}/2 \tag{3.4}$$

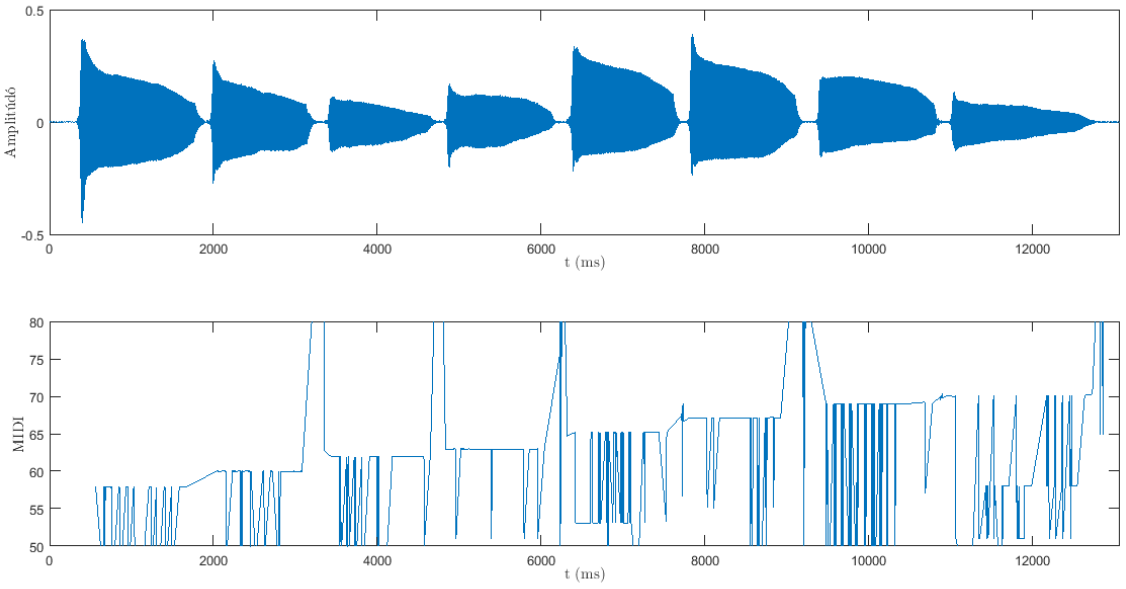
Az AMDF annyiban különbözik az SDF-től, hogy itt négyzetreemelés helyett abszolútérték-képzés szerepel, ezáltal a számítási idő rövidül. Ám ez egyben azt is jelenti, hogy az alapharmonikus kevésbé fog elkülönülni a felharmonikusoktól.



3.8. ábra. *A G húr hangjának időfüggvénye AMDF előtt és után*

A konstansválasztás itt is nagy problémát jelent. A húros hangszerek esetén célszerű kisebb szorzó (pl.: $k = 2$) használata a felharmonikus csúcsok közelsége miatt, míg a fúvós hangszerek esetén ezt az állandót ajánlott nagyobbnak választani (pl.: $k = 5$), mivel az alapharmonikus csúcsai jelentősen csillapodnak az idő múlásával.

Különböző gitárhangok vizsgálata során arra jutottam, hogy oktávhibák elkerülése érdekében a k-t 1,5 és 2,5 közé érdemes választani. A 3.11 képen látható, hogy a gitárok esetében tökéletesen működő $k = 1.7$ választás komoly oktávtévesztéseket okoz a szaxofon hangjának felismerésénél.



3.9. ábra. *Az AMDF által detektált hangmagasságok*

3.1.5. Special Normalisation of the Autocorrelation (SNAC) Function

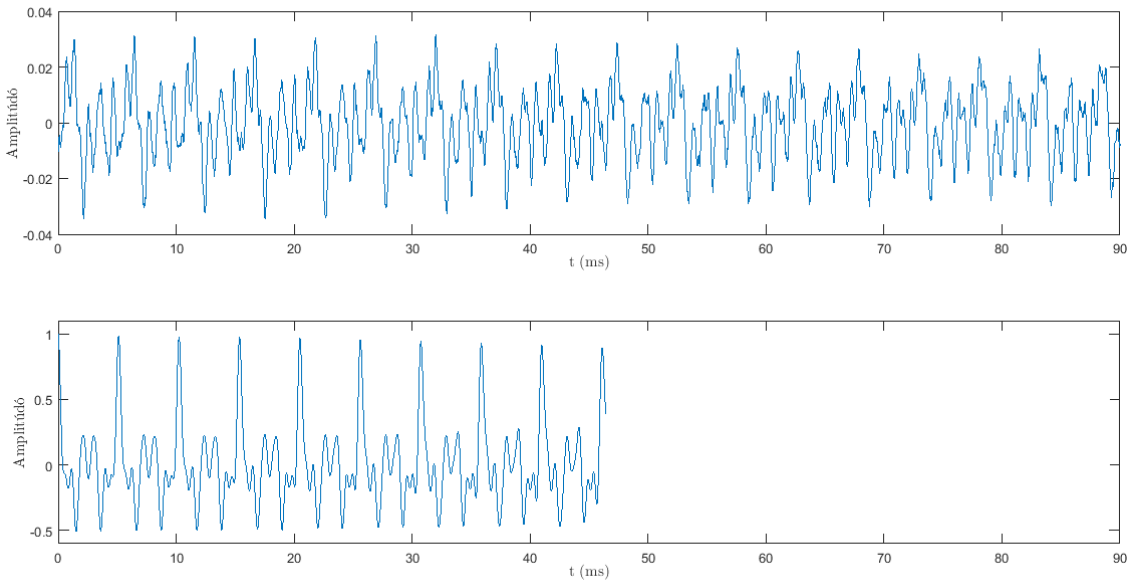
A „speciális módon” normált autokorrelációs függvény (angolul Special Normalisation of the Autocorrelation Function [SNAC]) az autokorrelációs függvény és az SDF előnyeit ötvözi.

A SNAC függvény definíciója a következő:

$$y(\tau) = 2\frac{\sum_{i=0}^{L/2-1} x_i x_{i+\tau}}{\sum_{i=0}^{L/2-1}(x_i^2 + x_{i+\tau}^2)}, \qquad 0 \leq \tau \leq L/2 \tag{3.5}$$

Habár a képlet alapján látható, hogy a SNAC függvénynek korábbi eljárásokhoz képest nagyobb számításikapacitásra van szüksége, fontos előnyökkel rendelkezik:

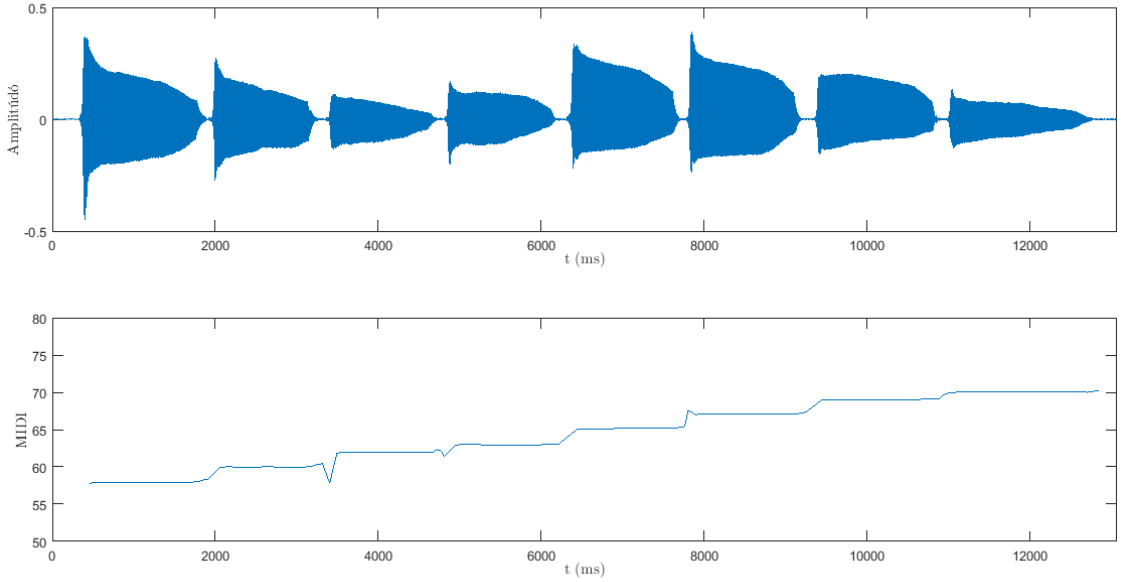
- Az ACF-hez hasonlóan nullátmenetek lesznek az időfüggvényben, így a csúcskeresés egyszerűsödik.
- Az alapharmonikus hasonló mértékben különül el a felharmonikusoktól, mint az SDF esetén, azonban ahol az SDF-nek minimuma van, ott a SNAC függvénynek maximuma.
- Az SNAC értékei 1 és -1 közé esnek, továbbá az alapharmonikusoknak megfelelő csúcsok stabilan 1 közelében maradnak.



3.10. ábra. *A G húr hangjának időfüggvénye a SNAC függvény alkalmazása előtt és után*

Az alapharmonikusokhoz tartozó csúcsértékek az SDF-hez vagy AMDF-hez képest kisebb mértékben csillapodnak, így a küszöbérték konstansa is pontosabban megválasztható. Ezt az értéket tapasztalati úton 0,9-nek vettem. Emellett a konstans mellett az algoritmus több különböző hangszer esetén is 1-2 centes pontossággal működött.

A korábban is alkalmazott szaxofonos hangminta SNAC függvény által számolt hangmagasságai a következőképpen alakultak:



3.11. ábra. *A SNAC függvény által detektált hangmagasságok*

3.2. Frekvenciatartománybeli eljárások

A jelfeldolgozás területén sokszor van szükségünk valamilyen frekvenciatartománybeli módszerre, mivel az az időtartománybeli analízishez képest sokkal alkalmasabb összetett jelek vizsgálatára. Digitális eszközök esetében a jelek frekvencitartományba való átalakítására leggyakrabban az úgynevezett gyors Fourier-transzformációt (angol Fast Fourier Transform [FFT]) használjuk.

Az FFT-nek számunkra van két fontos tulajdonsága. Az egyik, hogy számításigénye viszonylag nagy, így valósidejű megoldásokhoz erős hardver szükséges. A másik, hogy hangoláshoz megkövetelt pontosság elérésére nagy ablakméret szükséges. Amennyiben a mintavételi frekvencia F_s és az ablakméret L, akkor az FFT felbontása a következő:

$$f = \frac{F_s}{L} \tag{3.6}$$

Ez alapján ha 4096-os ablakméretet[6] választunk (ami még nem okoz jelentős késleltetést), a transzformáció felbontása 10.76 Hz lesz. Ez interpolációval tovább javítható, de még így is túl nagy ahhoz, hogy hangoláshoz használható legyen.

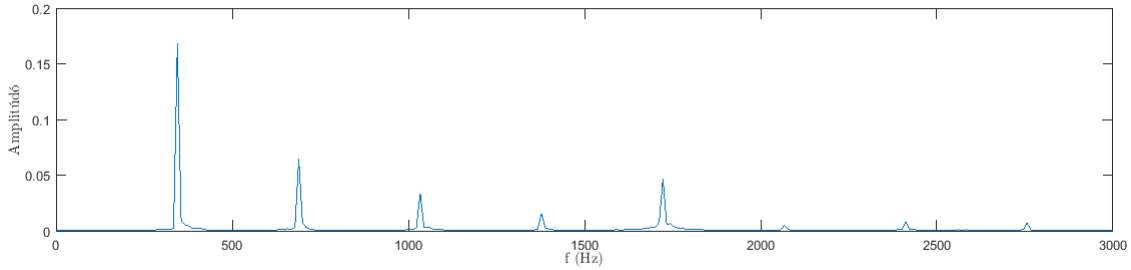
[6] Az FFT algoritmus 2^n mintából álló bemenő adatsorokkal működik.

Ezért mindenképp jelentős (akár egy másodperc közeli) késleltetéssel kell számolnunk.

3.2.1. Spektrális csúcsok módszere

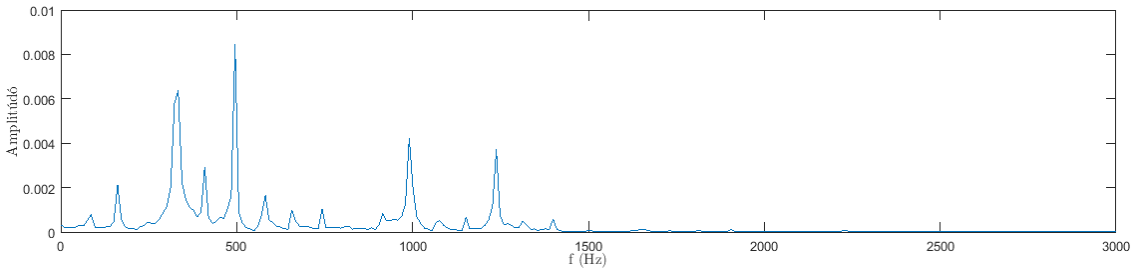
A frekvenciatartománybeli módszerek közül a spektrális csúcsok módszere tűnik a legkézenfekvőbb megoldásnak. Lényege, hogy miután transzformáltuk a bemenőjelet, a spektrumból megpróbáljuk megtalálni az alapharmonikushoz tartozó csúcsot.

Maximumkereséssel sokszor helyes eredményre juthatunk. Tipikusan fúvós hangszereknél jellemző, hogy az alapharmonikus rendelkezik a legnagyobb amplitúdóval. Ezt mutatja a 3.12 ábra is, amelyen egy fuvola kis F hangjának (349 Hz) spektruma látható (szándékosan nem dB-ben, hogy a csúcsok jobban elkülönüljenek).



3.12. ábra. *Fuvolahang spektruma*

Itt a maximumkeresés jó eredményre vezetne, azonban más hangszereknél (pl.: húros hangszereknél) a felharmonikusok könnyen dominálhatnak, így ott egy összetettebb algoritmusra van szükség. Erre egy példa a gitár alsó E húrja által keltett hang (kontra E, 82,4 Hz) spektruma (3.13 ábra).

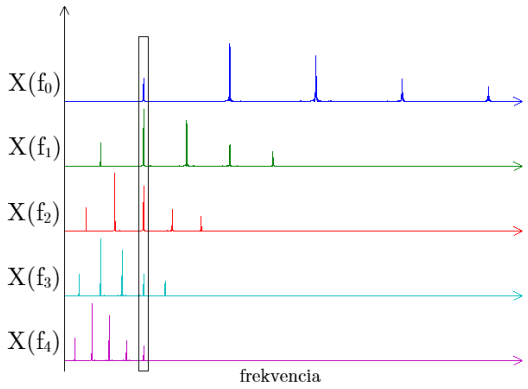


3.13. ábra. *Gitárhang spektruma*

Még abban az esetben is, ha találnánk egy jó algoritmust, amely még a virtuális hangmagasságokkal is megbírkózik, szembe kell néznünk a nagy késleltetéssel, ami kromatikus hangolóknál – ahol elvárt a tizedmásodperc környéki válaszidő – nagyon zavaró lehet.

3.2.2. Harmonic Product Spectrum (HPS)

A HPS módszer lényege, hogy a spektrum minden f_i helyen lévő értékét összeszorozzuk f_i egészszámú többszöröseinél lévő értékeivel. Így optimális esetben az alapharmonikus helyén lesz a legnagyobb érték, mivel a csúcsértékek az alapharmonikus frekvenciájának egészszámú többszöröseinél találhatóak[7].



3.14. ábra. *A HPS működése*

A HPS képlete a következő:

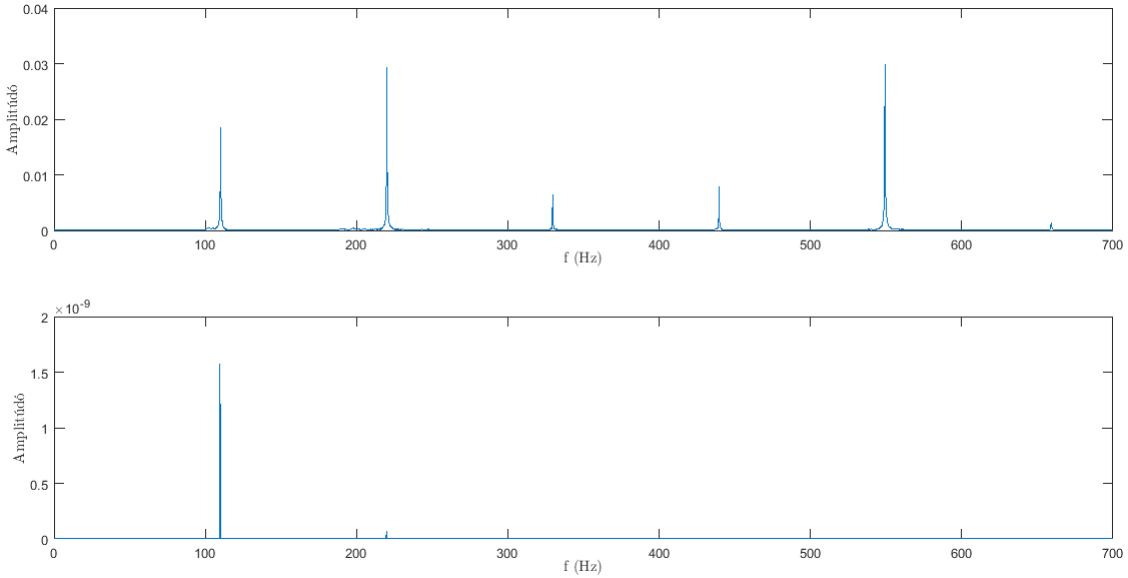
$$Y(f) = \prod_{r=1}^{R} X(fr) \tag{3.7}$$

ahol $X(f)$ a bemenő jel amplitúdóspektruma [2].

Az R megválasztása nem triviális feladat. Míg a felharmonikusokban gazdag húros hangszereknél ezt az értéket érdemes nagyobbnak (5-6) venni, addig a fúvós hangszerek esetén nem tanácsos nagy szám választása, mert akkor zaj is könnyedén kerülhet a szorzatba.

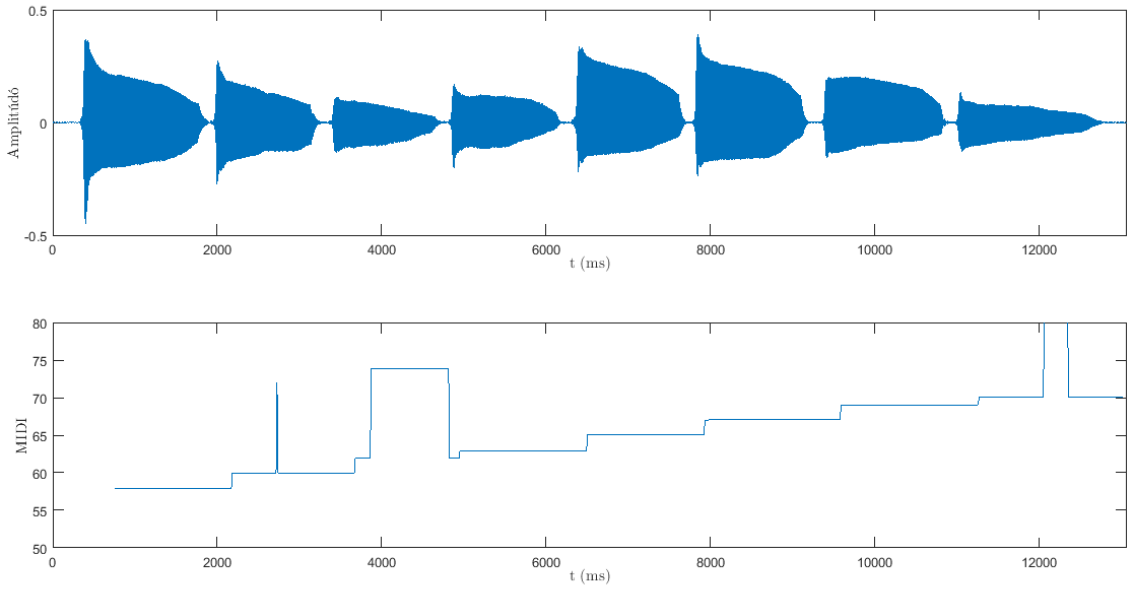
A 3.15 ábrán látható, hogy az A gitárhang (110 Hz) spektruma hogyan változik a HPS hatására. A felharmonikusok szinte teljesen eltűntek, így az alapharmonikus maximumkereséssel könnyen megtalálható.

[7]Ez tényleg csak ideális viszonyok közt igaz, mivel egyes hangszereknél (pl.: zongora vagy gitár) a felharmonikusok folyamatosan eltolódnak pozitív irányba.



3.15. ábra. *Az A húr hangjának spektruma HPS előtt és után*

A 3.16 kép a korábban használt szakszofonos hangminta HPS által számolt hangmagasságait mutatja.



3.16. ábra. *A HPS algoritmus által detektált hangmagasságok*

Ha nem vesszük figyelembe a harmadik és az utolsó hangnál fellépő oktávhibát, a HPS közel azonos pontossággal működik mint a SNAC függvény, azonban ehhez egy viszonylag nagy, 32768 mintából álló ablakkal kellett végigfutni a hangfájlon, ami jelentős késleltetést hoz a rendszerbe. Továbbá az R=3 választás más hangszereknél kevésnek bizonyulhat, így ott szép számmal jelenhetnek meg oktávhibák.

3.3. Tanulságok

Ebben a fejezetben több hangmagasság-felismerő módszer is bemutatásra került. Összességében elmondható, hogy amikor rövid válaszidőre van szükség, akkor az időtartománybeli eljárások sokkal pontosabbnak bizonyulnak a frekvenciatartománybeli megoldásokhoz képest, így kromatikus hangoláshoz sokkal alkalmasabbak. (A fejezetben ismertetett eljárásokat több hangmintával is teszteltem. Azok eredményeit szemléltető ábrák közül három a függelékben megtalálható.)

Azonban polifonikus hangolás esetében csak a frekvenciaspektrumból nyerhetünk használható információt. Mivel pontos visszajelzést várunk az alkalmazástól, ezért nagy ablakméretet kell alkalmaznunk. Ez jelentős késleltetést okoz, de polifonikus hangolóknál ez nem annyira nagy probléma[8].

[8] A polifonikus hangoló fő célja, hogy megmutassa, melyik húr hamis a lepengetett hat közül. Ha valamelyik hangolásra szorul, az már a kromatikus funkcióval pontosabban kivitelezhető.

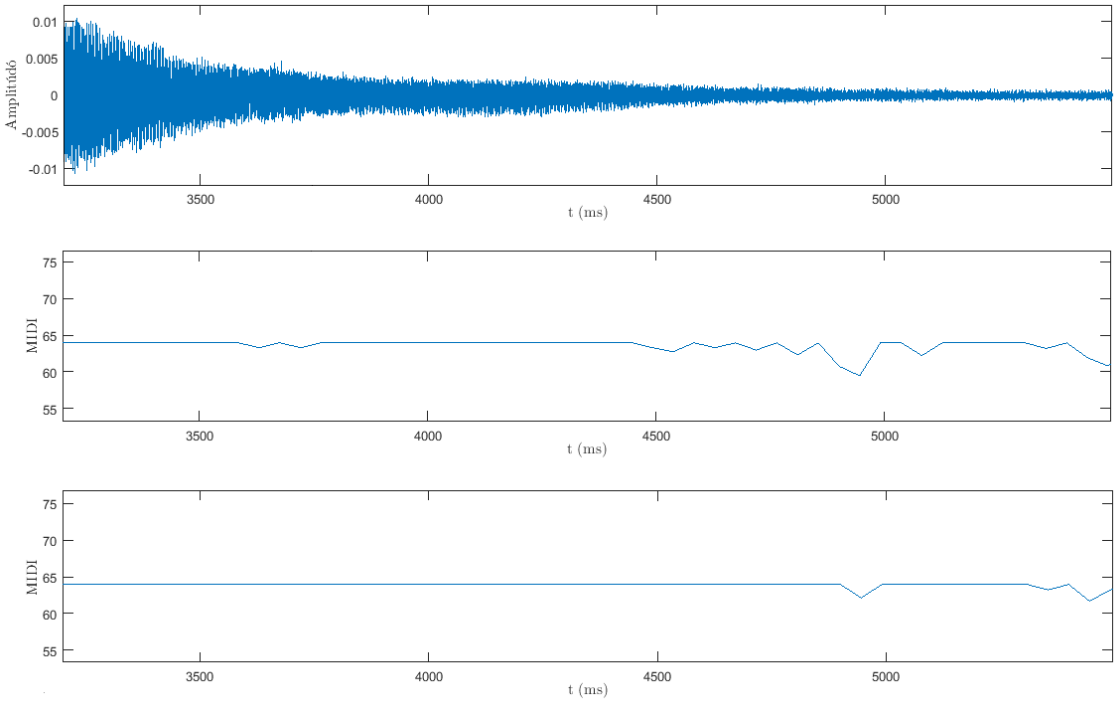
4. fejezet

A megvalósított algoritmusok

Ebben a részben a két hangolómodulhoz kiválasztott eljárást mutatom be. Mivel mindkét esetben fontos a csúcsok helyének minél pontosabb detektálása, ezért mindkét esetben parabolikus interpolációt alkalmaztam [5].

4.1. Kromatikus hangoló

Kromatikus hangoláshoz végül a SNAC függvényt választottam. A többi módszerhez képest pontosabb és robosztusabb. Egyetlen hátránya, hogy nagy számításigénye van, így sokáig az autokorrelációs függvény tűnt a legjobb megoldásnak, amely hasonlóan pontos, azonban kisebb energiájú jeleknél kevésbé precíz.



4.1. ábra. *Az ACF és a SNAC függvény által detektált hangmagasságok*

A 4.1 ábrán az látható, ahogy egy gitár felső E (MIDI: 64) húrja által keltett hang nagyon halk lecsengő szakaszának a végén az autokorreláció megadja magát, azonban a SNAC függvény még mindig képes felismerni a hangmagasságot.

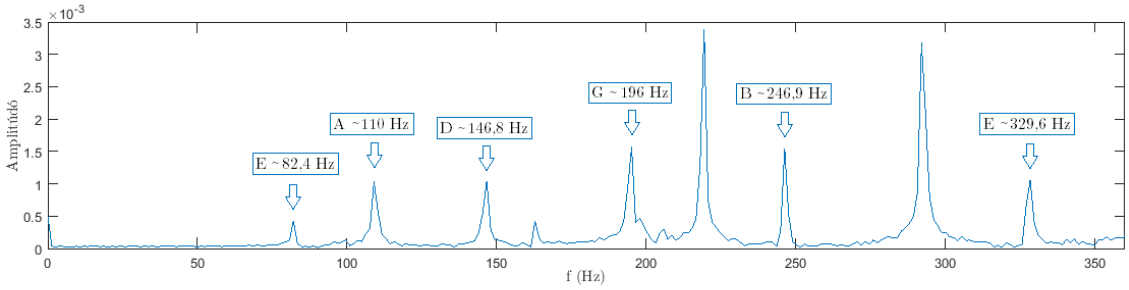
Mivel az a célom, hogy az alkalmazás képes legyen a négyhúros basszusgitár alsó 41.2 Hz-es E húrjának hangolására is, az ablakméretet ehhez kell igazítani. A SNAC függvény is az ablak feléből nyeri a hasznos információt, így legalább négy periódust kell lefedni, ami 0,097 másodperc. Ez 44100 Hz-es mintavételezés mellet 4282 mintát jelent, tehát egy 4500-as méretű ablak már elegendő a frekvencia megtalálására.

Ez az ablakméret durván egy tized másodperces késleltetést jelent, ami a jelenleg elérhető androidos hangolók közt igen jónak számít.

4.2. Polifonikus hangoló

Mint már említettem, polifonikus hangoláshoz mindenképp valamilyen frekvenciatartománybeli eljárásra van szükség. A legkézenfekvőbb megoldásnak a spektrális csúcsok módszere tűnik, hisz ha tudjuk, hol kell keresni a csúcsainkat – amit polifonikus hangolás során előre beállítunk – akkor könnyedén megtalálhatjuk a számunkra fontos értékeket.

A 4.2 ábra a klasszikus gitár összes (behangolt) húrjának lepengetéséből származó hang spektrumát mutatja, a keresendő frekvenciák bejelölésével.



4.2. ábra. *Gitárhang spektruma*

Látható, hogy a húrok alapharmonikusai egyszerűen detektálhatóak a spektrumból, azonban ehhez a felbontáshoz $2^{15} = 32768$ mintából álló ablakot kellett használni, és a pontossága még így is hagy kívánnivalót maga után.

Természetesen jogos a felvetés, miszerint az alacsonyabb húrok felharmonikusai beleszólhatnak a magasabb húrok alapharmonikusaiba. Ehhez mindenképp speciális csúcskiválasztó algoritmusra van szükség.

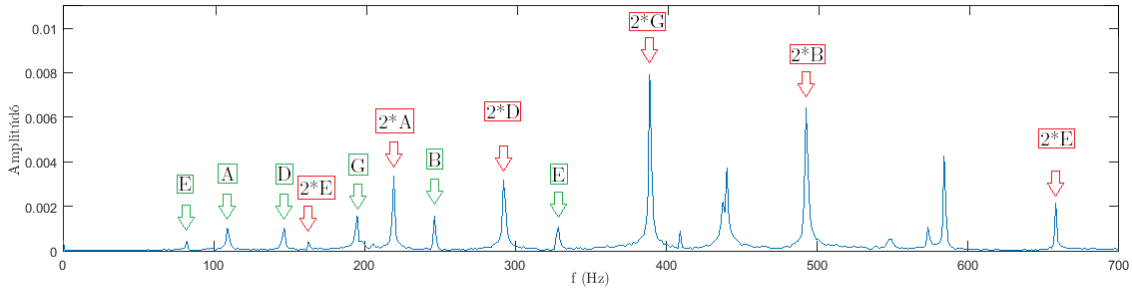
Az algoritmusról és arról, hogy miként javítható a pontosság, a következő részben teszek említést.

Gitárhúrok frekvenciájának meghatározása

Az előző részben arra a következtetésre jutottam, hogy a jel spektruma elegendő ahhoz, hogy meghatározzuk a gitár húrjainak pontosságát. Egy hathúros gitárt sokféle képpen lehet hangolni [1], azonban ez az alkalmazás a standard *E-A-D-G-B-E* hangolást veszi alapul.

Mivel ismerjük a keresendő frekvencia értékeket, ezért a frekvenciatartományban csak azok környezetét kell figyelembe venni. Az algoritmus a hat gitárhúr pontos hangmagasságának 50 centes sugarában vizsgálja a spektrumot. Emiatt 50 centnél nagyobb elhangolódás meghatározására jelenleg nem alkalmas a program.

Említettem, hogy még a 2^{15}-es ablakméret esetén interpoláció után sem volt megfelelő a pontosság. Egy megoldás lett volna erre az ablakméret növelése, de az *FFT* miatt a következő választható hossz 2^{16} (65536) lett volna, ami már túl nagy késleltetést jelentene. Ehelyett a gitárhúr jelentős felharmonikus tartalma miatt megtehetjük, hogy nem a húrok frekvenciájának alapharmonikusai körül keressük a csúcsokat, hanem azok kétszeresénél (első felharmonikusoknál). A 4.3 képen az összes húr megpengetéséből származó hang spektruma látható. Az alapharmonikusokat (zöld) és azok kétszeresét (piros) külön megjelöltem.



4.3. ábra. *Gitárhang spektruma*

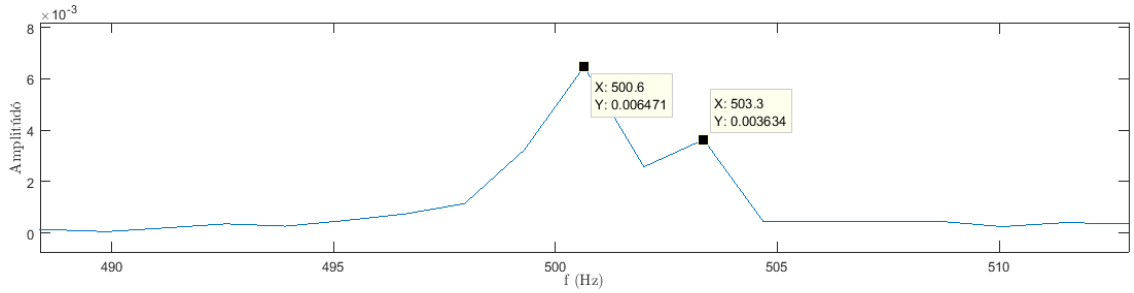
Ennek köszönhetően a detektálás pontossága a kétszeresére nőtt, így ez már alkalmasnak mondható polifonikus hangolásra.

A gitárhang spektrumának vizsgálata során fontos problémát jelent a húrok felharmonikusainak egymást zavaró hatása. Hogy ezeket a hatásokat kiszűrhessük, elsőnek ki kell számolni, pontosan mely felharmonikusok okozhatnak problémát. A húrokhoz tartozó alapharmonikusokat és azok felharmonikusait a 4.4 ábra szemlélteti.

	E	A	D	G	B	E
alapharmonikus	82,41	110	146,83	196	246,94	329,63
1. felharmonikus	164,82	220	293,66	392	493,88	659,26
2. felharmonikus	247,23	330	440,49	588	740,82	988,89
3. felharmonikus	329,64	440	587,32	784	987,76	1318,5
4. felharmonikus	412,05	550	734,15	980	1234,7	1648,2
5. felharmonikus	494,46	660	880,98	1176	1481,6	1977,8
6. felharmonikus	576,87	770	1027,8	1372	1728,6	2307,4
7. felharmonikus	659,28	880	1174,6	1568	1975,5	2637

4.4. ábra. *A húrok alapharmonikusai és felharmonikusai Hz-ben*

Amint látható, a B és a felső E húrok frekvenciáinak közelében található az A húr ötödik és az alsó E húr ötödik és hetedik felharmonikusa. Amennyiben a gitár tökéletesen be van hangolva, ez nem okoz problémát, mivel akkor ezek egymást erősítik. Azonban ez az esetek többségében nincs így. Erre egy példa a következő oldalon található 4.5 képen látható spektrumrészlet.



4.5. ábra. *A B húr első és az alsó E húr ötödik felharmonikusa*

Itt egy elhangolt gitár húrjai közül a B húr detektálása egyszerű maximumkereséssel nem adna jó eredményt, mert az alsó E húr ötödik felharmonikusa nagyobb amplitúdóval van jelen. Ehhez hasonló esetek miatt mindenképp szükséges egy algoritmus, amely még ezen felharmonikusok jelenléte esetén is képes megtalálni a két húr hangmagasságát.

Ezt az algoritmust Matlabban implementáltam, és a következő főbb lépéseket hajtja végre az amplitúdóspektrumon:

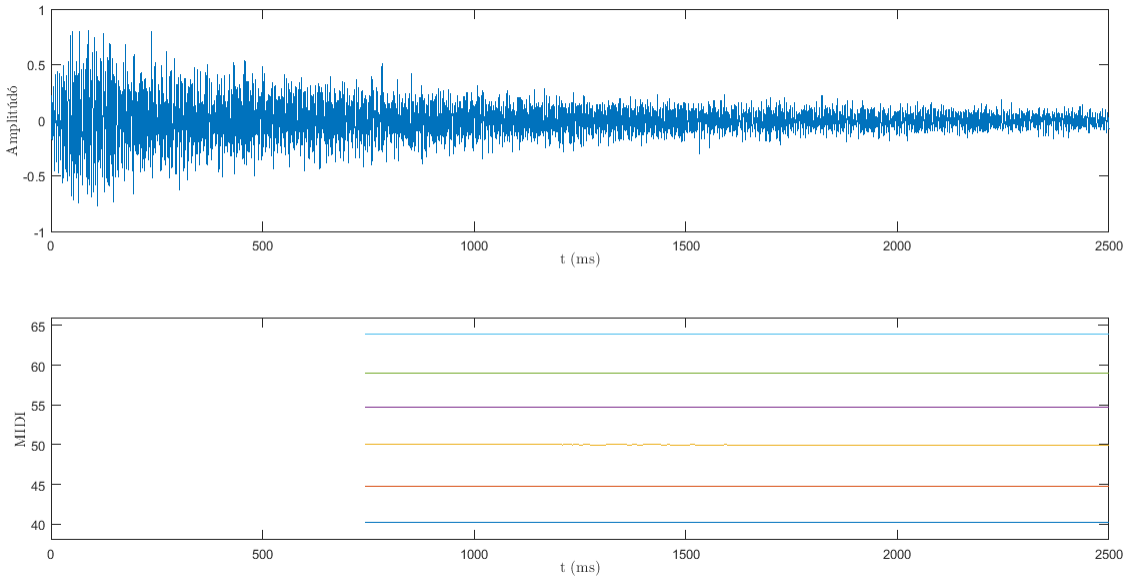
- Mind a hat húrra kiszámoljuk az alsó és felső határfrekvenciát (± 50 centes távolság alapján), majd ezeket átváltjuk indexekre a következő összefüggés alapján:

$$index = \frac{f \cdot L}{F_s} \tag{4.1}$$

- Az alsó E húrtól a G-ig, minden egyes húr esetében a kiszámolt határfrekvenciáknak megfelelő indexek közt maximumkereséssel majd interpolációval megállapítjuk a csúcsok helyét. Ezekből frekvenciákat számolunk, majd egy tömbbe mentjük őket.

- A B és a felső E húroknál elsőnek csúcskeresést kell végrehajtani a határfrekvenciák közt. (Csúcskeresésnél a legegyszerűbb módszert használtam, azaz minden értéket csúcsnak vettem amely előtt és után nála alacsonyabb érték szerepelt.) A megtalált csúcsoknak egyesével meghatározzuk a frekvenciáját, majd tömbe mentjük őket.
- A B húr esetében minden olyan csúcsot nyolcadrészére csillapítunk (empirikus úton megválasztott érték), amelynek frekvenciája a korábban megtalált alsó E húr ötödik felharmonikus frekvenciájának ± 2 Hz-es környezetében található. Miután minden csúcsot megvizsgáltunk, maximumkereséssel itt is megállapítjuk a frekvenciát[1].
- Ugyanezt megismételjük a felső E húrnál is, csak ott az A és az E húr frekvenciája alapján csillapítunk. (Tapasztalati úton megállapítottam, hogy az alsó E húr hetedik felharmonikusával nem kell foglalkozni, mert annak amplitúdója már alig érzékelhető.)

Az algoritmus teszteléséhez egy klasszikus gitár minden húrjának hangját külön rögzítettem, majd azokat az Audacity nevű audió-szerkesztő programmal egyforma hosszúra vágtam. Ezeket a hangsávokat egyesítve egy WAV fájlba mentettem. A 4.6 ábra mutatja, hogyan ismerte fel az algoritmus az egyszerre alhangzott húrok hangmagasságát.



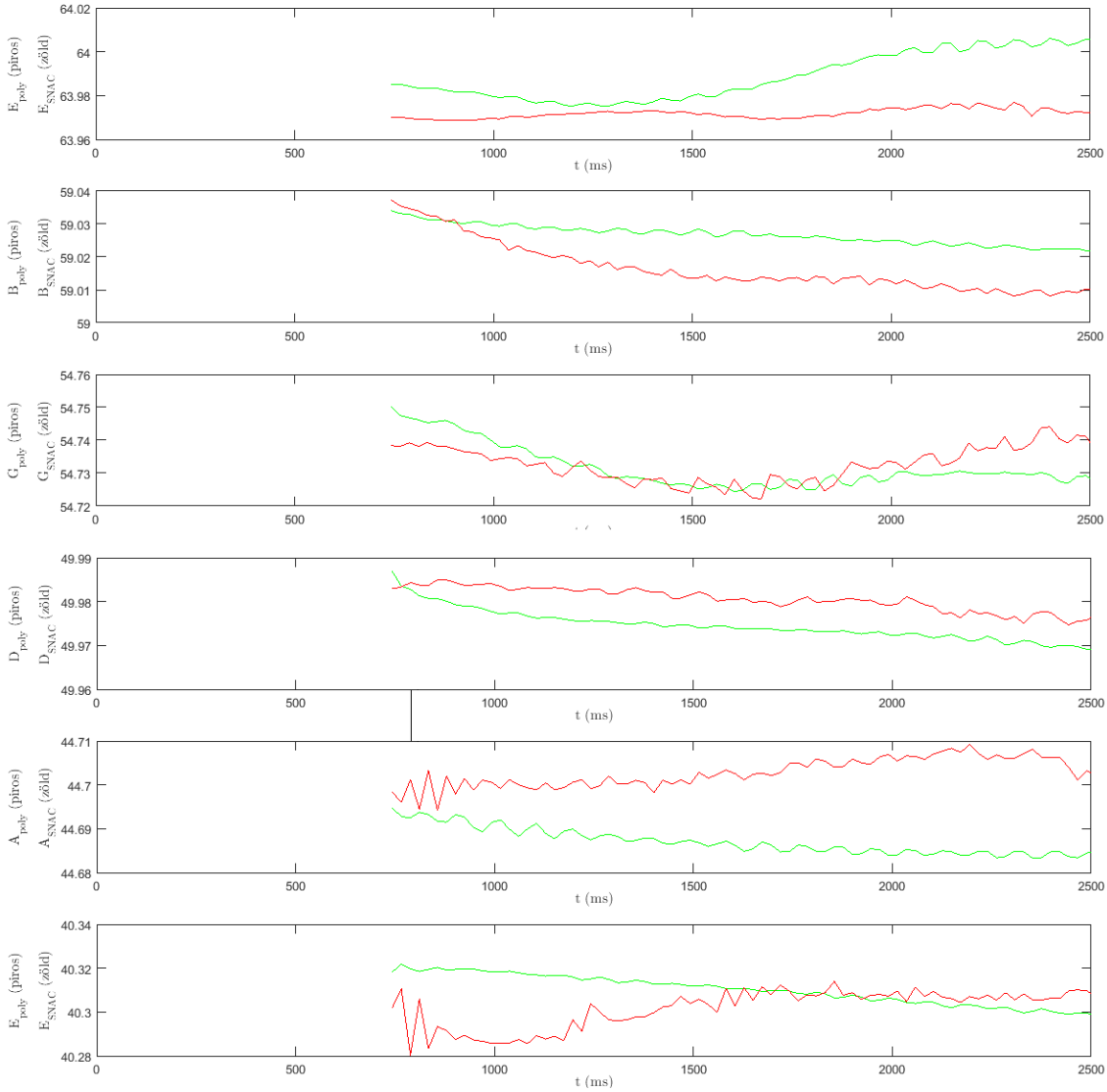
4.6. ábra. *A polifonikus hangolóalgoritmus által felismert hangmagasságok*

A képen pontosan nem látszik, de az ingadozás mértéke egyik húr esetében sem nagyobb három centnél, ami elég stabilnak mondható. Az ábrán is érzékelhető viszonylag nagy késleltetést a 2^{15}-es ablakméret okozza.

[1]Igaz, hogy ha a két húr harmóniában van, a két harmonikus egybeesik, Ekkor azt érjük el, hogy a keresendő csúcsot is csillapítjuk. Azonban ez a maximumkeresésnél nem jelent problémát, mivel a csillapítás nem annyira nagy, hogy a zajszint alá kerüljön a csúcs.

Összehasonlítás a SNAC függvénnyel

Ahhoz, hogy jobb rálátásunk legyen a módszer pontosságára, a kapott eredményeket összevetettem a SNAC függvény által számolt értékekkel. Ehhez a hangsávokat egyesítés előtt külön-külön hangfájlokba mentettem, majd azok hangmagasságait – a polifonikus megoldással megegyező ablakmérettel – kiszámoltattam a SNAC-re épülő algoritmussal is.



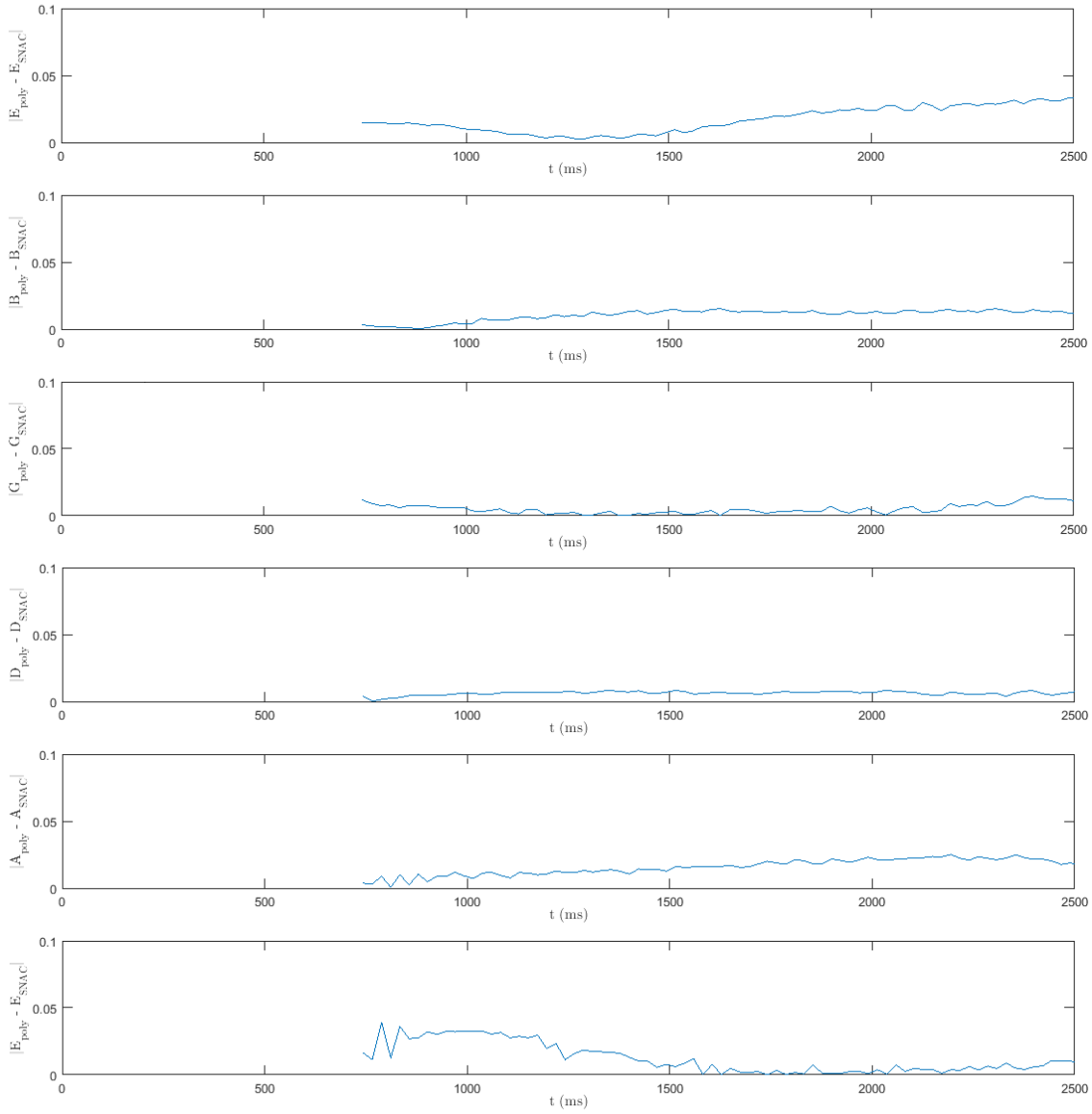
4.7. ábra. *A SNAC függvény (zöld) és a polifonikus hangolóalgoritmus (piros) által meghatározott hangmagasságok*

A két módszer eredményei közti eltérések elég változatosak. Látszik, hogy a kisebb ablakkal dolgozó, több átlagoláson átesett SNAC által számolt értékek sokkal kisebb mértékű ingadozást mutatnak, illetve az is, hogy ez az ingadozás a polifonikus megoldás esetében a frekvencia növekedésével csökken.

Ezen kívül jól látható, hogy a SNAC függvény által meghatározott értékek az idő előrehaladtával csökkenek, ami megfelel az elvártaknak[2].

[2]A gitárhúrok fizikájából következik, hogy pengetést követően a hangmagasság folyamatosan csökken.

Az eltérések mértékét a 4.8 képen ábrázoltam.



4.8. ábra. *A SNAC függvény és a polifonikus hangolóalgoritmus által meghatározott hangmagasságok közti eltérés*

Amint látható, az eltérés mértéke az esetek többségében nem haladja meg a három centet. A legnagyobb eltérés a két *E* húrnál tapasztalható, melyek esetében 4-5 centes különbség is előfordul. Sajnos ennek a differenciának az okára még nem sikerült rájönnöm, azonban ha ezt sikerül megoldanom, akkor ez az eljárás alkalmas lesz pontos polifonikus hangolás elvégzésére.

5. fejezet

Implementáció Android platformon

Ebben a fejezetben röviden ismertetem az Android operációs rendszerrel kapcsolatos tudnivalókat, majd bemutatom az előző fejezet végén választott eljárások Androidra történő implementálásának menetét.

5.1. Az Android bemutatása

Az Android egy Linux-kernel alapú mobil operációs rendszer, mely 2005-ben – az Android Incorporated nevű kaliforniai cég felvásárlásával – a Google tulajdonába került. Habár sokan mobiltelefonos és tabletes platformmal azonosítják, mára ezeken kívül karórák, televíziók vagy akár digitális fényképezőgépek is futtathatják. Mivel erősen érintőkijelző-orientált rendszerről van szó, így használata minden olyan eszközön kényelmes, amelynek kis kijelzője van, és az adatbevitel nem tipikusan egérrel és/vagy billentyűzettel történik.

5.1. ábra. *Az Androidot futtató Samsung Gear S2 karóra és a Nexus 6 mobiltelefon*

A főleg ARM chipeken futó[1] Android nagy előnye, hogy nyílt forráskódú, rugalmas és könnyen alakítható rendszer, emiatt mára az Apple és a Microsoft kivételével (amelyek saját fejlesztésű platfromjaikat használják) minden okostelefon-gyártó cég rendelkezik androidos modellekkel.

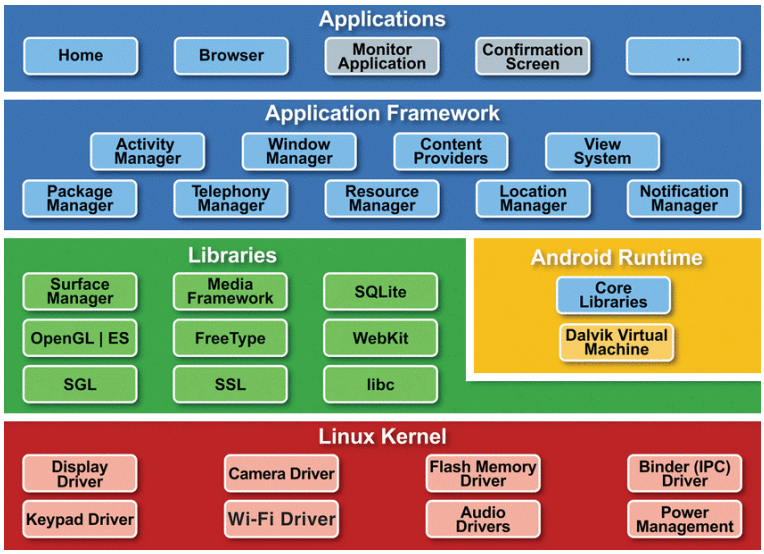
[1]Bár az androidos készülékek nagy része ARM processzort használ, egyre több olyan eszköz jelenik meg, amely az Intel alacsonyfogyasztású RISC (x86-64) chipjeit, az úgynevezett Atom processzorokat tartalmazza.

Az évek során rengeteget fejlődött a rendszer, és ez a fejlődés a verziókon is megfigyelhető. Míg a 2009 szeptemberében megjelent Android 1.6 (Donut) a többpontos érintést sem támogatta, addig a legújabb 6.0-ás (Marshmallow) verzió már fejlett memória kezeléssel és újlenyomatérzékeléssel is rendelkezik.

Ezen verziók nyomonkövetése fejlesztőként fontos feladat, mivel egyes verziók között komoly API-beli különbségek lehetnek[2]. Ezért fejlesztés előtt alaposan át kell gondolnunk, melyik az a minimum verzió, amelyet még támogatni fog az alkalmazásunk. Általánosságban igaz, hogy minél magasabb a minimális verziószám, annál több funkció használható, de az alkalmazás így kevesebb emberhez jut el.

5.1.1. Az Android szerkezete

A következő ábrán a platform szerkezete látható.



5.2. ábra. *Az Android operációs rendszer felépítése [15]*

A legalsó szinten van a Linux-kernel. Itt történik a memóriakezelés és a folyamatok ütemezése, illetve ezen a szinten vannak a hardvert kezelő eszközmeghajtók programjai is.

A Linux-kernel felett találhatóak a különböző programkönyvtárak, és részben rájuk épül az Android-futtatókörnyezet. Ennek szerves részét alkotja a Dalvik virtuális gép, amely lényegében a személyi számítógépeknél megszokott Java Virtual Machine (JVM) egy jelentősen újratervezett és optimalizált verziója, és ez felelős az Androidra készített Java-alkalmazások futtatásáért. A Dalvik virtuális gépen belül a memóriakezelés *garbage collectorral* történik, így a Java programozási nyelvnél megszokott módon itt sem kell a memóriafelszabadítással foglalkoznunk.

A legfelsőbb szinten vannak a Java-alapú megoldások (pl.: Home képernyő, főmenü stb.), amelyeket a Dalvik futtat. Lényegében ezek adják a látható operációs rendszert.

[2]Az API (Application Programming Interface) alkalmazások fejlesztését segítő, előre megírt függvények gyűjteménye. Segítségével a programozók egyszerűen kommunikálhatnak az operációs rendszerrel, vagy bármilyen hardvereszközzel.

5.1.2. Az APK állomány

Android platformra úgynevezett APK állományokat telepíthetünk, amelyek becsomagolva tartalmazzák az alkalmazásokat. Bármilyen formában (Google Play-ről letöltve, saját projekt fordítását követően, SD kártyára másolva stb.) eljuttatva a készülékünkre az APK-t, könnyedén telepíthetjük. A telepítésért a PackageManagerService felelős, és az alkalmazások a készülék memóriájára vagy bizonyos körülmények között külső SD kártyára is kerülhetnek.

Az APK állomány leginkább a Java világban megszokott JAR fájlhoz hasonlítható, de vannak jelentős eltérések. Telepítése során a felhasználónak jóvá kell hagynia az alkalmazás hozzáférési szintjeit (pl.: internet elérés, telefonhívás, üzenetküldés, hozzáférés az SD kártyához stb.), így a gyanús alkalmazások könnyen kiszűrhetőek[3].

Egy apk fájl tipikus tartalma a következő:

- Alkalmazástanúsítvány és metainformációk;
- Erőforrásadatok és erőforrások listája;
- Név, minimális verziószám, jogosultságok és könyvtárak;
- A Dalvik számára érthető formátumban lefordított osztályok;

5.1.3. Alkalmazások fordításásnak menete

Android platformon az alkalmazások fejlesztése általánosan Java nyelven történik, amelyhez egy SDK-t (Software Development Kit) biztosít a Google. Amennyiben alacsonyabb szintű funkciókat is el szeretnénk érni, lehetőségünk van natív kódot (C/C++) is készíteni az NDK (Native Development Kit) segítségével.

A Java nyelven írt alkalmazások a Dalvik virtuális gépen futnak felügyelten, a memóriakezelésért a futtatókörnyezet és a virtuális gép a felelős. Mint már említettem, a memória felszabadítását egy GC (Garbage Collector) végzi, ám ez nem jelenti azt, hogy felelőtlenül bánhatunk az objektumok (az osztályok példányai) létrehozásával.

Android fejlesztés során a forráskódot és a felhasználói felületet külön hozzuk létre. A felhasználói felület definiálására XML-állományokat használhatunk, de ez nem zárja ki, hogy a forráskód szintjén is létrehozzuk, elérjük és manipuláljuk azt. A projektleíró állomány (Android Manifest) szintén XML-formátumban érhető el. Ebben kell megadnunk a minimális verziószámot, az alkalmazás jogosultságait illetve a külső könyvtárak és alkalmazáskomponensek listáját.

[3]Felmérések szerint a Google Play-en található szoftverek 5 százaléka képes telefonhívást indítani a felhasználó beavatkozása nélkül.

Egy új Android-projekt létrehozása után a forráskód az *src* mappában, míg a felhasználói felület XML-állományai a *res* mappában találhatóak. A fejlesztői környezet automatikusan generál egy *R.java* állományt, az erőforrás-állományokat és a forráskódot azonosítókkal (id) köti össze.

Az Android-projekt fordításának eredménye az előző részben bemutatott *apk* állomány, melyet közvetlenül a mobileszközünkre telepíthetünk.

5.1.4. Egy alkalmazás komponensei

Egy androidos alkalmazás több különböző típusú komponensből épülhet fel, amelyek közül egyet is elég az alkalmazásnak tartalmaznia. Az Android négy fő alkalmazáskomponens-típust támogat, amelyek eltérő feladatokat látnak el.

Ezen komponenstípusok az alábbiak:

- Activity
- Service
- ContentProvider
- BroadcastReceiver

Az Android alkalmazás ugyanabból a komponenstípusból több komponenst is tartalmazhat.

Az *Activity* a leggyakrabban használt komponenstípus. Saját látható felülettel rendelkezik, és legtöbbször az alkalmazások több *Activity*-t is tartalmaznak, amelyek különböző célokat szolgálnak, de együtt adják a teljes funkcionalitást. Ilyenkor az alkalmazáson belül könnyedén válthatunk egyik *Activity*-ről a másikra. Például egy zenelejátszó applikációban a kezdő *Activity* a zenefájlok listás nézete. Amennyiben kiválasztunk egy zenefájlt, ez az *Activity* szünetelteti működését, és átadja helyét egy újnak, ami a zenelejátszó-funkciókat (Play/Stop gombok, tekerés stb.) jeleníti.

A *Service* komponensnek az *Activity*-vel ellentétben nincs felhasználói felülete, mivel háttérben futó feladatnak felel meg. A rendszer alapértelmezetten is több különböző szolgáltatást futtat, ezek biztosítják a megfelelő működést. Például a korábban említett zenelejátszó *Service*-ként is képes futni a háttérben, amíg mi más alkalmazást (pl.: böngésző) használunk.

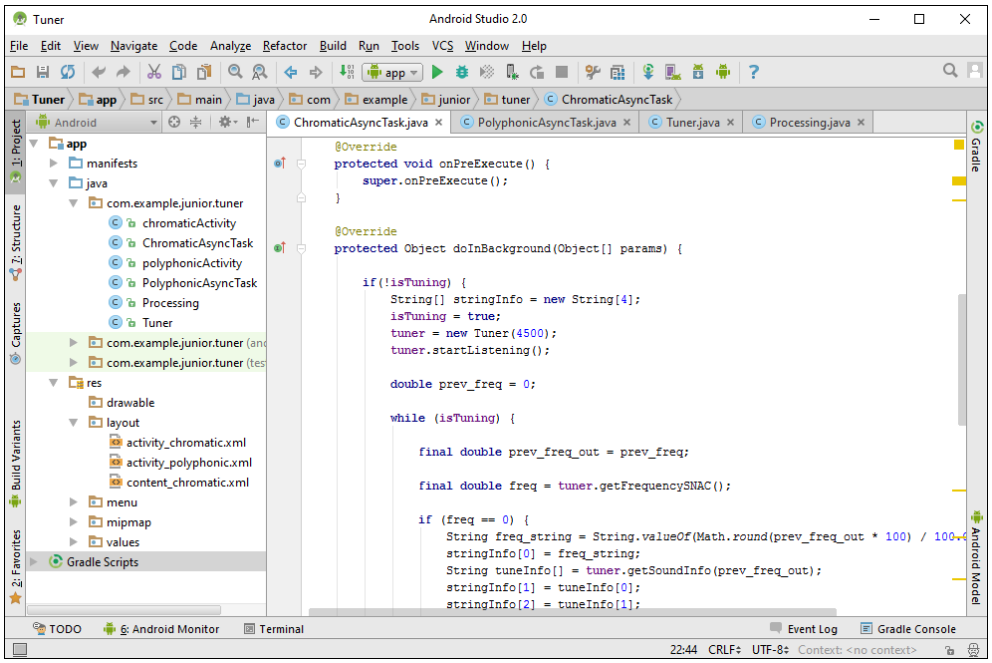
A *ContentProvider* feladata egy adatforrás kezelése és az arra vonatkozó kérések kiszolgálása. Ez viszonylag bonyolulttá teszi a komponenst, mivel az adatforrás bármi lehet (pl.: készüléken lévő SQLite adatbázis, weben keresztül elérhető RSS stb.). Például a telefonokban alapból megtalálható hívásnapló alkalmazások is *ContentProvider*-rel kezelik a készüléken tárolt hívásnapló-adatokat.

A *BroadcastReceiver* komponensnek az a feladata, hogy a különféle események hatására aktiválódjon, és valamilyen feladatot végrehajtson. Az Android a legtöbb eseményt úgynevezett *broadcast*-ok formájában jelzi (pl.: Wifi-re csatlakozás, GPS aktiválása, alacsony akkumulátorszint stb.). Alkalmazásaink *BroadcastReceiver*-ekkel feliratkozhatnak ezekre a

broadcast eseményekre, így ha azok bekövetkeznek, végrehajthatjuk az implementált funkcióinkat. Például egy online videónéző alkalmazás *BroadcastReceiver* komponensén keresztül gyenge internetkapcsolat esetén csökkentheti a sávszéelsséget [8].

5.1.5. A fejlesztőkörnyezet

Androidfejlesztésre leggyakrabban az Android Studio nevű szoftvert használják, ami egy IntelliJ IDEA[4] alapú integrált fejlesztőkörnyezet (IDE). Jelenleg ez a Google által hivatalosan támogatott IDE, melyet kifejezetten androidos alkalmazások fejlesztésére hoztak létre. Több operációs rendszerre is (Windows, OS X, Linux) ingyenesen elérhető.



5.3. ábra. *Az Android Studio 2.0*

Bár az Android Studio kissé erőforrásigényes, rengeteg fejlesztést segítő funkciója van, mint például:

- Android-specifikus gyors átnevezések és javítások;
- API-k könnyű és gyors elérése;
- Grafikus UI tervező (F.0.4 ábra);
- Sablonalapú tervezőegység az általános alkalmazások fejlesztéséhez;
- Beépített támogatás a Google Cloud Platformhoz;

A fejlesztés során végig ezt a szoftvert alkalmaztam, illetve tesztelésre egy Samsung Galaxy S4 és egy LG G2 okostelefont használtam.

A következő részben a hangolóalkalmazás megvalsósításának menetét mutatom be.

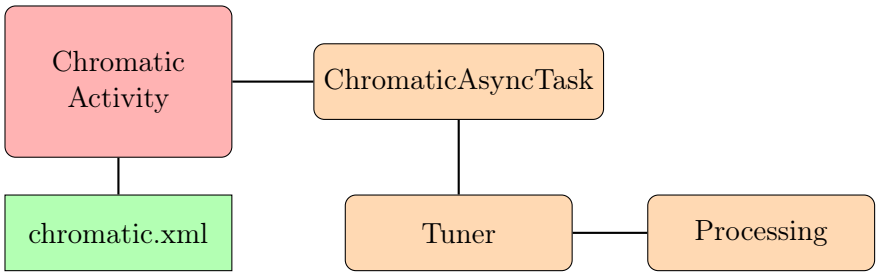
[4]Az IntelliJ IDEA egy Java IDE a JetBrains fejlesztésében.

5.2. A kromatikus hangoló implementálása

Összességében az alkalmazás két *Activity*-re épül. Egyik a kromatikus, míg a másik a polifonikus hangoló felületét adja.

Ebben a részben az alkalmazás kromatikus hangolófunkciójának fejlesztését mutatom be. A fejlesztés során a harmadik fejezet végén kiválasztott SNAC függvényre épülő algoritmust implementáltam.

A hangoló struktúrája a 5.4 ábrán látható.



5.4. ábra. *A kromatikus hangoló moduláris felépítése*

A hangoló öt fő modulból áll. Egy Activityből és a hozzátartozó XML-állományból, továbbá három Java osztályból.

5.2.1. Chromatic Activity

Ez az *Activity* biztosítja a kapcsolatot az alkalmazás és a felhasználó közt. Mikor elindítjuk a programot, a *chromatic* nevű XML állomány alapján létrehozza a kromatikus hangoló felületét, majd elkezdi a hangolást.

A következő fontosabb részekből áll:

- **startTuning metódus**[5]: A nevéből is következik, hogy ez a tagfüggvény felelős a hangolás megkezdéséért. Meghívásakor létrehoz egy példányt a *ChromaticAsyncTask* osztályból, majd kiadja annak *execute* parancsát. Ekkor a *ChromaticAsyncTask* objektumon belül implementált függvények lefutnak. Tulajdonképpen ezek hajtják végre a hangolás számításait, és az értékek kiírását a felhasználói felületre.

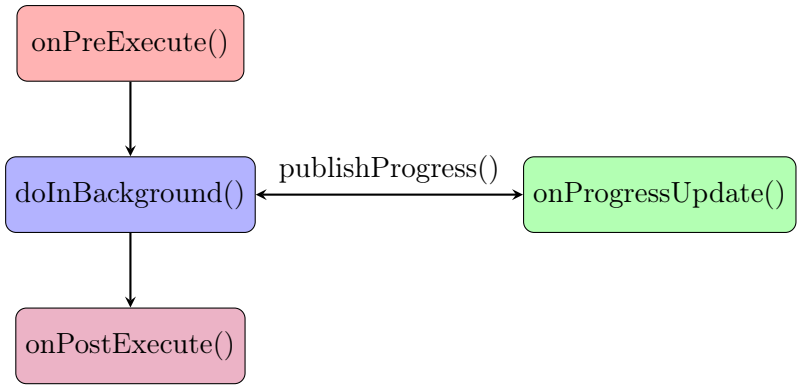
- **stopTuning metódus**: Leállítja a hangolást, és felszabadítja a korábban deklarált *AsyncTask* objektumot. Amennyiben az alkalmazás háttérbe kerül, vagy újra láthatóvá válik, akkor ezeknek megfelően a hangolás is leáll, vagy újraindul.

- **startPolyphonicActivity metódus**: Amennyiben szeretnénk polifonikus hangolásra váltani (aminek külön *Activity*-je és felülete van), akkor ez a függvény kerül meghívásra. Ebben az esetben a jelenleg futó kromatikus hangoló szünetelteti működését, és elindítja a másik hangolófunkciót.

[5]Metódusnak hívjuk az osztályok tagfüggvényeit. Mivel a Java nyelben minden függvény valamilyen osztály tagfüggvénye, így tetszés szerint hívhatóak mindkét módon.

5.2.2. ChromaticAsyncTask osztály

A *ChromaticAsyncTask* az *AsyncTask*-ból származik, amely egy alapértelmezett androidos osztály. Lényege, hogy viszonylag könnyedén tudunk párhuzamosan műveleteket végezni a háttérben úgy, hogy közben módosíthatjuk a UI értékeit [3]. Esetünkben ez azért fontos, mert a hangolásnak folyamatosan futnia kell, és közben másodpercenként többször felül kell írnia a felhasználói felület értékeit (Például ki kell íratnia a játszott hang jelét, frekvenciáját és elhangolódásának mértékét centben.)



5.5. ábra. *Az AsyncTask osztály működése*

Erre az osztály a következő két függvényt használja, amelyeket céljainknak megfelően felül kellett írni:

- **doInBackground metódus**: A háttérszámítások ebben a függvényben történnek. Itt inicializálunk egy példányt a *Tuner* osztályból. A konstruktor segítségével egy 4500 mintából álló ablakméretet adunk meg a *Tuner* objektumnak. Ezután egy *while* ciklusba lépünk, ami egy *boolean* változó (*isTuning*) segítségével ellenőrzi, hogy nem kell-e leállítani a hangolást. (Az *Activity stopTuning* függvényével lényegében ezt a változót billentjük *false*-ba.) A cikluson belül történik meghívásra a deklarált Tuning objektum *getFrequencySNAC* metódusa, ami a SNAC függvényre épülő algoritmust hajtja végre, és visszatér a beolvasott ablak hangmagasságával, majd ezeket az adatokat kiküldi a *publishProgress* utasítással.

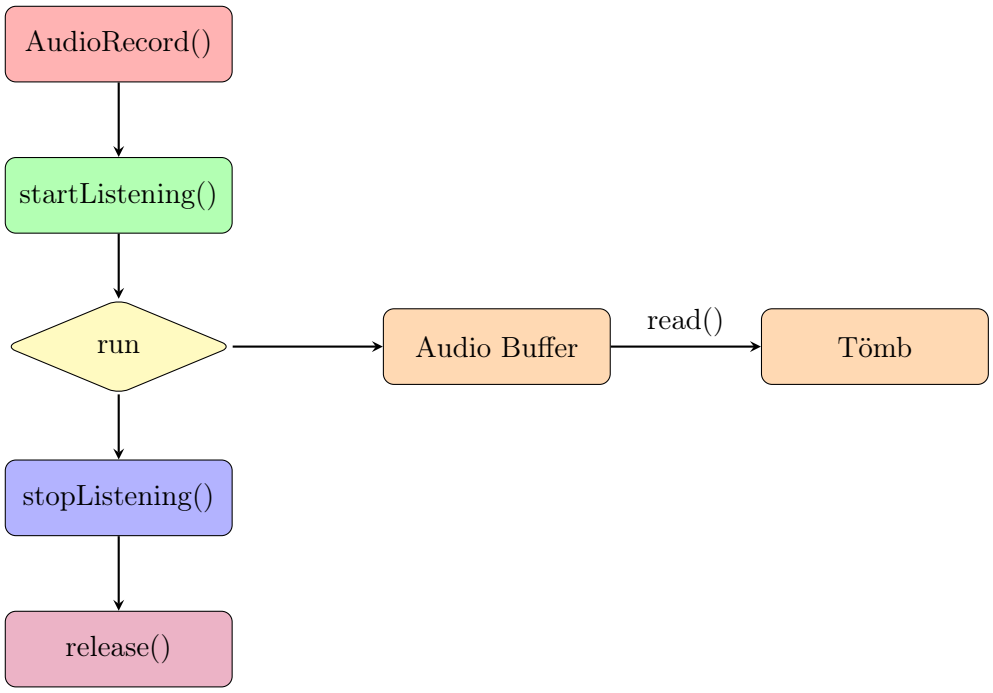
- **onProgressUpdate metódus**: Az *onProgressUpdate* a *doInBackground* metódus által kiküldött adatokat kapja meg, majd ezen adatokat feldolgozza, és módosítja a UI-t.

5.2.3. A Tuner osztály

A *Tuner* osztály mind a kromatikus, mind a polifonikus hangoló esetében használatra kerül. Ez vezérli a mikrofonból tömbbe való olvasást, illetve a hangmagasságkiszámító függvények itt kerülnek meghívásra a *Processing* osztályból.

5.2.4. Adatok kinyerése a mikrofonból

Androidon a hangbemenetről (jack bemenetről vagy beépített mikrofonból) történő olvasásra több osztály is elérhető. Az egyik leggyakrabban használt módszer az *AudioRecord* osztály alkalmazása, melynek nagy előnye, hogy közvetlenül képes a mikrofon bufferéből tetszőleges méretű tömbbe írni az adatfolyam értékeit [4]. Az *AudioRecord* használatának folyamatát a 5.6 ábra mutatja be.



5.6. ábra. *Az AudioRecord osztály alkalmazása*

Az *AudioRecord* objektum inicializálásakor a konstruktor paramétereiként meg kell adnunk a következőket:

- forrás (mikrofon)
- mintavételi frekvencia (44100 Hz)
- csatorna (MONO)
- kódolás (PCM 16bit)
- audio buffer mérete (kromatikus: 4500, polifonikus: 32768)

Következő lépésben a *startListening* függvénnyel elindítjuk az audio buffer feltöltését. Ekkor van lehetőség a *read* metódussal tömbbe írni a buffer tartalmát. Ezután a tömböt átadjuk a *Processing* osztály *getFreqBySNAC* tagfüggvényének paramétereként, ami visszatér a kiszámolt frekvenciaértékkel.

Amennyiben szünetel a hangolás, a *stopListening* metódussal leállítjuk a buffer feltöltését, majd a *release* függvénnyel felszabadítjuk az *AudioRecord* erőforrásait.

5.2.5. A Processing osztály

A tényleges számításért felelős függvények a *Processing* osztályban találhatóak. Ezek statikusak, így az osztály példányosítása nélkül meghívhatóak.

Mind a kromatikus, mind a polifonikus hangoló esetében kulcsfontosságú tényező a csúcsok megtalálása. A kellő pontosság érdekében mindkét esetben a korábban említett parabolikus interpolációt kellett alkalmazni [5].

A kromatikus hangoló alapját a SNAC függvényre épülő algoritmus alkotja, melyet a *getFreqBySNAC* nevű metódusként implementáltam. Paraméterként egy *double* értékekből álló tömböt vár, és az ebből számolt, szintén *double* frekvenciaértékkel (Hz) tér vissza.

A függvény a következő lépéseket hajtja végre:

- A *Processing*-en belül implementált SNAC függvényt végrehajtjuk a tömbön.
- Nullátmeneten alapuló csúcskeresést végzünk el az adatsoron. Ehhez minden második nullátmenet közt maximumkeresést kell végrehajtani.
- Az alapharmonikusnak megfelelő csúcsok detektálása oly módon, hogy a harmadik fejezetben említett küszöbszint feletti csúcsokértékek helyeit egy tömbbe mentjük.
- Az előző pontban meghatározott csúcshelyek közt eltelt időt a mintavételi frekvencia ismeretében kiszámítjuk, majd vesszük azok reciprokát, így megkapjuk a frekvenciaértékeket.
- A kapott frekvenciaértékeket átlagoljuk[6], így ezáltal is nő a pontosság.

A kiszámolt frekvenciaértékből az első fejezetben ismertetett összefüggéssel könnyedén kiszámítható a hang MIDI kódja, amiből tisztán látszik az elhangolódás mértéke, továbbá a MIDI kódból egy egyszerű állapotgéppel megkaphatjuk a zenei hang betűjelét.

[6]A lejátszott hang frekvenciája minél alacsonyabb, annál kevesebb csúcs található egy ablakon belül. Emiatt kevesebb értékből képezhető átlag, azonban a mintavételezés pontossága a frekvencia csökkenésével nő, így ez a két hatás kiegyenlíti egymást

5.2.6. A felhasználói felület

A felhasználói felület megtervezésénél az egyszerűség volt a szempont. Ezért összesen három fajta információt jelenít meg a felület: a játszott hang betűjelét (E, D#, G stb.), az elhangolódás típusát (magas, alacsony vagy jó) és mértékét (centben).

A UI alapjának egy BOSS digitális hangoló előlapi nézetét választottam (5.7 ábra). Ezt képszerkesztő programmal addig alakítottam, amíg csak egy kijelző, egy gomb és egy ledsor maradt (illetve feliratok).

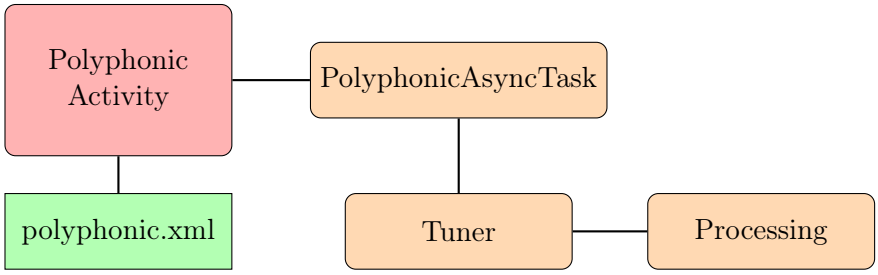
5.7. ábra. *A felhasználói felület alapjául szolgáló BOSS TU-80*

5.8. ábra. *A kromatikus hangoló felülete*

A kijelzőn megjelenik a játszott hang betűjele és az elhangolódás mértéke centben. A ledsoron látható az elhangolódás típusa, illetve bal oldalon van egy gomb a polifonikus hangoló megnyitásához. A ledek az 5 centnél nagyobb eltérést jelzik piros fénnyel.

5.3. A polifonikus hangoló implementálása

Ebben a fejezetben az alkalmazás polifonikus hangoló komponensének fejlesztését mutatom be. Az applikáció ezen funkciójának struktúrája a 5.9 ábrán látható.



5.9. ábra. *A polifonikus hangoló moduláris felépítése*

5.3.1. Polyphonic Activity

Az alkalmazás indításakor alapértelmezettként a kromatikus hangoló felülete jelenik meg. Amennyiben át szeretnénk váltani a többhúros hangolásra, azt egy gombnyomás segítségével megtehetjük. Ekkor a kromatikus hangoló szünetelteti működését, és átadja a futást a *Polyphonic Activity*-nek. Ilyenkor az létrehozza a *polyphonic.xml* fájlban tárolt UI-t, és elkezdi a hangolást.

Az itt implementált fontosabb függvények a következők:

- **startTuning metódus**: A kromatikus hangolónál leírtakhoz hasonlóan ez a függvény kezdi el a hangolást. Meghívásakor *true*-ba billenti az *isTuning* logikai változót, ami által ellenőrizhető a futás állapota. Ezután deklarál egy *PolyphonicAsyncTask* objektumot, majd meghívja annak *execute* tagfüggvényét, ami hatására az *AsyncTask*-on belüli metódusok megkezdik működésüket.

- **stopTuning metódus**: *False*-ba állítja az *isTuning* flaget, és felszabadítja a korábban deklarált *AsyncTask* objektumot.

- **startChromaticActivity metódus**: Amennyiben szeretnénk visszatérni kromatikus hangolásra, akkor egy gombnyomás hatására ez a függvény kerül meghívásra. Ekkor a *Polyphonic Activity* leállítja futását, és elindítja a másik hangolófunkciót.

5.3.2. A PolyphonicAsyncTask és a Tuner osztály

A kromatikus hangolóhoz hasonlóan a háttérben történő számítások és a közben történő UI módosítások az *AsyncTask* két tagfüggvényével valósulnak meg:

- A *doInBackground* metóduson belül inicializálunk egy *Tuner* objektumváltozót 32768-as ablakhosszal. Ezután egy *while* ciklusban hívjuk meg ennek az objektumnak a *getPolypolyphonic* metódusát, ami a gitár hat húrjának frekvenciaértékével tér vissza. Ezeket egy tömbbe mentjük, majd ezt a tömböt adjuk át a *publishProgress* utasítással.

- Miután az *onProgressUpdate* függvény megkapta az értékeket, a felhasználói felület megfelelő részét módosítja.

A *getPolyphonic* függvény meghívásakor a *Tuner* osztályon belül elkezdődik az audio buffer feltöltése. Innen az adatok bekerülnek a megadott ablakméretnek megfelelő nagyságú tömbbe, majd ezt a tömböt adjuk át paraméterként a *Processing* osztály statikus *getPolyFrequencies* metódusának, amely visszatér a kiszámolt frekvenciaértékekkel.

5.3.3. A getPolyFrequencies függvény

A fourier transzformációt és a megfelelő csúcsok kiválasztását a *Processing* osztály *getPolyFrequencies* tagfüggvénye hajtja végre. Ennek lépései a következők:

- A hat keresendő húr frekvenciáját megadjuk, majd abból az 50 centes alsó és felső határfrekvenciákat meghatározzuk.
- A határfrekvenciákat átváltjuk indexekre, hogy az algoritmus tudja, melyik két index közt kell keresni az egyes csúcsokat.
- A fourier transzformációt elvégezzük a *FastFourierTransformer* osztály segítségével [6]. Ezután a kapott komplex vektor elemeinek venni kell az abszolútértékét, hogy megkapjuk az amplitúdóspektrumot.
- Az alsó E húrtól a G húrig a határfrekvenciáik közt maximumkereséssel megállapítjuk az egyes frekvenciaértékeket.
- A B és E húrok esetén a harmadik fejezet végén ismertetett eljárás alapján határozzuk meg a frekvenciákat.

A megtalált frekvenciaértékek egy lokális hatelemű tömbváltozóba kerülnek, így a függvény lefutása után ezzel tér vissza.

5.3.4. A felhasználói felület

A polifonikus hangoló felhasználói felületének kialakításakor – az egységesség megtartása érdekében – a kromatikus hangoló felületét vettem alapul. Az egyetlen különbséget a ledek hiánya jelenti.

5.10. ábra. *A polifonikus hangoló felülete*

A kijelzőn egyszerre látható a hat húr állapota. Amennyiben valamelyik húr el van hangolva, akkor az alkalmazás egy *kereszttel* (#) jelzi ha túl magas, vagy egy *bével* (b) ha túl alacsony. Behangolt húr esetén az annak megfelelő betűjel mellett nem található jelzés.

6. fejezet

Modulok értékelése

Ebben a fejezetben az alkalmazás teszteléséről és meglévő applikációkkal való összehasonlításáról írok, majd ezek függvényében értékelem a megvalósított hangolómodulokat.

A hangolófunkciók tesztelését valós hangszerekkel (fuvola, elektromos és klasszikus gitár), illetve számítógép hangszórójáról lejátszott hangmintákkal végeztem.

A kromatikus hangoló

A kromatikus hangoló tesztelésekor a sebességet és a pontosságot vizsgáltam.

Összehasonlítási alapnak a *Google Play*-ből ingyenesen elérhető (több mint egy millió letöltéssel rendelkező) *PitchLab Guitar Tuner* nevű alkalmazást választottam. Tesztelésre egy Samsung Galaxy S4 és egy LG G2 telefont használtam.

6.1. ábra. *Összehasonlítás a PitchLab-bel*

A modul használat közben kifejezetten gyorsan reagált. *Android Studio*-val elvégzett tesztek alapján másodpercenként több mint tízszer jelez vissza a felhasználói felület a játszott hang magasságáról. Ez az összehasonlításkor is jól érzékelhető, ugyanis a hangváltásokra gyorsabban felel, mint a *PitchLab*.

Több hangszerrel tesztelve arra a következtetésre jutottam, hogy az alkalmazás pontosan jelzi a játszott hang állapotát, azonban zajra (pl. hangosabb beszéd vagy a TV hangja) kissé érzékeny, így a hangolást mindenképp érdemes csendesebb környezetben végezni. Ezen a téren alulmarad az internetről letölthető társához képest, amely zajosabb környezetben is képes stabilan mutatni a hangmagasságot.

A polifonikus hangoló

Ahogy várható volt, a polifonikus modul nem fut túl gyorsan, köszönhetően a FFT-nek. Tesztek alapján $0,7$ másodpercenként jelez vissza a hangoló, ami első ránézésre soknak tűnhet, azonban használat közben egyáltalán nem volt zavaró.

A pontosság tesztelésére többféle képpen elhangolt klasszikus és elektromos gitárt használtam. A tesztek eredményei alapján a negyedik fejezet végén említett pontatlanság futás közben is érzékelhető volt, főleg az alsó E húr esetében. Sokszor fordult elő, hogy jól behangolt húrok esetén is az E hangját túl alacsonynak mutatta, és érdekes módon ez a hiba sokszor az A húrnál is előjött. A többi hang esetén megbízhatóan működött a hangoló.

6.2. ábra. *A HT-6 az alsó E húr esetében többször is hibázik*

Az alkalmazásboltban viszonylag nehezen található polifonikus funkciót támogató applikáció. Hosszas keresgélés után összesen kettőt találtam, és ezek közül is csak az egyik, a *HT-6 FastTune* nevű szoftver tűnt használhatónak, így ezt választottam összehasonlítási alapnak (6.2).

A *HT-6* valamivel gyorsabban működik, illetve rendelkezik extra funkciókkal is, mint

például más gitárhangolások támogatása.

Azonban pontosság tekintetében neki is vannak hibái. Ugyanúgy gondban van az alsó E és A húrok hangjának meghatározásánál, tehát úgy tűnik, fejlesztés során más is szembetalálkozott ezzel a problémával.

Ahhoz, hogy megértsük ennek a pontatlanságnak az okát, a jövőben további tesztekre lesz szükség.

7. fejezet

Összefoglalás

Dolgozatom célja az volt hogy fejlesszek egy kromatikus és polifonikus hangolásra alkalmas szoftvert Android platformra.

A fejlesztés előtt ismereteket gyűjtöttem különböző idő- és frekvenciatartománybeli hangoló eljárásokkal kapcsolatban. Ezeket az algoritmusokat MATLAB segítségével teszteltem, majd a tesztek eredményei alapján kiválasztottam az alkalmazás számára legalkalmasabb módszereket.

Az alkalmazásfejlesztés során utánajártam, milyen módszerek vannak az androidos készülékek mikrofonjából történő adatolvasásra, illetve mely ingyenesen elérhető függvénykönyvtárak használhatóak Fourier-transzformáció végrehajtására. Továbbá megismerkedtem az androidos felhasználói felületek tervezésének folyamatával.

Az alkalmazás megvalósítása után annak kromatikus és polifonikus modulját különböző hangszerek által keltett hangmintákkal teszteltem, majd összehasonlítást végeztem a Google alkalmazásboltjából ingyenesen letölthető megoldásokkal is.

7.1. Továbbfejlesztési lehetőségek

Az alkalmazás struktúrájának köszönhetően könnyen bővíthető új funkciókkal. Ilyen például a referencia frekvencia megadása, aminek köszönhetően nem csak a 440 Hz-es A hanghoz képest lehetne hangolni a hangszereinket. Polifonikus hangolónál egy könnyebb plusz funkció lehet a *capo* támogatása, illetve egy bonyolultabb továbbfejlesztés lenne a standard E-A-D-G-B-E gitárhangolás helyett más állások támogatásának bevezetése (pl.: *Drop D* hangolás: D-A-D-G-B-E, *Drop C* hangolás: C-G-C-F-A-D stb.).

Az algoritmusokon is lehet még fejleszteni, hogy kevésbé legyenek érzékenyek a zajra, illetve, hogy gyorsabban fussanak. Ezeken kívül a polifonikus hangolóalgoritmus pontosságán (főleg a E és A húr esetében) is érdemes még javítani.

Ezek kijavítása után, és az említett extra funkciók hozzáadása után akár a Google alkalmazásboltba is fel lehetne tölteni az applikációt.

A polifonikus hangolást továbbgondolva lehetne egy teljesen automatikusan működő akkordfelismerő modult fejleszteni, mely több hangszer esetében is működhetne, és előzetes beállítások nélkül lenne képes felismerni akár 5-6 hang kombinációját. Ez egy jóval bonyo-

lultabb feladat, és teljesen új ismereteket követel, ezért akár diplomatervként is megállná a helyét.

7.2. Utószó

Az androidos és iOS-es készülékek mind szoftveresen, mind hardveresen nagy ütemben fejlődnek, így a zenei alkalmazások száma is gyorsan növekszik. Ma már nem ritka, hogy profi zenészek is használnak tablet- vagy telefonalapú megoldásokat [12].

Úgy gondolom, hogy az okostelefonos hangolók professzionális felhasználásra is alkalmasak, mivel kellő pontossággal képesek működni. Talán a legnagyobb hibájuk (főleg az androidos készülékeknek), hogy az inputra lassabban reagálnak, mint a digitális hangolók, azonban ez idővel könnyen változhat. Viszont legnagyobb előnyük, hogy rengeteg ember számára gyorsan, egyszerűen és sok esetben teljesen ingyen elérhetőek.

Irodalomjegyzék

[1] Alternate guitar tunings. http://howtotuneaguitar.org/tuning/alternate-guitar-tuning-chart/.

[2] Patricio de la Cuadra, Aaron Master, Craig Sapp. Efficient pitch detection techniques for interactive music. Center for Computer Research in Music and Acoustics, Stanford University
International Computer Music Association 2001, Havana, Cuba.

[3] Android Developer. *AsyncTask*, 2016. http://developer.android.com/reference/android/os/AsyncTask.html.

[4] Android Developer. *AudioRecord*, 2016. https://developer.android.com/reference/android/media/AudioRecord.html.

[5] Matt Donadio. How to interpolate the peak location of a dft or fft if the frequency of interest is between bins. http://dspguru.com/dsp/howtos/how-to-interpolate-fft-peak.

[6] Apache Software Foundation. *Class FastFourierTransformer*, 2016. http://commons.apache.org/proper/commons-math/apidocs/org/apache/commons/math3/transform/FastFourierTransformer.html.

[7] Dr. Tevesz Gábor. *Mikrokontroller alapú rendszerek - Elektronikus jegyzet.* 2016. https://www.aut.bme.hu/Course/VIAUA348.

[8] Ekler Péter, Fehér Marcell, Forstner Bertalan, Kelényi Imre. *Android-alapú szoftverfejlesztés.* SZAK Kiadó, 2012.

[9] Sonic Research Inc. St-300 mini strobe tuner instruction manual. https://www.turbo-tuner.com/media/ST300mini_Manual.pdf.

[10] Imhof Iván. http://gsfanatic.com/hu/blog/hangolas-felejtsd-el. Cikk az elektronikus hangolók történetéről.

[11] Philip McLeod. *Fast, Accurate Pitch Detection Tools for Music Analysis.* PhD thesis, University of Otago, Dunedin, New Zealand, May 2008.

[12] Jordan Rudess, John Petrucci. Hourglass ala morphwiz and guitar. https://www.youtube.com/watch?v=np7t3t449cE.

[13] Peterson Electro-Musical Products. Strobe tuners: Their history and how they work. https://www.youtube.com/watch?v=U9HJiiLz_5w.

[14] Heinz Alfred Brockhaus, Hugo Riemann. *Zenei lexikon I–III.* Zeneműkiadó, 1985.

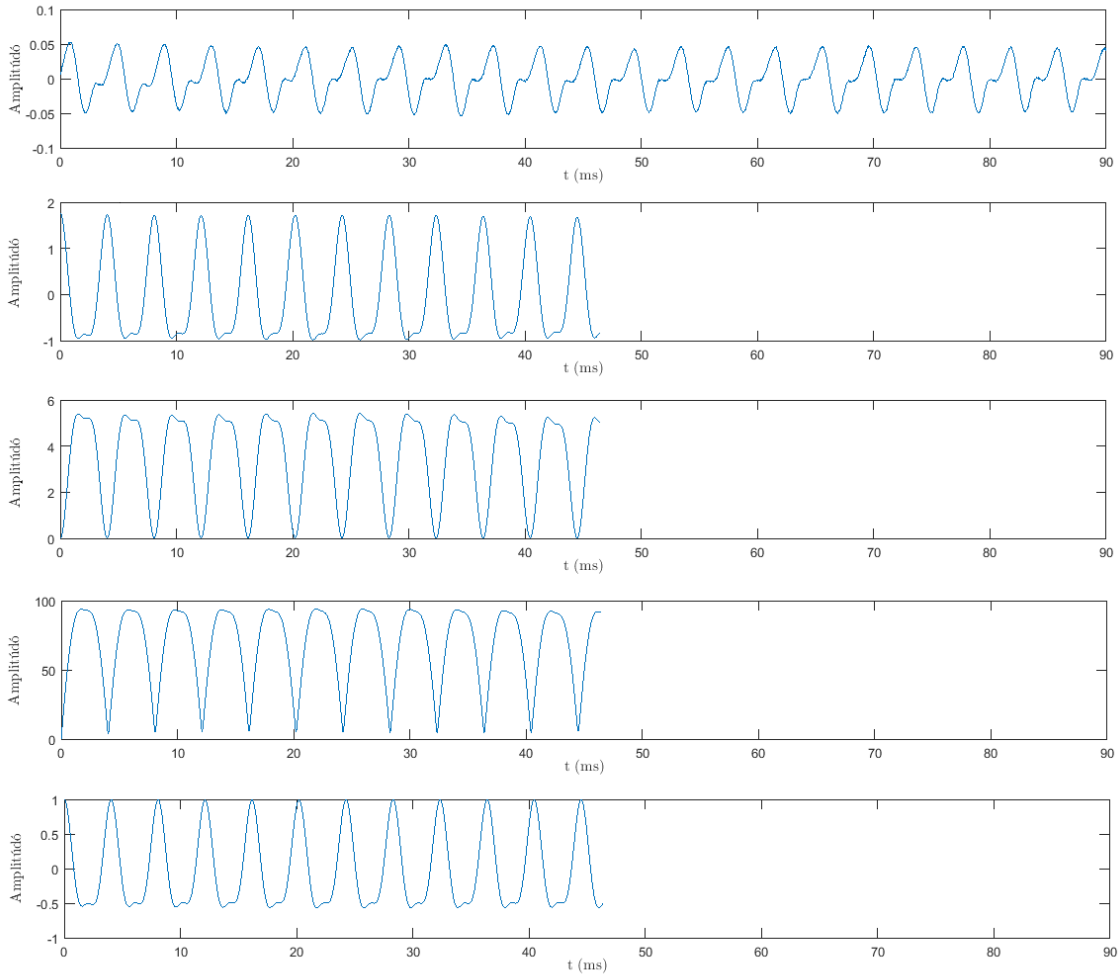
[15] SatworkS. Understanding android architecture project structure. http://satworks.blogspot.hu/2010/08/android-2-understanding-android.html.

[16] Gungl Szilárd. Énektudást fejlesztő alkalmazás, 2015. BSc Szakdolgozat, BME-MIT.

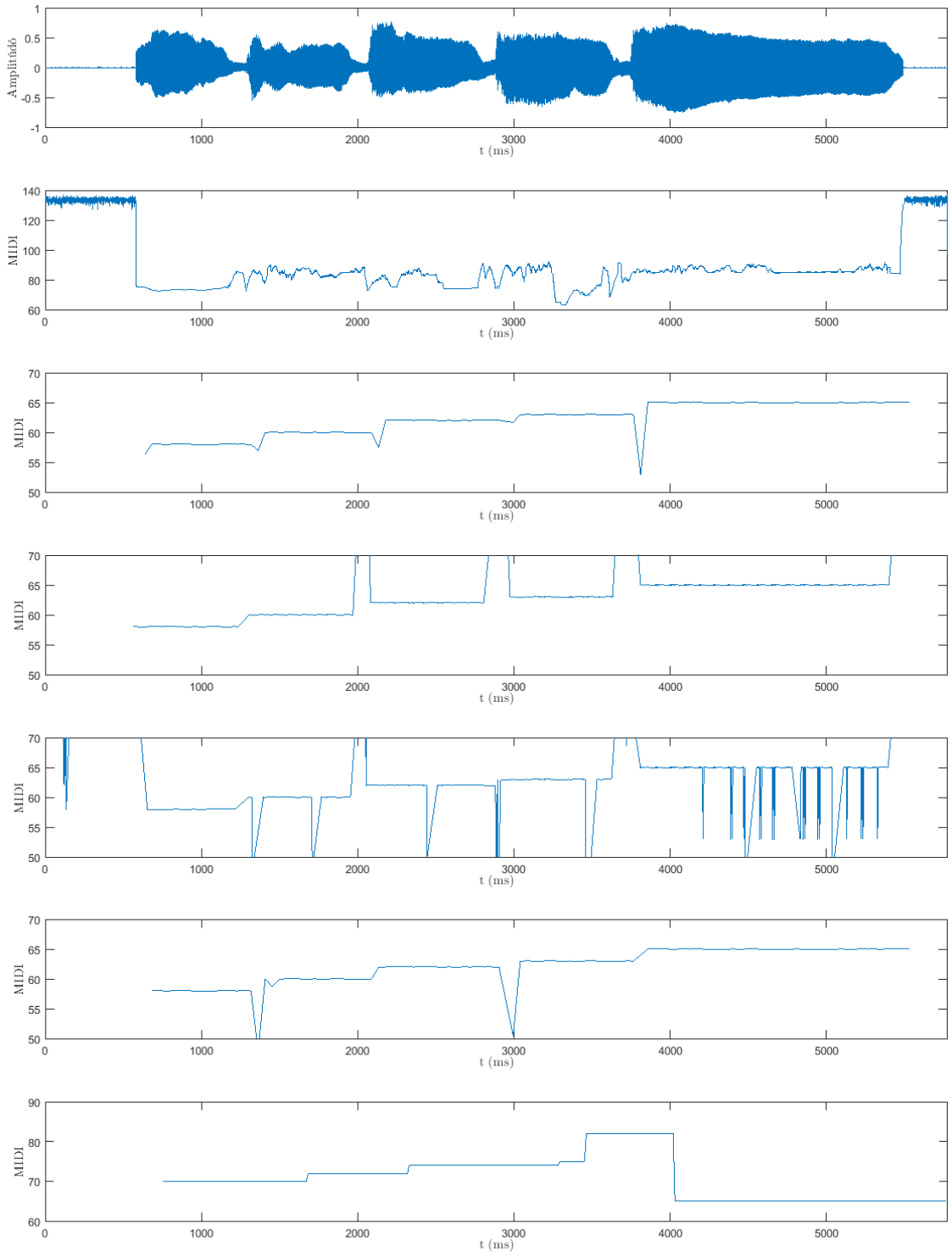
[17] Joe Wolfe. Note names, midi numbers and frequencies. https://newt.phys.unsw.edu.au/jw/notes.html.

Függelék

Időtartománybeli függvények kimenőjele

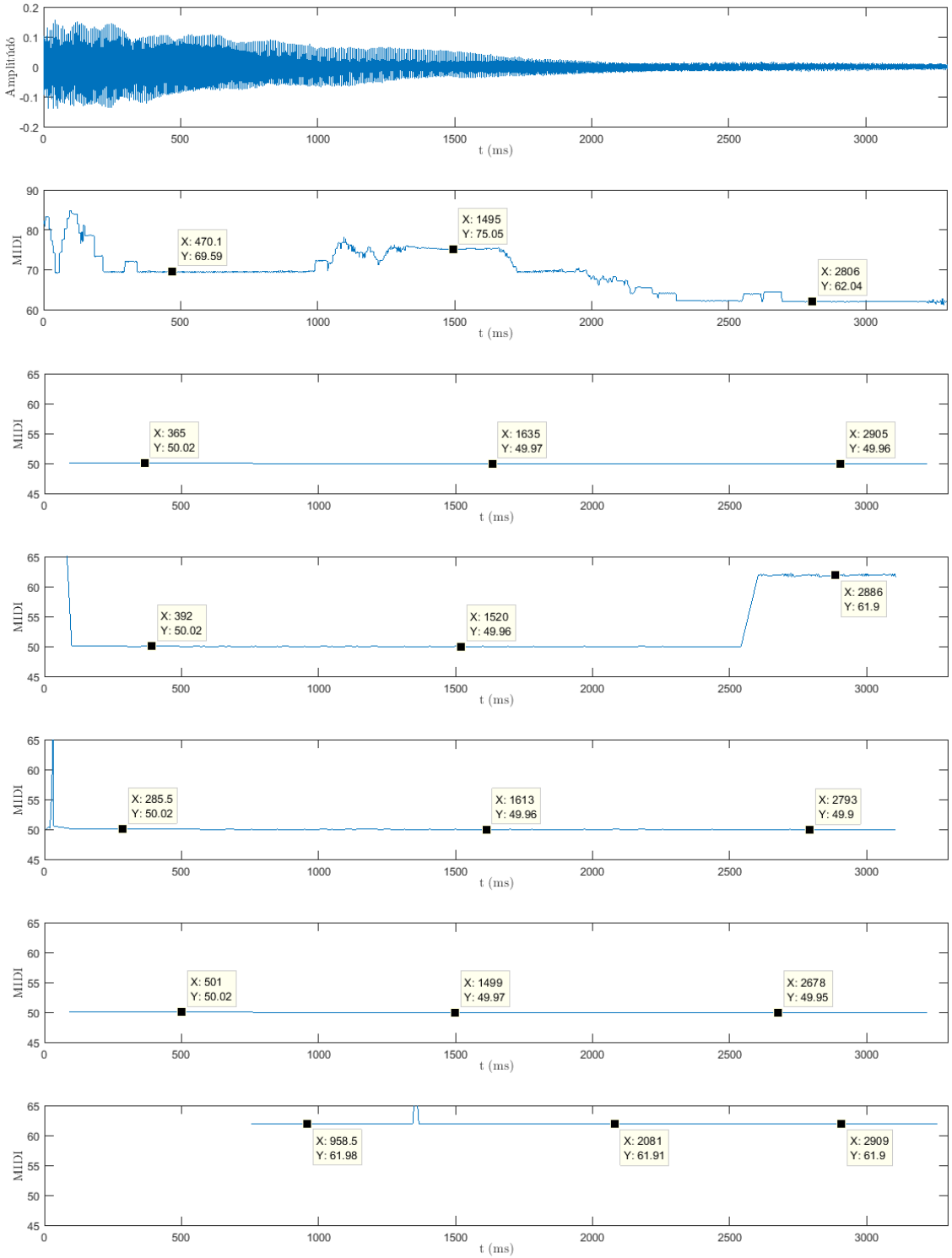


F.0.1. ábra. *Egy klasszikus gitár B húrjának időfüggvénye ACF, SDF, AMDF és SNAC előtt illetve után*



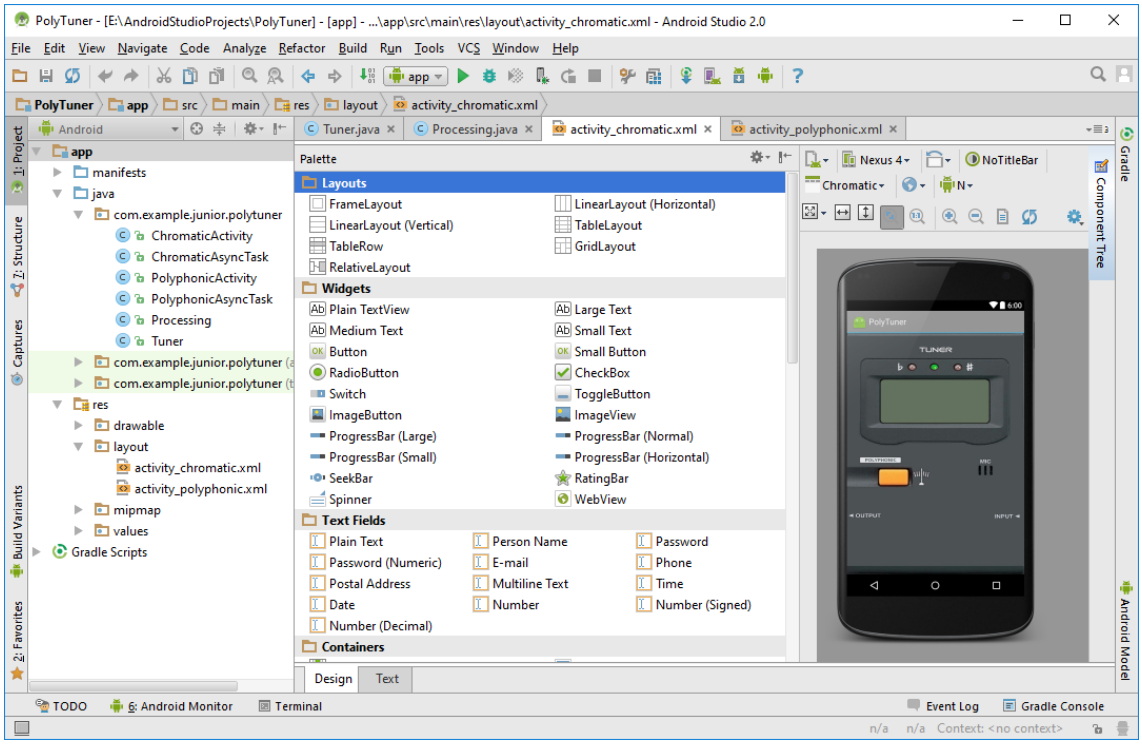
F.0.2. ábra. *Egy trombita C-D-E-F-G (MIDI: 58-60-62-63-65) hangjainak felismerése ZCR, ACF, SDF, AMDF, SNAC és HPS segítségével*

Eljárások összehasonlítása



F.0.3. ábra. *Egy elektromos gitár D (MIDI: 50) húrjának felismerése ZCR, ACF, SDF, AMDF, SNAC és HPS segítségével*

Android Studio



F.0.4. ábra. *Az Android Studio UI-tervező modulja*