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

Horváth Gergely

RITMUSÉRZÉK-FEJLESZTŐ ALKALMAZÁS FEJLESZTÉSE

KONZULENS

Dr. Bank Balázs

BUDAPEST, 2017

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott **Horváth Gergely**, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot/ diplomatervet 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 hitelesített 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.

Kelt: Budapest, 2017. 12. 09.

...…….
Horváth Gergely

Összefoglaló

Kevésbé tapasztalt zenészek gyakori hiányossága, hogy a zenélés során rossz tempóval játszanak vagy a tempót nem képesek egyenletesen tartani. Ennek szerepe a dobosok esetében a legjelentősebb. Ahhoz, hogy ezt a hiányosságot korrigálják, az egyik lehetőség a metronóm használata, ami egy előre beállított tempó alapján határozza meg az ütések ideális helyét. A dolgozatom célja egy olyan alkalmazás fejlesztése volt, amely visszajelzést ad a felhasználóknak a játszott zene tempójáról és képes annak folyamatos követésére. Az általam elkészített alkalmazás a számítógép által vett hang alapján valósítja meg a tempó kijelzését. Emellett alkalmas a metronóm tempójától való eltérés visszajelzésére is.

A dolgozatom során megismerkedtem a digitális jelfeldolgozás lépéseivel és egy olyan algoritmust fejlesztettem ki, ami alkalmas a zene tempójának pontos kiszámítására. Az irodalomkutatás alkalmával meglévő tempófelismerő eljárásokat tanulmányoztam. A saját algoritmusomat ezek felhasználásával, igyekeztem kifejleszteni és alkalmassá tenni a probléma megoldására. A tervezés során több lehetőséget is kipróbáltam és összehasonlítottam ahhoz, hogy megtaláljam az ideális megoldást.

Az algoritmusok összehasonlításához Matlabot használtam, és előre rögzített hangfelvételek segítségével állapítottam meg az eljárások hatékonyságát. A legjobbnak ítélt megoldást végül C++ nyelven implementáltam, JUCE fejlesztőkörnyezet felhasználásával. Az elkészült grafikus program önállóan futtatható. Az alkalmazásom eredményeit végül összehasonlítottam a Matlabban kapott eredményekkel, így meggyőződve annak helyes működéséről.

Az elkészült szoftver több meglévő megoldást foglal magába. A tempófelismerő funkció más alkalmazások által elérhető mobiltelefonokra és számítógépekre. Az ütemen belüli pontosság kijelzéséhez pedig, eddig elektromos dobok használatára volt szükség. Az elkészített munkám célja volt, hogy mindkét funkciót egyben megvalósítsa és elérhetővé tegye azokat akusztikus hangszeren játszók számára is.

Abstract

A common deficiency of beginner musician is, that they play with incorrect tempo or they can't keep a constant tempo. The role of this skill is especially important for drummers. To correct this deficiency one option is to use a metronome which is gives back beats with a fix tempo. The goal of my thesis was to develop a software which can provide feedback for the user about the currently played tempo. The application also follows the tempo changes. My software is using the audio input of the computer to calculate the tempo and give this information to the user. Furthermore it is able to provide information about the difference between the metronome beats and the played beats.

In connection with my thesis, I have learned a lot about digital signal processing and I have developed an algorithm which can calculate the accurate tempo of a music. During my literature research I have learned about available BPM (Beat Per Minute) detection algorithms. I have used this knowledge for the development of my own algorith. I have tried out various algorithms and I have compared these solutions during the development.

For the comparison I have used Matlab and I have tested the system with recorded sounds to measure its precision. Finally, I have implemented the best solution in C++ and I have used the JUCE development tool for it. The final programme has a graphical user interface and can be executed independently. I have also compared the application with the results of the Matlab algorithm, so I was able to check its correctness.

The application includes different features that were formerly available in separate devices. The tempo detection function has been implemented on mobile phones and also on computers before. Some electronic drum kits are able to give information about the time difference between the played beats and the metronome beats. My goal with my work was to make a software which is includes both functions and can be used for practice with acoustic musical instruments.

1. Bevezetés

A feladat egy olyan ritmusérzék-fejlesztő alkalmazás megvalósítása, amely futtatható általános számítógépen és a bemenő jelet a géphez csatlakoztatott mikrofonból nyeri. Az alkalmazás nagy segítséget nyújthat a zenészeknek a tempótartás, illetve tempóváltás gyakorlásában és elérhetővé teszi ezt a funkciót az akusztikus hangszereken zenélők számára is.

1.1 Meglévő megoldások

A fejezetben bemutatásra kerülnek a dolgozat témájához hasonló meglévő programok. Ezek a megoldások összehasonlítási alapként szolgáltathatnak a munkám során.

BeatSeeker:

A Max for Live az Ableton cég terméke, ami tartalmaz a dolgozat témájához hasonló funkciót, a neve BeatSeeker [Andrew Robertson – 2015]. A program előnye, hogy a zenésznek nem metronómra kell játszania, hanem a gép végzi a tempó felismerését. A BeatSeeker alkalmas dob és egyéb ritmikus játék elemzésére. A funkciót arra használják a Max for Live alkalmazásban, hogy a playback hangok lejátszásának tempóját az élő zene tempójához igazítsák. A zenész jelezheti az alkalmazásnak, ha játéka tempója nem állandó, ekkor a program a korábban játszott zene tempóját veszi alapul a további működése során.

1. ábra: BeatSeeker alkalmazás felülete

BPM-Detector:

A Sandberg Sound cég terméke a BPM-Detector egy Androidos eszközökre elérhető alkalmazás [SandergSound - 2013]. Ez egy egyszerű BPM (Beat Per Minute / Percenkénti ütésszám) számláló szoftver.

2. ábra: BPM-Detector felülete

Az applikációt elindítva a 2. ábrán látható kép fogadja a felhasználót. Az eszköz mikrofonján vett hang tempóját jeleníti meg. A felső *Range* gombbal kiválasztható, hogy a kijelzett tempó mely tartományba essen.

1.2 Dolgozat célja

A dolgozat célja egy olyan program elkészítése, amely Windows operációs rendszeren futtatható. Kimenetként a bemenő hang tempóját jelzi ki. Az alkalmazás két elérhető funkciót biztosít, az egyik a tempó automatikus felismerése és kijelzése, a másik az ütések összehasonlítása egy előre beállított metronómjellel.

Tempódetektálás:

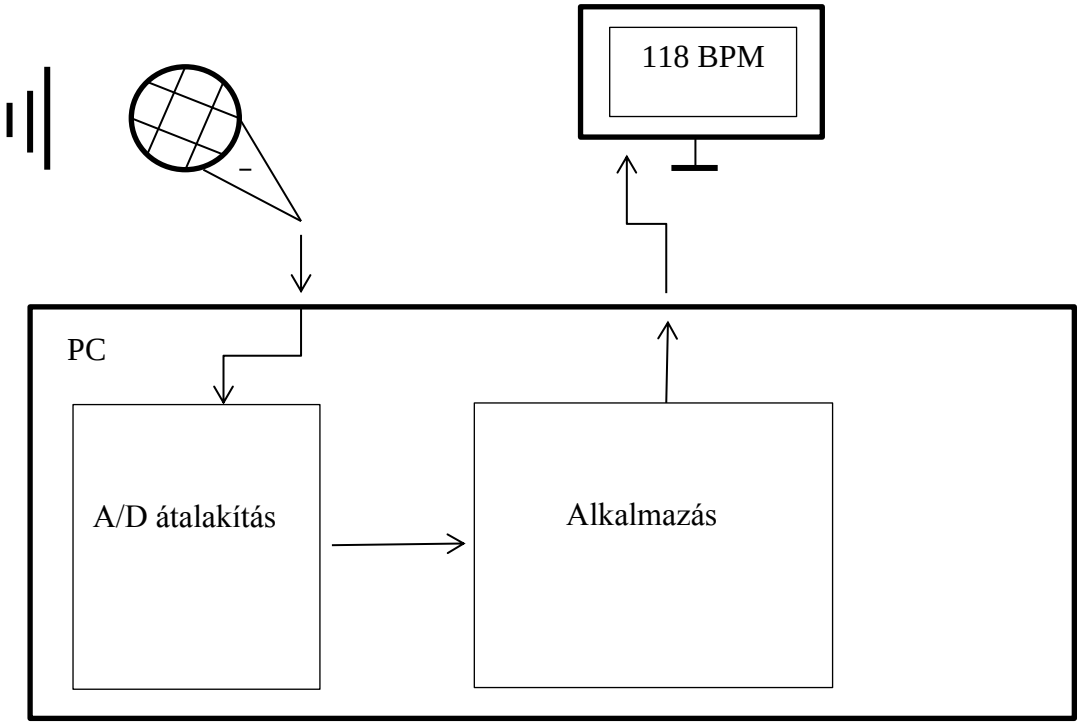
A funkció célja, hogy egy szabadon játszott zenéről képes legyen megállapítani annak tempóját és ezt a tempót élőben visszajelezze a program felhasználója felé. Adott esetekben képes kell, hogy legyen a tempó megállapítására több hangszer együttes játéka esetén, így használhatóvá válik, akár zenekari próbákon is.

A feldolgozásnak rövidebb idő alatt kell végbemennie, mint a feldolgozott hang hossza, ahhoz hogy a tempó valós időben visszajelezhető legyen. A kijelzés frissítési ideje fejlesztői döntés, a tervezés során tapasztaltak alapján, mindenesetre feltételezhető, hogy a tempó nem változik néhány másodpercen belül jelentős mértékben. A megvalósítás bármilyen számítógépen implementálható megoldással történhet, de a cél, hogy minél stabilabb visszajelzést kapjon a felhasználó és egyszerűen tudja használni a programot. Ideális esetben a megoldás képes a tempóváltozások követésére, illetve szüneteltetett játék után azt folytatva az aktuális tempó újbóli kijelzésére.

Tempókövetés:

A funkció célja, hogy a felhasználó visszajelzést kapjon arról, hogy éppen siet vagy késik egy előre beállított tempóértékhez képest. Ez nagy segítséget nyújt, akár színpadi játék, akár zenekari próbák során, ahol egyértelmű és gyors visszacsatolásra van szükség.

A metronóm tempóját előre be kell állítani. A használat során az alkalmazás visszajelzést ad a metronómtól való eltérés irányáról és mértékéről. Feltételezhető, hogy ahogyan az előző funkció esetében is, a zenész egy-két másodpercen belül nem siet vagy késik jelentősen. A visszajelzés frissítési ideje a fejlesztés során tapasztaltaknak megfelelően előre beállított.



3. ábra: Alkalmazás absztrakt modellje

A zenészeknek korábban ahhoz, hogy ezeket a funkciókat élvezhessék, elektronikus eszközöket kellett vásárolniuk. Vannak olyan elektromos dobok a piacon, amelyek tartalmaznak ritmusérzék-fejlesztő funkciót, ezzel segítve a gyakorlást. Az általam implementált megoldás ezzel szemben elérhetővé teszi a tempó folyamatos figyelését akusztikus hangszereken játszó zenészek számára is. A meglévő megoldások egy része fizetős, mint az 1.1-es fejezetben ismertetett BeatSeeker, vagy túl lassan követik a tempóváltozást, ahogyan például BPM-Detector. Ezért a dolgozatom célja egy olyan alkalmazás fejlesztése, ami több funkcióval szolgálja ki a zenészeket, minél pontosabb visszajelzést ad és mindenki számára ingyenesen elérhető.

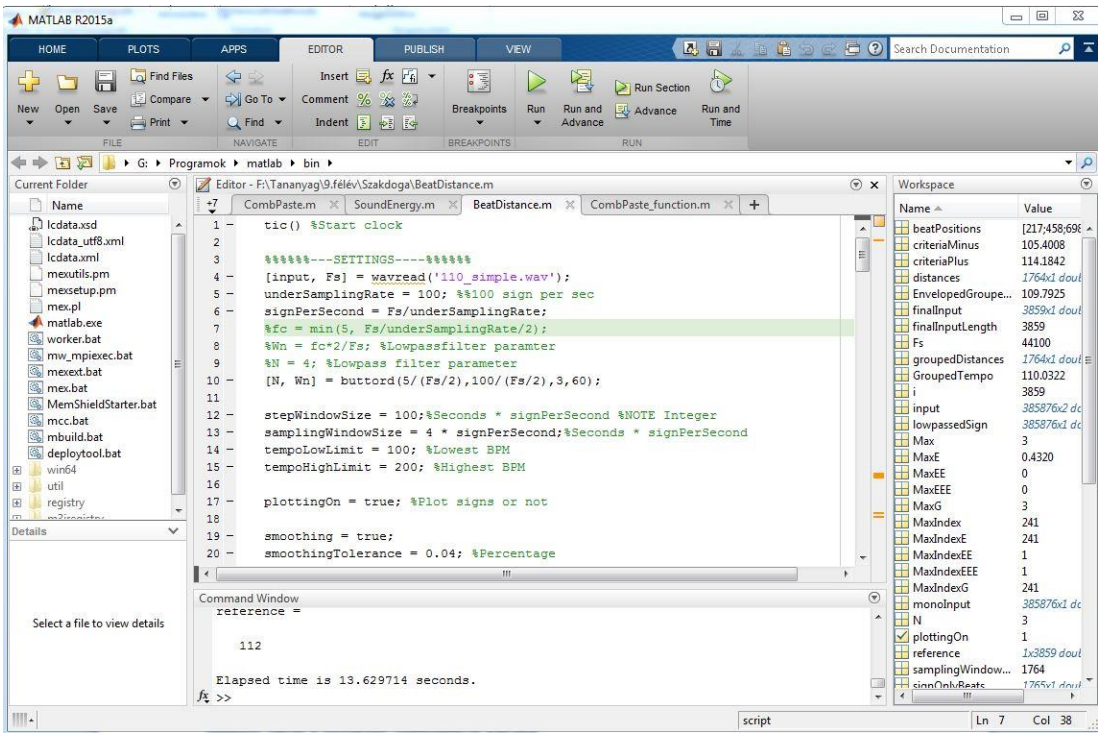
2. Választott eszközök

2.1 Matlab

Az eljárások kifejlesztésére és tesztelésére használt eszköz a Matlab, ami lehetőséget biztosít különböző matematikai műveletek elvégzésére és tervezési eljárások megvalósítására egy egyedi programozási nyelv használatával.

Előnyei, hogy számos eszköz áll rendelkezésre, amely meggyorsítja a tervezést, ilyenek a különböző szűrőtervezők, vagy az előre implementált matematikai függvények. Tökéletes eszköz az eredményének szemléltetésére, rendkívül rugalmas és könnyen használható ábrázolási eljárásokat tartalmaz. A dolgozatban található grafikonok jelentős része ezzel a programmal készült, melynek felülete a 4. ábrán látható.

Nagy segítség a használata során a körülötte kialakult nagy létszámú és nagy tapasztalattal rendelkező közösség, ez jelentősen megkönnyíti a megoldás megtalálását egy-egy adott problémára, továbbá az alkalmazás funkcióinak bővülése folyamatos.



4. ábra: Matlab felülete

Használt verzió: R2015a

2.2 JUCE

A JUCE egy zenei alkalmazások fejlesztését támogató nyílt forráskódú keretrendszer. Asztali és mobil platformokra történő fejlesztésre egyaránt használható. Főként a grafikus interfészei és audio plug-in-jai miatt használják. A célja, hogy ugyanazt a programkódot fel lehessen használni Windows, Mac OS és Linux operációs rendszerekre történő fejlesztés során is. Segítségével összetett alkalmazások fejleszthetőek egyedi igények szerint, továbbá támogatja zeneszerkesztő programokban használható audio pluginok fejlesztését is.

Amiket rendelkezésre bocsát:

1. Könnyen szerkeszthető felhasználói felület

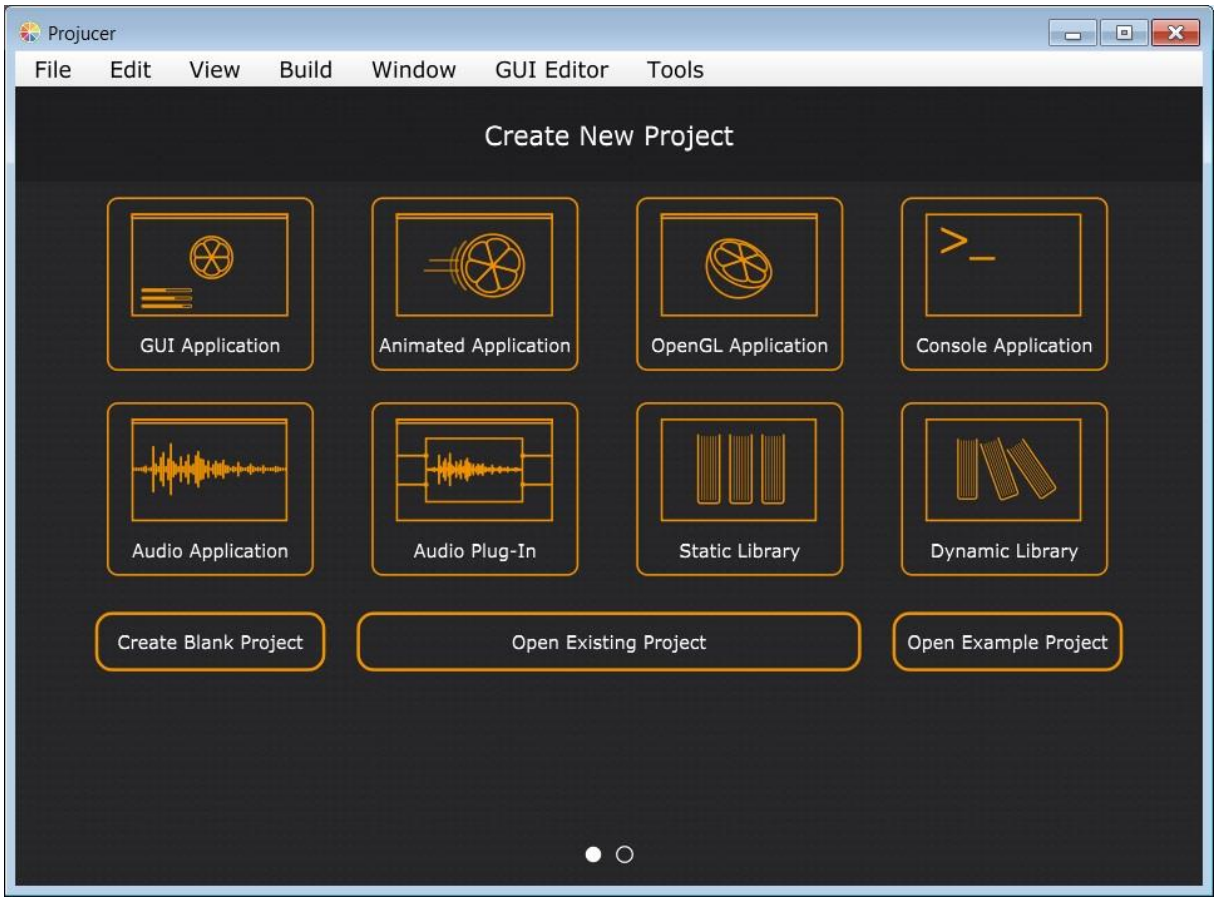
2. Egyedi fordító és futtató környezet

3. Plugin fejlesztési lehetőség

4. Széleskörű formátumtámogatás

A fejlesztés nyelve C++, melyhez bármilyen IDE használható úgy, mint Microsoft Visual Studio, Xcode vagy Code::Blocks, stb. Az újabb verzió lehetőséget biztosít mobil platformokra való fejlesztésre is.

Használatával elérhetővé válnak az úgynevezett JUCE modulok, amelyek különböző előre implementált osztályok és programrészletek. Ezek között megtalálhatóak grafikus elemek, például grafikonok és gombok, emellett széles körű audio csomagot biztosít. Az audio csomagok segítségével könnyen kezelhető, a számítógép hangkártyája, illetve egyszerűen olvashatóak különböző audio fájltípusok. Továbbá támogatja a digitális szűrőtervezési eljárásokat is.



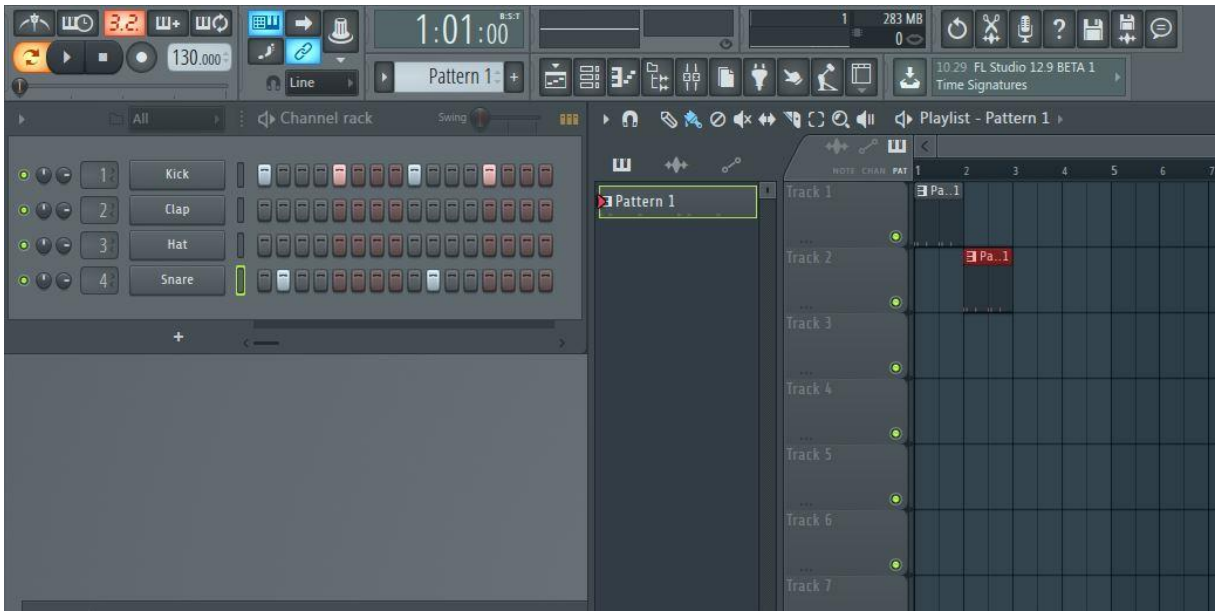
5. ábra: JUCE felülete

Használt verzió: JUCE 5.2

Megjegyzés: Az alkalmazás telepítése során ügyelni kell az elvárt szoftverek megfelelő állapotára és verziójára. A tapasztalataim szerint ezt a verziót célszerű a Microsoft Visual Studio 2017-es változatával használni, amely a legújabb C++ könyvtárakat tartalmazza. Továbbá Windowson elvárt az Internet Explorer, ugyanis a JUCE ezt használja és, ha nem elérhető, nem ad visszajelzést hiba esetén.

2.3 FL Studio 12

Az FL Studio egy digitális audio munkaállomás. A dolgozat készítése során a referenciajelek előállítására használtam. Segítségével könnyen generálható bármilyen zenerészlet és beállítható annak tempója.

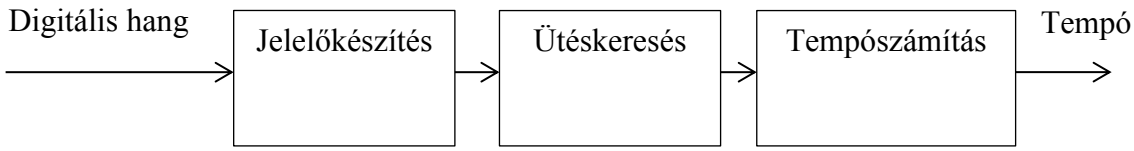


6. ábra: FL Studio 12 felülete

4. Tempó- és ütemdetektálás Matlabban

Először az algoritmusokat Matlabban implementáltam és hasonlítottam össze. Ebben a fejezetben ismertetem az algoritmusok részletes működését és az összehasonlítás eredményeit.

4.1 Algoritmusok



7. ábra: Az algoritmus általános blokkvázlata

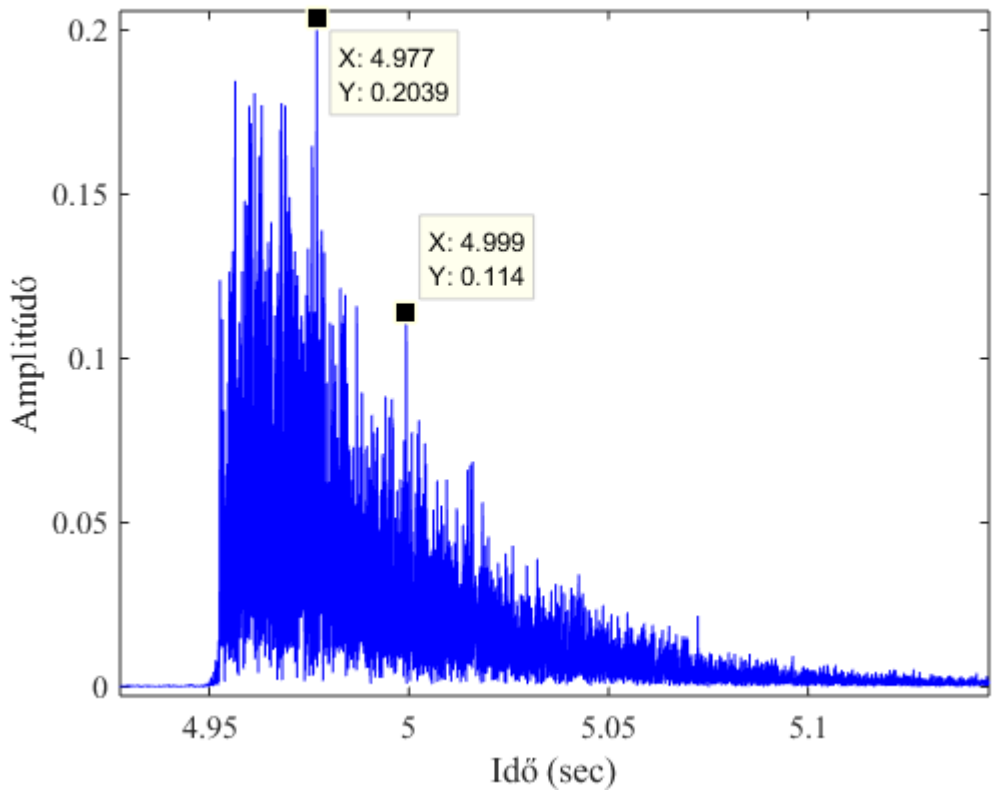
A 7. ábrán a bemenő jel *y* egy egycsatornás jel, ami a következőképpen áll elő az eredeti kétcsatornás jelből *x*:

$$y = \frac{(|x(right)| + |x(left)|)}{2}$$

Átlagolásra kerül a két csatorna egyenirányított jele, ügyelve arra, hogy a csatornák jelei ne oltsák ki egymást akkor sem, ha ellentétes fázisban érkeznek be.

4.1.1 Jelelőkészítés és szűrőválasztás

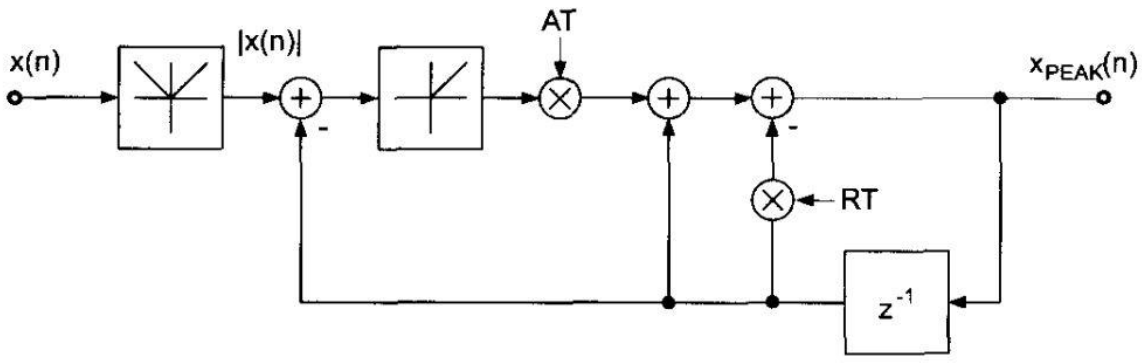
A jel előkészítésére azért van szükség, mert egy ütés sok amplitúdóváltozást tartalmaz mind a felfutás, mind a lecsengés során, azaz egy ütés több csúcsot is tartalmazhat. A 8. ábrán ez figyelhető meg. Az ilyen gyors változásokkal teli jelet nehéz feldolgozni, ezért a cél egy simább jel kinyerése. Ennek érdekében az alábbi módszereket vizsgáltam meg. [Balázs -2013]



8. ábra: A bemenő jel abszolútértéke

Burkológörbe-generálás:

A burkológörbe a hanghullám maximális amplitúdójának időbeli lefolyását leíró görbe. Az általam választott burkológörbe generálási eljárás blokkvázlata a 9. ábrán látható. [Szigetvári - 2014]



9. ábra: Burkológörbe előállításának blokkvázlata [Zölzer - 2011]

Az algoritmus két beállítható paraméterrel rendelkezik, ezek az *AT* (Attack Time) felfutási idő, valamint az *RT* (Release Time) lecsengési, elengedési idő. Ezen

paraméterek beállításával határozható meg, hogy az előállított burkológörbe milyen mértékben kövesse az eredeti jel változásait a felfutás és a lecsengés során.

A működés lebontva a következő:

1. A burkoló első jele $x_{PEAK}(1)$ megegyezik az eredeti jelben találhatóval $x(1)$.

2. Ha a soron következő bemeneti jel értéke $x(i)$ nagyobb a burkoló előző indexű jelénél $x_{PEAK(i-1)}$. A növekmény az AT paraméterrel korrigálva hozzáadásra kerül a burkoló előző értékének (1-RT)-vel korrigált értékéhez.

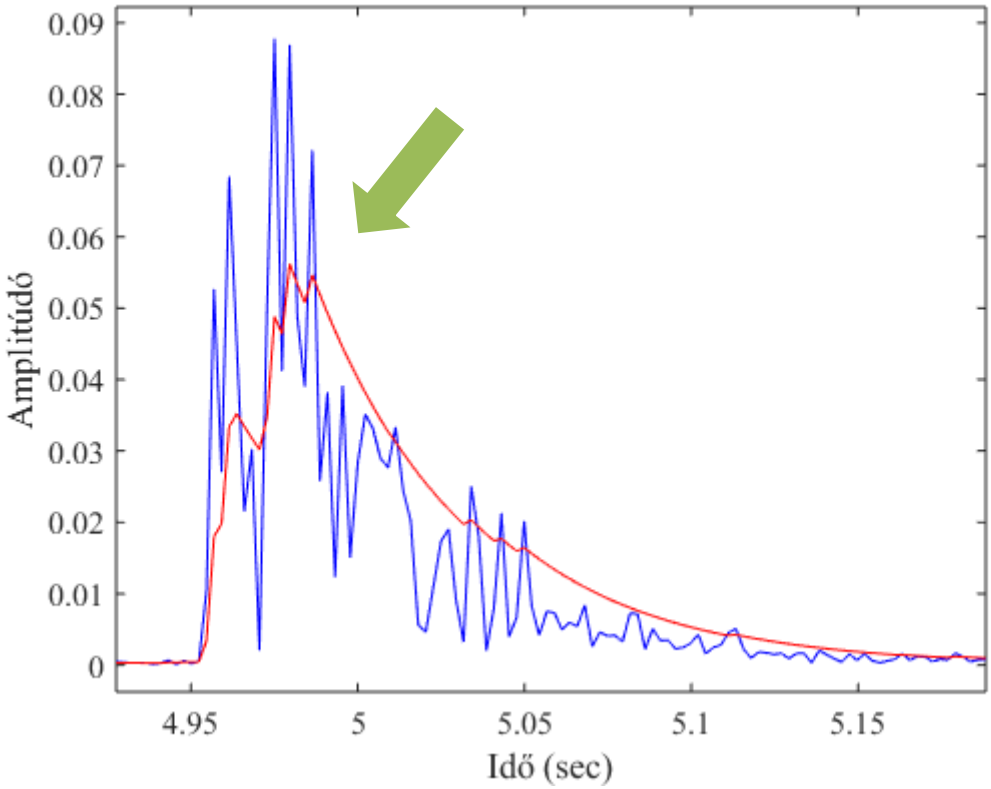
$$x_{PEAK}(i) = (x_{PEAK}(i) - x_{PEAK}(i-1)) * AT + x_{PEAK}(i-1) * (1 - RT)$$

Ha a soron következő bemeneti jel értéke $x(i)$ kisebb a burkoló előző indexű jelének nagyságánál $x_{PEAK(i-1)}$. A burkoló aktuális értéke az előző jel RT-vel korrigált értéke.

$$x_{PEAK}(i) = (1 - RT) * x_{PEAK}(i-1)$$

Ez a lépés a bemeneti jel végéig ismétlődik.

A leírt működés nyomonkövethető a 9. ábrán, a következő 10. ábra pedig a művelet eredményét demonstrálja.



10. ábra: A jelre illesztett burkológörbe eredménye (*AT*=0.3, *RT*=0.05) (kék: feldolgozandó jel, piros: burkológörbe)

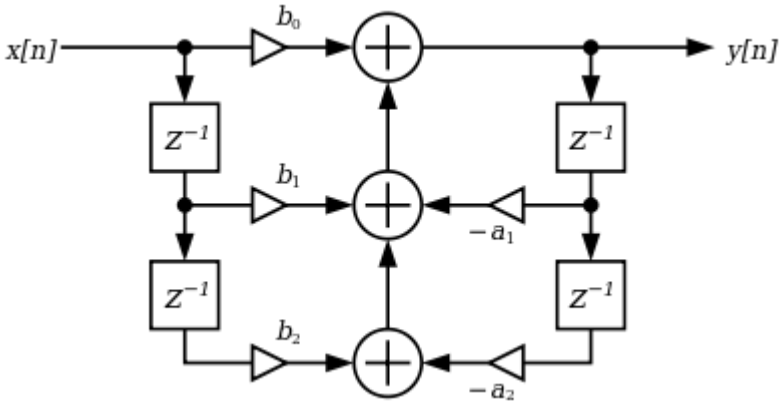
Az ábrán az eredeti jel kékkel látható, a pirossal ábrázolt jel pedig a burkológörbét jeleníti meg. Megfigyelhető, hogy továbbra is több csúcsot tartalmaz az ütés burkolójának jele, ezeket az amplitúdó változásokat jelöli 10. ábrán látható zöld nyíl, de elmondható, hogy az előállított jel közelíti az általam elérni kívánt alakot.

Aluláteresztő szűrés:

Az ideális szűrők lehetővé teszik egy adott frekvenciasáv veszteségmentes áteresztését, míg a nem kívánt frekvenciatartomány jeleit teljes mértékben elnyomják. Az ideális szűrőt nem lehet megvalósítani, ezért a cél a minél jobb közelítés megtalálása. Ezért esett a választásom a Butterworth szűrőre. [Lipovszki - 2012]

Széles körben használt szűrőtervezési eljárás a Butterwoth-féle eljárás. Az *N* paraméter a szűrő fokszámát határozza meg, a *Wn* pedig a törésponti frekvenciát. Minél magasabb a szűrő rendje, annál élesebb a vágás. A Butterwoth szűrőket széles körben alkalmazzák, ugyanis igen könnyen méretezhetőek. Előnyük a simaságuk és a monoton

csökkenő frekvenciamenetük. A magasabb fokú szűrők megközelítik az ideális aluláteresztő szűrő frekvenciafüggvényét.



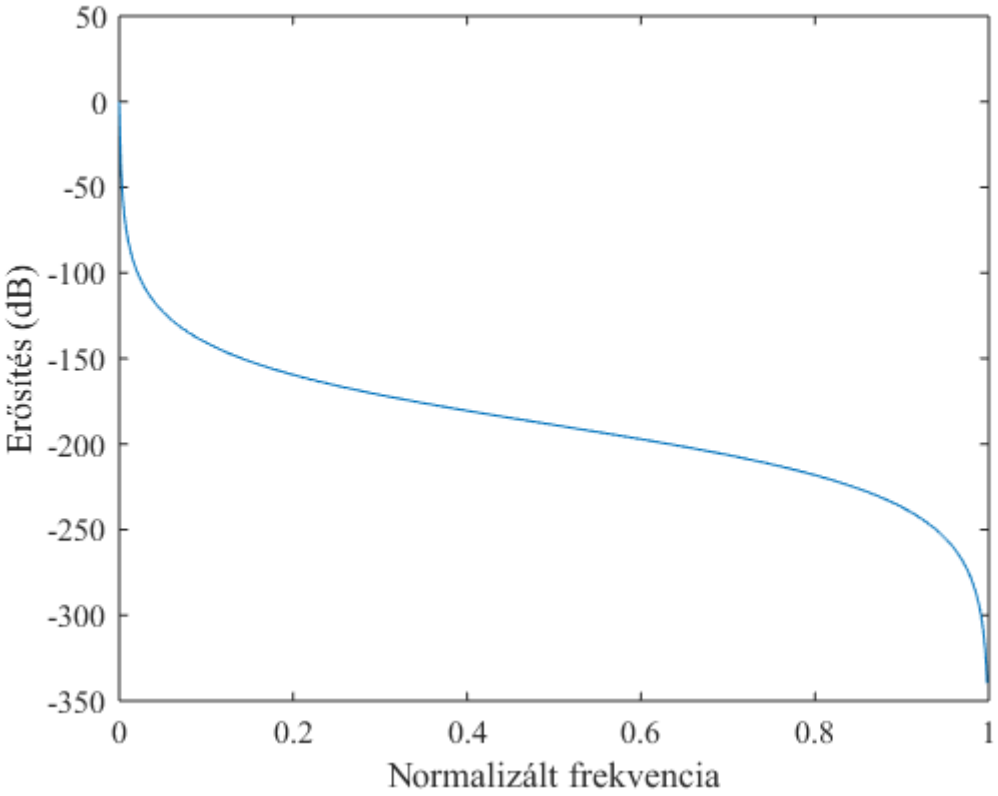
11. ábra Szűrő blokkvázlata [Lipovszki - 2012]

Ez a szűrőtípus az IIR (Infinite Impulse Response = Végtelen impulzusválaszú) szűrők családjába tartozik, egy ilyen szűrő blokkvázlata látható a 11. ábrán. Az *x* a bemenő jel, az *y* pedig a kimenő jel. Az *a* és *b* változók a szűrőhöz tartozó koefficiensek, amelyek függenek a szűrő rendjétől *N*, a töréspontí frekvenciától *Wn* és a szűrő típusától, ami az esetemben *lowpass*, azaz aluláteresztő. A visszacsatolt rendszer matematikai alakját a következő egyenletrendszer reprezentálja [Kabal - 2004]:

$$A = a[i] * y[i-1] + \cdots + a[N] * y[i-N]$$

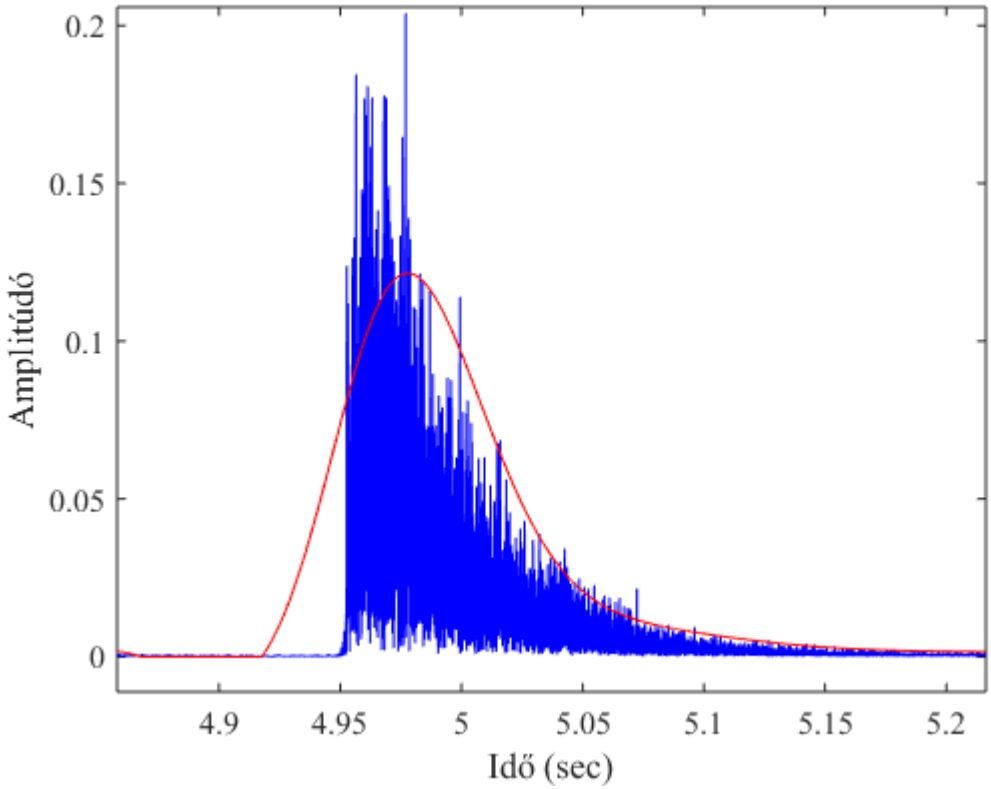
$$B = b[0] * x[i] + b[1] * x[i-1] + \cdots + b[N] * x[i-N]$$

$$y[i] = \frac{1}{a[0]} * (B - A)$$



12. ábra: A tervezés során használt szűrő frekvenciaválasza (*N=3, Wn = 4.5e-4*)

Az algoritmusok tervezése során használt Butterworth aluláteresztő szűrő frekvenciaválasza a 12. ábrán látható.



13. ábra Az aluláteresztő szűrés hatása (kék: eredeti jel, piros: aluláteresztő szűrés eredménye)

A 13. ábrán kékkel az eredeti jel, míg pirossal a szűrt jel figyelhető meg. Az ábrán látható eredmény kétirányú szűrést reprezentál, így korrigálva a szűrés során fellépő fázistolást. A tempószámítás során nem szükséges a keletkezett fázistolást korrigálni, mert ez az ütések egymáshoz képesti viszonyát nem befolyásolja. [Fodor - 2014]

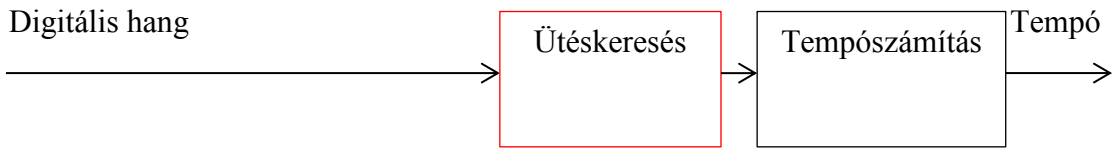
Összefoglalás:

Megállapítható, hogy a *Burkológörbe-generálás* egyedüli alkalmazása nem elegendő a kívánt jel előállításához, de az így előállt jelen alkalmazott *Aluláteresztő szűrés* már alkalmassá teszi a jelet a feldolgozásra. Azonban az *Aluláteresztő szűrés* közvetlenül az eredeti jelen alkalmazva elégséges a feladat ellátására, ezért a tervezés során ezt a módszert választottam a jel előfeldolgozásához.

4.1.2 Ütéskeresés

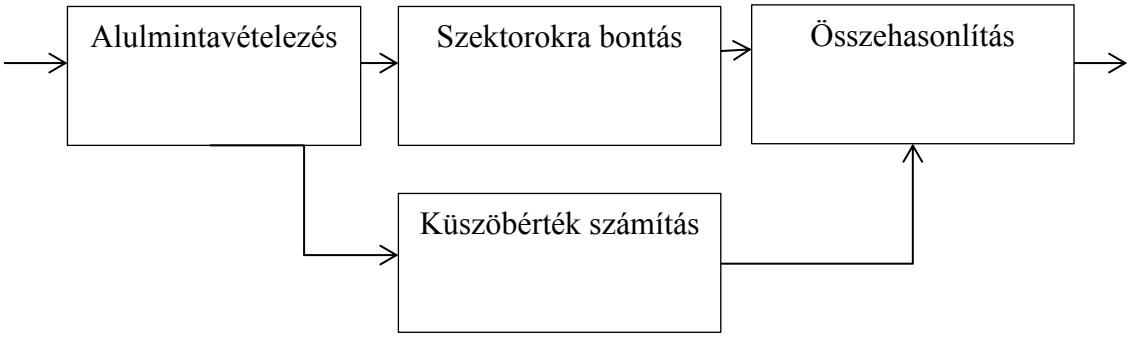
A jel előfeldolgozása után az ütések megkereséséhez az alábbi lehetőségeket vizsgáltam meg.

Hangenergia alapján működő algoritmus:



14. ábra: Hangenergia alapján működő algoritmus blokkvázlata

Ezen ütemkeresési eljárás során a 4.1.1-es fejezetben ismertetett jelelőkészítés lépése elhagyható és a digitális hangon azonnal végrehajtható az úgynevezett ütéskeresés. Az így előállt algoritmus lépései a 14. ábrán láthatóak.



15. ábra: Hangenergia alapján történő ütéskeresés lépései

A 15. ábrán az ütéskeresés lépései követhetőek, amelyek a következők:

1. Alulmintavételezés az *UndersamplingRate* bemeneti paraméter alapján. A kimenet: y_{us}.

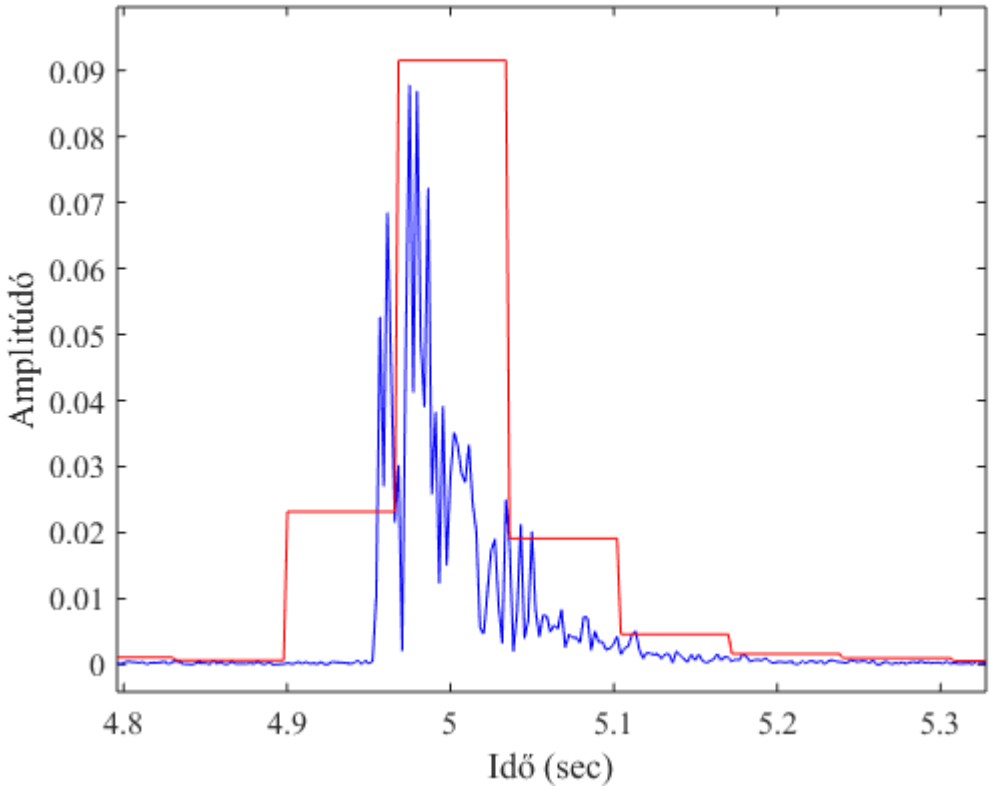
2. A már alulmintavételezett jelen végighaladás és a szektorok összenergiájának meghatározása. (*SS*=*SectorSize*)

$$y_s(i) = \frac{\sum_{j=i*SS}^{(i+1)*SS} y_{us(j)}}{SS}$$

3. Küszöbérték meghatározása a teljes jel összenergiája alapján vagy a jel egy részének összenergiája alapján.

$$c = \frac{\sum_{j=0}^{length(y_S)} y_S}{length(y_S)}$$

4. Az előző lépések után, ahol az y_s aktuális értéke nagyobb a küszöbértéknél c-nél, ott ütés van. [Ziccardi - 2015], [Frédéric - 2003]



16. ábra: Szektorok összenergiája (kék: eredeti jel, piros: szektorok energiája)

A 16. ábrán a piros jel reprezentálja a szektorok összenergiáját. Megjegyzendő, hogy a módszer a szektorokra osztás miatt veszít a pontosságából a szektorok szélességétől függően.

Lokális maximumkeresés:

A lokális maximumkeresés módszeréhez szükség van 4.1.1-es fejezetben ismertetett jelelőkészítésre. Miután a jel előkészítése megtörtént a cél az ütések minél pontosabb megtalálása. Mivel az előkészítés után egy ütés mintája egy csúcsot tartalmaz, ezért alkalmazható a maximumkeresés módszere.

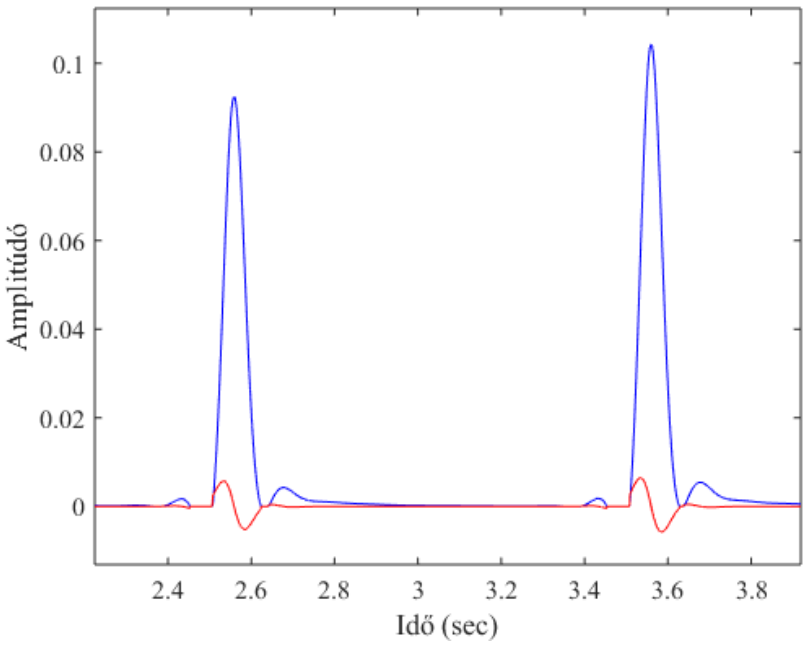
Lépések:

1. Alulmintavételezés az *UndersamplingRate* bemeneti paraméter alapján. A kimenet: y_{us}.

2. Maximumkeresés.

$$y_{us}(i-1) < y_{us}(i) > y_{us}(i+1)$$

A módszer használatával pontosan megállapíthatóak az ütések helyei a jelben. Erre a célra alkalmas emellett a jel deriváltja is, ami ideális esetben az ütések helyén vált előjelet, ahogyan ez a 17. ábrán látható, azonban a maximumkeresés módszere egyszerűbben implementálható és a számításigénye is alacsonyabb. Ezért az ütéskereséshez a tervezés során a maximumkeresést használtam.

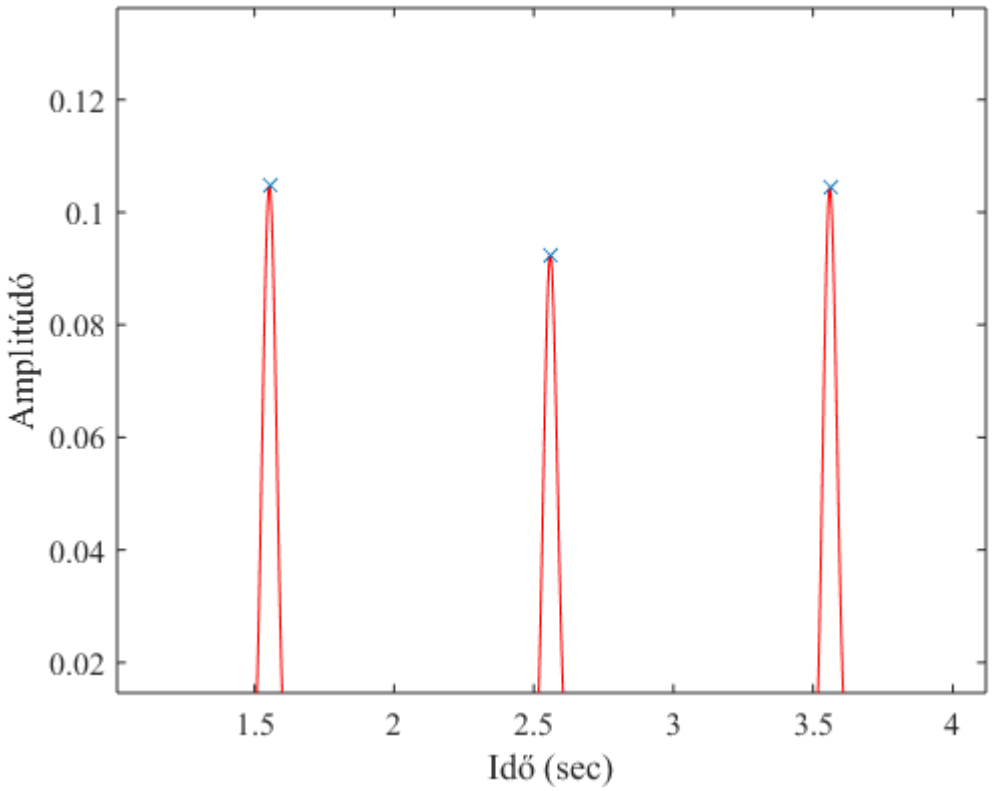


17. ábra: A jel deriváltja (kék: aluláteresztő szűrés eredménye, piros: a jel deriváltja)

4.1.3 Tempószámítás

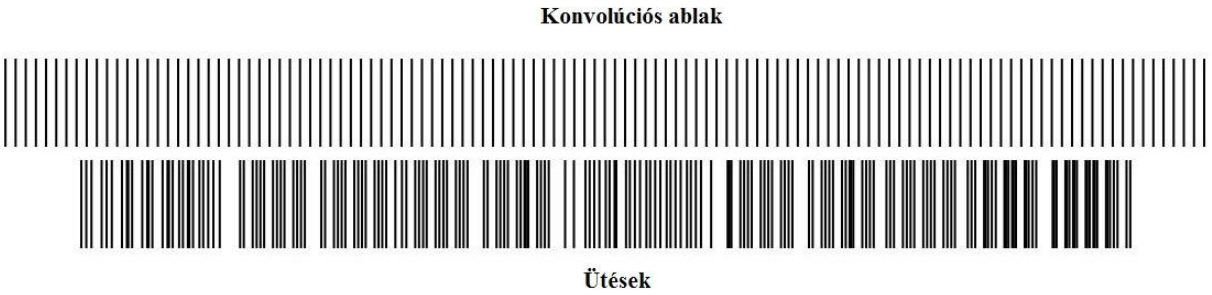
A ütések helyeinek meghatározása után, ezek alapján a tempószámítás lépése következik.

Ablakillesztéses módszer



18. ábra: Detektált ütések *x*-el jelölve (piros: aluláteresztő szűrés eredménye, kék: ütések helye)

A kiindulási állapotban rendelkezésre állnak a jelben lévő ütések és azok helyei, ahogyan a 18. ábrán is látható. A tempó kiszámításának ötlete [Joe - 2014] cikk szerint a következő: egy adott tempóérték alapján meghatározható egy úgynevezett konvolúciós ablak. A módszer lényege, hogy ezt az ablakot próbálja ráilleszteni az ütésekre. Az illesztést különböző tempóértékek által meghatározott ablakokkal elvégezve és az illesztések eredményeit összehasonlítva, kiválasztható a legjobban illeszkedő ablak, amelyből visszaszámolható a tempó.



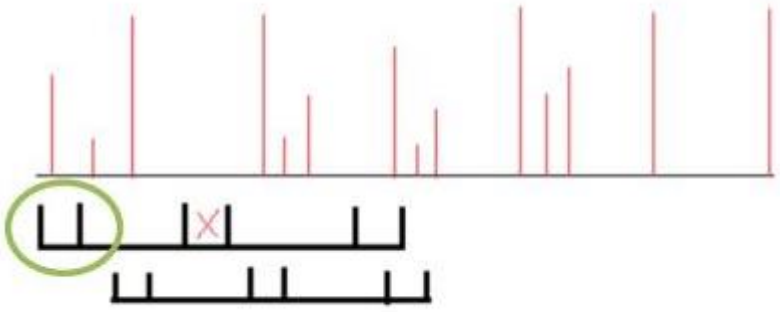
19. ábra Konvolúciós ablak illesztése [Joe - 2014]

A 19. ábra felső részén látható a konvolúciós ablak, amelyet a módszer a teljes jelre illeszt. Egy illesztés annál sikeresebb, minél több ütés esik egybe az ablak jeleivel.

Az általam használt eljárás kismértékben eltér az előzőekben ismertetettől. Az eltérés, hogy nem a teljes jelre próbál illesztést végrehajtani az algoritmus, hanem egyszerre csak a soron következő három ütésre, így könnyebben feldolgozható a folytonosan beérkező jel.

Lépések:

1. Az ütésekre a megadott *DemoTempo* paraméter alapján egy úgynevezett fésűt próbál illeszteni az eljárás. A fésű ablakainak szélessége a fésű szélességétől függően változik, értéke az ablakok távolságának negyede.



20. ábra: Fésű illesztése az ütésekre

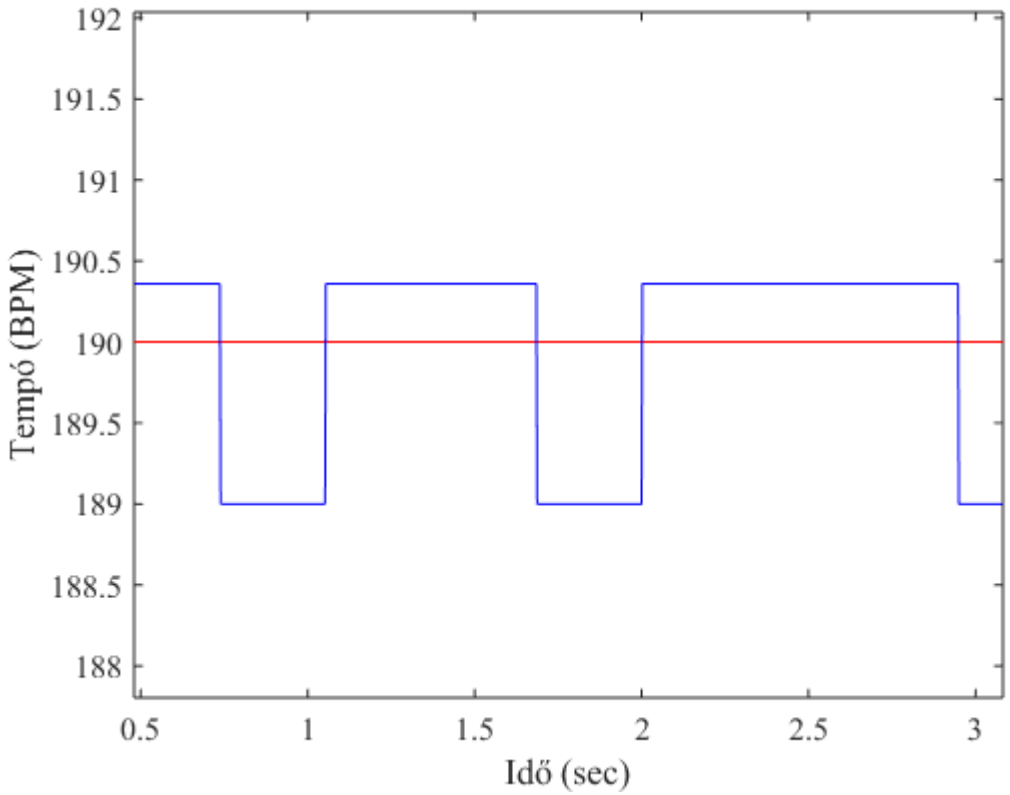
A 20. ábrán zölddel bekeretezett rész jelöli a fésű egy ablakát. A két fekete ábra pedig a fésűket demonstrálja.

2. Amennyiben az illesztés sikeres, azaz minden ablakban található ütés, a tempó kiszámításra kerül a fésű által lefedett ütések távolságai alapján.

$$tempo = \frac{60 * SignPerSecond}{BeatDistance}$$

A *SignPerSecond* a másodpercenkénti mintaszámot határozza meg. A *BeatDistance* az ütések távolsága mintaszámban értelmezve.

3. Ezután a fésű illesztése továbbhalad a jelen. Sikertelen illesztés esetén a fésű továbbhalad a következő ütésig.



21. ábra: Az ablakillesztéses módszer eredménye (piros: metronóm tempója, kék: algoritmus eredménye, zene: 190 BPM tempójú egyenletes dobütések)

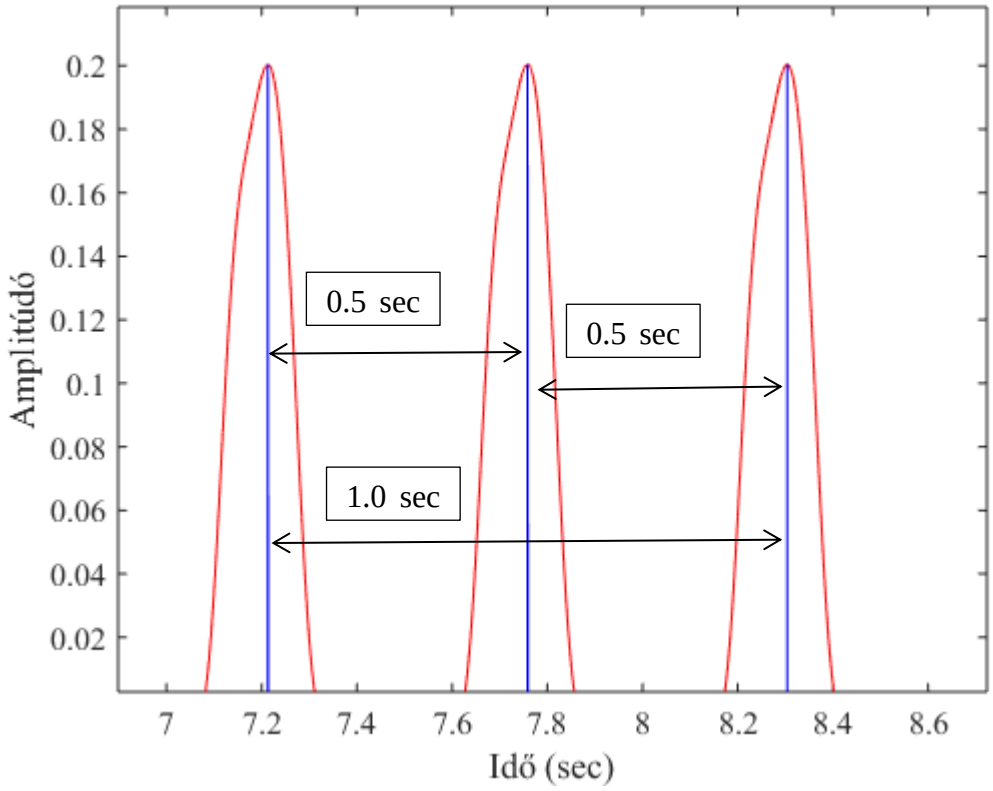
A 21. ábrán pirossal a referenciatempó látható, ami ebben az esetben 190 BPM. A kékkel ábrázolt érték az algoritmus által kiszámolt tempót reprezentálja. Az algoritmus részletes eredményei a 4.1.5-ös fejezetben kerülnek ismertetésre.

Ütéstávolság gyakoriságán alapuló módszer

A módszer alapötlete a következő: Egy adott ütés helyét összemérve az összes többi ütés helyével különböző távolságértékek állnak elő. Az így kiszámolt távolságok közül a leggyakoribb alapján visszaszámolható a tempó. [Joe - 2014] Az általam használt eljárás a következő:

Lépések:

1. Az ütések távolságainak kiszámolása a *SamplingWindowSize* nagyságú tartományban.

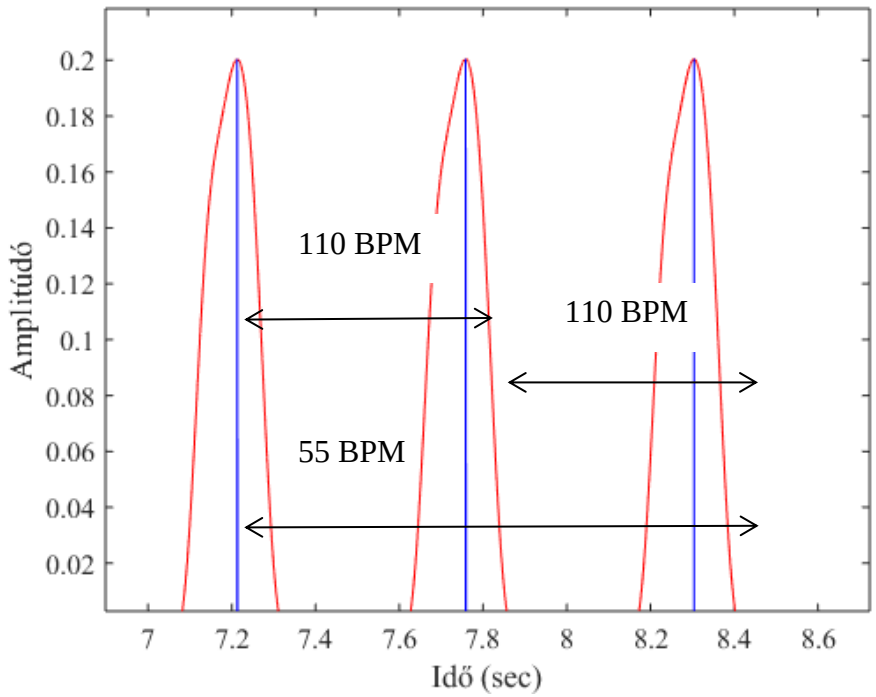


22. ábra: Az ütések távolságainak meghatározása (piros: előfeldolgozás utáni jel, kék: ütések helyei)

Az így meghatározott értékek például a következők lehetnek:

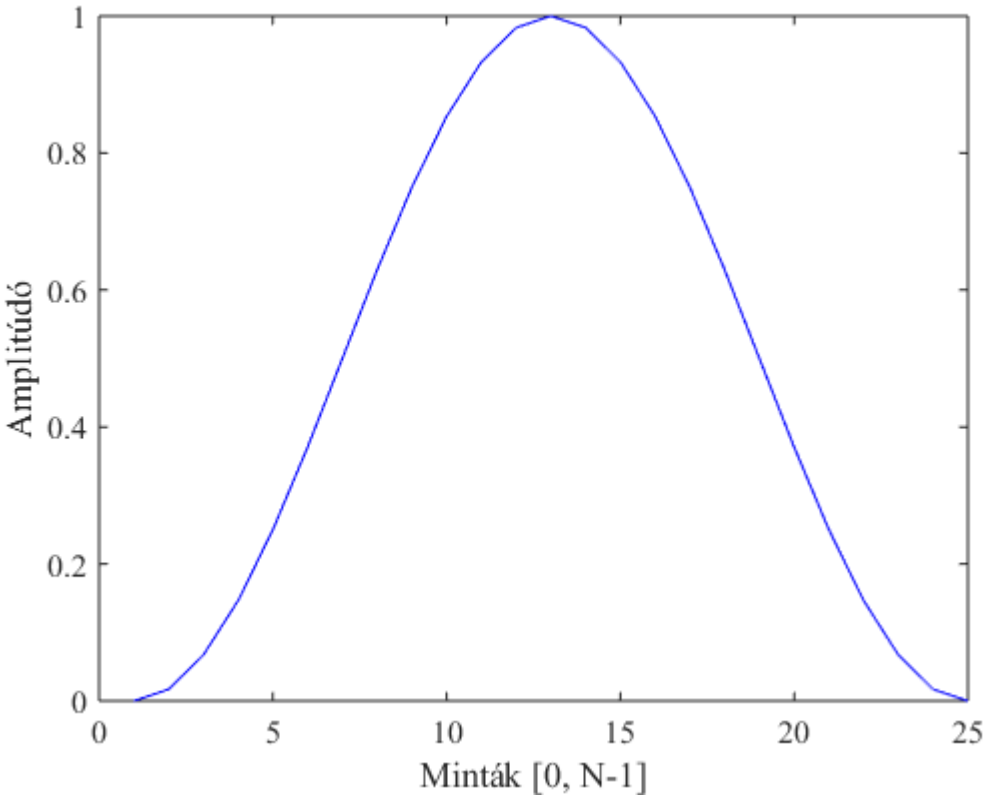
Távolság (sec)	Gyakoriság
1.6	21
0.5	12
0.25	2

2. A távolságértékek gyakoriságának csoportosítása a *TempoLowLimit* és *TempoHighLimit* által meghatározott távolságok közé. Tegyük fel, hogy ezek az értékek 100 BPM és 200 BPM. A lépés célja, hogy a zenékben a tempóértékek többszöröseit is figyelembe vegye az algoritmus és csoportosítsa őket a limitek közé. Például 110 BPM esetén vegye figyelembe a 55 BPM-hez tartozó távolságértékeket is, ahogyan a 23. ábra demonstrálja. Ezzel az eljárással több információhoz juthatunk a tempót illetően. Az így előállított hisztogram a 25. ábrán tekinthető meg.



23. ábra: Ütéstávolságok csoportosítása (piros: előfeldolgozott jel, kék: ütések helyei)

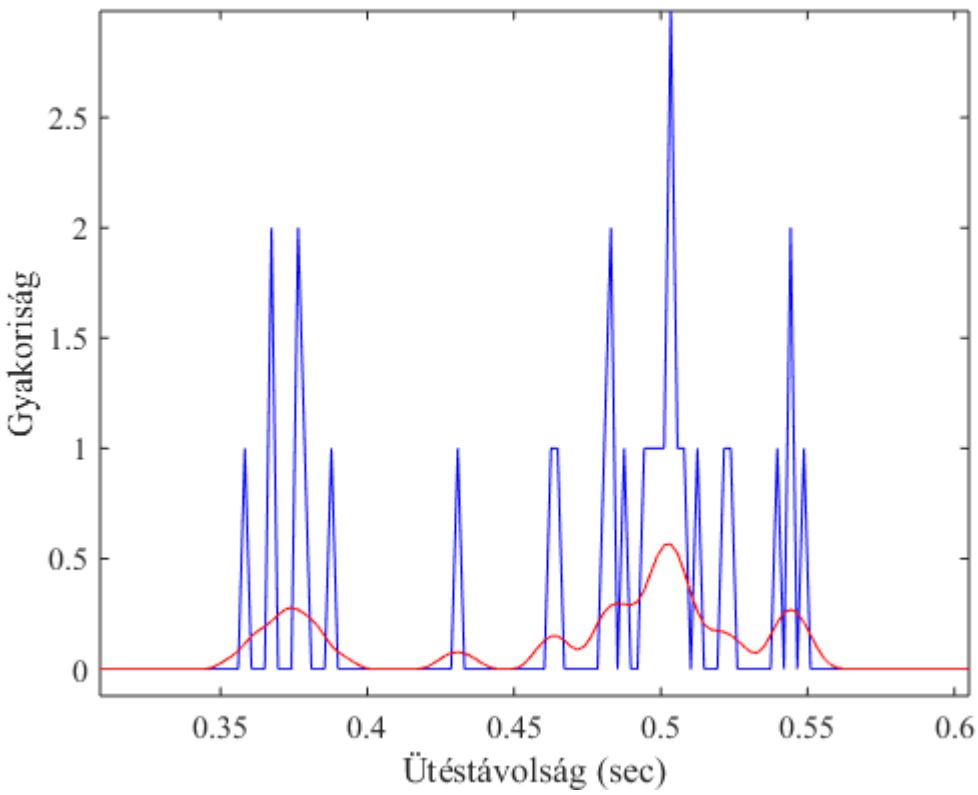
3. A meghatározott távolságértékek egy Hann ablak alapú szűrés segítségével simításra kerülnek [Kabal - 2004].



24. ábra: Hann ablak (*N*=25)

Az 24. ábrán látható Hann ablak értékei a következő egyenlettel határozhatóak meg, amelyben *N* az ablak szélességét jelöli:

$$y(n) = 0.5 - 0.5cos\frac{2\pi n}{N}$$



25. ábra: Távolságértékek hisztogramja (kék: eredeti gyakoriságok, piros: Hann szűrés eredménye)

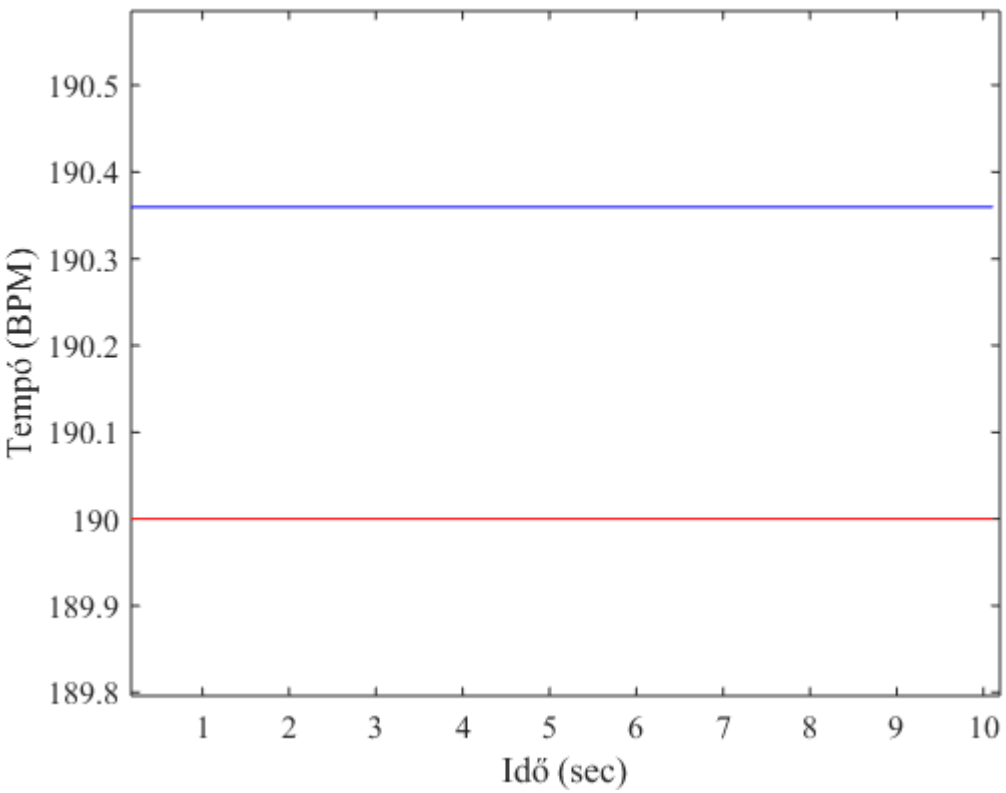
Az 25. ábra *x* tengelye a távolságok nagyságát jelzi, az *y* tengelye pedig a távolságok gyakoriságát. A piros jel az eredeti távolságértékek 13 szélességű Hann ablakkal történő átlagolásának eredményét mutatja.

Erre a lépésre azért van szükség, mert előfordulnak egymáshoz nagyon közeli távolságértékek különböző gyakorisággal és az ezekben tárolt információ a maximumkeresés során elveszne. Ennek az eljárásnak köszönhetően az eredményt ezek az értékek is befolyásolják és így pontosabb végeredmény kapható.

4. Ezután a három leggyakrabban előforduló távolságérték alapján kiszámításra kerülnek a tempók.

$$tempo = \frac{60 * SignPerSecond}{BeatDistance}$$

Végül az előző ciklusban kiszámolt tempóhoz legközelebb eső érték kerül kiválasztásra.



26. ábra: Ütéstávolság alapján számolt eredmény (piros: metronóm tempója, kék: algoritmus eredménye, zene: 190 BPM tempójú egyenletes dobütések)

A 26. ábrán pirossal a referencia tempó látható, ami 190 BPM. Kékkel pedig az algoritmus által kiszámolt érték figyelhető meg. Látható, hogy az eredmény alig tér el a referenciatempótól.

4.1.4 Algoritmusok bemeneti paraméterei

UndersamplingRate: Alulmintavételezés aránya. Az eredeti bemenő jel mintavételi frekvenciája 44100 Hz, azaz másodpercenként 44100 jelet tartalmaz. A tempó megállapítása szempontjából ez feleslegesen magas érték és nagymértékben lassítja az algoritmust, ezért van szükség az alulmintavételezésre. Az általam használt érték 50, azaz az eredeti felbontás másodpercenkénti 882 jelre csökken.

SectorSize: A feldolgozás során a jel szektorokba osztásra kerül. Azt, hogy egy ilyen szektor hány jelből álljon, ez a paraméter határozza meg. Ez a paraméter kizárólag a hangenergia alapján működő ütéskeresés során használt. Az általam használt érték 20 minta.

[N, Wn]: Az aluláteresztő szűrő fokszáma és törésponti frekvenciája. Ezek a paraméterek csak azokban az algoritmusokban használandóak, amelyekben a jelelőkészítés aluláteresztő szűréssel történik.

StepWindowSize: A kijelzés gyakoriságát határozza meg mintaszámban megadva. Az általam használt érték 0.2 másodpercnek megfelelő.

SamplingWindowSize: Azt a mintaszámot adja meg, amellyel egy adott frissítési intervallumban számol az algoritmus. Például 2 másodperc.

TempoLowLimit/TempoHighLimit: A kijelzendő minimális és maximális tempót határozza meg BPM-ben. Az általam használt értékek 100 BPM és 200 BPM.

DemoTempo: A várt tempó értéke BPM-ben megadva.

4.1.5 Összehasonlítás

Az összehasonlítás szempontjai:

I: Futási idő, számításigény

II: Pontosság

III: Tempóváltozások követésének képessége

I. Futási idő, számításigény: A program működése során az egyik legfontosabb paraméter az algoritmus számításigénye. A Matlabban történt tervezés során a teljes feldolgozandó jel rendelkezésre állt, azonban a végleges implementáció során valós időben kell elvégezni a feldolgozást. Ez azt jelenti, hogy egy hangrészlet feldolgozási ideje nem lehet hosszabb a részlet hosszánál.

II. Pontosság: A cél a minél pontosabb tempódetektálás. Elvárás, hogy ha a felhasználó 60 BPM-es tempóval játszik, ne kapjon más, például 64 BPM-es tempóról visszajelzést.

III. Tempóváltozások követésének képessége: Zenélés során nem ritka, hogy szándékos a tempó emelése vagy csökkentése. A legideálisabb, ha ezeket a változásokat a program kezelni tudja, továbbá előnyt jelent, ha nem csak a fokozatos változásokat képes követni az alkalmazás, hanem a nagyobb tempóugrásokat is. A nagymértékű és hirtelen tempóváltásokról elmondható, hogy ritkán fordulnak elő, de ideális esetben az alkalmazás képes ezeket is kezelni.

A vizsgált algoritmusok:

Az algoritmusok a 4.1.1, 4.1.2 és 4.1.3-as fejezetben ismertetett lépések kombinációiként állnak össze a következőképpen:

A: Hangenergia alapján működő algoritmus:

Az algoritmus nem tartalmazza a jelelőkészítés lépését. Az ütéskeresés az átlagos hangenergia számítás módszerével történik. A tempószámításhoz az ütéstávolságok gyakoriságán alapuló módszert használja.

B: Ablakillesztéses módszer:

A jelelőkészítéshez aluláteresztő szűrést használ. Az ütéskeresés a lokális maximum elvén működik. A tempószámításhoz pedig az ablakillesztéses módszert használja.

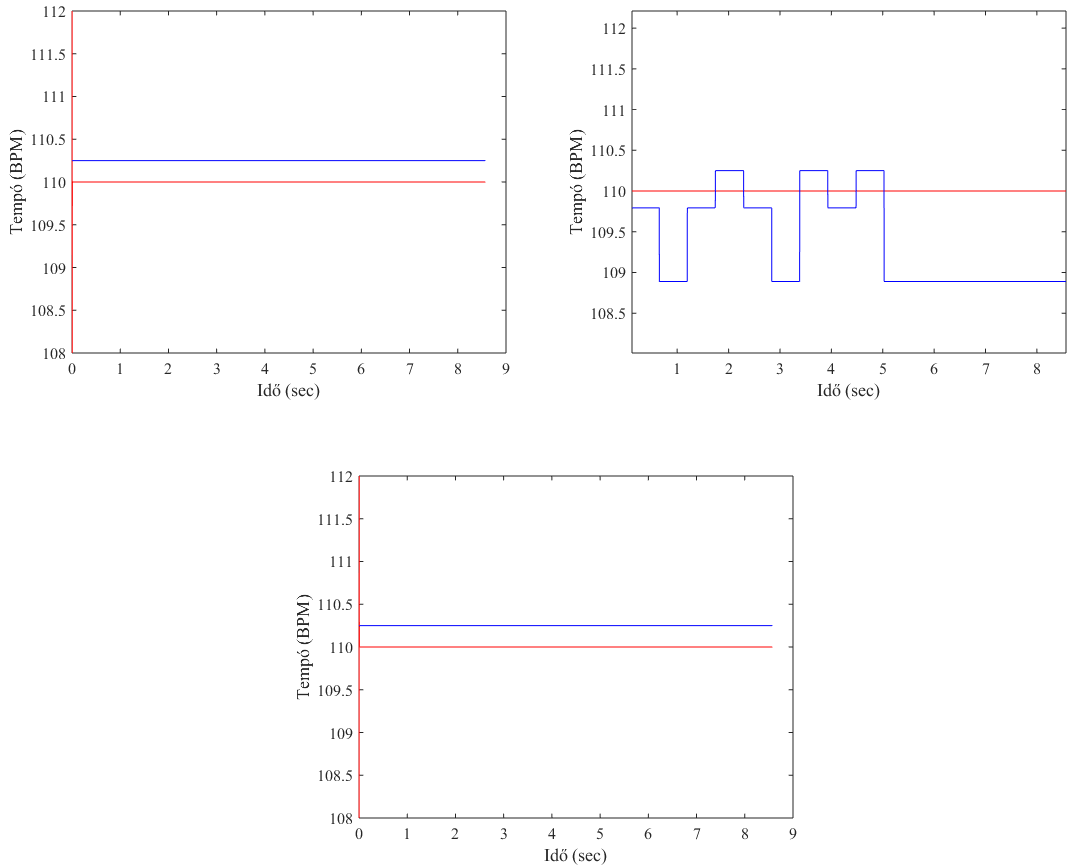
C: Ütemtávolságok gyakoriságán alapuló módszer:

A jelelőkészítéshez aluláteresztő szűrést használ. Az ütéskeresés a lokális maximum elvén működik. A tempószámítás az ütemtávolságok gyakoriságán alapuló módszerrel történik.

A pontos mérés érdekében az algoritmusok közös paraméterei megegyeznek és az összehasonlításhoz használt referenciajelek számítógép által generáltak. A jelek a következők:

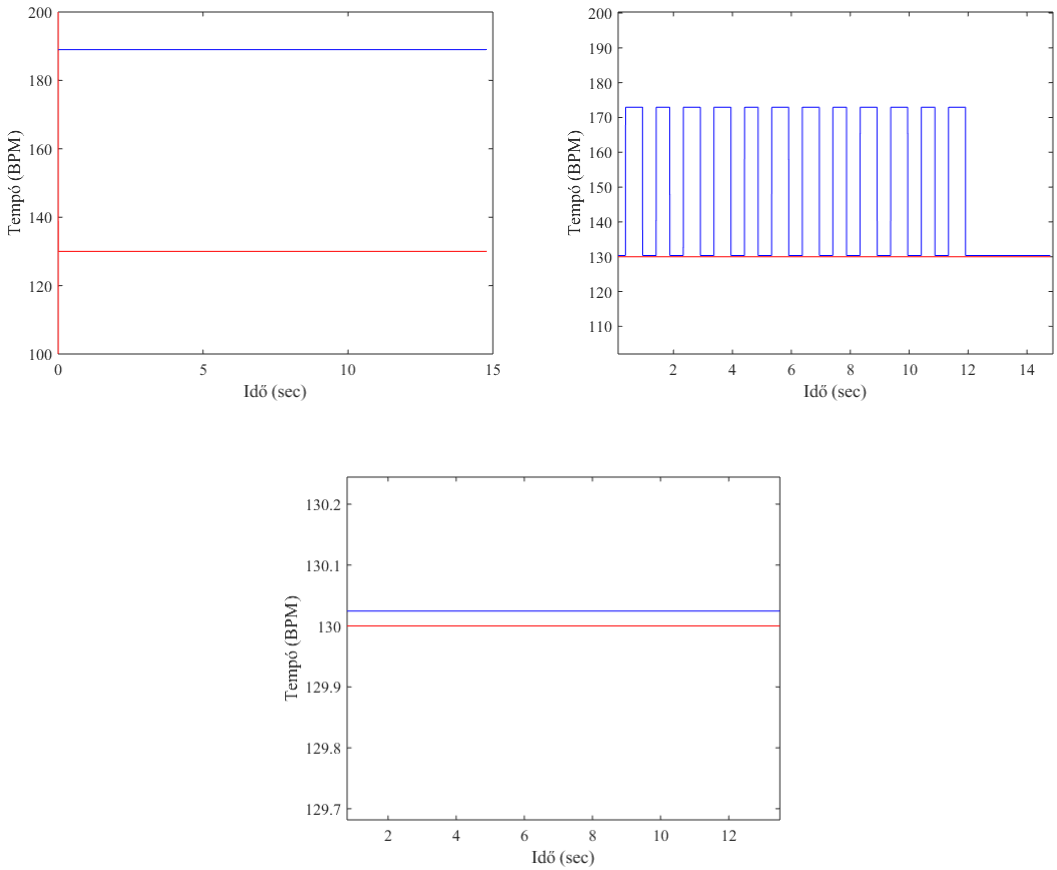
Jelölés	Név	Tempó	Leírás
a.)	110_simple.wav	110BPM	Egyenletes dobütések.
b.)	130_simple_cin.wav	130BPM	Egyenletes cintányérütések.
c.)	half_increase.wav	129-129.5-130BPM	Egyenletes dobütések fél BPM tempóemelkedéssel.
d.)	110_130_change.wav	110-130BPM	Egyenletes dobütések 110BPM-el, majd 130BPM-es egyenletes cintányérütések.
e.)	jb_midi_118.wav	118BPM	James Brown zeneszám dobjátéka midi-ből generálva.

Az eredmények ismertetése során a grafikonok *x* tengelye az időt ábrázolja másodpercben, az *y* tengely pedig a tempót BPM-ben meghatározva. Az ábrán pirossal jelölt értékek a metronóm tempóját, a kékkel jelöltek pedig az algoritmusok eredményeit mutatják.



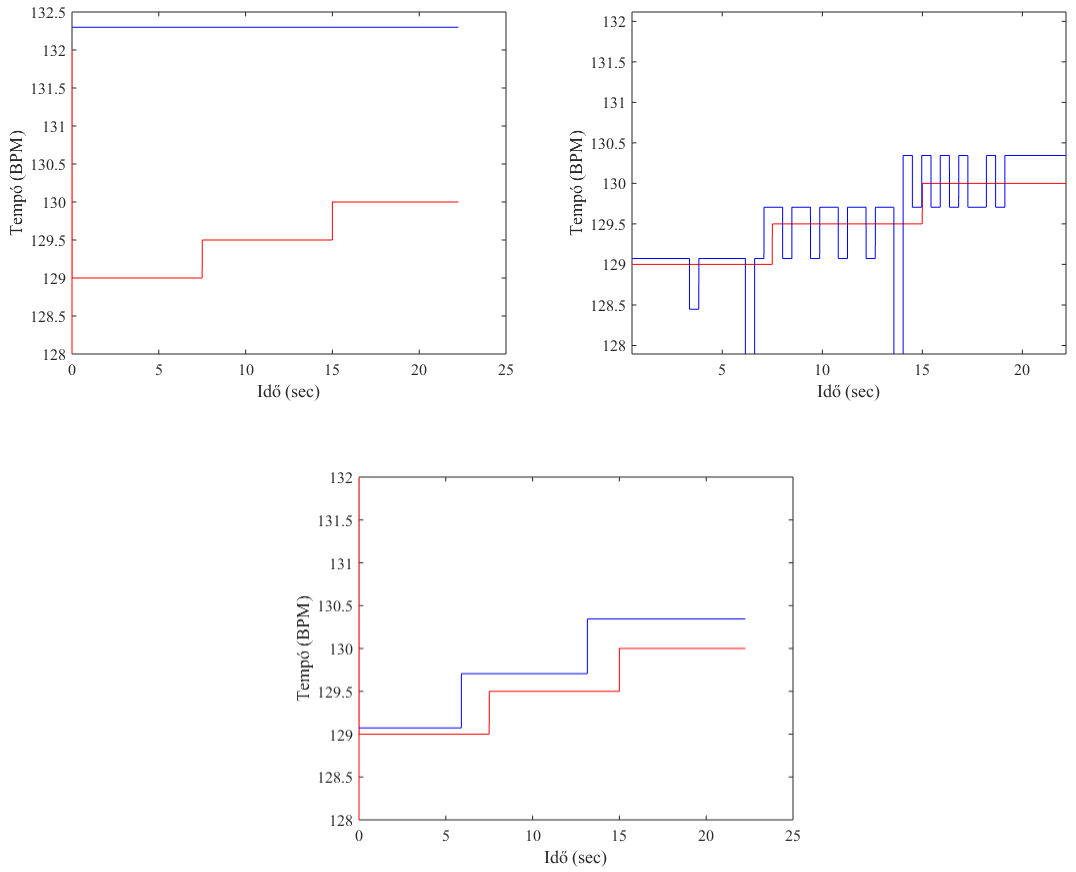
27. ábra: Hangenergia alapján működő algoritmus *(A)*, Ablakillesztéses módszer *(B)* és Ütemtávolságok gyakoriságán alapuló módszer *(C*) algoritmusok eredményei az *a.)* 110_simple.wav zenére (piros: metronóm tempója, kék: algoritmus eredménye)

Az 27. ábrán látható hogy mindhárom algoritmus eredménye pontos, a metronómtempótól való eltérés nem jelentős. A *B* algoritmus esetében megfigyelhető a kijelzés ingadozása.



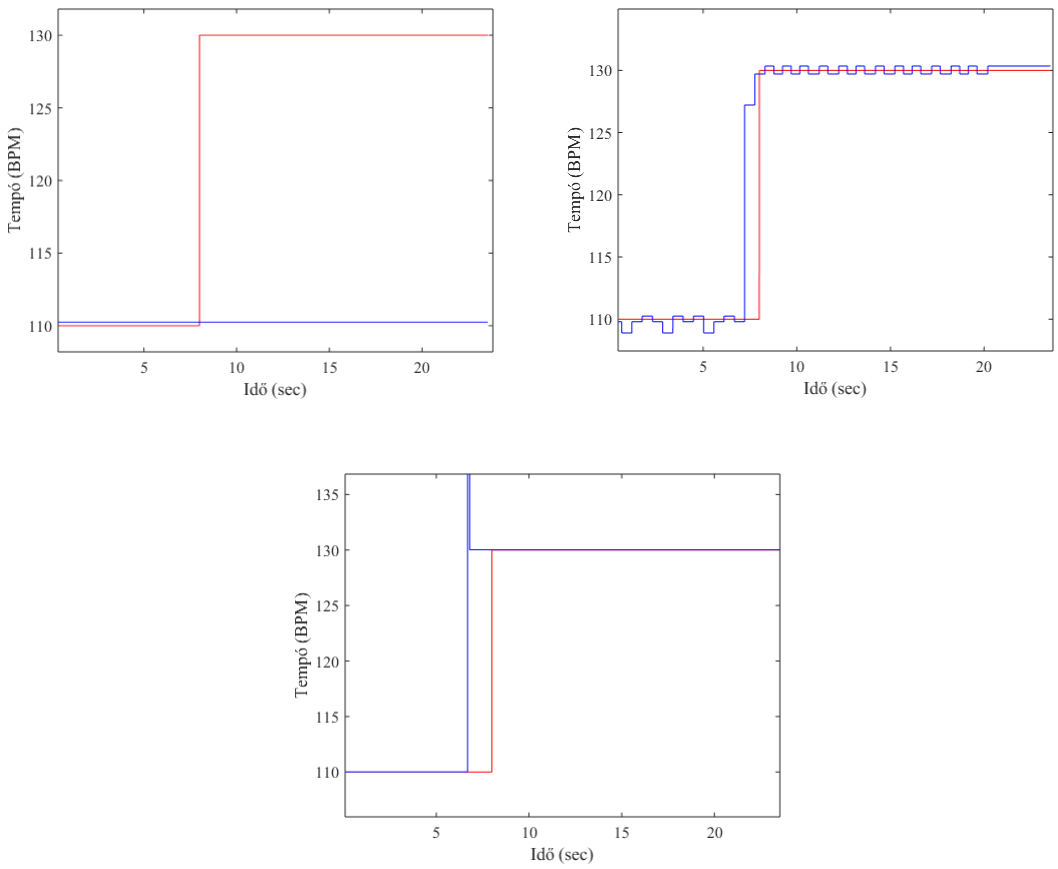
28. ábra: Hangenergia alapján működő algoritmus *(A),* Ablakillesztéses módszer *(B)* és Ütemtávolságok gyakoriságán alapuló módszer *(C*) algoritmusok eredményei az *b.)* 130_simple_cin.wav zenére (piros: metronóm tempója, kék: algoritmus eredménye)

A 28. ábrán látható, hogy az *A* algoritmus eredménye helytelen. A *B* eredménye periodikusan változik és a változás mértéke jelentős. A *C* értéke helyes, közel van a referenciához.



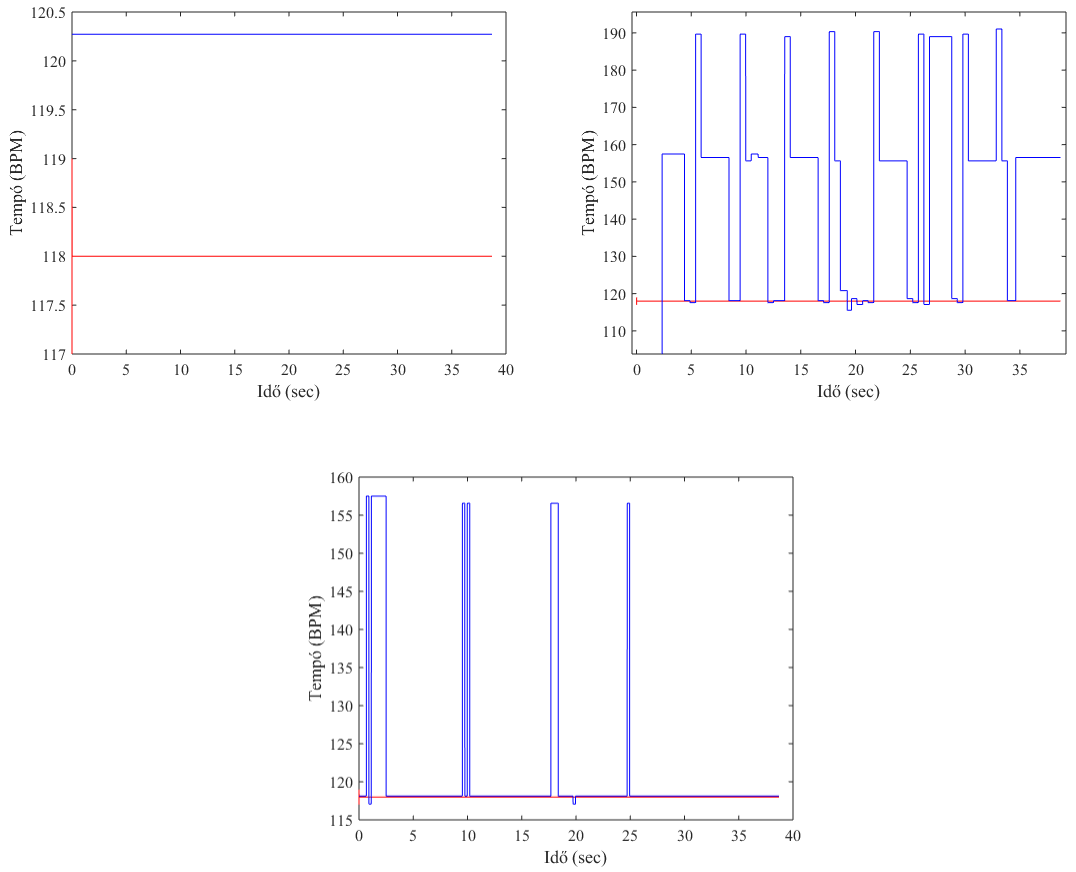
29. ábra: Hangenergia alapján működő algoritmus *(A)*, Ablakillesztéses módszer *(B)* és Ütemtávolságok gyakoriságán alapuló módszer *(C*) algoritmusok eredményei az *c.)* half_increase.wav zenére (piros: metronóm tempója, kék: algoritmus eredménye)

Ebben az esetben a tempó 7.5 másodpercenként emelkedik 0.5 BPM-et. Az *A* eljárást leszámítva a másik két eredmény jónak tekinthető, a változásokat pontosan követik az algoritmusok.



30. ábra: Hangenergia alapján működő algoritmus *(A)*, Ablakillesztéses módszer *(B)* és Ütemtávolságok gyakoriságán alapuló módszer *(C*) algoritmusok eredményei az d.*)* 110_130_change.wav zenére (piros: metronóm tempója, kék: algoritmus eredménye)

Ebben az esetben a tempó 8 másodpercnél 110 BPM-ről 130 BPM-re változik. A *B* és *C* megoldások megfelelően felismerik a változást és követik azt. Az *A* megoldás nem képes a változás követésére.



31. ábra: Hangenergia alapján működő algoritmus *(A)*, Ablakillesztéses módszer *(B)* és Ütemtávolságok gyakoriságán alapuló módszer *(C*) algoritmusok eredményei az e*.)* jb_midi_118.wav zenére (piros: metronóm tempója, kék: algoritmus eredménye)

A 31. ábrán az *C* eljárás a legpontosabb és az *A* tévedése sem jelentős mértékű. A *B* megoldás rendkívül bizonytalan.

A számításigény méréséhez egy 8 másodperc hosszúságú hangot használtam fel. A pontosságmérés oszlopában az extrém értékek figyelmen kívül hagyása mellett számított eltérések átlaga látható.

Algoritmus	**I. szempont** Futási idő	**II. szempont** Pontosság	**III. szempont** Tempóváltozások követése	**Megjegyzés**
A	0.2 sec	1.9 BPM	-	Pontatlan és nehezen követi a változásokat.
B	0.3 sec	0.3 BPM	+	Előre ismerni kell a zene tempóját.
C	0.6 sec	0.35 BPM	+	Nem kell előre ismerni a zene tempóját.

4.1.6 Módszer kiválasztása

Az összehasonlítás eredményeinek értelmezése a következő:

Az *A* algoritmus számításigény tekintetében a legkedvezőbb. A szektorokra osztás következménye, hogy a pontossága a három eljárás közül a legrosszabb. Ennek hatása, hogy a tempóváltozásokat nem tudta megfelelően lekövetni. További nehézség, hogy körülményes megállapítani az ideális szektorméretet, ugyanis ha túl kicsi vagy túl nagy szektort választunk, az algoritmus pontatlan lesz.

A *B* algoritmus a számításigényt tekintve közepesen teljesített. A pontossága az összefoglaló táblázat alapján a legjobb, azonban ez annak köszönhető, hogy az eredmény számítása során nem vettem figyelembe az extrém nagy eltéréseket, valamint a bizonytalan kijelzést. Az algoritmus egyik hátránya, hogy nem képes a tempó stabil kiszámítására, csak közelíti azt, ahogyan ez az ábrákon is látható. További hátrány, hogy előre ismerni kell a zene tempóját, hogy a fésű illesztését végre tudja hajtani az eljárás.

A *C* algoritmus számításigény tekintetében a leggyengébb volt, azonban a feldolgozás a teljes zene hosszánál kevesebb időt vett igénybe és feltételezhető, hogy ez az idő a C++-ban történő implementáció során kedvezőbb lesz. A legpontosabb eljárás és további előnye, hogy jól követi a tempóváltozásokat is. A működéséhez nem

szükséges a zene tempójának előzetes ismerete. Ezeket a szempontokat figyelembe véve a *C* algoritmusra esett a választásom és ezt valósítottam meg a végleges implementáció során.

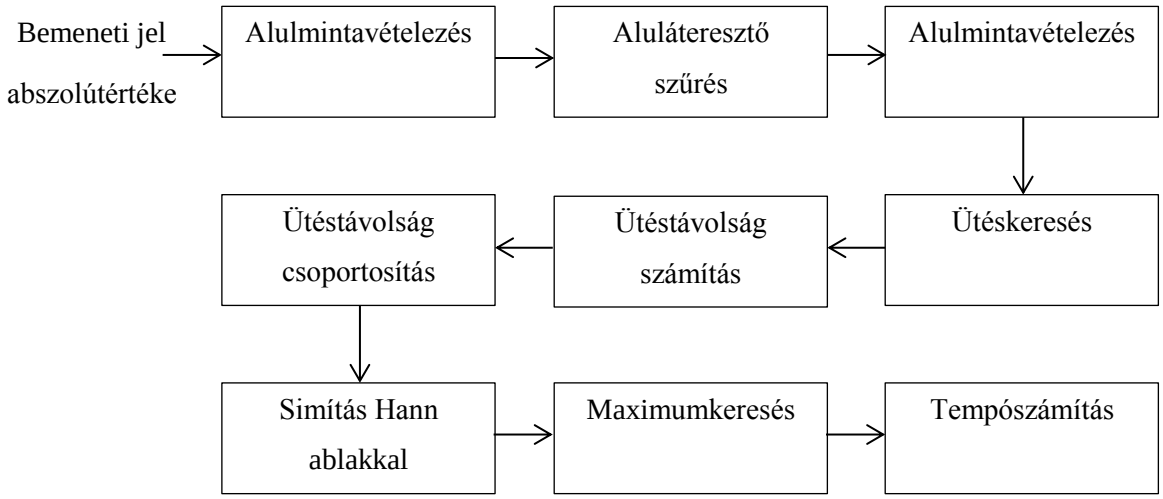
5. Implementáció

Az implementáció C++ nyelven történt a JUCE fejlesztőkörnyezet felhasználásával. A cél a bevezetőben ismertetett funkciók megvalósítása volt, a tervezés során kifejlesztett algoritmus felhasználásával.

5.1 Környezet

A JUCE fejlesztőkörnyezet által biztosított funkciók jelentősen megkönnyítették a fejlesztést. Az *AudioIODeviceCallback* osztály által egy tömbben tárolva kinyerhető volt a mikrofonjel. A környezet lehetőséget biztosított az audio eszközök és hangcsatornák kiválasztására és a mintavételi frekvencia beállítására. A fejlesztés során felhasználtam a JUCE honlapján elérhető demókat. A demókban ismertetésre kerülnek a program által biztosított grafikus elemek, valamint részletes információk olvashatóak az eszköz eseménykezeléséről, osztályok hierarchiájáról és az elvárt paraméterekről. A projekt szerkesztéséhez, fordításához és futtatásához Microsoft Visual Studiot használtam.

5.2 Program működésének részletei

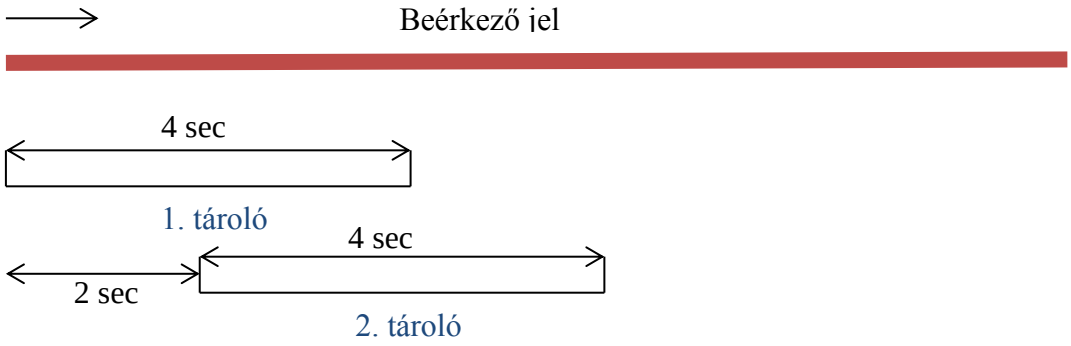


32. ábra: Az implementált algoritmus lépései

Az algoritmus lépései a 32. ábrán nyomon követhetőek. Ezek a következőek:

- Abszolútérték képzés: a bementi jelnek a feldolgozás első lépésében vettem az abszolút értékét.
- Alulmintavételezés: a feldolgozandó jel mintavételi frekvenciája 44100 Hz. Ezt lecsökkentettem 8820 Hz-re a kedvezőbb számításigény érdekében. A művelet eredményében a jelek átlagát használtam.
- Aluláteresztő szűrés: aluláteresztő szűrést végeztem egy másodrendű Butterworth szűrővel.
- Alulmintavételezés: a 8820 Hz mintavételezésű jel felbontását lecsökkentettem 882 Hz-re, ezzel tovább csökkentve a számításigényt.
- Ütéskeresés: Az ütések helyeit a lokális maximumkeresés módszerével állapítottam meg.
- Ütéstávolság számítás: a detektált ütések egymástól mért távolságait meghatároztam és eltároltam a memóriában. A tárolt információ a távolságértékek gyakorisága.
- Ütéstávolság csoportosítás: a 4.1.3-as fejezetben ismertetett ütéstávolság gyakoriságán alapuló módszer szerint csoportosítottam a távolságértékeket, így előállítva azok hisztogramját.
- Simítás Hann ablakkal: a távolságértékeken simítást végeztem egy 25 egység szélességű Hann ablakkal.
- Maximumkeresés: megkerestem a három leggyakrabban előforduló távolságértéket.
- Tempószámítás: a kiválasztott értékek alapján kiszámoltam a leggyakoribb tempóértékeket és az aktuális tempóhoz legközelebbi értékkel frissítettem a kijelzendő paramétert.

A számításhoz vett hang hossza a *Precision* paramétertől függ, azaz kettő vagy négy másodperc. A tempó meghatározásának gyakorisága a *Precision* paraméterrel megegyező, azaz egy és két másodperc. A beérkező jeleket két tárolóba helyeztem el a 33. ábrán látható módon elcsúsztatva azokat:

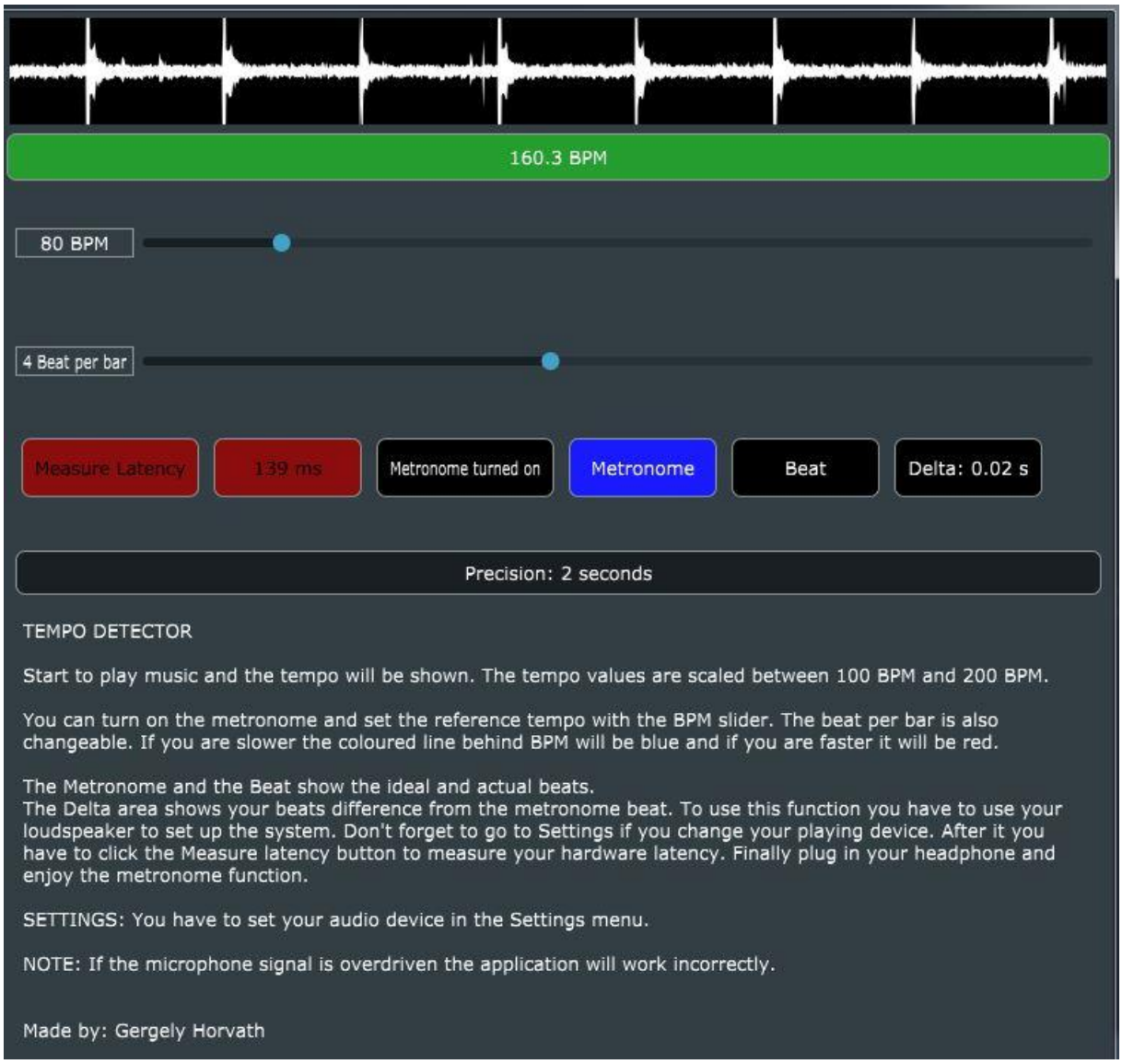


33. ábra: Beérkező jel tárolókba helyezése

A feldolgozás a két tároló értékein felváltva ment végbe, így megnőtt a számítás pontossága. A tempó kiszámításának tartománya 100 BPM és 200 BPM közötti volt. A kijelzéshez és a globális változók értékeinek frissítéséhez a JUCE által biztosított *Timer* osztályt használtam, amely az előre beállított időközönként lefuttatta a definiált eljárást. Szintén a *Timer* osztályt használtam a metronóm szimulálásához.

5.3 Bemutatás és tesztelés

Az alkalmazás elindításához követelmény egy *.exe* kiterjesztésű fájl futtatására alkalmas számítógép, valamint a program működtetéséhez szükség van a számítógéphez csatlakoztatott mikrofonra. A program fordításához elvárás Microsoft Visual C++ 14.1.



34. ábra: Az elkészített program felülete

A programot elindítva a felhasználót a 34. ábrán látható kép fogadja. Az alkalmazás működéséhez a beállítások menüpont alatt ki kell választani a megfelelő hangbementet és be kell állítani annak csatornáit, valamint mintavételi gyakoriságát.

A 34. ábrán a legfelső sávban látható a bementi jel. A használat során ügyelni kell arra, hogy a jel ne legyen túlvezérelt, mert ez zavarja az algoritmus működését. Ilyen esetben lejjebb kell állítani a mikrofon erősítését. Az ábrán a zöld sávban a zene tempója látható. A tempó frissítésének gyakorisága a *Precision* gomb segítségével állítható. Az elérhető értékek 1 és 2 másodperc. Alacsonyabb tempó mellett a hibamentes eredménykijelzés érdekében érdemes a nagyobb értéket választani.

A program másik funkciója a metronóm funkció. A kijelzett tempóérték alatt található csúszkával beállítható a metronóm tempója és a második csúszkával kiválasztható a metronóm üteme. Az elérhető ütemszámok egytől nyolcig terjednek. A funkció bekapcsolásához a *Metronome turned off/on* gombot kell használni. Bekapcsolt funkció esetén a gomb melletti *Metronome* felíratú terület a beállított ütemben villogással jelzi a metronóm ütéseit. A program a képi visszacsatolás mellett az audio kimeneten is kisugározza az ütemmérő jelét. További segítség a gyakorló számára, hogy visszajelzést kaphat az ütései helyéről és hibája nagyságáról. Ehhez szükség van a hangkártya feldolgozási ideje okozta késleltetés kimérésére. A mérés során biztosítani kell, hogy a hangkimenet jele érzékelhető legyen a bemeneten. A késleltetés megállapításához a *Measure Latency* gomb használandó, a használathoz feltétel a zajmentes környezet. A kalibráció szükség esetén megismételhető, az eredmény pedig milliszekundumban kerül kiírásra. A kalibráció elvégzése után újra csatlakoztatható a számítógéphez a zenész által használt fej- vagy fülhallgató. A csatlakoztatás után a beállítások menüben újra be kell állítani az audio eszközöket. Mindezek után zenélés közben a *Beat* feliratú terület a zenész ütéseit jelzi vissza és a *Delta* érték megmutatja számára ezek eltérését a metronóm jelétől másodpercben kifejezve.

További segítséget nyújt, hogy a kijelzett tempó háttere zöld, ha a játszott zene tempója megegyezik az előre beállítottal. Ha késik a zenész, a háttér kék színt vesz fel, míg gyorsabb zene esetén piros színnel ad visszajelzést. A kijelzés színe a metronóm tempójától való eltérés mértékétől függően világosabb, illetve sötétebb.

A program a tesztelés során az elvártak szerint teljesített. A működéshez előfeltétel volt a zajmentes környezet. A pontosságot külső metronóm segítségével teszteltem és az alkalmazás mérési hibája nem haladta meg a 0.5 BPM-et. Alkalmas volt a tempóváltozások követésére, a beállási ideje 2 másodperces pontossági beállítás mellett 1-2 másodperc körüli volt. A nem gépi, azonban metronómhoz igazított dobszólók esetében is jól teljesített és precíz visszajelzést adott. Alkalmanként elveszítette az aktuális tempót és hamis értéket jelezett ki, de ezt 3 másodpercen belül korrigálta. A működés részletes eredményei a további fejezetben kerülnek ismertetésre.

5.4 Értékelés

A program értékeléséhez összehasonlítottam az eredményeit a Matlabban implementált megoldás során tapasztaltakkal, így meggyőződve arról, hogy a végleges verzió működése megegyezik a tervezett algoritmus működésével. A pontosság teszteléséhez metronómot használtam, valamint a metronómhoz igazított dobszóló felvételeket. A tesztelés során összehasonlítottam az általam elkészített programot a bevezetőben ismertetett BPM-Detector alkalmazással is.

5.4.1 Összehasonlítás Matlabbal

A felhasznált tesztadatok megegyeznek a tervezés során használtakkal.

Jelölés	Név	Tempó
a.)	110_simple.wav	110BPM
b.)	130_simple_cin.wav	130BPM
c.)	half_increase.wav	129-129.5-130BPM
d.)	110_130_change.wav	110-130BPM
e.)	jb_midi_118.wav	118BPM

Alkalmazás eredményei a különböző zenékre:

a.) A kijelzés stabil volt és 0.5 BPM pontosságú.

b.) A kijelzés stabil volt és teljesítette 0.5 BPM-es pontosságot.

c.) Az alkalmazás a zene kezdeti tempóját helyesen felismerte és a tempóváltozásokra egy másodpercen belül reagált. A teszt végén alkalmanként eltérő értéket mutatott, de az eltérés mértéke nem haladta meg az 1 BPM-et.

d.) A kijelzés stabil volt és pontos. A hirtelen tempóváltást helyesen jelezte ki az alkalmazás és a beállási ideje fél és egy másodperc közötti volt.

e.) A teszt során többször jelzett helytelen értéket a program, de a zene jelentős részében a megfelelő tempót mutatta.

Ezek alapján elmondható, hogy a véglegesen implementált megoldás teljesítette a tervezés során felállított kritériumokat.

5.4.2 Összehasonlítás más programmal

Az általam elkészített programot összehasonlítottam a mobiltelefonra elérhető BPM-Detector nevű alkalmazással. Az összehasonlítás során az alábbi eredményekre jutottam.

A metronómmal való tesztelés során látható volt, hogy a megoldásom 0.5 BPM-el pontosabb kijelzést tesz lehetővé. A tempó változása során a BPM-Detector körülbelül egy-két másodperccel lassabban reagált. A metronómhoz igazított egyenletes tempójú dobszólók esetében mind a két megoldás hibátlanul teljesített, a kijelzés stabil és pontos volt. A legnagyobb különbséget az énekhangot tartalmazó zenék vizsgálata során tapasztaltam. Ebben az esetben a konkurens program helyesen működött, míg az általam elkészített megoldás az énekhangot tartalmazó részeken hajlamos volt a helyes érték elvesztésére. Ennek oka, hogy az én megoldásom kizárólag a hang burkológörbéjét használja a feldolgozás során.

6. Összefoglalás

A dolgozat elkészítése kapcsán megismerkedtem a digitális jelfeldolgozás alapjaival. Különböző burkológörbe készítési eljárásokat próbáltam ki, melyeket a hangfeldolgozás mellett számos területen alkalmaznak, például a tőzsdei adatok feldolgozása során is. A munkám jelentős részét a különböző tempófelismerő algoritmusok kifejlesztése tette ki Matlabban. A feldolgozás nehézségét az okozta, hogy az általam megismert meglévő tempódetektáló eljárások esetében a cél a teljes hang tempójának megállapítása volt. Az én esetemben ezzel szemben valós idejű feldolgozásra kellett felkészíteni a programot, így kevesebb rendelkezésre álló információból kellett a zene gyorsaságát meghatározni. Végül három különböző algoritmust hasonlítottam össze, hogy kiválaszthassam a legideálisabbat a végső implementációhoz. Az implementáció során nagy segítséget nyújtott a Matlabban való megvalósítás során szerzett tapasztalat, így a fejlesztést célirányosan tudtam elvégezni.

Az elkészült program sikeresen teljesítette a felállított követelményeket, miszerint alkalmas egy zenész gyakorlásának segítésére. Egyetlen hangszer jelének feldolgozása esetén hibátlan működést mutatott. A gyengesége az összetettebb hangok elemzése volt, különösképpen az énekhangok esetén. A fejlesztés során további feladatot jelentett, a feldolgozás felgyorsítása. A tempószámító algoritmus mellett használt grafikus elemek frissítése jelentősen leterhelte a számítógép erőforrásait. A munkám alkalmával a metronómjel meghatározásához és más frissítendő értékek számításához a JUCE által biztosított *Timer* osztályt használtam fel, melynek funkciója, hogy egy előre beállított periodicitással lefuttasson egy kódrészletet. A *Timer* hátránya, hogy nem képes teljes garanciát vállalni a periodicitás állandóságára, ezért az általa kiváltott események gyakoriságát külön ellenőrizni kell. Az ismertetett pontatlanságból adódóan a metronóm pontos tempója nehezen volt beállítható. A tesztelés további tapasztalata volt, hogy a program erőforrásigénye magas és a hibátlan működéshez ezt biztosítani kell számára, így érhető el a pontos kijelzés és a megakadásmentes futtatás.

Elmondható, hogy az általam elkészített program a meglévő megoldásokkal szemben egyszerre több gyakorlást segítő funkciót is magában foglal. A tempó egyszerű kijelzésére használható például az 1.1-es fejezetben ismertetett BPM-Detector. Az ütemen belüli pontosság kijelzésére, azonban jellemzően csak elektromos dobok

képesek. Például a Roland TD-3 típusú dobfelszerelés tartalmaz *Coach (edző)* módot, amelynek keretein belül a *Time Check* funkció használható a metronóm ütéseitől való eltérés meghatározására. [Roland - 2004] Az általam elkészített szoftver mindkét használati módra lehetőséget biztosít, továbbá nem szükséges hozzá elektromos dobfelszerelés, hanem hagyományos akusztikus hangszerekkel is használható. Futtatásához pedig kizárólag egy napjainkban minden háztartásban megtalálható számítógépre van szükség, így nincs szükség speciális eszközökre. A használata a grafikus felületnek köszönhetően egyszerű és bárki által könnyen elsajátítható.

Az alkalmazás továbbfejlesztési lehetőségei:

1. A beérkező hang szétbontása a hangmagasság alapján. A felbontás történhet magas, közép és mély tartományokra. Ezután a tartományok tempóját külön- külön meghatározva pontosabb kijelzésre nyílna lehetőség.

2. A JUCE fejlesztőkörnyezet támogatja mobilalkalmazások fejlesztését is. A jövőbeli cél az alkalmazás elkészítése mobilplatformokra is.

3. A program többszálúvá tétele után és a szálbiztosság garantálását követően. Használhatóvá válhatna a JUCE *HighResolutionTimer* osztálya, ami lehetővé tenné a pontosabb eseménykezelést és kijelzést.

4. A metronóm pontosabb működése érdekében az ütemek helyének meghatározásához a kisugárzandó jelsorozat mintaszámai használhatóak. Ezzel garantálhatóvá válna a metronóm pontossága és hatékonyabbá tehető lenne a szoftver.

4. További cél lehet az alkalmazás erőforrásigényének csökkentése.

7. Melléklet ismertetése

BPMDetector.exe: az elkészült szoftver futtatható verziója.

BPMDetector.cpp: az elkészített program forráskódja.

BPMDetector.zip: az alkalmazás fejlesztési projektje, fordítható változata.

BeatDistance.m: Ütéstávolságok gyakoriságát használó algoritmus Matlab kódja. Az ütéskereséshez a maximumkeresés módszerét használja. A 4.1.5. fejezetben *C* jelű algoritmusként szerepel.

SoundEnergy.m: Ütéstávolságok gyakoriságát használó algoritmus, amely az ütéskereséshez a hangenergia alapján működő módszert használja. A 4.1.5. fejezetben *A* jelű algoritmusként szerepel.

*CombFilter.m:*A 4.1.5. fejezetben *B* betűjellel rendelkező ablakillesztéses módszer Matlab kódja.

README.txt: A melléklet rövid összefoglaló leírását tartalmazza.

Matlab kiegészítő fájlok: A Matlabban elkészített algoritmusok által használt funkciók fájljai *_funcion.m* végződéssel rendelkeznek. Ezek szükségesek a kódok helyes működéséhez.

Hangfájlok:

110_simple.wav, 130_simple_cin.wav, half_increase.wav, 110_130_change.wav, jb_midi_118.wav, 190.wav

8. Rövidítések

AAX: Avid Audio eXtension

A/D: Analóg/Digitális

AU: Audio Units

BPM: Beat Per Minute = Percenkénti ütés szám

Hz/kHz: Hertz/kiloHertz

IDE: Integrated Development Environment = Integrált fejlesztői környezet

ms: milliszekundum

MIDI: Musical Instrument Digital Interface

IIR: Infinite Impulse Response = Végtelen impulzusválaszú

VST: Virtual Studio Technology = Virtuális stúdió technológia

WAV: Waveform Audio File Format

9. Irodalomjegyzék

[Andrew Robertson - 2015] Andrew Robertson – BeatSeeker - 2015

https://www.ableton.com/en/packs/beatseeker/ (2017.12.06.)

[Balázs - 2013] Balázs János – Beat Detection and Correction For Djing Applications, Budapest University of Technology and Economics, Msc thesis - 2013

[Frédéric - 2003] Frédéric Patin – Beat Detection Algorithms - 2003

http://www.flipcode.com/misc/BeatDetectionAlgorithms.pdf (2017.12.06.)

[Fodor - 2014] Dr. Fodor Dénes – Digitális jelfeldolgozás, Pannon Egyetem - 2014

[Joe - 2014] Joe Sullivan – Beat Detection Using JavaScript and the Web Audio API - 2014

http://joesul.li/van/beat-detection-using-web-audio/ (2017.12.06.)

[JUCE – 2017] JUCE Api Modules – 2017

https://juce.com/doc/modules (2017.12.06.)

[Kabal - 2004] Peter Kabal – Discrete Time Signal Processing, Electrical &Computer Engineering McGill University - 2004

[Lipovszki - 2012] Dr. Lipovszki György – Jelfeldolgozás és számítógépes irányítás, EDUTUS Főiskola – 2012

[Roland - 2004] Roland Corporation - Ronald TD-3 Owner's Manual - 2004

[Ziccardi - 2015] Marco Ziccardi – Beat Detection Algorithms – 2015

http://mziccard.me/2015/05/28/beats-detection-algorithms-1/ (2017.12.06.)

[SandergSound - 2013] SandergSound – BPM-Detector

https://play.google.com/store/apps/details?id=com.BPMDetector2 (2017.12.06.)

[Szigetvári - 2014] Szigetvári Andrea, Horváth Balázs – Bevezetés a zenei informatikába, Typotex Kiadó, 2014

[Zölzer - 2011] Udo Zölzer – DAFX: Digital Audio Effects, John Wiley & Sons, Ltd – 2011