MŰEGYETEM 1782

SZAKDOLGOZAT-FELADAT

Szilágyi Bence Ágoston (PAT051)
Villamosmérnök hallgató részére

Tempódetektálás EFM32 Giant Gecko 32 bites mikrokontrollerrel

Különösen ütőhangszeresek és dobosok, de más hangszeres zenészek esetében is fontos, hogy tudjanak állandó tempóval játszani, tehát ne lassuljanak vagy gyorsuljanak. A tempótartást hagyományos módon metronómmal szokták gyakorolni, ugyanakkor a metronóm, amellett, hogy akusztikus ellenőrzési pont, segítség is, így nem modellezi pontosan a valós zenei szituációt. Ezen okból lenne hasznos egy olyan beágyazott rendszer, ami kizárólag a mikrofon által felvett hang alapján meghatározná a zene aktuális tempóját és azt kijelezné, így a zenész valós időben visszacsatolást kapna arról, hogy tartja-e az általa megcélzott tempót.

A hallgató feladata egy EFM32GG-STK3700 fejlesztői kártyát, ill. a hozzá kapcsolt elektret mikrofont és mikrofonerősítőt felhasználó, tempódetekálást megvalósító prototípus elkészítése, amely a zene tempóját az LCD kijelzőn megjeleníti.

A hallgató feladatának a következőkre kell kiterjednie:

- Ismerje meg az ütés- és tempódetektálás alapvető módszereit az irodalom alapján. Térképezze fel az elérhető tempódetektáló alkalmazások tulajdonságait, képességeit.
- Valósítson meg MATLAB környezetben különböző tempódetektáló módszereket, vagy egy kiválasztott módszer variációit, ill. vizsgálja meg, hogyan befolyásolják a módszerek paraméterei (pl. ablakméret) a felismerési pontosságot és a késleltetést.
- Tesztelje az EFM32GG-STK3700 fejlesztői kártya AD és DA átalakítóját, ill. valósítsa meg a szintillesztést a mikrofonerősítő kimenete és az AD átalakító bemenete között.
- A MATLAB környezetben fejlesztett algoritmusok közül válassza ki, melyik felel meg az EFM32 Giant Gecko mikrokontroller számítási kapacitásának, és ezt implementálja a fejlesztőkártyán.
- A mikrokontrolleren implementált algoritmust ugyanazon algoritmus MATLAB változatával verifikálja, a mikrokontroller memóriájába feltöltött hangminták felhasználásával.
- Az elkészült prototípust valós környezetben tesztelje.

Tanszéki konzulens: Dr. Bank Balázs, docens

Budapest, 2020. október 7.

……..
Dr. Dabóczi Tamás
tanszékvezető
egyetemi tanár, DSc

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

1117 Budapest, Magyar Tudósok krt. 2. I. ép. I.E.444.
Telefon: 463-2057, Fax: 463-4112
http://www.mit.bme.hu • e-mail: mitadm@mit.bme.hu

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

Szilágyi Bence Ágoston

TEMPÓDETEKTÁLÁS EFM32 GIANT GECKO 32 BITES MIKROKONTROLLERREL

BSc Szakdolgozat

KONZULENS

Dr. Bank Balázs

BUDAPEST, 2020

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott **Szilágyi Bence Ágoston**, 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 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, 2020. 12. 11.

...…….

Szilágyi Bence Ágoston

Összefoglaló

Dobosok és ütősök esetén elengedhetetlen a tempótartás fejlesztése, és ez - ha kisebb mértékben is - más hangszeren játszó zenészek esetén is így van. Ebben nagy segítséget jelent egy olyan eszköz, amely a mikrofonjel elemzése alapján a játszott tempóról visszacsatolást ad. A feladatom egy tempódetektáló algoritmus kifejlesztése volt, mely rövid, pár másodperces zeneminták tempóját képes meghatározni, majd ezt egy általam választott beágyazott rendszeres megoldással implementáltam. A zenei hangfelvételek tempójának meghatározása egy, az ezredforduló környékétől kezdve igencsak széles körben vizsgált probléma, így számos forrást találtam, melyeket felhasználtam az algoritmusom elkészítése során. Bár a tempódetektálás feladatát mindegyik forrásban sajátos módon valósították meg, számos hasonlóság felfedezhető köztük, így például az algoritmus főbb lépési egyértelműen elkülönülnek. Az algoritmusom lépéseinek megválasztása és azok továbbfejlesztése után a paramétereket úgy kellett hangolnom, hogy a dolgozatom által kijelölt alkalmazásra, azaz dobos hangszerek által keltett ritmusok tempójának felismerésére, a lehető legnagyobb pontossággal működjön. Az algoritmus fejlesztését MATLAB-ban, a beágyazott rendszerre történő implementációt pedig C nyelven tettem. A választott eszközöm, melyen az algoritmust implementáltam egy STK3700-as Giant Gecko mikrokontroller, melyhez analóg módon egy külső mikrofont és előerősítőt illesztettem, melynek jelét a Gecko beépített ADC-je dolgozza fel. Az implementáció során a legnagyobb nehézséget a mikrokontroller fizikai korlátai szabták, pontosabban a limitált memóriaméret és számítási kapacitás. A MATLAB-ban megírt algoritmusomat oly módon kellett módosítanom, hogy a fellépő limitációk mellett is megfelelő működést biztosítson. Az elkészült eszközöm dobosok számára gyors, és kellően sűrű visszacsatolást nyújt a játszott zene tempójáról; azaz minden egyes másodpercben a legutóbbi 3 másodpercre vonatkozóan. Tiszta hangzás esetén kielégítő pontossággal és biztonsággal működik, háttérzaj esetén azonban a pontosság csökken. Az eszközt dobos hangminták feldolgozására konfiguráltam, így a rövid, hangos ütéseket tartalmazó zenék tempóját képes a legnagyobb pontossággal meghatározni. Az eszköz autonóm működésű, nem igényel felhasználói beavatkozást, valamint elemről is működtethető, tehát könnyen hordozható.

Abstract

In the case of drummers and percussion players, it is essential to improve the tempo, and this is true, albeit to a lesser extent, of musicians playing other instruments. A tool that provides feedback on the pace played based on the analysis of the microphone signal is a great help in this. My task was to develop a tempo detection algorithm that can determine the tempo of short, few-second long music samples, and then implement this with an embedded system. Determining the tempo of musical recordings is a problem that has been studied quite extensively since the turn of the millennium, so have I found several sources that I could use to create my algorithm. Although the task of tempo detection was implemented in a specific way in each source, several similarities can be discovered between them, so that, for example, the main steps of the algorithm are clearly separated. After choosing the steps of my algorithm and further developing them, I had to tune the parameters to work with the highest possible accuracy for the application selected by my dissertation, i.e., to recognize the tempo of the rhythms produced by drum instruments. I developed the algorithm in MATLAB and the implementation on the embedded system in C language. My chosen device on which I implemented the algorithm was an STK3700 Giant Gecko microcontroller, to which I connected an external microphone and a preamplifier, the signal of which is processed by Gecko's built-in ADC. During implementation, the greatest difficulty was due to the physical limitations of the microcontroller, more specifically the limited memory size and computing capacity. I had to modify my algorithm written in MATLAB to ensure proper operation even with the limitations that occur. My finished device is suitable for a drummer to provide fast and sufficiently dense feedback on the pace of the music played; that is, in each second for the last 3 seconds. It works with satisfactory accuracy and safety if the noise level is low, although accuracy is reduced with the occurrence of background noise. I have configured the device to process drum sound samples so that it can determine the tempo of music with short, loud beats with the utmost accuracy. The device operates autonomously, does not require user intervention, and can also be operated from a battery, so it is easy to carry.

1 Bevezetés

Dobosok és ütősök esetén elengedhetetlen a tempótartás fejlesztése, és ez - ha kisebb mértékben is - más hangszeren játszó zenészek esetén is így van. Ebben nagy segítséget jelentene egy olyan eszköz, amely a mikrofonjel elemzése alapján a játszott tempóról visszacsatolást adna.

Konkrétabban megfogalmazva a feladatom egy tempódetektáló algoritmus kifejlesztése, mely rövid, pár másodperces zenemintáknak képes a tempóját meghatározni. A zenei hangfelvételek tempójának meghatározása egy, az ezredforduló környékétől kezdve igencsak széles körben vizsgált probléma [1]. A tempódetektálás olyan területeken alkalmazható, mint például több hangsáv szinkronizálása egyidejű lejátszáshoz, ütés-szinkronizált audio effektek [2], vagy zeneszámok automatikusan történő loop-olása (magába hurkolt lejátszása). Persze ez csak pár felhozott példa, az alkalmazhatóság köre szélesebben kiterjed.

Ezt az algoritmust úgy kellett megírnom a saját alkalmazásomhoz, hogy könnyen tesztelhető legyen különböző hangmintákra, illetve lehetőség legyen azokra történő hangolására; magyarán könnyen paraméterezhető legyen. Ezután pedig a paramétereit úgy kellett beállítanom, hogy a számomra kijelölt alkalmazás során megfelelő működést biztosítson. Mivel a célul kitűzött felhasználói réteg a dobosok és ütős hangszeren játszó zenészek, ezért az algoritmusnak ilyen hangszerek által keltett zenei ritmusok hangmintáira kell jól működnie.

Feladatom volt továbbá ezt az alkalmazást egy beágyazott rendszerbe ültetnem. A beágyazott rendszerek jellemző tulajdonsága, hogy intenzív fizikai és információs kapcsolatban állnak a környezetükkel, és autonóm működésűek. Ezeknek a jellemzőknek az általam előállított eszközre is igaznak kell lenniük. Fizikai kapcsolatban kell az eszközöm álljon a környezetével, mivel folyamatosan szükséges mikrofon segítségével hangmintákat vegyen, amiből később meg tudja határozni annak tempóját. Információs kapcsolatban kell álljon a környezetével, mivel az algoritmus eredményeként számolt tempót folyamatosan ki kell jeleznie a felhasználó számára. És mindezt autonóm működve kell tegye, bármilyen felhasználói vagy külső számítógépes beavatkozás igénybevétele nélkül.

1.1 Zeneelméleti alapfogalmak

Annak érdekében, hogy a későbbiekben a tempódetektálást és hozzá kapcsolódó témaköröket tárgyalni tudjam, szükséges néhány zeneelméleti fogalmat tisztázni. Ezeket a következő bekezdésekben félkövérrel jelölöm, forrásként Siska Ádám interneten közzétett cikkét használtam fel [3].

Ritmusérték alatt az egyes konkrét hangok időbeli terjedelmét értjük. Az általunk vett alapidőtartam hosszát **egész hang**nak nevezzük. A további fő ritmusértékek az egész hang tört részei, jellemzően annak páros osztással vett hányadai, tehát: *fél, negyed, nyolcad, tizenhatod* stb. hang. Ezek mellett persze tetszőleges egész számú részre felbonthatjuk a ritmusértékeinket, pl.: *triola* - hármas osztás, *kvintola* - ötös osztás, *szextola* - hatos osztás stb. Megjegyzendő, hogy ezek a felosztások tetszőlegesen sokszor egymásba is ágyazhatóak. Például gyakran beszélünk olyan hangról, melynek hossza a negyedhang hosszának harmadával egyenlő.

Ahogy minden más kommunikációs csatorna esetében, úgy a zenében is rendkívül fontos a tagolás. A legtöbb zenében hangsúlyos és hangsúlytalan elemek követik egymást, valamilyen rendszer szerint. Egy ilyen periodikus egység az **ütem**. Az **ütemmutató** egy törtszám, amely a zene metrikus alapbeosztását mutatja. Megadása úgy történik, hogy megmondjuk, hogy milyen ritmusérték adja az ütem alapegységét, továbbá megmondjuk azt is, hogy ebből az alapidőtartamból hány darab adja ki az ütem teljes hosszát. Így például a négy negyedes ütem négy darab negyedhangból áll, a hat nyolcados ütem pedig hat darab nyolcadból.

A **tempó** egy ritmusérték hosszát időben vett mértékegységhez rendeli. Tulajdonképpen nem más, mint egy utasítás arra, hogy egységnyi idő alatt hány egyforma ritmusértéket kell játszanunk. Az egységnyi idő általában a perc. A tempó mértékegysége leggyakrabban a **BPM**, azaz **Beats Per Minute** – szó szerint: ütések száma percenként. Az ütések ebben az esetben az ütemmutató második számát jelentik, azaz az ütem alapegységét. Így pl. a 145 BPM-es tempó négy negyedes ütemmutató esetében azt jelenti, hogy egy perc alatt 145 negyedet kell játszanunk. Nagyon ritkán változik egy zeneszámon belül a tempó, így általánosságban a zeneszámokhoz hozzárendelhető egy BPM mennyiség.

1.2 Irodalmi áttekintés

Több, a BME hallgatói által íródott szakdolgozatot és diplomatervet olvastam, amelyekben foglalkoztak a zenei tempódetektálás témájával [4][5][6], továbbá interneten keresztül találtam különböző szakmai cikkeket is. Bár a tempódetektálás feladatát mindegyik forrásban sajátos módon valósították meg, számos hasonlóság felfedezhető köztük.

Ha egy módszer több, egymástól független alkalmazásban is jónak bizonyult, az jó jel arra, hogy megbízható eleme az algoritmusnak. Ezeknek a megoldásoknak a felfedezése nagy segítséget nyújtott az algoritmusom elkészítése során, és én is alkalmaztam őket.

A források szinte mindegyike megegyezik abban, hogy a tempódetektáláshoz vizsgált jelben először is az ütések időben definiált helyeit határozzák meg, azonban ennek módjai közt már jelentős eltérések vannak a források közt. Az egyetemi szakdolgozat és diplomamunkák közt jellemző volt a bemeneti jel abszolútértékének burkológörbéjét képezni, mint például a [6] forrásban; így a szűrt jel deriváltján jól láthatóan elkülönülve megjelennek az ütések, melyeknek szoftveres algoritmussal való detektálása már könnyebb feladat, egy szigorított maximumkeresés futtatható utána a jelen. Más forrásokban viszont más módszereket is találtam az ütéshelyek detektálására. Az [1] forrásban például az ütések tranziensének vizsgálatával állapította meg a szerző azoknak időben vett helyeit, a [8] forrásban pedig spektrális analízis segítségével. A [7] forrásban számos alternatíva olvasható az ütésdetektálásra, az előbb felsoroltak is megtalálhatók benne, a cikkben alkalmazott ütésdetektáló módszer pedig ezeknek keveréke. Én a felsorolt módszerek közül hallgatótársaimmal megegyező megoldást választottam, azaz a vizsgált jel abszolútértékének burkológörbéjén kerestem ütéshelyeket.

A tempódetektálás egyértelműen elkülöníthető második fázisa a már megtalált ütéshelyek felhasználásával azok közti domináns periodicitás megtalálása. Bár a konkrét módszer eltérő lehet források közt, lényegében mégis mindegyik hasonló eredményt ér el. Például a [7] és [8] forrásokban autokorrelációs függvény segítségével számítják ki a szerzők a zenei mintában szereplő periodicitást. Többen ezt úgy tették meg, hogy megvizsgálták az ütések közt eltelt időtartamok gyakoriságát, és ebből egy hisztogramot állítottak elő; mint például a [4] forrásban is. A [7] forrás pedig a zene periodicitására

jellemző adatstruktúrát bár nem hisztogramnak nevezi, hanem periodicitás-detektáló függvénynek, lényegében ugyanazt az információt tartalmazza, és a feldolgozása is hasonló eljárást igényel. Én is a hisztogramos módszert alkalmaztam.

A hisztogram további feldolgozására pedig többen alkalmaztak valamilyen simító eljárást, például Hann-ablak segítségével [4], esetleg az ütéstávolságok előfordulásának valamilyen súlyozását [7], hogy a zenei játékban fellépő pontatlanságokat kiküszöböljék. Az így előállt görbe segítségével, bár több opció is fent áll, valamilyen egyszerű lépéssel már meghatározható a bemeneti hangminta tempója.

Összességében kijelenthető, hogy a források alapján a tempódetektáló algoritmusok egyértelműen elkülöníthető lépései a zenei mintában az ütéshelyek felfedezése, majd ez után az azok közti időtávolságok alapján a fellépő periodicitások meghatározása, végül pedig ennek további feldolgozásával a minta tempójának meghatározása.

1.3 Elérhető megoldások

1.3.1 Metronóm

A legkönnyebben elérhető megoldás a tempótartás gyakorlására zenészek számára a metronóm eszközök használata. Ezek egy előre beállított tempót diktálnak a felhasználó számára, általában hangjelzések formájában, de gyakran adnak vizuális jeleket is. Segítségükkel a kijelölt tempó tartása gyakorolható, illetve az adott tempójú zene kíséretéhez egy biztos alapot ad.

Ezek számtalan alkalmazási helyen és formában léteznek, az eredeti mechanikus megoldástól kezdve a különböző beágyazott rendszerekig. Utóbbiak nagyobb mértékben paraméterezhetőek is a felhasználó által, mint például az ütések száma ütemenként, vagy a ritmus típusa (negyedek, nyolcadok, triola stb.). Példa egy ilyen eszközre a Rhythm Watch Mini [9].

1. ábra: Rhythm Watch Mini RW30 digitális metronóm

Bár a metronóm egy praktikus és széles körben alkalmazott eszköz, a jelen szakdolgozat témájául kitűzött probléma megoldására nem alkalmas, mivel a felhasználó csak egy viszonyítási alapot kap az elérni kívánt tempóhoz, nem kap visszajelzést arról, hogy ilyen alap nélkül mennyire pontosan lenne képes magától azt a tempót tartani.

1.3.2 Audio keverőpult

Az audio jelfeldolgozás egyik speciális területén keresgélve találhatunk olyan elektronikus eszközöket, melyek képesek egy zenei jel tempójának megállapítására, illetve ritmusának detektálására.

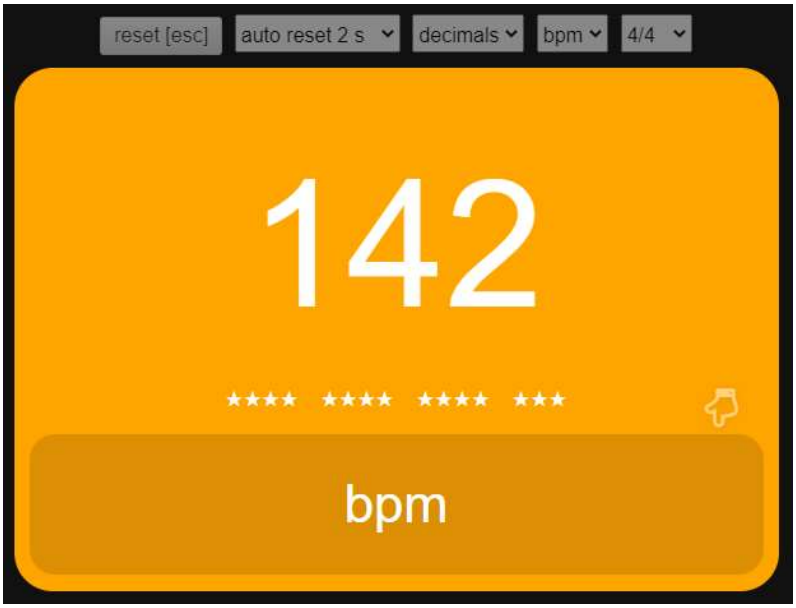
Egyes hangtechnikában alkalmazott keverőpultok például tartalmaznak beépített tempódetektáló funkciót. Egy példa erre a Pioneer SVM-1000 nevű terméke [10].

2. ábra: Pioneer SVM-1000 keverőpult

Azonban ezek a termékek természetesen jelen szakdolgozatban kitűzött célra nem alkalmazhatóak, már csak azért sem, mivel az ilyen típusok rendszerint igen magas vételárral rendelkeznek.

1.3.3 Tap BPM

Amennyiben praktikus létező megoldásokat akarunk keresni, azokat kell vizsgálni, melyek mindenki számára elérhetőek. Számtalan weboldal létezik, mely segítségével megállapítható egy egyenlő időközökkel elválasztott ütésekből álló ritmus tempója. Ezek a bizonyos „Tap BPM” oldalak nem hang alapján detektálnak tempót, ennél sokkal egyszerűbb a működésük, használatukhoz a felhasználónak kell az ütések pillanataiban egyet kattintania, illetve gombot lenyomnia.



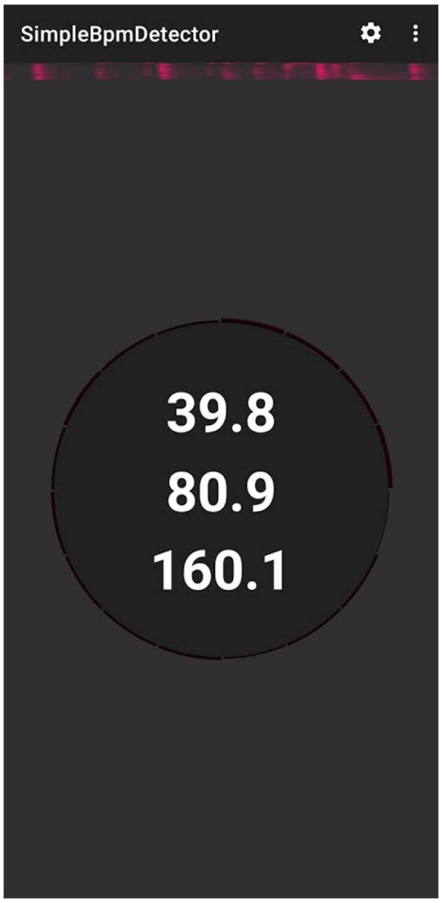
3. ábra: Tap BPM weboldal [11] kezelőfelülete

Ilyen és ehhez hasonló alkalmazások elérhetőek okostelefonokra is, azonban a metronómmal egyetemben nem tudják ellátni a kívánt feladatot. Itt a tempódetektálás folytonos fizikai kapcsolatot igényel a felhasználóval (periodikus gombnyomogatás formájában), zenélés közben pedig ez természetesen nem elvárható.

1.3.4 Tempódetektáló applikáció

Olyan alkalmazások is találhatóak, melyek többet nyújtanak, mint egy, az előző pontban részletezett „Tap BPM" applikáció. Ezek már jobban hasonlítanak az általam megvalósítani kívánt eszközre, mivel mikrofon adatait használják fel tempó megállapításához.

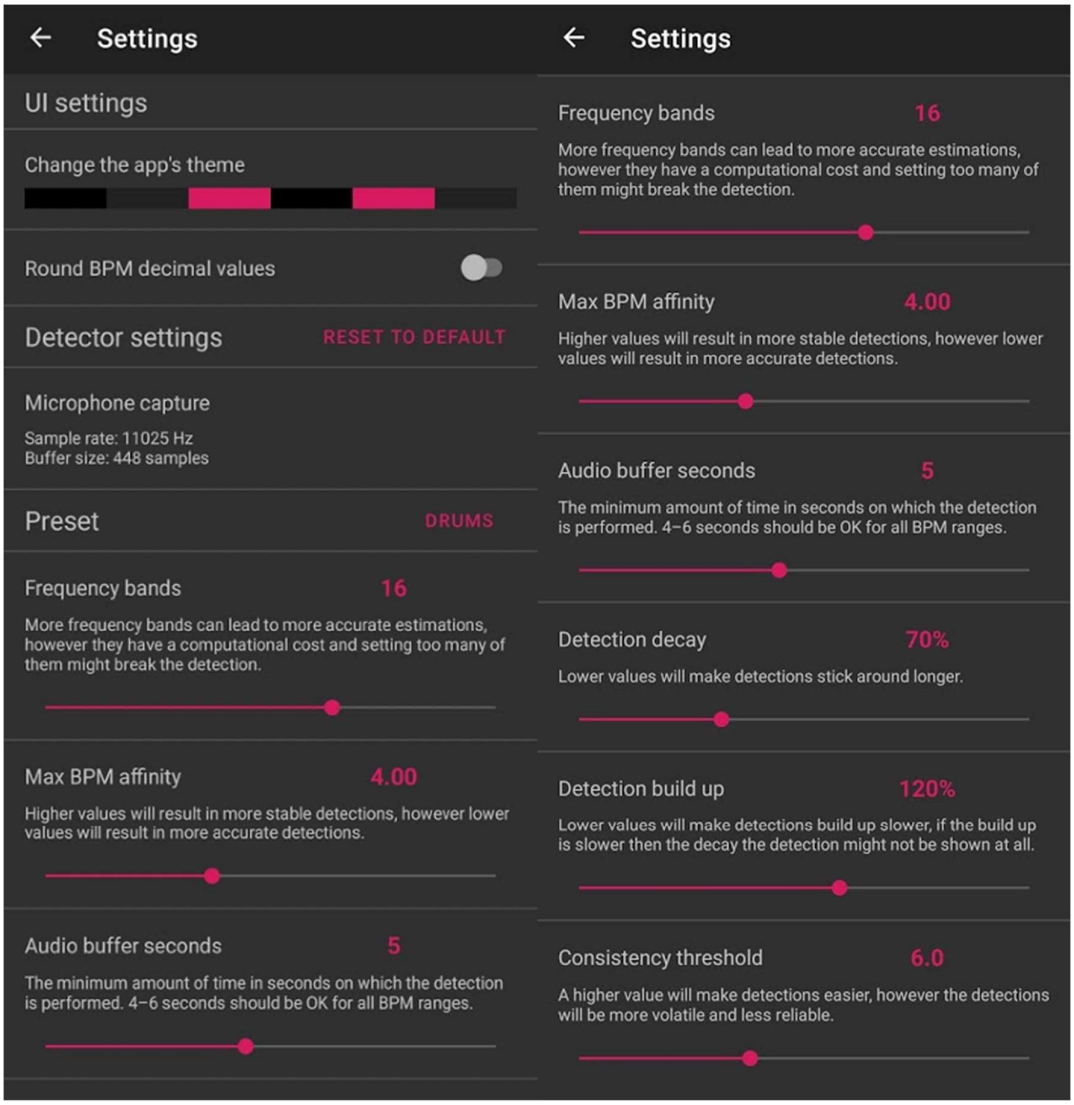
Az Android alapú mobil eszközökre telepíthető applikációk tárházában, a Google Play Store-ban a legnépszerűbb ilyen alkalmazás a Simple BPM Detector [12]. Működés közben, valamint egy 80 BPM tempójú dobos szólót a telefon közelében lejátszva így néz ki:



4. ábra: Simple BPM Detector használat közben

Jól látható, hogy az alkalmazás nem csupán egy, hanem három tempót is megállapít a vizsgált hangmintára. S bár a zene valós tempója 80 BPM (azaz a negyedek által meghatározott tempó), kijelez ezen kívül 40 és 160 BPM-hez közeli értékeket is. Ennek oka az, hogy a zenében folyamatos nyolcados cintányérhang hallható, ami a negyedekhez képest kétszeres, 160 BPM-es tempót eredményezi. Továbbá a szóló leghangosabb ütései, a pergődobos hangok csupán minden második negyednél szólalnak meg, félhangonként. Ez utóbbi eredményezi a negyedekhez képest feleakkora, 40 BPM-es tempót.

Az én eszközöm azonban csak egy értéket jelez ki kimenetként, így az algoritmusomat úgy kellett terveznem, hogy az az egy is kielégítő eredményt nyújtson.



5. ábra: Simple BPM Detector beállítások ablaka

A Simple BPM Detector-ra és a vele hasonló alkalmazásokra általánosságban igaz, hogy bár részletesen paraméterezhetőek, tapasztalataim alapján ezzel párhuzamosan a paraméterek helyes megválasztását a felhasználóra bízzák, nem nyújtanak optimális beállításokat, legfeljebb 1-2 preset beállítást.

A vizsgált alkalmazás az enyémtől eltérő beállítható paramétekkel rendelkezik, meg is figyelhető ezek alapján, hogy az algoritmusa is különbözik attól, amit én megvalósítottam. Ez is egy példa arra, hogy a tempódetektáló módszerekre számos alternatíva létezik.

A fentiekkel párhuzamosan az is megállapítható, hogy a feladatkiírásomnak megfelelő alkalmazásra ezek a létező telefonos szoftverek kevésbé használhatóak, mivel

általánosan kijelenthető, hogy túl nagy késleltetéssel képesek csak a zene tempóját megállapítani, így nem tudnak időben visszajelzést nyújtani a dobos számára.

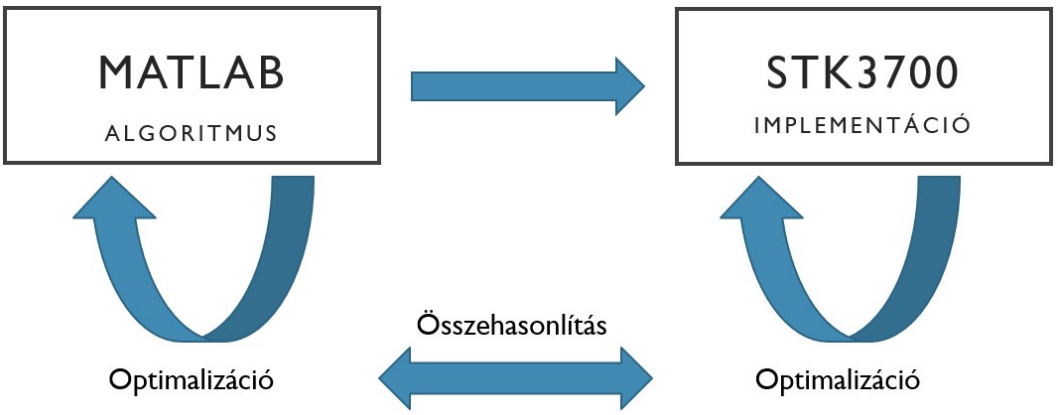
1.4 A dolgozat célja

A szakdolgozatom eredményeképp előállt eszköznek olyannak kell lennie, amely viszonylag olcsó elemekből épül fel, hordozható és felhasználói beavatkozás nélkül üzembe helyezhető. Az eszköz esztétikájának növelése nem része a feladatomnak.

Az eszközömnek a feladat kiírásának és a bevezetőben megfogalmazottaknak megfelelően kell működnie, azaz visszacsatolást adnia a dobos zenész számára a játszott zene tempójáról, egész számú BPM-ben kifejezve, valós időben. A tempódetektálás minél pontosabb kell legyen, valamint minél gyakoribb visszajelzést kell adjon, hogy a dobos annak megfelelően tudjon korrigálni a játékán. A rendszer paraméterei úgy kell beállítva legyenek, hogy a program a valós használatban várható bemeneti értékek esetén adjon pontos eredményt.

A program futása során esélyes, hogy a valós játszott tempónak egész számú többszörösét, illetve hányadát találja meg. Azaz a negyedek közti ütéstávolságok helyett pl. a nyolcados, tizenhatodos, vagy esetleg félhangos időközöket állapítja dominánsnak a hangmintában. Bár ez nem ideális, nem minősül rossz működésnek, mivel tulajdonképpen azok is ugyanannak a játszási sebességnek felelnek meg a dobos szemszögéből; a dobos tudni fogja, hogy sikerül-e tartania a kívánt tempóját. Például, ha a zenész a játszott ritmusában minden egyes nyolcadot hangsúlyozva játszik, akkor nem meglepő (a dobos számára sem), ha a negyedes tempó kétszeresét jelzi ki az eszköz.

1.5 Megvalósítás menete



6. ábra: A feladat elkészítésének diagramja

Az algoritmus fejlesztését MATLAB-ban, a beágyazott rendszerre történő implementációt pedig C nyelven tettem.

A megvalósítási folyamat része volt az algoritmus fejlesztése, illetve az implementáció közben a folyamatos optimalizáció, hogy az előállt eredmény minél jobban illeszkedjen az elérni kívánt célhoz. Ez alatt azt értem, hogy a tempódetektáló algoritmus minél pontosabb legyen, valamint a választott fejlesztőkártya fizikai limitációin belül is megfelelő működésre legyen képes. Utóbbiakat részletesen tárgyalom a 3.5. fejezetben. Ehhez természetesen a két felületen írt programokat folyamatosan össze is kellett hasonlítanom egymással, a fejlesztési folyamat nem lineárisan ment végbe, sok tesztelést, visszacsatolást és újratervezést igényelt.

A tesztelésnek fontos szerepe volt a fejlesztési folyamat egésze alatt. Mind a MATLAB-os tervezői fázisban, mind az implementációs fázisban, mind pedig az eszközöm és programom végleges formájának előállása után teszteket kellett végeznem, hogy megbizonyosodjak a helyes működéséről.

2 Algoritmus fejlesztése

Az algoritmusom fejlesztését MATLAB környezetben végeztem. Az algoritmusom lépéseinek kialakításához felhasználtam az 1.2. fejezetben leírtakat, melyeket a szakdolgozatom alkalmazásához szabtam egyes helyeken. A kódomhoz sem kiindulási alapként, sem később nem emeltem át máshonnan részleteket, így a kódolást teljes mértékben én végeztem.

2.1 MATLAB fejlesztőkörnyezet bemutatása

A MATLAB [13] egy modern, interaktív programrendszer, amely tudományos és mérnöki- matematikai számítások elősegítésére készült. Gyártója a The MathWorks Inc. (USA). Nevét a MATrix LABoratory rövidítéséből kapta. Eredetileg FORTRAN nyelven készült, majd később teljes egészében átírták C nyelvre és állandó tökéletesítésének köszönhetően jelenleg numerikus számítások területén a világ egyik legjobb terméke.

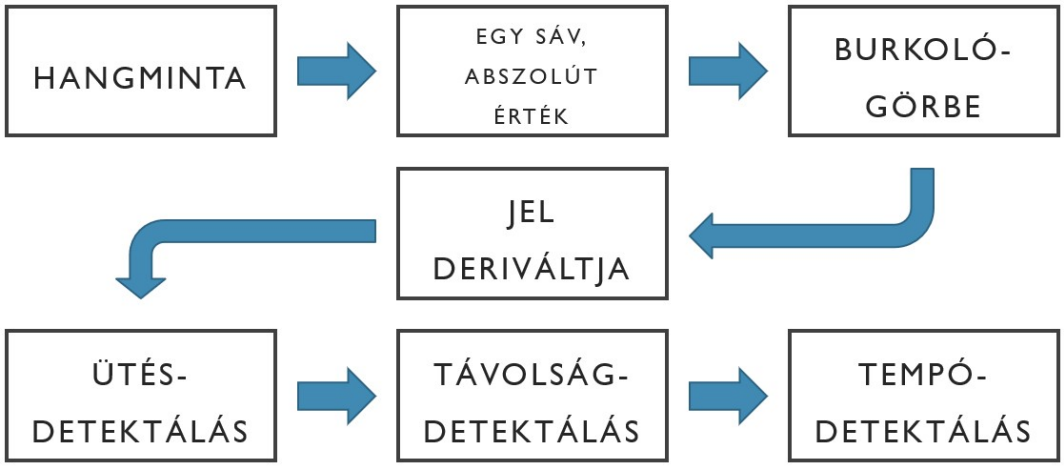
A MATLAB egy magasszintű programozási nyelv. Parancsszavai a matematikai képletek és függvények értelemszerű alkalmazásából adódnak. Parancsvezérelt üzemmódban, vagyis interpreterként dolgozik. Parancsainak száma nagy, de megtanulásuk könnyű, alkalmazásuk egyszerű, elegáns. A rögtön figyelmeztető hibajelzések és a jól megalkotott Help-menük a parancsokban, vagy összeférhetetlenségek, vagy hibák gyors kiszűrését eredményezik. A remek két- és háromdimenziós grafikus megjelenítési képességek és a Windows környezet külön előnyöket szolgáltat a felhasználó számára.

A MATLAB egy "nyitott" programcsomag, mert lehetőséget biztosít újabb funkciók (parancsok) beépítésére ún. M-file-ok vagy ún. MEX-file-ok formájában. Az M-file-ok *.m kiterjesztésű szövegfile-ok, amelyek MATLAB parancsokat vagy egyéb, a felhasználó által definiált függvényeket tartalmazhatnak. Ilyen specifikus (applikációs) feladatok megoldására íródtak a kiegészítő programok, az ún. Toolbox-ok. A MEX-file-ok Fortran- vagy C nyelvű, MATLAB parancsra lefordított *.exe file-ok, amelyek a továbbiakban úgy viselkednek, mint a MATLAB gyárilag megírt rutinjai. A MATLAB programcsomagról egy részletesebb alapvető ismertető a [14] forrásnál megjelölt weboldalon olvasható.

A fentiek alapján számomra megfelelő választás volt a feladat megvalósítását végző algoritmust MATLAB környezetben fejleszteni. A MATLAB, és az általam továbbá felhasznált Signal Processing Toolbox tökéletes ehhez hasonló jelfeldolgozási folyamatok megírására. Segítségével könnyen kezelhetőek a hosszú tömbökbe foglalt változók, számos beépített függvény könnyíti meg a tervezési folyamatot, valamint nagyon könnyű azonnali visszajelzést kapni az algoritmus működéséről. Utóbbi annak is köszönhető, hogy könnyű ábrázolni a tárolt, s kiszámolt adatsorokat; én is sok ábrázolást használtam a fejlesztés során.

Az egyetemi tanulmányaim során több tárgy keretén belül is használtam MATLAB-ot, így szerencsére nem okozott különös nehézséget a használata a szakdolgozatom elkészítése során sem.

2.2 A tempódetektáló algoritmus



7. ábra: A tempófelismerő algoritmus blokkvázlata

Az algoritmus felépítése az itt látható főbb lépésekre különíthető el, melyek szekvenciálisan kerülnek végrehajtásra. A folyamat az 1.2. fejezetben leírtak alapján állt elő, a megismert, jól elkülönülő lépések alapján.

Ahogy korábban is írtam, a tesztelésnek fontos szerepe volt a fejlesztési folyamat egésze alatt. Az algoritmust az alkalmazásomhoz szabtam, a folyamatos tesztelés során úgy módosítottam rajta részleteket, hogy a lehető legjobb eredményeket adja dobos bemeneti hangminták esetén. Példa erre, hogy az ütésdetektálás lépésben a valódi ütéshelyek detektálása, azaz a derivált jelben található összes lokális maximum szűrésének módja is így állt elő.

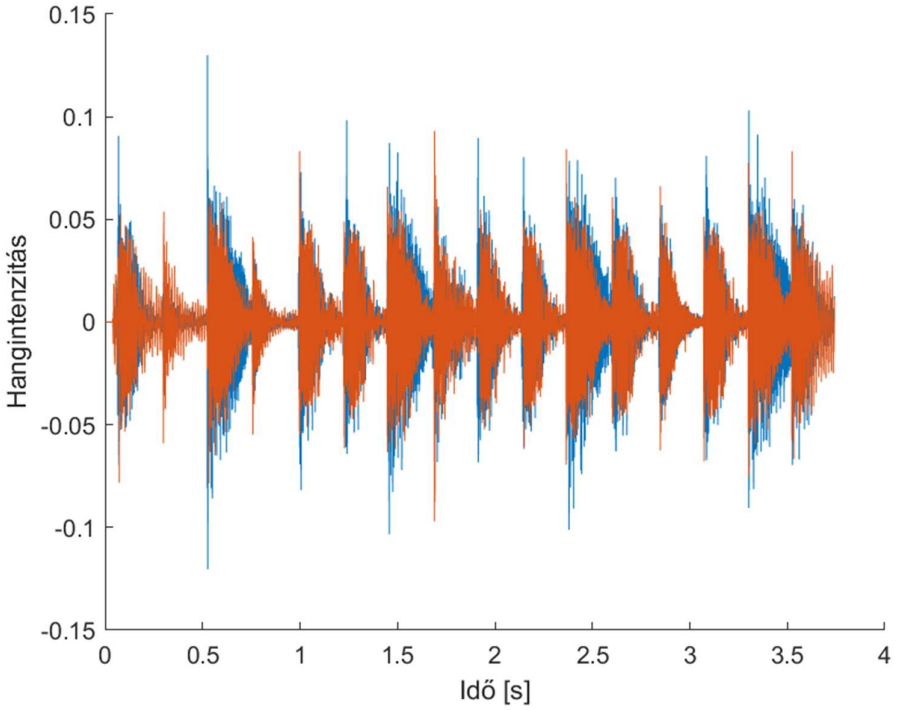
Az algoritmus által felhasznált paraméterek értékei is tesztelés segítségével álltak elő, többnyire heurisztikus módon, majd újratesztelés során manuálisan próbáltam velük a hibákat korrigálni.

Az algoritmus lépéseit egyenként, részletesen bemutatom.

2.2.1 Hangminta betöltése

Az algoritmus kiindulási pontja egy hangminta, jelen alkalmazásban néhány másodpercnyi, amin a jelfeldolgozási folyamatot el szeretnénk végezni, és a tempóját meghatározni. Munkám során vegyesen használtam különböző, internetről letöltött hanganyagokat erre a célra, illetve saját magam által, mikrofonnal felvett mintákat is. Mivel az elkészült projektet elsősorban dobosok által való felhasználásra szánom, ezért a minták túlnyomó többsége tisztán dobos ütésekből állt, más hangszer hangját nem tartalmazta, valamint zajt is csak nagyon kis mértékben.

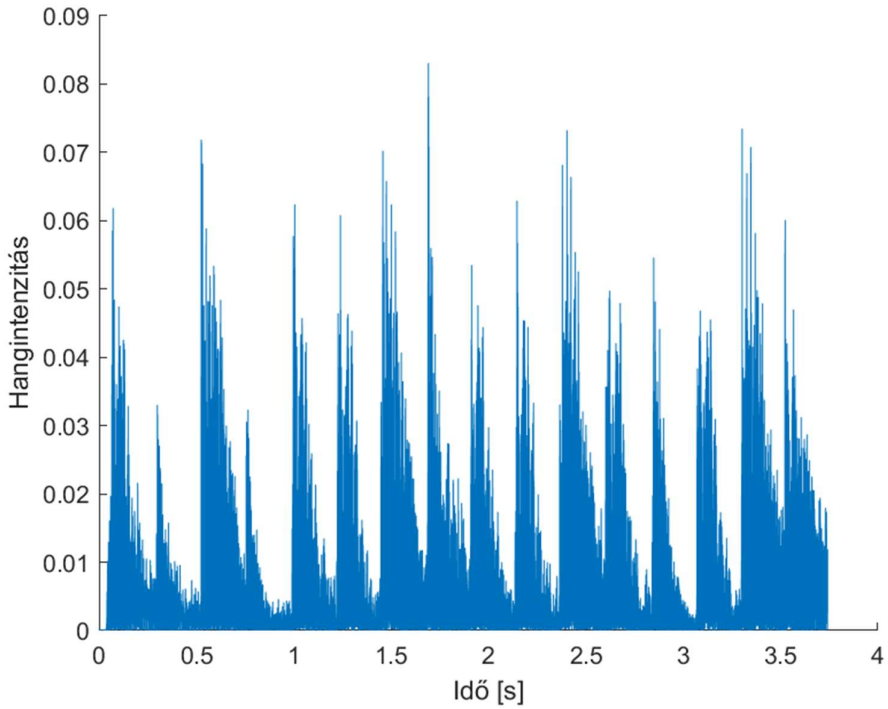
Az algoritmus bemutatása alatt használt példa: Blur – Song 2 című zeneszám [15]. Ennek az első majdnem 4 másodpercét használtam itt fel, mely 2 ütemnyi dobos szólót tartalmaz. Tempója: 130 BPM



8. ábra: Bemenő kétsávos hangminta

2.2.2 Abszolútérték képzés

Mivel a felhasznált hangminták mind kétsávosak voltak, így azok elemenkénti átlagolásával első lépésként létrehoztam egy mono hangsávot. Ennek ezután abszolút értékét képeztem, mivel minden esetben a hangintenzitás abszolút nagyságára vagyunk kíváncsiak.

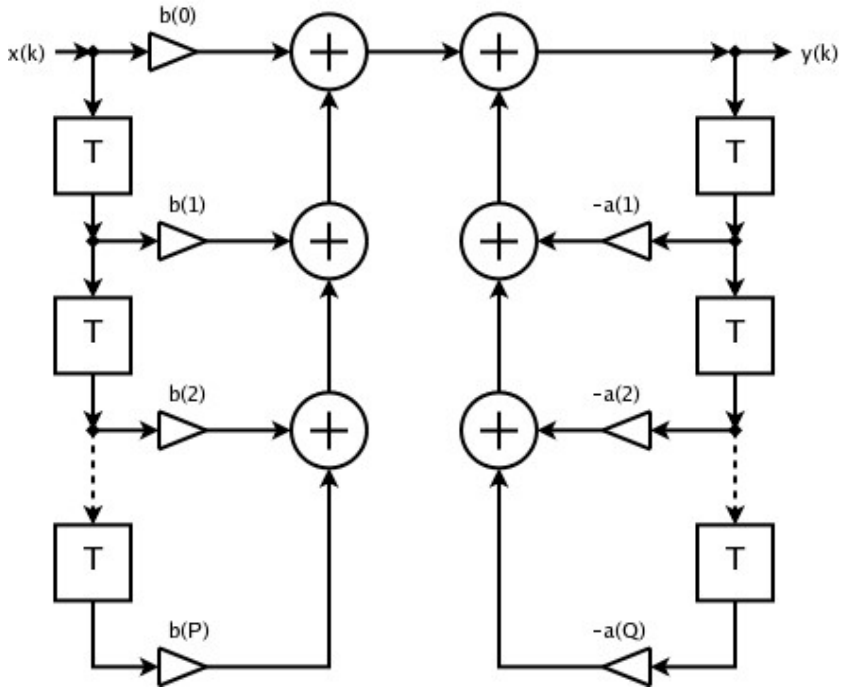


9. ábra: Mono hangsáv abszolút értéke

2.2.3 Burkológörbe előállítása

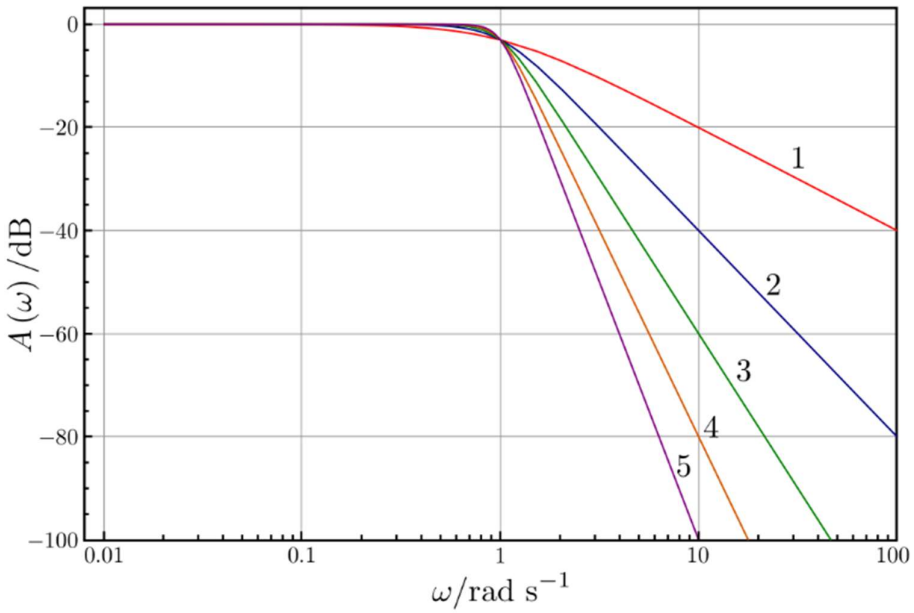
Hogy a hangmintában a hangszeres ütések időhelyeit meg tudjuk határozni, az eddigi, abszolút értékekből álló mintánknak képeznünk kell a burkológörbéjét. Így kiesnek a magas frekvenciájú összetevők, és a függvény csúcsai közt jól láthatóan elkülönülnek az ütések által keltett hangok. Ez a művelet egy aluláteresztő szűrő segítségével megvalósítható.

Mivel egy mintavett, digitális jelet kell feldolgoznunk, így egy digitális, diszkrét idejű szűrőre van szükségünk. A digitális szűrők két típusából, azaz a véges impulzusválaszú szűrők (FIR filter) és végtelen impulzusválaszú szűrők (IIR filter) közül az utóbbit választottam, mivel a felhasználás szempontjából ideális szűrőtulajdonságok elérése mellett kisebb számításigénnyel rendelkezik, ami mérvadó szempont mikrokontrolleres implementáció során [16].



10. ábra: IIR szűrő blokkdiagramja [17]

A választás a diszkrét idejű, végtelen impulzusválaszú szűrők közül a Butterworth szűrőre esett. Bár a szűrő amplitúdómenete a töréspontí frekvencia közelében nem rendelkezik nagy meredekséggel, áteresztő tartományban amplitúdóhelyesen viszi át a frekvenciakomponenseket [18]. Megjegyzendő, hogy ez az alkalmazás esetében bármelyik IIR szűrőtípus ideális választás lett volna, mivel ebben az alkalmazásban nem kritikus a frekvenciamenet: csak csúcsokat keresünk a jelben.



11. ábra: Különböző fokszámú Butterworth szűrők amplitúdómenete [19]

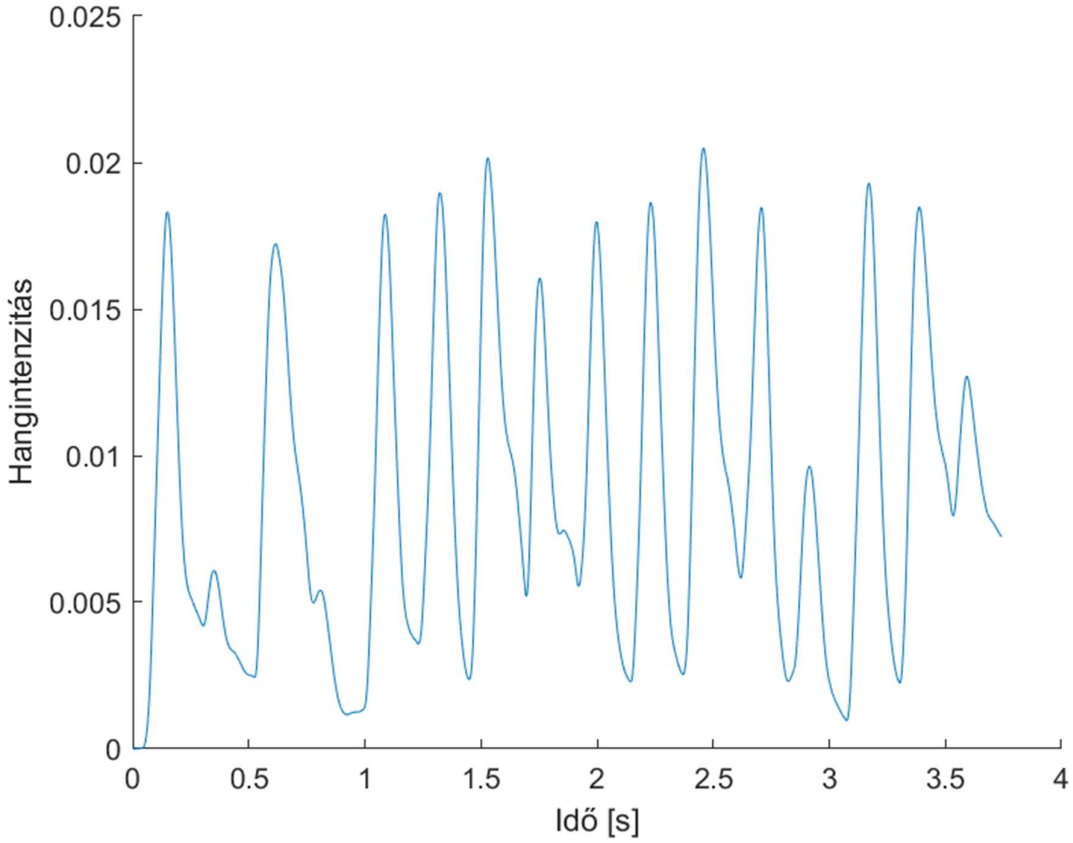
A szűrő fokszámát 3-nak, törésponti frekvenciáját pedig 8 Hz-nek választottam meg. Ezek az értékek próbálgatássorozatok eredményeként álltak elő, mivel ezek mellett álltak elő a legkielégítőbb eredmények. Ez alatt azt értem, hogy ezek az értékek mellett jól elkülöníthetően megjelentek a burkológörbén az ütések. Azok amplitúdói nem torzultak túlzott mértékben (mint túl alacsony törésponti frekvencia esetén), valamint nem jelentek meg további csúcsok, melyek befolyásolhatják az ütéshelyek detektálását (mint túl magas törésponti frekvencia esetén).

Ennek megfelelően a szűrőm rendszeregyenlete:

$$y(n) = b_0x(n) + b_1x(n-1) + b_2x(n-2) + b_3x(n-3) - \\ - a_1y(n-1) - a_2y(n-2) - a_3y(n-3)$$

Ahol az $A = \{1, a_1, a_2, a_3\}$ és $B = \{b_0, b_1, b_2, b_3\}$ szűrőparamétereket MATLAB segítségével számoltam ki a mintavételi frekvencia ismeretében, 8 Hz-es törésponti frekvenciával.

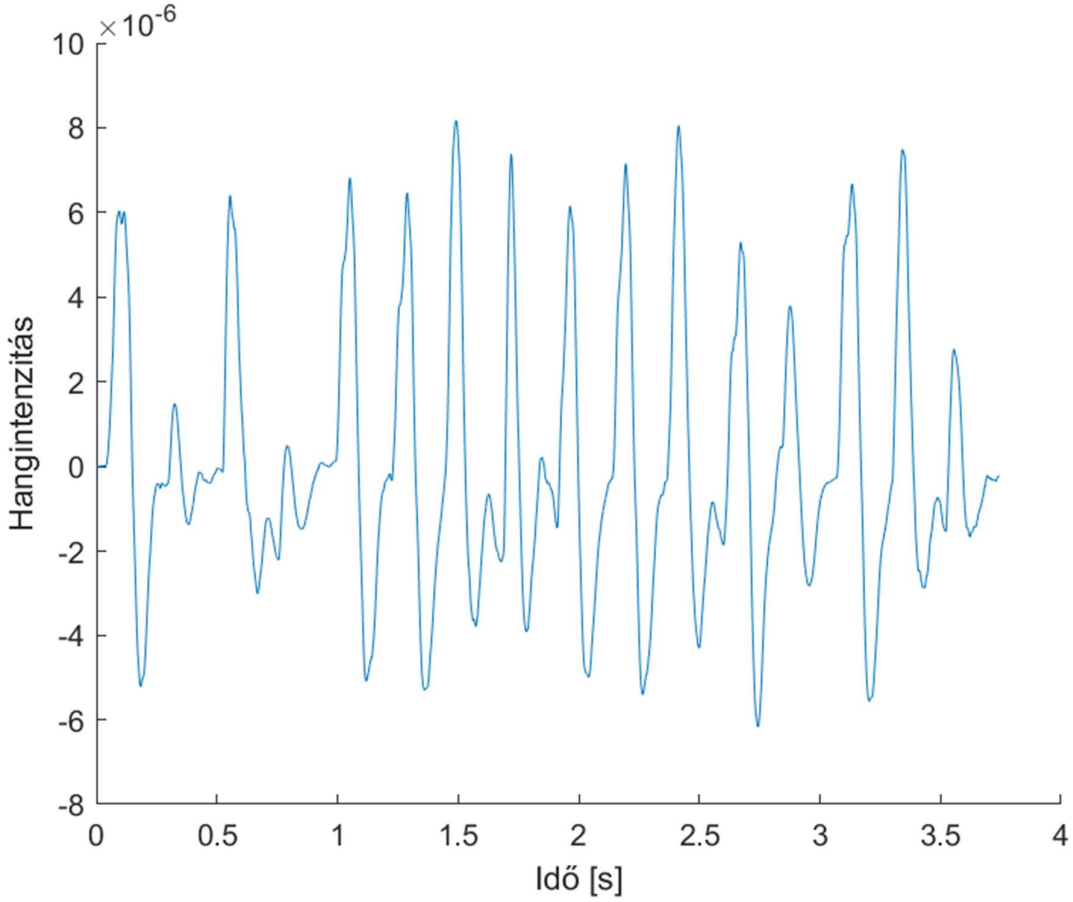
A dokumentációban példaként feldolgozott mintához tehát a következő burkológörbét állítottam elő:



12. ábra: Jel burkológörbéje

2.2.4 Jel deriválása

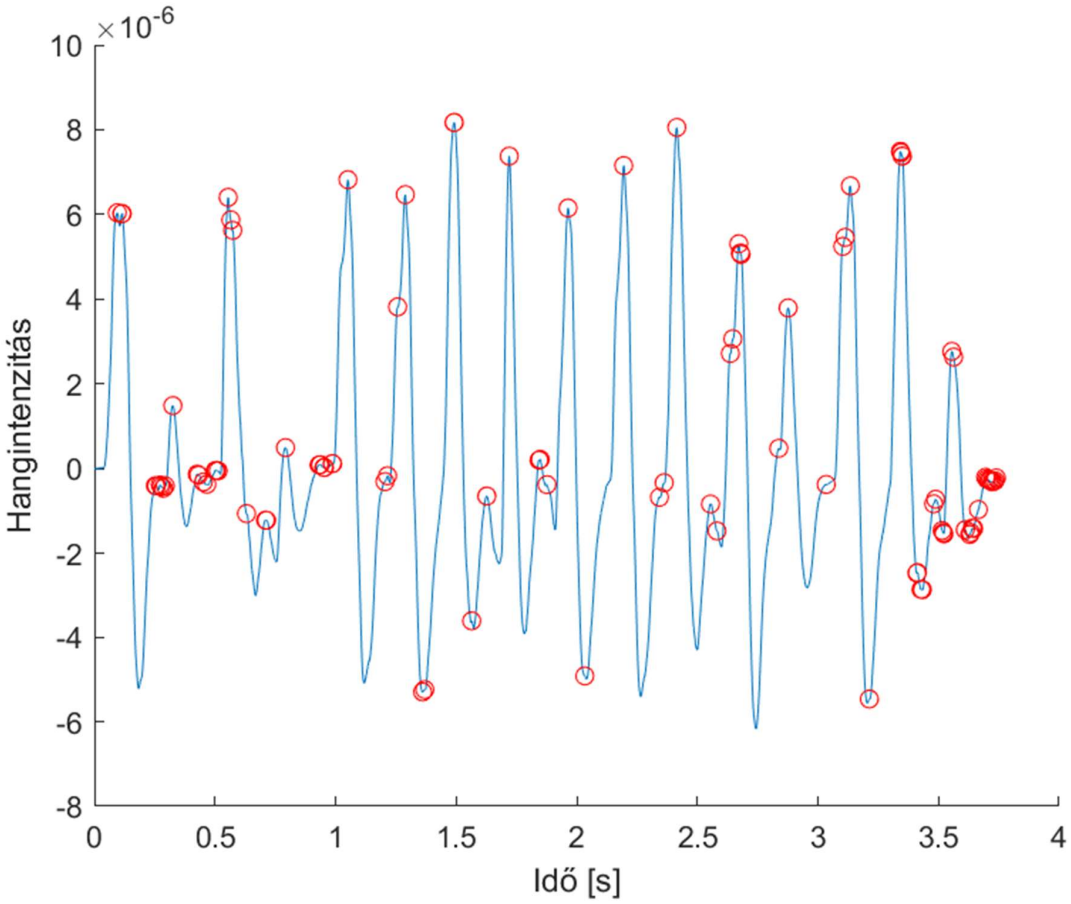
A következő lépésben a burkológörbe deriváltját kell képzelni, azon a jelen kell a csúcsokat keresni. Így pontosabban lehet az ütések helyeit meghatározni, valamint hosszabban tartó hangok esetén is pontos eredményt kapunk, mivel ily módon a hangintenzitás változásának nagyságát vizsgáljuk, mely az egyes hangok megszólalásának elején a maximális.



13. ábra: Szűrt jel deriváltja

2.2.5 Ütésdetektálás

A derivált jelen először megkerestem az összes lokális maximumot, azaz minden olyan értéket, mely nagyobb mindkét szomszédjánál.



14. ábra: Szűrt jel deriváltja, maximumok megjelölve

Látható, hogy maximumból rengeteget tartalmaz a jel, és bár tartalmazza a valós ütéshelyekhez tartozó csúcsokat is, ezeken túl rengeteg fals csúcsot is tartalmaz, amelyeket ki kell szűrni. Ezt a szűrést két lépésben értem el.

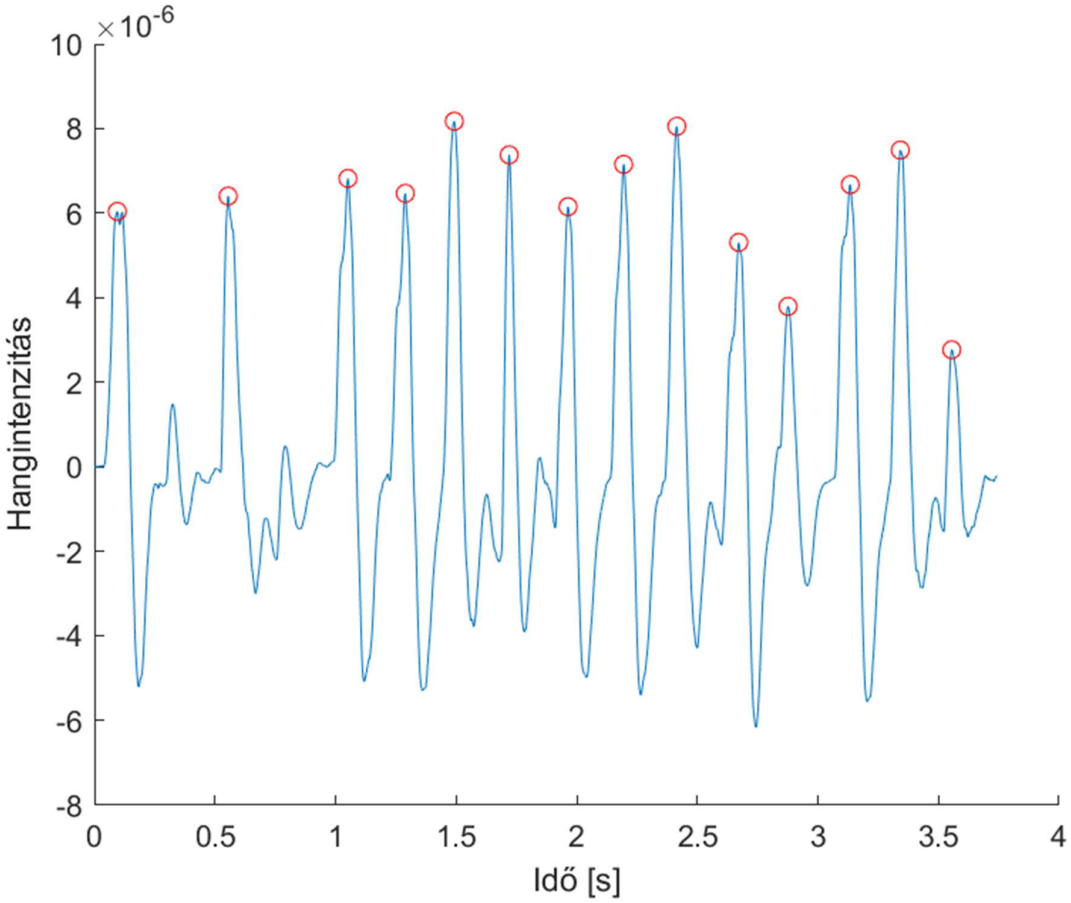
Az első lépésben paraméterként megadható egy intenzitásarány, melyet a teljes szűrt hangminta deriváltjának abszolút maximumához mérek. Annak ezen hányada meghatároz egy jelszintet, és minden az alatti lokális maximum eltávolításra kerül. Segítségével kiszűrhető a rengeteg 0 körüli és az alatti maximum. A példában az intenzitásarány konstans értéke 4.

A második lépés két paraméterrel rendelkezik: egy, mely megadja azt a maximum BPM-et (Beats Per Minute, azaz tempó), amit mérni szeretnénk, másik pedig azt a maximum ütésszámot, amennyit egy negyed (azaz Beat) alatt mérni szeretnénk. Példámban előbbi értéke 180, utóbbié 16. Ezen két paraméter és a mintavételi frekvencia segítségével kiszámolható egy minimális elfogadható ütéstávolság:

$$\text{min. ütéstáv. [minta]} = \frac{\text{Fs [Hz]} \cdot 60}{\text{max BPM [1/perc]} \cdot \text{max ütés negyedenként}}$$

Ami egy minta darabszámban kifejezett mennyiség, és azt adja meg, hogy mi az az intervallumnagyság, amelyen belül semmiképp nem léphet fel egynél több ütés, azaz csúcs a jelben. Ezt felhasználva az algoritmus ki tudja szűrni a valós csúcsok közelében található fals csúcsokat, és minden egyes szomszédos maximum-pár esetén eltávolítja az alacsonyabb hangintenzitással rendelkezőt, amennyiben azok túl közel vannak egymáshoz.

A lépések eredményeképp előáll a valós ütéshelyek helyeit tartalmazó tömb:

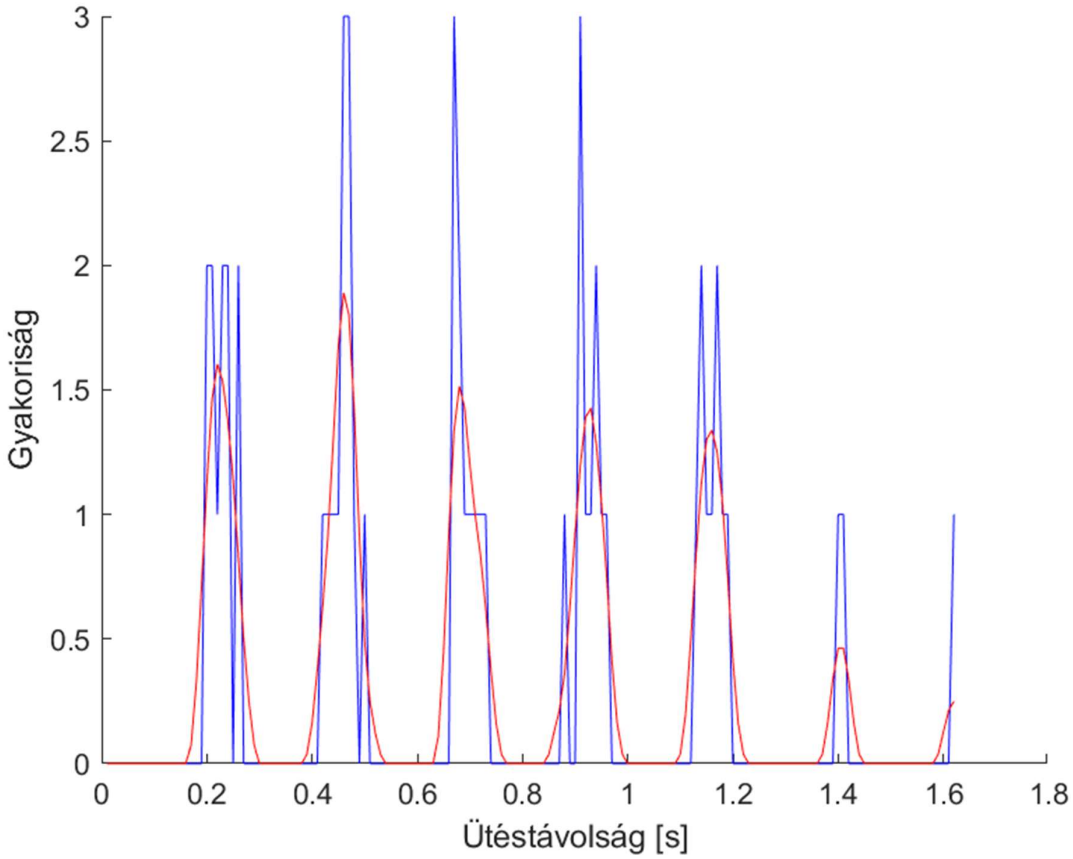


15. ábra: Szűrt jel deriváltja, ütéshelyek megjelölve

2.2.6 Távolságdetektálás

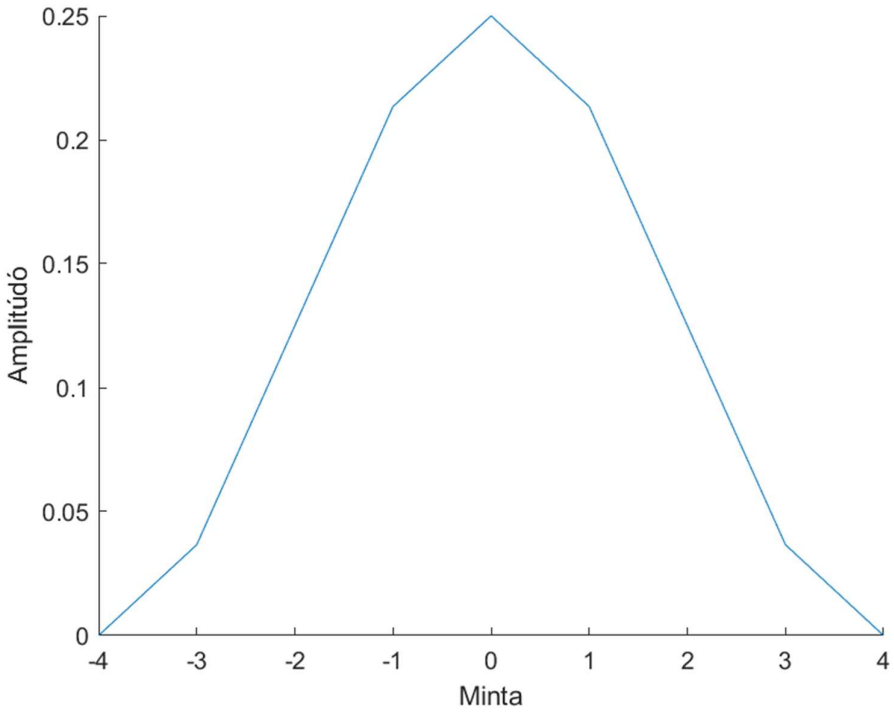
A következő lépés a már megtalált ütések időben számított helyei közötti távolságok mérése. Ennek egy jól működő módja az ütéstávolságokból egy hisztogramot előállítani, azaz az egyes ütéstávolságokhoz előfordulási gyakoriságot rendelni [1].

Az ütéstávolságok mérésének felbontása eleinte, és ebben, az algoritmus szemléltetésére szánt fejezetben: 10 ms (ezt később csökkentettem a pontosság növelése érdekében). Paraméterként szükségünk van az összehasonlítások számára, mely megmondja, hogy minden egyes ütést az őt követő hány darab ütéssel mérünk össze. Példámban ennek értéke 5.



16. ábra: Ütéstávolságok hisztogramja

Az ábrán tehát kék színnel szerepel a hisztogram. Hogyha egy zenében az ütések közt egy adott időköz pontos egész számú többszörösei lennének, akkor a hisztogramon csupán ezeken a helyeken látnánk 1-1 tüskét. Azonban látszik, hogy a zenében előforduló ütések közti időközök nem tökéletesek, mindenképpen van egy bizonyos szórásuk. Hogy ezt a hatást kompenzáljuk, a hisztogramon elvégeztem egy Hann ablakos [20] simítást, amely segítségével ezeket a pontatlanságokat kiátlagoltam. Paraméterként itt szükségünk van az ablak szélességére, példámban ennek értéke 9, azaz 0,09 s. Az ablak egységnyi területre normált alakját használtam az ábrázolás könnyítésének szempontjából, a működésre ennek nincs kihatása.



17. ábra: 9 elemű, normált területű Hann-ablak

A simítás során a hisztogram és az ablakfüggvény konvolúciója adja az új függvényt, és a következő képlettel számítható, amennyiben az ablak maximum értékét a 0 helyen veszi fel:

$$(Hann * Hisztogram)[n] = \sum_{m=-4}^{4} Hann[m] \cdot Hisztogram[n-m]$$

Ennek eredménye látható a 16. ábrán piros színnel, itt már jól láthatóan elkülönülnek az egyes ütéstávolságokhoz tartozó csúcsok – továbbá egy játszani kívánt ütéstávolsághoz csak egy csúcs tartozik. A csúcsok magassága pedig jó becslőt ad az egyes ütéstávolságok gyakoriságára a mintában.

2.2.7 Tempódetektálás

Az algoritmus kimenete a tempó értéke. Azonban a hisztogram, és annak simított változata alapján ezt többféleképpen meg lehet határozni, én két módszert próbáltam. A legegyszerűbb megoldás a simított hisztogram maximum helyének meghatározása volt. Példánk esetében a függvény maximuma 0,46 másodpercnél lép fel, innen könnyen kiszámolható, hogy a BPM = 60/0,46 = 130,4348. Ez az eredmény kielégítő, mivel közel esik a tényleges 130-as tempóhoz, egészre kerekítve azonos. Kedvező módon ez az egyszerű megoldás nyújtja a legideálisabb, azaz a legtöbb esetben helyes és pontos

eredményt. A simított hisztogramban maximum helyet keresve nem csupán azt használjuk ki, hogy az ütéstávolságok bizonyos értékek közelében csoportosulnak, hanem ezek a csoportok közül megkeressük a zenében a leggyakoribbat. Ez sok esetben a tényleges zenei tempót fogja eredményül adni, mivel a zenében található legkisebb periodicitás általában a fő tempóhoz tartozik; továbbá, ha mégsem, a tempónál gyorsabb, azaz nem hangsúlyos ütések gyakran vannak annyira halkak, hogy azokat az algoritmus nem is számítja ütésnek. Ez ismét azt fogja eredményezni, hogy a tényleges tempó ütésközei a leggyakoribbak a jelben.

Egy másik módszerrel is kísérletezgettem az algoritmus fejlesztése során, ami bár a maga céljára megfelel, nem erre van szükségem a végső algoritmusomban. Mindenesetre felvázolom ezt a módszert is. A hangmintában meghatározható az előforduló legrövidebb időközhöz tartozó tempó egy fésűs módszer segítségével. A módszer során különböző időközű fésűket illesztünk a hisztogramra, és keressük a legjobb illesztést. Pontosabban, az összes ütésköz végigpásztázásra kerül, és a simított hisztogramnak az ilyen időközönként fellépő értékeit összeadjuk – persze csak az első N darabot, ahol N egyenlő azzal a paraméterrel, ami megadta, hogy az ütéseket az őt követő hány db ütéssel mértük össze a hisztogram előállításához. Ezzel a módszerrel megállapíthatjuk, hogy melyik az a legkisebb ütések közti időköz, aminek a többszörösei is megtalálhatóak a mintában. Ezzel a második módszerrel a meghatározott legjobban illeszkedő fésű a 0.23 másodperces beosztású fésű lett, és az így meghatározott BPM = 60/0,23 = 260,8696.

Nem csoda, hogy utóbbi módszerrel pontosan kétszeres tempót határoztunk meg. Az első módszerrel a negyedek közti ütéstávolsághoz tartozó tempót sikerült detektálnunk, utóbbi pedig a nyolcadokhoz tartozó időközt találta meg, mivel az a legkisebb időköz, amit a hangminta tartalmaz, és ez pont fele a negyedeknek.

Ez a mérések közti eltérés nem jelent problémát az alkalmazás szempontjából, hiszen mindkettő ugyanannak a tempónak feleltethető meg. Mindenesetre előnyösebb a negyedes, tényleges tempót kijeleznie az eszközömnek, így az első tempódetektáló módszert alkalmaztam, mely ugyanezt az elvet követve általánosságban is jobb eredménnyel jár.

2.3 MATLAB kód kezelhetősége

Az algoritmus készítése során folyamatosan optimalizálnom kellett azt a megfelelő működés elérése érdekében, és az algoritmus által felhasznált konstansok értékei is többször változtatásra kerültek. A kódban külön megírtam a fontos adatsorok megjelenítését végző sorokat, így könnyű volt azonnali visszajelzést kapnom a működésről. Ilyen ábrákat használtam fel az algoritmus működésének leírásánál is jelen dolgozatban. Segítségükkel össze tudtam hasonlítani egy adott beállításokkal rendelkező algoritmus működését különböző bemenetekre, illetve adott bemenet mellett adódó kimeneteket valamelyik paraméter változtatása mellett. Ennek eredményeképp állt elő a végleges algoritmus, illetve a hozzá tartozó konstansok értékei.

Törekedtem arra, hogy a MATLAB-ban megírt kódom végső változatát könnyű legyen később C nyelvre átírni. (Természetesen ez az adatmegjelenítést végző részekre nem vonatkozik.) A fejlesztés kezdetén még rövid, a MATLAB nyelvében natív kifejezéseket és függvényeket használtam a folyamat megkönnyítésének érdekében, azonban később, a programrészeket átírva, tartózkodtam ezek használatától. Ez azt jelenti, hogy kerültem a MATLAB saját függvényeinek használatát, valamint a sajátos, leegyszerűsített tömbkezelő műveleteinek alkalmazását. Ezek helyett inkább manuálisan, ciklusos módszerekkel hajtottam végre az egyes lépéseket, melyeket már sokkal könnyebb volt C nyelvre átírnom (a MATLAB egyik sajátosságától eltekintve, miszerint ott a tömbök indexelése 1-től kezdődik, a C nyelvben megszokott 0-tól indexeléstől eltérően).

MATLAB-ban a kódomat először egy hosszú script-be írtam le, azonban ezután ezt jól elkülönített függvényekbe foglaltam (mind az algoritmust végző, mind az adatmegjelenítő programrészeket), és írtam egy új scriptet, ami ezeket áttekinthetően és formázottan hívja meg. A paramétereket is könnyen állíthatóvá tettem. Így lehetőség van a könnyű tesztelésre, különböző hangmintákat különböző paraméterek mellett nagy számban lehet tesztelni. Az eredmények alapján lehetséges fejleszteni az algoritmust, valamint pontosítani a paraméterek értékeit az alkalmazhatóság spektrumának szélesítése érdekében, illetve egyes alkalmazások esetén a tempódetektálás pontosságának növelése érdekében. A szakdolgozatomnak azonban nem állt fókuszában ennek elérése.

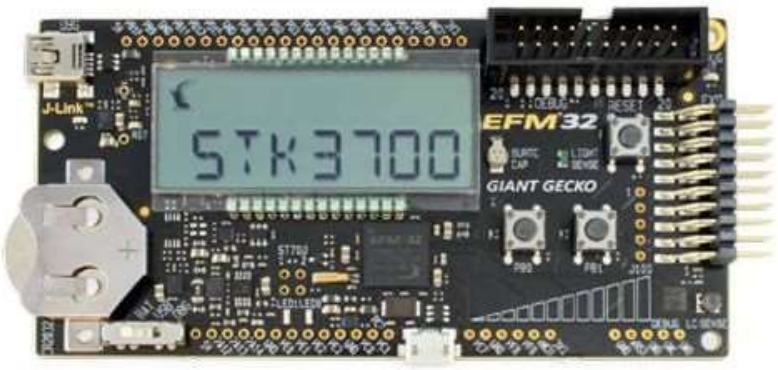
Az algoritmus paramétereit, konstansait dobos és ütős hangmintákon tesztelve állítottam be az előírt alkalmazás szempontjából kielégítő eredményt nyújtva. Így természetesen az előállt végső program is ilyen alkalmazás esetén fog kellően jó eredményt mutatni. Mindemellett megjegyzendő, hogy a megírt kódom felhasználásával az algoritmus könnyen kívánt alkalmazáshoz szabható.

3 Implementáció beágyazott rendszeren

Az algoritmus beágyazott rendszeren történő implementációját Simplicity Studio fejlesztőkörnyezetben végeztem, C nyelven. A választott eszközöm, melyen az algoritmust implementáltam pedig az EFM32 STK3700 sorszámú, fantázianevén Giant Gecko mikrokontroller. Az implementáció magába foglalta az eszközöm hardveres kialakítását, azaz egy külső periféria illesztését analóg módon; valamint a MATLAB-ban megírt algoritmusom erre az új rendszerre való átírását, és a fellépő fizikai limitációknak megfelelő módosítását.

3.1 Fejlesztőkártya bemutatása

Az általam szakdolgozathoz használt fejlesztőkártya a Silicon Labs [21] EFM32 „Giant Gecko" STK3700, ARM Cortex-M3 CPU [22] alapú mikrokontrollere. Az EFM32 Giant Gecko MCU (mikrokontroller egység) család ezen tagja funkciókban gazdag platformot nyújt, melynek főbb felhasználási területei a prototípus- és applikációfejlesztés, valamint azok kiértékelése [23].



18. ábra: Giant Gecko mikrokontroller [24]

A szakdolgozatom szempontjából fontos paraméterei és perifériái a következők:

- Simplicity Studio fejlesztőkörnyezet által támogatott.
- USB (Universal Serial Bus) csatlakozón keresztül számítógéppel összekötve működési feszültség alá helyezhető, számítógépről a programkód a mikrokontrollerre tölthető, valamint a program futás közben számítógépről debuggolható.

- Gombelem segítségével is táplálható, így működés közben is hordozható, mobilis eszköz.
- 48 MHz-es nagyfrekvenciás kristály-oszcillátor órajellel rendelkezik.
- 1 MB (NAND) flash és 128 KB RAM (Random Access Memory) memóriaméretekkel rendelkezik.
- 160 szegmenses LCD (Liquid Crystal Display), segítségével számok, illetve bizonyos karakterek kijelezhetők, a program által meghatározott tempó ide kiírható.
- Beépített 12 bites ADC (analog-to-digital converter) és DAC (digital-to-analog converter), az ADC legfeljebb 13 MHz-es órajellel képes működni. Az ADC segítségével (az általam használt beállításokkal) az adott bemeneti lábon megjelenő 0 és 3,3 V közötti feszültségtartományban fellépő értékek átalakítás után a 0-4095 (decimális) értékkészlettel kódolhatók. A DAC-t csupán tesztelési célokra használtam.

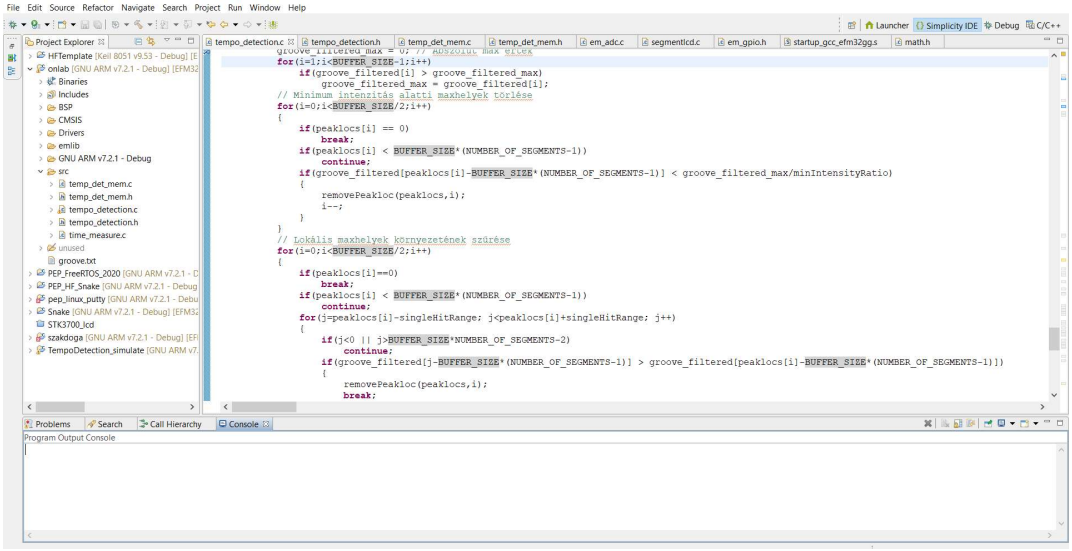
Azért választottam a Giant Gecko mikrokontrollert, mert a korábbi egyetemi tanulmányaim során, pontosabban a Méréstechnika és Információs Rendszerek Tanszék által oktatott specializációs tárgyaim keretein belül is erről az eszközről tanultunk, így rendelkeztem már bizonyos szintű tapasztalattal a használatáról és képességeiről. A Beágyazott és Ambiens Rendszerek tárgy laborjain személyesen is kipróbálhattam, hogyan használható jelfeldolgozási folyamatok elvégzésére az eszköz, és kíváncsi voltam, hogy egy ilyen, nagyobb komplexitású feladat elvégzése során hogyan teljesít. Így ideális választásnak tekintem, hogy a konzulensem, Bank Balázs által kiírt tanszéki témát, egy tempódetektáló funkciót igénylő alkalmazást beágyazott rendszeren, azon belül is Giant Gecko mikrokontrolleren implementáltam.

Nem utolsó szempont az sem, hogy specializációra kerülésem óta tulajdonomban volt egy, a tanszéktől kölcsönkapott darab, így már a szakdolgozat projektem kezdetekor rendelkezésemre állt.

3.2 Simplicity Studio fejlesztőkörnyezet bemutatása

A Simplicity Studio [25] a Silicon Labs saját fejlesztőkörnyezete, mely segítségével különböző típusú mikrokontrollerekre alkalmazások fejleszthetőek, illetve a programokból gépi kódot generálva az eszközre tölthetőek. Többek között az EFM32

típusú MCU család, köztük a STK3700-as Giant Gecko használata esetén is ideális (és magától értetődő) választás ezen szoftver használata, mivel számos eszközzel könnyíti meg a fejlesztési folyamatot.



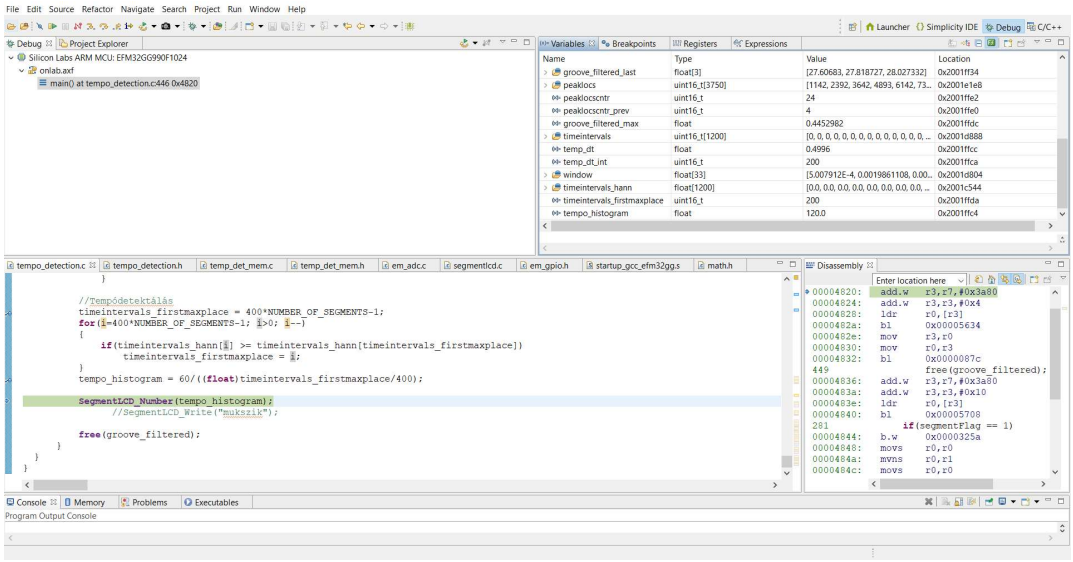
19. ábra: Simplicity Studio kezelőfelülete fejlesztés közben

A jól megszokott alkalmazásfejlesztő platformok funkcióin túl két olyan lehetőséget biztosít a Simplicity Studio, ami különösen nagy segítséggel volt a szakdolgozatom készítése során. Mindkettő a szoftver ún. Debug üzemmódja alatt érhető el, amelyre automatikusan átvált, ha a projektet a gépi kód generálása után a fejlesztőkörnyezeten keresztül a mikrokontrollerre töltjük, és futtatjuk.

Egyik nagy segítség, hogy a Debug üzemmódnak hála a programot mikrokontrolleren való futása közben bármikor megállíthatjuk, illetve a nem szorosan hardveres időzítéshez kapcsolódó programrészeket lépésről lépésre tudjuk végrehajtani. A kódban töréspontok is beállíthatóak, melyekkel elérhető, hogy az általunk kívánt ponton álljon meg a program futása (mikor azt eléri), így az azt követő programrész alaposan megvizsgálható.

Másfelől pedig a Simplicity Studio rendkívül hasznos funkciója, hogy Debug üzemmódban a Variables fül alatt, miközben éppen meg van állítva a program futása, lehetőség van a mikrokontroller memóriájában lévő lokális változók (az éppen megállított pillanat kontextusában értelmezve) megtekintésére. Így például egy vizsgált programrész alatt lépkedve figyelemmel kísérhető azoknak megváltozása, ami nagyon hasznos funkció hibák megtalálására. Vagy ami az én esetemben szintén fontos, a program megállítása

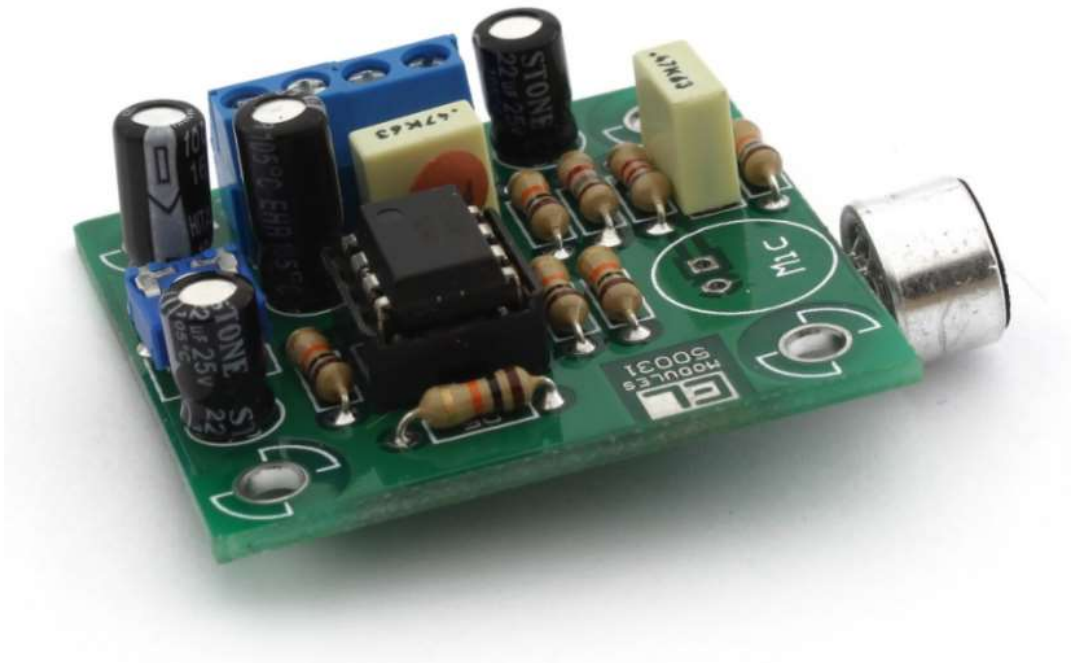
után exportálni tudok egy tömbben tárolt hangmintát, így azt MATLAB-ba bevitel után akár ábrázolni is tudom.



20. ábra: Simplicity Studio kezelőfelülete Debug üzemmódban

3.3 Hardveres rendszer bemutatása

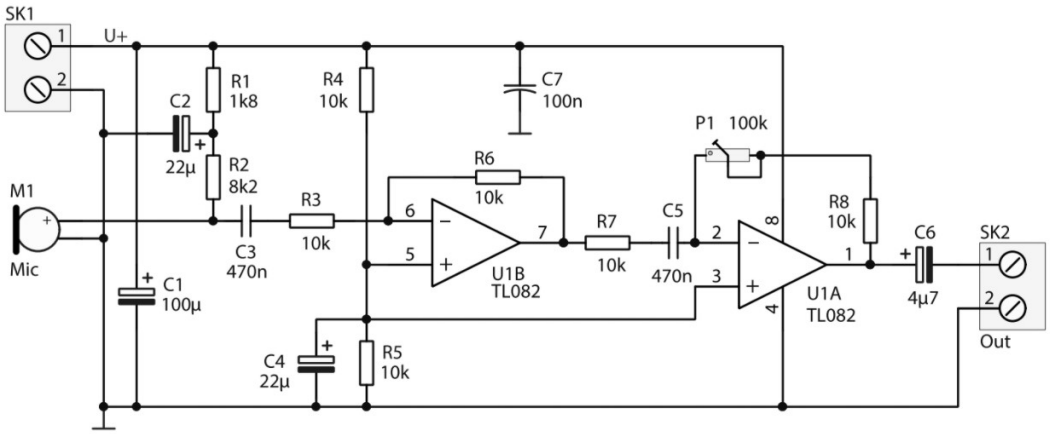
A hang detektálásához a hardveres rendszeremhez készen vásároltam egy EM-60031-es elektret mikrofont és előerősítőt [26]. A gyártói oldalon a következő kép található róla:



21. ábra: EM-60031 elektret mikrofon előerősítő

A nyomtatott áramkör jobb szélén látható az ezüst színű, henger alakú elektret mikrofon. A mikrofon több helyzetben (elöl, hátul, élében) is beültethető, de rövid vezetékekkel akár ki is emelhető a panelről.

Ez egy kompakt és könnyen használható áramkör, ami tökéletesnek bizonyult a feladatom szempontjából. A következő kapcsolási rajzzal rendelkezik:

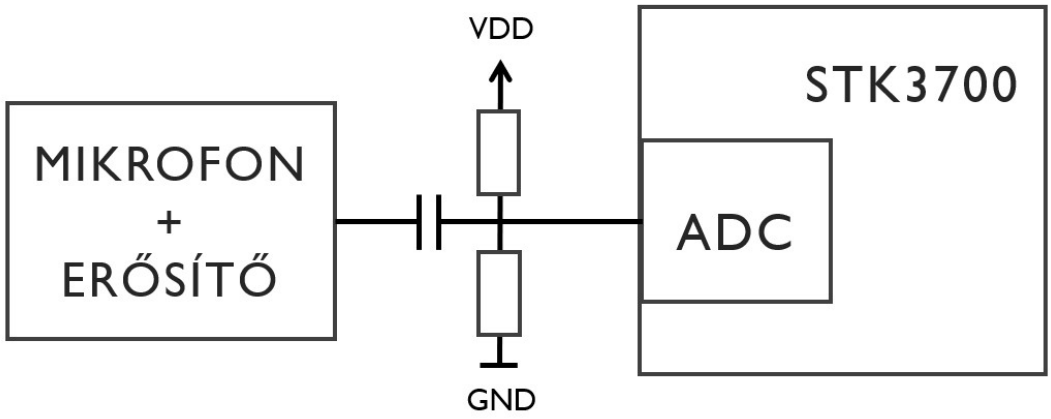


22. ábra: EM-60031 elektret mikrofon előerősítő kapcsolási rajza

A két fokozatú műveleti erősítős kapcsolásban az első fokozat egy fix 10-szeres, míg a másodiknál egy trimmer potenciométerrel állítható az erősítés.

Az igényelt tápfeszültsége egyenáramú és szűrt 9…12 V, én egy 9 voltos alkálielemmel tápláltam.

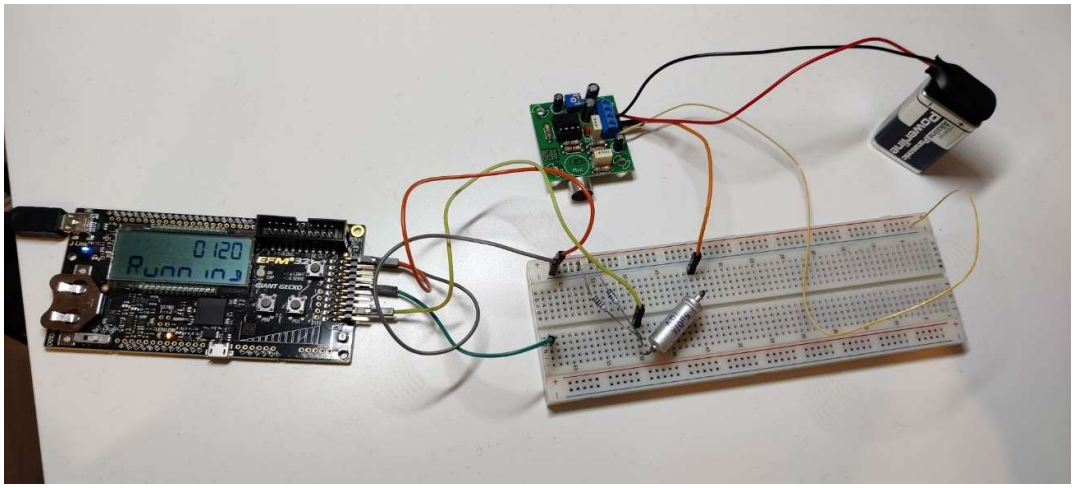
Használatához a mikrokontrollerrel földes összeköttetést kellett biztosítanom, az erősítő kimenetén pedig megjelenik a mikrofon erősített jele.



23. ábra: fizikai blokkvázlat

Az ábrán látható módon illesztettem az erősítőt a mikrokontrollerhez. Az ábra vázlatos, valójában több összeköttetést alkalmaztam, így az ábrán nem szerepel a mikrofonerősítő tápellátása, valamint az erősítő és a mikrokontroller földes összeköttetése. A mikrofonerősítő kimenete egy 0 V középértékű jel, azonban a Giant Gecko ADC-je 0 V és egy előre meghatározott referenciafeszültség (alapértelmezett és általan használt érték: 3,3 V) közti feszültségértékeket képes érzékelni. Így szükséges volt megemelni az erősítő jelszintjét a tápfeszültség (3,3 V) felével, amit egy egyszerű ellenállásosztóval tettem meg. Mindkét ellenállás értéke 10 kΩ. A mikrofonerősítő kimenetén egy 5 µF-os kondenzátor található, így az meggátolja az 1,65 V-os egyenfeszültség káros visszahatását az erősítőre.

A beméréshez és teszteléshez jelgenerátort és oszcilloszkópot használtam, ezek segítségével vizsgáltam és hitelesítettem a bemenő jelszint megemelését. Szintén oszcilloszkóp segítségével állítottam be a mikrofon erősítését, azaz az erősítő áramkörön lévő potenciométert. A beállítás olyan, hogy a Gecko ADC-jén megjelenő jel feszültsége hangos ütések esetén is 0 és 3,3 V közt található, viszont megközelíti azokat, azaz az erősítést próbáltam a meglévő határok alatt maximalizálni. Végső eredményként az alábbi hardveres rendszert állítottam elő:



24. ábra: Hardveres megvalósítás

Az ábrán látszik bal oldalt a Giant Gecko mikrokontroller, tőle jobbra a készen vásárolt EM-60031-es mikrofonerősítő és hozzá tartozó elektret mikrofon, mely egy 9 V-os elemet használ tápként. A kontroller és erősítő összeköttetését jumper kábelekkel és breadboard-dal valósítottam meg, az ellenállásosztó is itt található.

Látszik továbbá a képen, hogy a mikrokontrollerből egy további kábel is ki van vezetve, ennek az az oka, hogy ennek segítségével végeztem a tesztelés nagy részét. A Gecko-t oly módon programoztam fel, hogy az ADC által beolvasott értékeket egyből kiküldtem a DAC kimenetén beolvasásuk pillanatában. Így lehetőségem volt a beolvasott jelet feldolgozás előtt, közvetlenül is vizsgálnom.

3.4 Megvalósítás egyszeri lefutással

A legelső megvalósításban a programkódot még nem valós idejű futáshoz írtam, hanem úgy, hogy egy előre megadott hangmintára egyszer lefusson az algoritmus. Ehhez a Gecko memóriájában eltároltam egy hangmintát, az implementált algoritmust erre futtattam le egyszer, és ennek határoztam meg a tempóját. Így ki tudtam próbálni, hogy helyesen működik-e a MATLAB nyelvről C-re átírt kódom, kitapasztaltam milyen limitációk állnak fent, és meg tudtam tervezni, hogy milyen módosításokra van szükség.

Először is a mikrokontroller memóriájában el kellett helyeznem a hangmintát, amelyen az algoritmus futhat; viszont egy olyan mintára volt szükség, amely a valósidejű használat alatt előálló minta értékkészletének megfelel. Ehhez MATLAB segítségével egy WAV (Waveform Audio File Format) fájlban tárolt hangmintát konvertáltam oly módon, hogy az megfeleljen a Gecko ADC-je által érzékelt értékeknek. Ez azt jelenti,

hogy a hangsávot a [-1; 1] tartományról a [0; 4095] tartományra transzformáltam lineárisan, majd értékeit egészekre kerekítettem. A Gecko memóriájában eltároltam egy így előállított 2 másodperc hosszú mintát, ez képezte az algoritmus bemeneti vektorát.

A Butterworth szűrő paramétereit MATLAB segítségével állítottam elő a mintavételi frekvencia ismeretében, és használtam fel konstansként a C kódban. Ezen túl a Hann-ablak értékeit is konstansként kezeltem, szintén MATLAB segítségével előállítva.

A tempódetektáló algoritmust lépésről lépésre átírtam C nyelvre, miközben folyamatosan teszteltem és összehasonlítottam a MATLAB-os részeredményekkel, hogy meggyőződjek helyesen működik az új környezetben is. Figyeltem végig a hatékonyságra (mind memóriahasználat, mind futásidő szempontjából), valamint a paraméterezhetőségre.

Az így előállt program fejlesztése során fel tudtam térképezni a Giant Gecko hardveréből adódó korlátokat, amiket a következő fejezetben fejtek ki.

3.5 Mikrokontroller fizikai korlátai

A feladatom megvalósításának egyik legnagyobb nehézségét az STK3700-as mikrokontroller fizikai korlátai szabták. Ez két fő részből áll: az eszköz limitált memóriamérete, valamint limitált számítási kapacitása. Az ezen korlátokból adódó problémákat különböző módszerekkel kellett áthidalnom.

3.5.1 Limitált memória

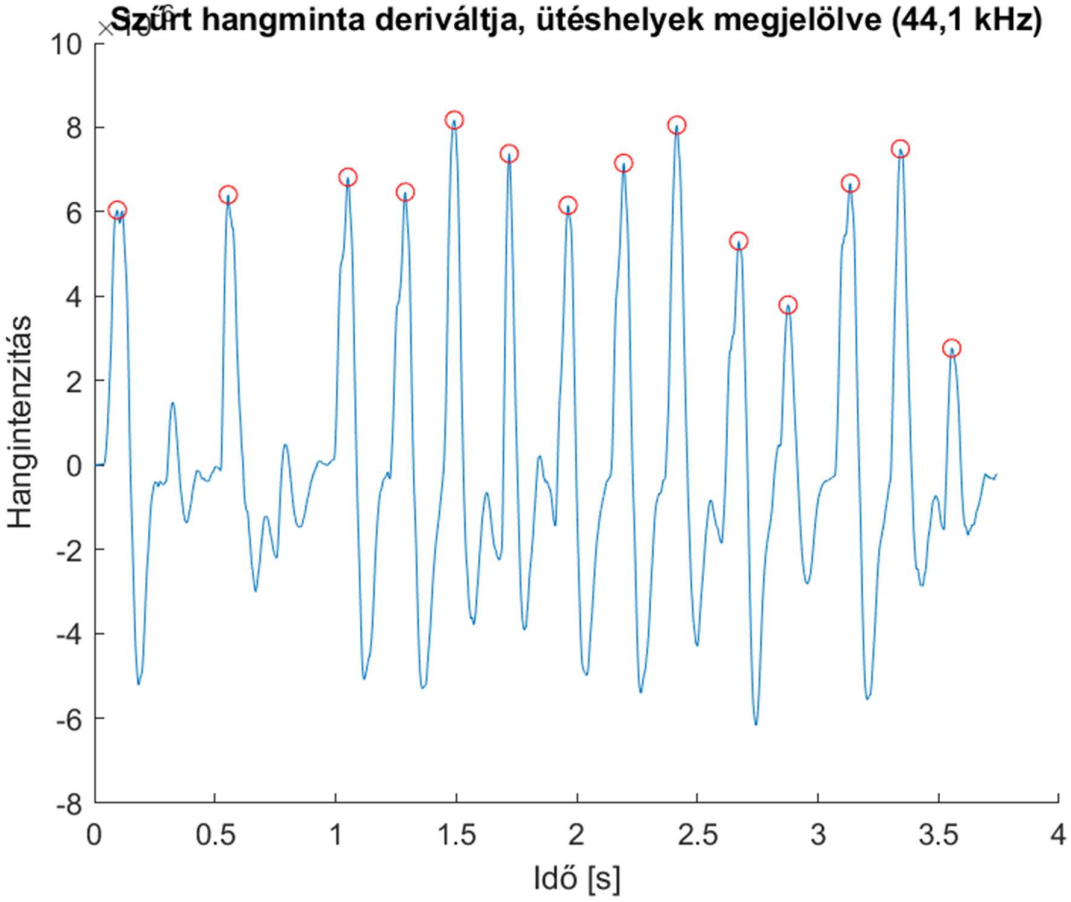
Az STK3700-as fejlesztőkártya 128 KB RAM-mal rendelkezik, ebben a keretben el kell férjen bármely időpillanatban az összes éppen használt változó értéke. A jelfeldolgozási folyamatban legalább egy lépés erejéig kénytelen vagyok lebegőpontos számokat (float) használni, mivel az abszolútértékek burkológörbéjének képzésénél (digitális szűrés) a sok tizedesjegy pontosságú számábrázolás elengedhetetlen. Dupla hosszúságú lebegőpontos számokra (double) azonban nincsen szükség, float változók használatával elérhető közel tökéletes egyezés a MATLAB-ban sokkal nagyobb felbontású változókkal számított eredménnyel (lásd 4. fejezet). Ez a float típusú tömb az, amely a legnagyobb helyet foglalja a memóriában, így a memóriát, mint szűk keresztmetszetet vizsgálva, ennek a tömbnek próbáltam a méretét minimalizálni.

Egy float változóban tárolt érték a memóriában 4 byte-ot foglal el. Mivel tudjuk, hogy a teljes rendelkezésre álló memória 128 kilobyte, így könnyen kiszámítható, hogy

$$\frac{128\ byte \cdot 2^{10}}{4\ byte} = 32\ 768$$

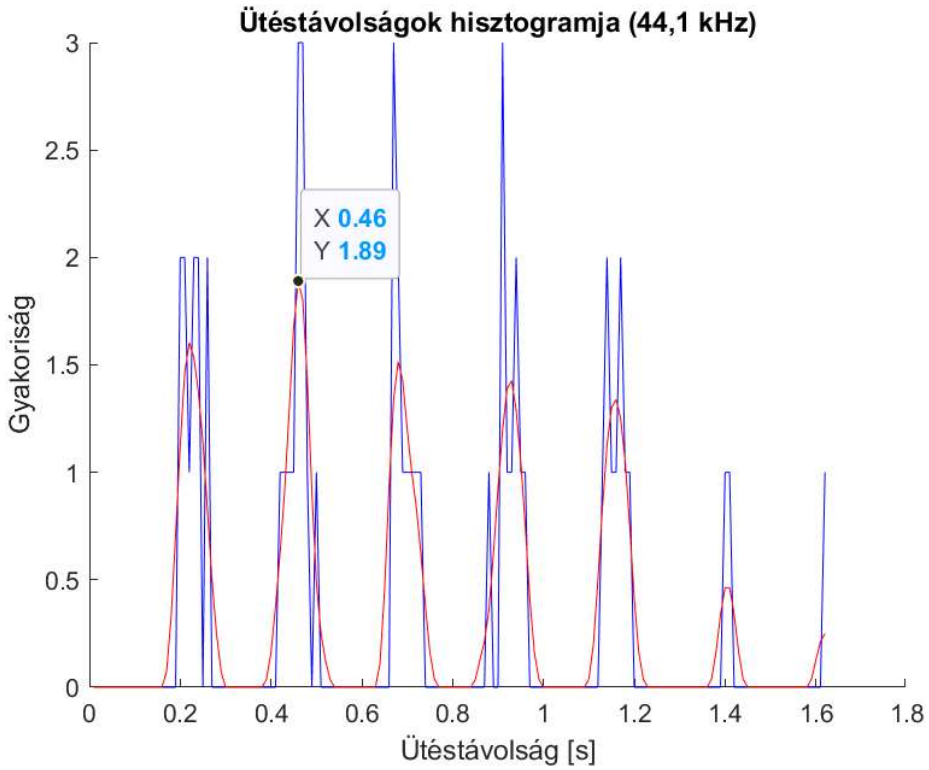
az összesen tárolható float változók száma egy időben. Ha a hangtechnikában standard 44,1 kHz-es mintavételi frekvenciával működne az eszközöm, akkor egyszerre egy másodpercnyi minta feldolgozása nem lenne lehetséges a fejlesztőkártyám segítségével, mivel 44 100 db float változó nem fér el egy adott időpillanatban a memóriában.

Szerencsére ilyen magas mintavételi frekvenciára nincsen szükség tempódetektálás során. Tesztjeim alapján ennél jóval alacsonyabb érték esetén is pontos eredményt kaptam a tempóra, mivel az ütéshelyek így is jól elkülöníthetően felfedezhetők a jelben. Az alábbi ábrán egy 44,1 kHz mintavételi frekvenciájú jelből láthatunk egy pár másodpercnyi mintát, az algoritmus egy későbbi szakaszában: itt kék színnel látható a már szűrt jel deriváltja, rajta pedig piros színnel bejelölve a valódinak érzékelt ütéshelyek.



25. ábra: Ütésdetektálás 44,1 kHz mintavételi frekvencia mellett

Ebben az esetben az alábbi hisztogram állt elő:

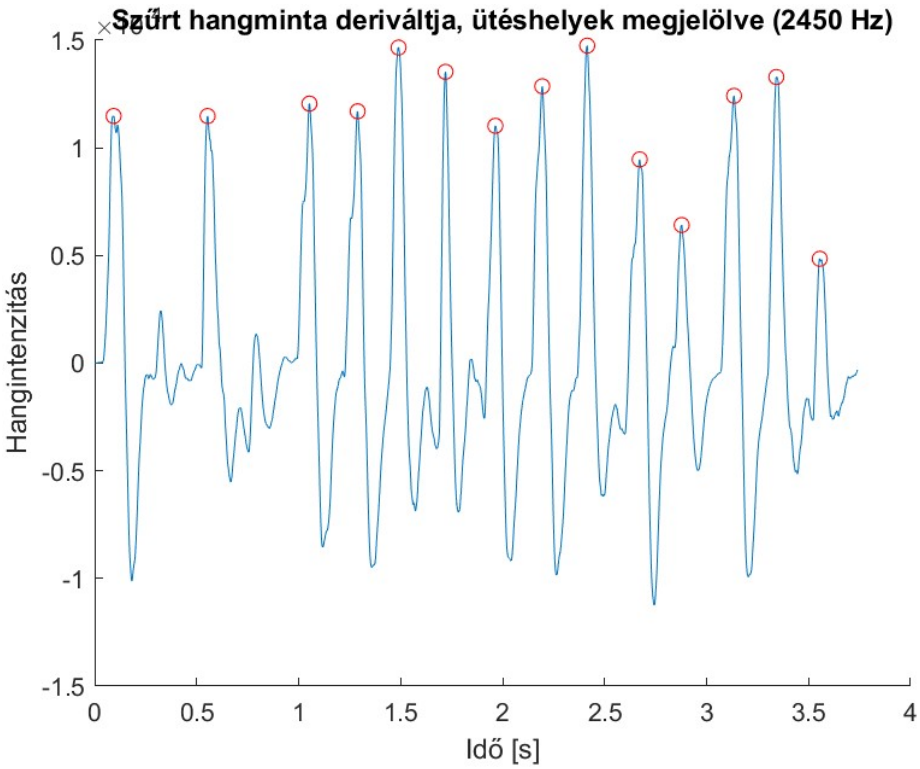


26. ábra: Hisztogram 44,1 kHz mintavételi frekvencia mellett

A Hann-ablakkal simított jel maximuma 0.46 másodpercnél lép fel, így a mintára számolt tempó: $60\ s\ \div 0.46\ s \approx 130\ BPM$

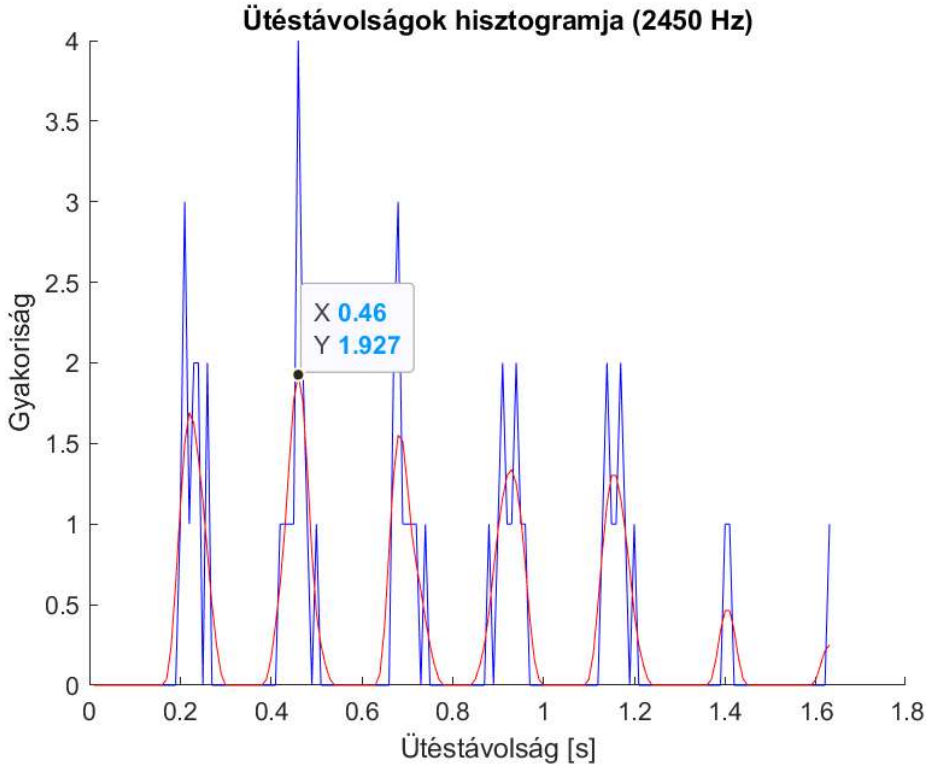
Ezután megvizsgáltam alacsonyabb mintavételi frekvenciák esetén is a kapott eredményeket, megkeresve a legalacsonyabb olyan mintavételi frekvenciát, ahol még pontos eredményt kapok.

A következő eredményeket 2450 Hz-es mintavételi frekvencia esetén kaptam:



27. ábra: Ütésdetektálás 2450 Hz mintavételi frekvencia mellett

Az ábra szinte tökéletesen egyezik a 44,1 kHz frekvenciával mintavett jel esetén rajzolt ábrával, az ütéshelyek pontosan ugyanúgy megjelennek. Itt az alábbi hisztogram állt elő:



28. ábra: Hisztogram 2450 Hz mintavételi frekvencia mellett

Itt látható, hogy a Hann-ablakkal simított hisztogram maximuma szintúgy 0.46 másodpercnél lép fel, mint 44,1 kHz esetén, tehát a számolt tempó is azonos lesz, 130 BPM. Megállapítható tehát, hogy 2450 Hz-es mintavételi frekvenciával is képesek vagyunk egy hangmintának kellő pontossággal meghatározni a tempóját.

Kényelmi szempontból az általam vizsgált frekvenciák 44100 Hz egész számú hányadai voltak, 2450 Hz-el mintavett jelhez csupán elég volt az eredeti jelmintának minden 18. mintáját figyelembe vennem (mivel $44100 \div 2450 = 18$). Itt megjegyzendő, hogy ilyen alacsony mintavételi frekvenciánál megjelenik az átlapolódás jelensége. Ezt sem átlapolódásgátló szűrővel, sem más eszközzel nem küszöbölöm ki a programomban. Azonban ez nem is baj, mivel az ütések detektálásához csupán a jel pillanatnyi teljesítményére van szükség, a burkológörbe így is helyesen előállítható.

2450 Hz-es mintavételi frekvenciánál memóriaméret szempontjából nem szükséges alacsonyabbat választani, mivel 2450 db float változó eltárolása (azaz ilyen mintavételi frekvencia esetén 1 másodpercnyi minta) is csak a RAM $2450 \div 32768 \approx 0.0747 \approx 7$ %-át teszi ki, így rengeteg helyet hagyva az egyéb változók tárolására is, melyek memóriaigénye mellesleg szintén csökkent a mintavételi frekvencia csökkentésével.

Emellett az implementáció során úgy optimalizáltam a kódomat, hogy a bármely adott pillanatban használt memória méretét minimalizáljam. Dinamikusan kezeltem a hosszú (egy másodperc mintáit tartalmazó) tömbök által használt memóriaterületeket, azokat használat után felszabadítottam. Néhány esetben egyes tömböket egy ciklus során felülírtam új értékekkel (pl. szűrt jel → szűrt jel deriváltja), ha a régi tömbre azután nincs szükség.

A fejlesztés során ezek a lépések elegendőnek bizonyultak a memóriaprobléma megoldására, a mikrokontrolleren található RAM nem kerül a program futása során soha betelt állapotba. A végső programban a mintavételi frekvenciát 2450 Hz-ről 2500 Hz-re növeltem (pontosságon ez nem változtat), mivel a Gecko 48 MHz-es órajelének az egy egész számú hányada, így ilyen frekvenciájú hardveres interruptokat pontosabban tud előállítani.

3.5.2 Limitált számítási kapacitás

A C nyelvre átírt programot az algoritmus felépítésénél taglalt részeire bontottam, majd ezeknek egyesével megmértem a futási idejüket.

A futási idő mérésére a mikrokontroller egy beépített speciális időzítőjét/számlálóját használtam, melynek neve: Cycle Count Register. Ez a regiszter a processzor indulásakor nulláról indul, és minden órajelütemben eggyel nő az értéke. A számláló 32 bites, a 14 MHz-es default órajel esetén pedig

$$T_{max} = \frac{1}{14\ MHz} \cdot 2^{32} = 306\ s\ \approx 5\ min$$

hosszúságú futási időt is tudunk mérni. Így tetszőleges programrész futási ideje mérhető, ha a regiszter értékét előtte, valamint utána is lekérdezzük, és a két értéket kivonjuk egymásból. (Szükséges emellett korrigálni a számítással és a regiszterolvasással eltöltött időt, azaz kivonjuk az órajelciklusok számát, ami a regiszterből történő olvasással és kivonással telik.)

2500 érték hosszú hangminta feldolgozása esetén a következő eredményeket kaptam futási időkre:

Művelet	Futási idő [s]
Abszolútérték képzés ideje	0,009728
Szűrt jel előállításának ideje	0,492940
Jel deriválásának ideje	0,022324
Maxhely keresés ideje	0,048151
Maxhely szűrés ideje	0,064330
Hisztogram & tempódetektálás	0,028666
Összesen	0,666139
Ami mintaként egyszer kell fusson	0,637473
Ami kiírásonként kell fusson	0,028666

Azt a megfigyelést tettem, hogy a futási idő túlnyomó részét a burkológörbe előállítása teszi ki, továbbá minden másodpercnyi mintára le kell fusson egy olyan programrész, melynek időbeli hossza 0,64 másodperc.

Mivel azt szerettem volna, hogy a program másodpercenként kijelezzen egy tempót, egy másodpercnyi minta pedig nem elegendő a tempó pontos meghatározásához, ezért minden egyes másodpercben több másodpercnyi mintát szükséges feldolgozni. Látható, hogy egy másodperc alatt nem lehetséges több másodpercnyi minta szűrése (burkológörbe előállítása), mivel 2 másodpercnyi minta szűrése már 1,28 másodpercnyi számítást venne igénybe. Szerencsére erre nincs is szükség, mivel minden másodpercnyi mintát elég egyszer szűrni. A másodpercek közt a digitális szűrő állapotváltozóinak eltárolásával (6 érték) pedig előállítható hiba nélkül a folytonos burkológörbe.

A megoldásom az lett, hogy másodpercenként lefut a táblázatban sárgával jelölt (első öt) programrész az aktuális másodperc mintájára. Ennek eredményeképp előáll minden egyes másodpercben a detektált ütések helye, ezt az információt pedig eltároltam a következő két másodperc idejére is. A hisztogramot minden másodpercben felépítettem a programomban, azonban nem csak az aktuális másodpercen belül detektált ütések közti időintervallumokkal, hanem a három legutóbbi másodperc ütéshelyei alapján. Ezzel a módszerrel másodpercenként ki tudtam jelezni a legújabb három másodpercnyi

hangminta számolt tempóját, így egy jó középutat teremtve a pontosság és visszajelzési sebesség közt.

3.6 Valós idejű futás

A fentebb kifejtett megfontolásokkal élve megírtam a végső programomat, mely már a mikrofon adatait használva, valós időben hivatott a hallott hangminta tempóját megállapítani.

Az ADC és DAC használatához inicializálnom kellett ezeket a perifériákat. Az egyenlő időközönként történő mintavételezéshez pedig fel kellett konfigurálnom egy timeres megszakításrutint. Mivel az az ADC legfeljebb 13 MHz-el képes működni, a Gecko 48 MHz-es alapórajeléből egy 4-es előosztóval 12 MHz-es órajelet állítottam elő. Ennek az órajelnek további leosztásával előállt a 2500 Hz gyakoriságú megszakításrutin, mely az ADC átalakítás eredményét beolvassa.

A perifériák, valamint megszakításrutin részletes konfigurációját a Beágyazott és Ambiens Rendszerek c. tárgy laborgyakorlataihoz kiadott anyagokban leírtak alapján végeztem [27].

A hangminták rögzítésére két globális buffert hoztam létre, melyekben egy-egy másodpercnyi hangminta fér el. Minden egyes másodperc alatt a megszakításrutin az egyiket feltölti az új másodperc értékeivel, a másik buffer, mely az előző másodperc hangmintáját tartalmazza, pedig feldolgozásra kerül a végtelen cikluson belül futó törzsprogramban, ami a jelfeldolgozási (azaz tempódetektáló) folyamatot tartalmazza. A megszakításrutin minden másodperc végén jelzést ad a főprogramnak (a buffer feltöltésre került) egy bináris flag segítségével, amire a törzsprogram a jelfeldolgozás végeztével várakozik. Két különálló bufferre azért volt szükség, hogy az esetleges felülírást (jelfeldolgozási folyamat közben a megszakítás felülírja a feldolgozandó értékeket) elkerüljem.

A bufferből a hangminta betöltését úgy végeztem, hogy első lépésként kiszámoltam a buffer átlagát, majd elemenként az átlagtól való abszolút eltérésüket helyeztem a továbbiakban feldolgozandó tömbbe. Így előállt az abszolútértékeket tartalmazó tömb, ahol a 0 szintet a jel középértéke képzi; ezzel kiküszöböltem a jelben jelen lévő DC komponens pontatlanságát.

A program írása közben megejtettem egy módosítást az algoritmuson, ami a tempódetektálás pontosságát növeli, mégpedig az ütések közti távolságdetektálás pontosságán javítottam. Eleinte a távolságdetektálást 10 ms pontossággal írtam meg, azonban későbbi megfontolás után ezt 2,5 ms-re csökkentettem.

A távolságdetektálásból adódó pontatlanság, mivel egy kerekítés, ezért ez egy időben kifejezhető hiba, worst case esetben állandó érték, a felbontás fele (mivel a kerekítési határon a felbontás felével módosulhat az érték). Minél nagyobb a vizsgált tempó, annál kisebb az egyes ütések közti távolság, így annál nagyobb hibát eredményez ez a kerekítés a tempódetektálásban. A 150 BPM-es maximális tempót céloztam meg, mint pontosan detektálni kívánt tempó, mivel ennél nagyobb értékek ritkán lépnek fel az átlagos dobosnál. Így erre próbáltam optimalizálni a felbontást. 150 BPM esetén a negyedes ütéstávolság: 60/150 = 0,4 s.

Amennyiben a távolságdetektálás felbontása 10 ms, így a worst case pontatlanság:

$$\frac{60\ s}{0{,}4\ s - 0{,}005\ s} BPM - \frac{60\ s}{0{,}4\ s} BPM \approx 1{,}898\ BPM$$

Ami nem egy jó érték, mivel a worst case hiba több, mint 0,5 BPM, így ez a pontatlanság akár rossz eredményre is vezetheti az algoritmust. Ennek persze a statisztikai esélye kicsi, egyrészt mivel a kerekítésnek nem ekkora a valószínű hibája, másrészt pedig a hisztogramban sok ütéstávolság szerepel, amik együttesen állítják elő az algoritmus eredményét. Mindenesetre ezt a pontatlanságot szerettem volna 0,5 BPM alá vinni, hogy ne okozhasson 150 BPM vagy az alatti tempó esetén hibát.

Ehhez a távolságdetektálás felbontásának pontosságát a négyszeresére növeltem, azaz 2,5 ms-es beosztásra. Így a worst case hiba 150 BPM esetén:

$$\frac{60\ s}{0{,}4\ s - 0{,}00125\ s} BPM - \frac{60\ s}{0{,}4\ s} BPM \approx 0{,}47\ BPM$$

Ez már nem okozhat hibát akkor sem, ha egy ütéstávolságból kerül a tempó kiszámításra, ami egész számra kerekítve kerül kijelzésre.

Apró, de annál fontosabb megjegyzés, hogy a felbontás növelésével párhuzamosan ezzel arányosan növeltem a hisztogramot simító Hann-ablak méretét is 9-ről 35 eleműre. Mivel a felbontás négyszeresére nőtt, így négyszeres szélességű ablakkal való simítással érhető el olyan eredmény, amire az ablak szélességét optimalizáltam. A négyszeres, 36 elemű helyett 35 elemű ablakszélességet választottam, mivel a

programkód megírásának könnyítése érdekében páratlan elemű ablakra volt szükségem. Továbbá még azt is megjegyezném, hogy ezzel a lépéssel jelentősen megnőtt a hisztogram-simítás, azaz a Hann ablakkal való konvolúció futási ideje. Ilyen pontosság elérése mellett ez a növekedés nem okoz gondot; viszont eleinte 1 ms-es felbontással is kísérleteztem, ekkor a közel 90 elemű Hann-ablakkal végzett konvolúció már túl sok időt vett igénybe, és nem tudott az algoritmus másodpercenként lefutni. Ez az oka annak, hogy a 2,5 ms-es felbontásnál nem választottam pontosabbat.

Ezen kívül más módosítást nem ejtettem az algoritmuson, az új programkeretbe átültetés, sok tesztelés és összehasonlítgatás után elkészült a végső, működő program.

Ezen a ponton megjegyezném, hogy az előállt C nyelvű kódot is könnyen paraméterezhetőre írtam. Az algoritmus bemutatásánál említett paraméterek is állíthatóak, továbbá a mintavételi frekvencia, valamint a vizsgált blokkok száma (azaz, hogy az utolsó hány másodpercet vegye figyelembe a program a tempó számításánál) is könnyen állítható.

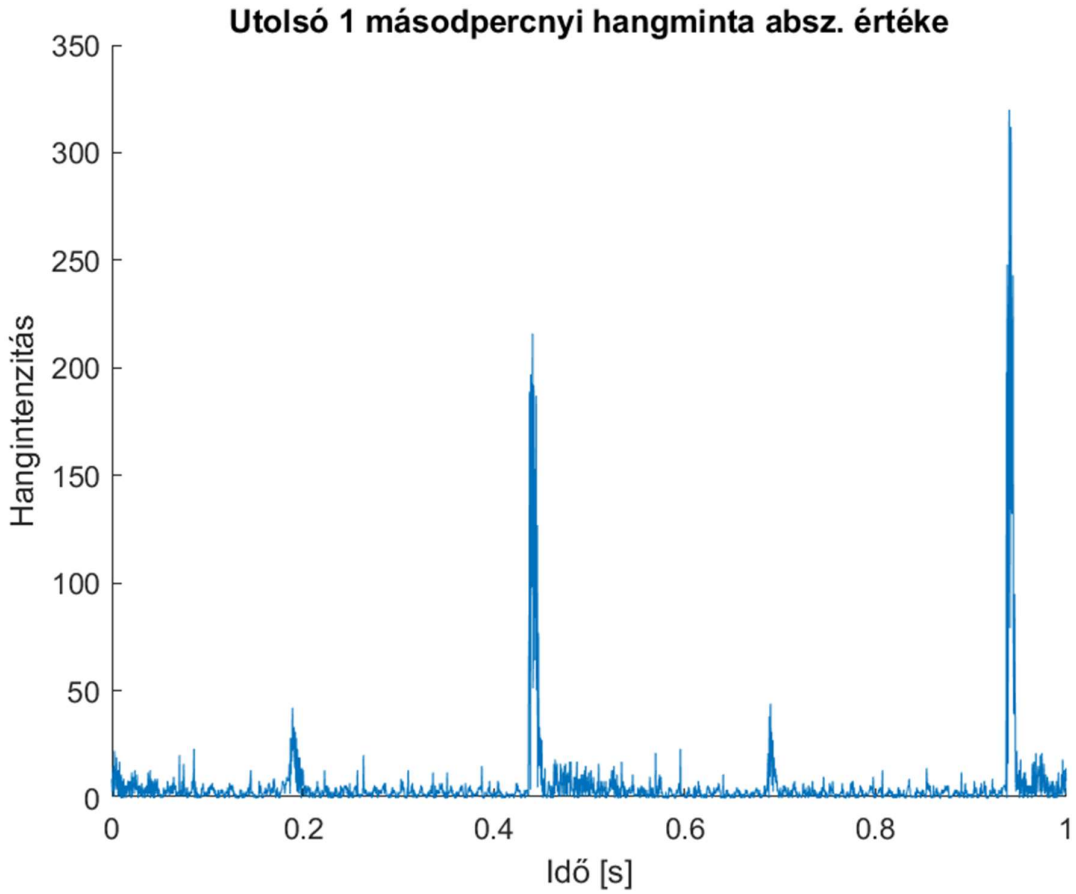
4 Tesztelés és értékelés

Az eszközöm működésének teszteléséhez a jelfeldolgozási folyamatot futása közben lépésenként megállítva exportáltam a részeredményeket MATLAB-ba, így ott ábrázolni tudtam őket. Hogy meggyőződjek a program helyességéről, azonos bemenetre lefuttattam a MATLAB jelfeldolgozó algoritmusát is, és annak részeredményeivel hasonlítottam össze a Gecko-ból exportált adatokat.

4.1 Hangminta feldolgozásának ellenőrzése

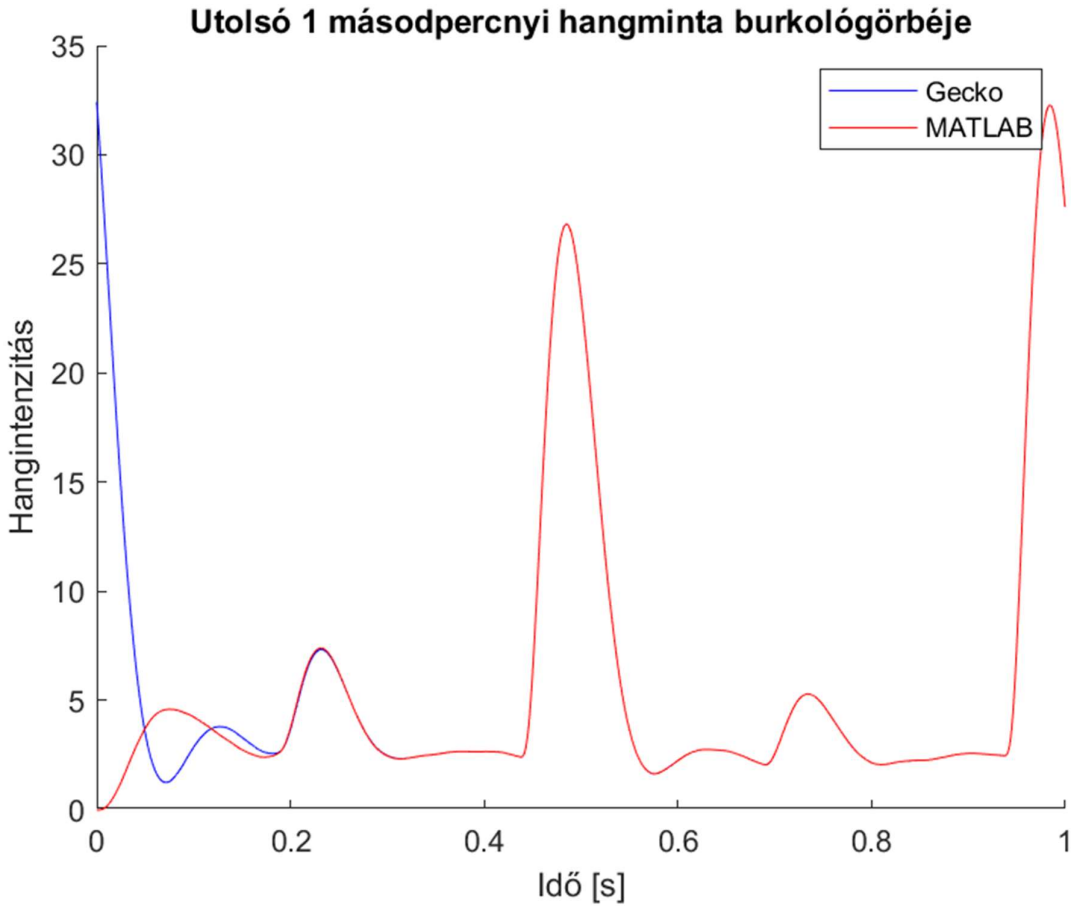
Ehhez a teszthez a mikrofon közelében egy pontosan 120 BPM tempójú dobos ritmust játszottam le hangszórón, mely egy ütemet ismételget. Az ütem négy negyedet tartalmaz, minden első és harmadik negyed egy nagydob ütés, minden második és negyedik negyed egy pergődob ütés. Emellett minden nyolcadban egy halkabb, kisérő cintányérütés hallható.

Körülbelül 10 másodperc után állítottam meg a program futását, a legújabb másodperc mintájának feldolgozása előtt. A folyamat részeredményeit lépésenként elmentettem, majd a MATLAB-ban megírt kódom részeredményeivel összehasonlítottam.



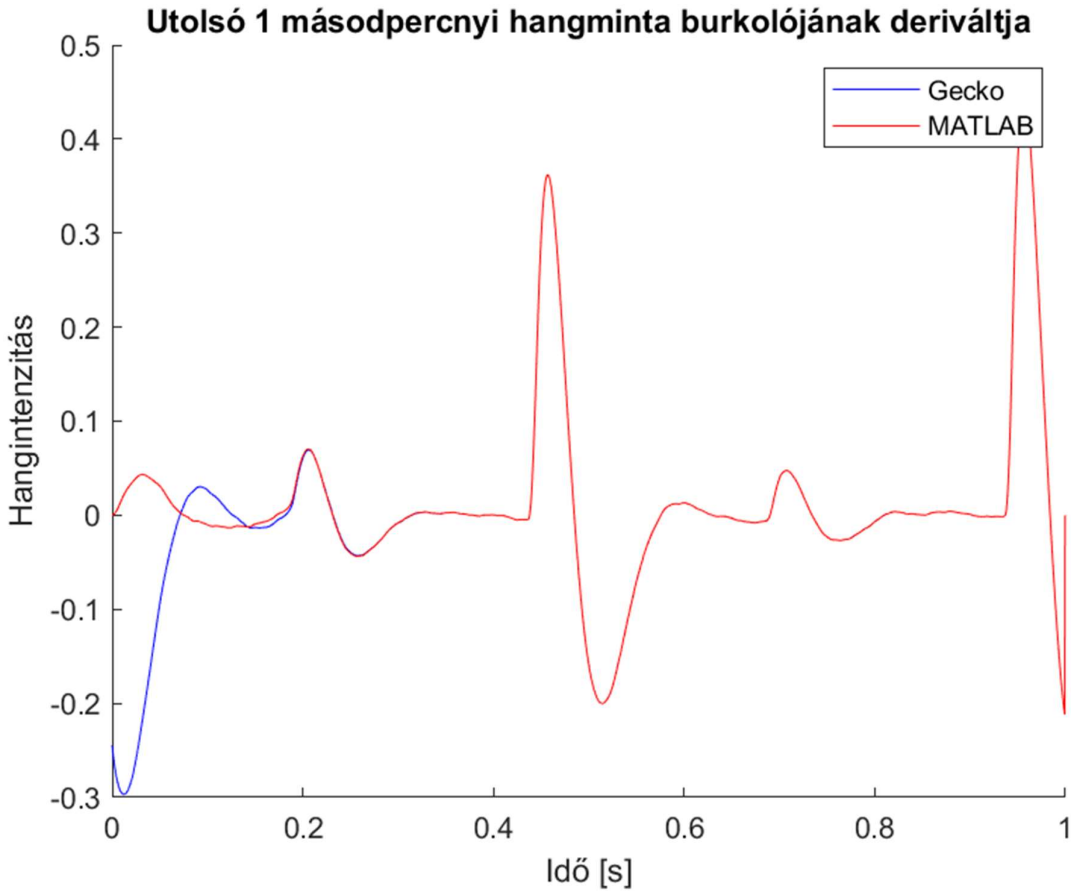
29. ábra: A vizsgált hangminta abszolút értéke

A képen látható a gecko által rögzített legújabb 1 másodpercnyi hangminta. A hangintenzitás az ADC kódolásában értelmezendő, azonban a minta betöltésekor minden egyes értékből ki lett vonva az egész mintából számolt átlagérték, majd ennek képeztem az abszolút értékét. Ezt a mintát töltöttem be MATLAB-ba is, és erre futtattam le az ott megírt programot is.



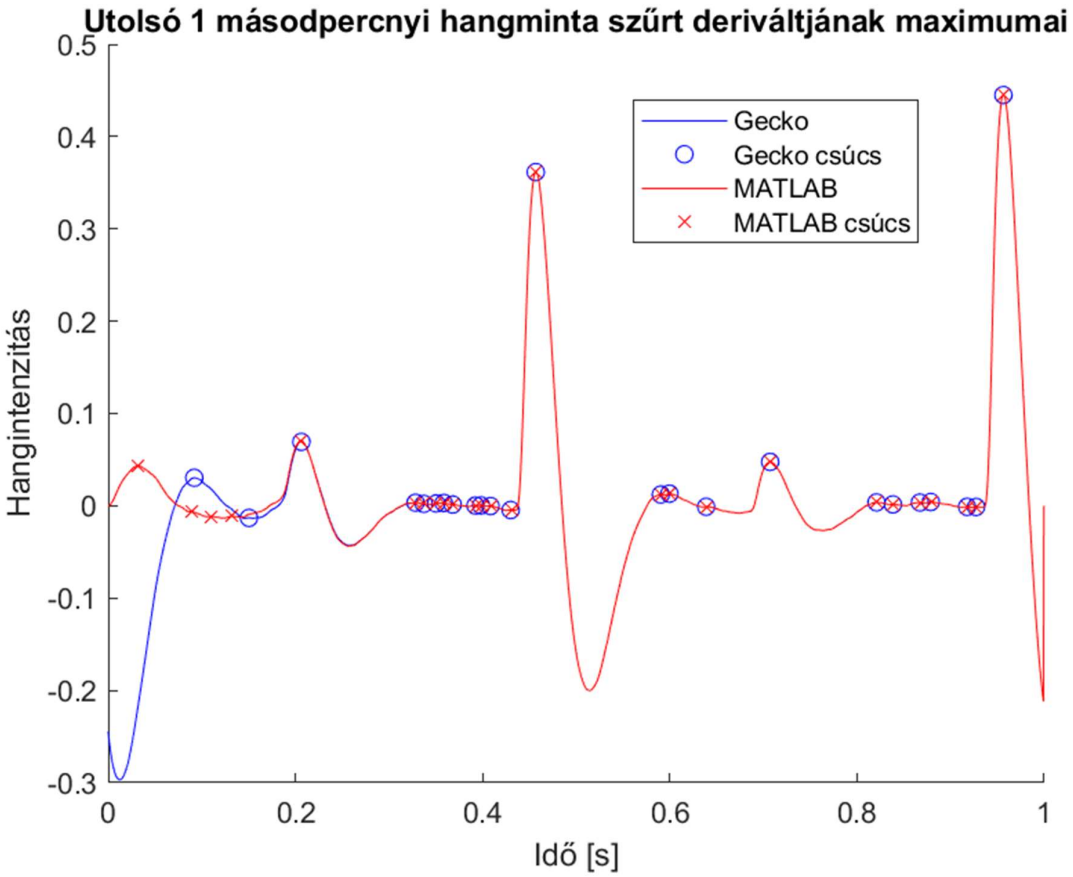
30. ábra: Szűrt minták összehasonlítása

A Butterworth szűrés után a fent látható eredmény állt elő a Gecko-n, illetve a MATLAB program futása során. Az egyetlen fellépő különbség, az a minta elején látható eltérés. A Gecko jele pozitív kezdőértékű, és felfedezhető benne az elején is egy hangimpulzus burkolója (egész pontosan az előző másodperc végén lévő pergő ütésé). A MATLAB jele 0 kezdőértékű, és egy rövid tranziens után beáll a Gecko jelére. Ennek oka, hogy a Gecko az előző másodperc feldolgozása végén elmentette az állapotváltozókat, majd az aktuális másodperc feldolgozásánál ezeket az értékeket töltötte be a szűrés kezdetekor. A MATLAB-ban nincsenek jelen ezek az értékek, így ott a szűrés állapotváltozói kezdetben 0 értékűek.



31. ábra: Szűrt minták deriváltjának összehasonlítása

Deriválás után is a feljebb leírt különbséget fedezhetünk fel, a magyarázat is az előző esetével azonos. Ezen a különbségen kívül a két jel precízen egyezik egymással.



32. ábra: Szűrt minták deriváltjainak lokális maximumai

A két program az ábrán látható lokális maximumokat számolta a jelekben. Itt is csupán a vizsgált másodperc elején található különbség, a többi lokális maximum egyezik a két program közt.

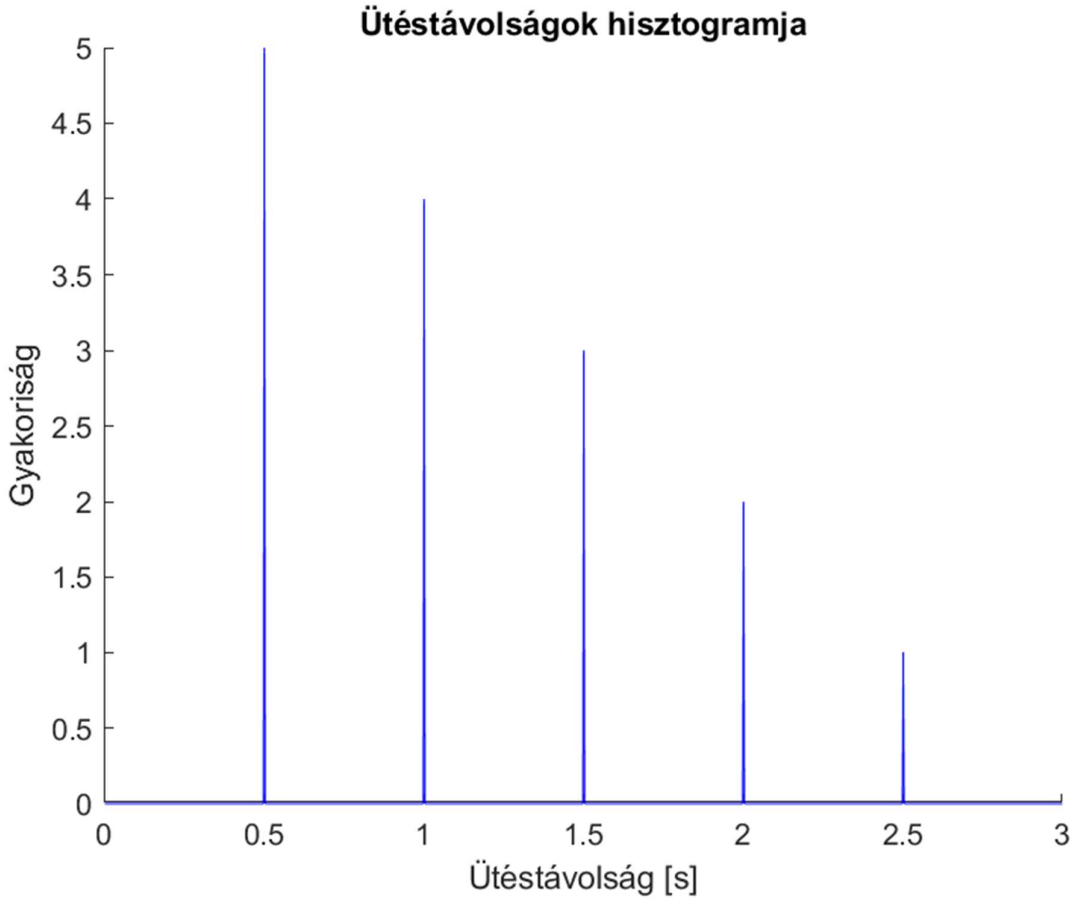
A lokális maximumok közül mindkét program két darabot ítélt valódi ütéshelynek, ezek pedig a tömb 1142. és 2391. elemei (0-tól indexelés esetén). Ez azt jelenti, hogy a nyolcados hangokat a program kiszűrte, mivel hangerejük nem érte el a kívánt nagyságrendet. Ez előnyös, mivel így nem zavarnak be a nyolcados ütéstávolságok a tempódetektálásba, csak a nagydob és pergődob által diktált negyedes ütéstávolságok a mérvadók.

Mivel a MATLAB programnak csak ez a két csúcs áll rendelkezésére, így a hisztogram kirajzolásának nincs sok értelme, egy darab ütéstávolságot képes detektálni ezen az 1 másodperces mintán. Csupán ennyi információ felhasználásával a következő tempót határozza meg:

$$Időköz\ [s] = \frac{2391 - 1142}{2500\ Hz} = 0{,}4996\ s\ \approx 0{,}5\ s\ (mivel\ 2{,}5\ ms\ a\ felbontás)$$

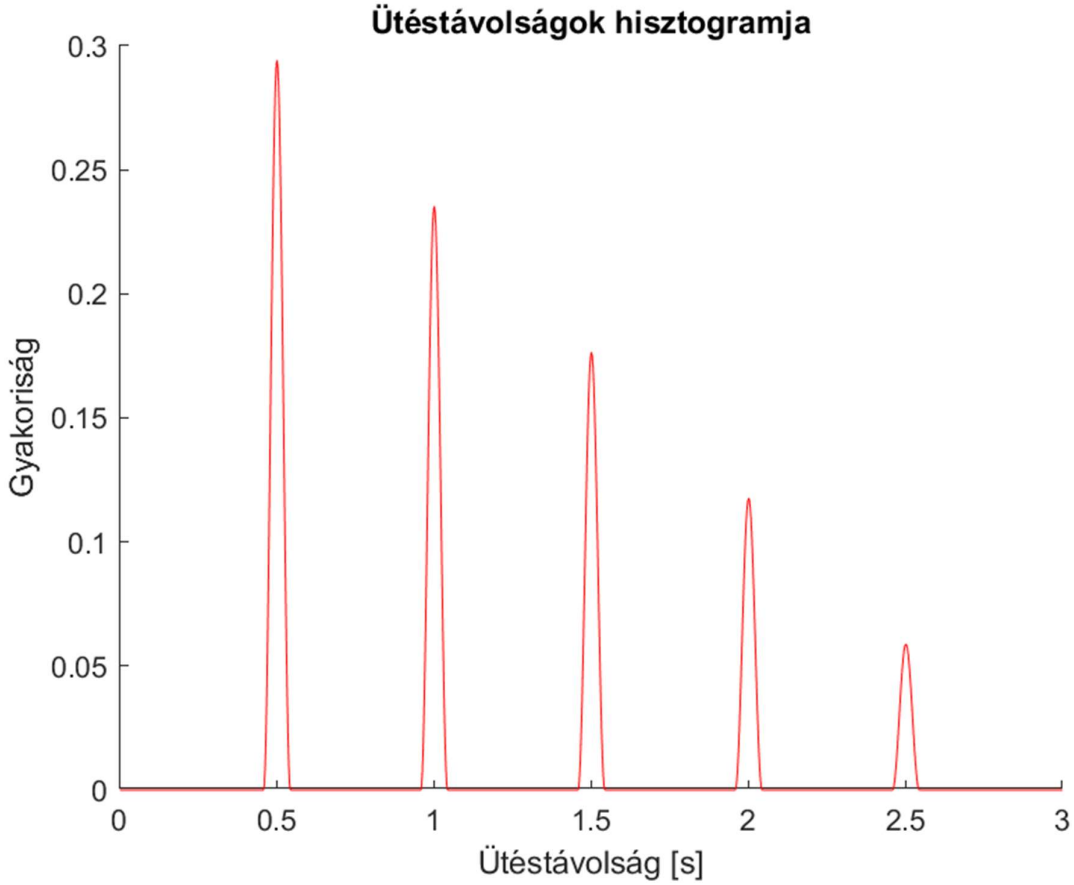
$$Tempó\ (BPM) = \frac{60\ s}{0{,}5\ s}\ BPM = \ 120\ BPM$$

Bár a tempódetektálás ebben az esetben egy ütésköz alapján is pontos volt, pontatlanabb játék esetén nem lenne elég egy ütésközt vizsgálni. A Gecko memóriájában szerepel az új másodperc ütéshelyen túl az előző 2 másodperc ütéseinek helye is. Ezek alapján a következő hisztogramot állította elő:



33. ábra: Gecko által előállított hisztogram az utóbbi 3 másodpercre nézve

Megjegyzendő, hogy a hangminta nagyon pontosan tartotta a tempót, így nem található szórás a 120 BPM-nek megfelelő ütéstávolság (és annak egész számú többszörösei) körül. Mindenesetre a következő Hann ablakkal simított hisztogram állt elő:



34. ábra: Gecko által előállított simított hisztogram az utóbbi 3 másodpercre nézve

Mivel a simított hisztogram maximuma a tömbnek 200. eleme, ezért a program a következő tempót számolta:

$$Tempó\ (BPM) = \frac{60\ s}{200 * \ 0{,}0025\ s}\ BPM = \ 120\ BPM$$

És ki is jelezte az LCD-n a helyes eredményt.

4.2 Értékelés

Az elkészült eszközöm számos tesztre helyes eredményt jelzett ki, s bár valós dob mellett nem tudtam kipróbálni, számos hangszóróval lejátszott mintával igen, és túlnyomó többségben helyes és pontos eredményt mutatott.

Elmondható, hogy a tempódetektálás gyors, és kellően sűrű visszacsatolást nyújt a dobos számára (másodpercenként a legutóbbi 3 másodpercre vonatkozóan), ami kielégítőbb, mint a feladatomat megvalósító legjobb létező megoldás (amit találtam), a

telefonos tempódetektáló applikáció. Nem mellesleg sok esetben pontosabb eredményt is nyújtott az eszközöm nála, így ez is sikernek tudható be.

Az eszköz gyengesége a zajban rejlik. Amennyiben túl nagy a háttérzaj, ami el tudja nyomni a vizsgálni kívánt hangmintát, illetve éles hangokat is tartalmaz, amik szintén ütésként megjelennek a jelben, felborulhat az algoritmus integritása és hibás eredményt jelezhet ki. Megjegyzendő, hogy ez a hatás egy irányított mikrofon használatával kiküszöbölhető lenne, amely a dobra irányítva annak a keltett hangjait rögzítené a legerősebb jellel, minden más irányból érkező zajt pedig elnyomna. Összességében mindenesetre kijelenthető, hogy az eszközöm a specifikációknak megfelel.

A fent részletezett teszt mellett készítettem egy videót, melyen az eszköz működését tesztelem és demonstrálom. Egyrészt egy metronómon beállított tempó felismerése látható rajta, majd egy szabad kézzel dobos gumipadon játszott ritmus tempójának felismerése. A videó a [28] forrás által megjelölt YouTube weboldalon megtekinthető.

5 Összefoglalás

Az elkészült eszköz a specifikációknak eleget tesz, azaz alkalmas egy dobos számára gyors, és kellően sűrű visszacsatolást nyújtani a játszott zene tempójáról; azaz minden egyes másodpercben a legutóbbi 3 másodpercre vonatkozóan. Tiszta hangzás esetén kielégítő pontossággal és biztonsággal működik, háttérzaj esetén azonban a pontosság csökken. Az eszközt dobos hangminták feldolgozására konfiguráltam, így a rövid, hangos ütéseket tartalmazó zenék tempóját képes a legnagyobb pontossággal meghatározni. Az eszköz autonóm működésű, nem igényel felhasználói beavatkozást, valamint elemről is működtethető, tehát könnyen hordozható.

A szakdolgozatom által meghatározott projekt nagyon hasznos volt számomra tanulás szempontjából. Megismerkedhettem a jelfeldolgozás alapjaival, ami egy olyan témakör, amivel ezelőtt nagyon keveset foglalkoztam, mégis nagyon érdekelt. Kipróbáltam milyen egy jelfeldolgozó algoritmust fejleszteni, tanultam létező megoldásokról, valamint MATLAB használatában is sokkal járatosabb lettem. A beágyazott rendszereket illetően is nőtt a tudásom, sikeresen implementáltam egy általam megírt algoritmust egy mikrokontrolleren, amihez külső perifériát is illesztettem analóg módon.

A legnagyobb kihívást a mikrokontroller memóriamérete és számítási kapacitása okozta, pontosabban a program ezek által megszabott határokon belül történő megvalósítása. Nagy sikerélményt nyújtott, mikor láttam, hogy kellő munka és optimalizálás után sikeresen a beágyazott rendszerbe ültethető az algoritmusom, ami a fent álló fizikai limitációk mellett is tökéletesen működik. Jó volt ezt a folyamatot elejétől végéig személyesen megtapasztalni.

5.1 Továbbfejlesztési lehetőségek

1. A MATLAB-ban megírt algoritmusom könnyen tesztelhető különböző hangmintákra, így különböző hangmintákat különböző paraméterek mellett nagy számban lehetne tesztelni. Az eredmények alapján lehetne fejleszteni az algoritmust, valamint pontosítani a paraméterek értékeit az alkalmazhatóság spektrumának szélesítése érdekében, illetve egyes alkalmazások esetén a tempódetektálás pontosságának növelése érdekében.

2. Az algoritmus egy másik beágyazott rendszerbe is ültethető, ezt tovább könnyíti a már megírt C nyelvű kód is. Egy nagyobb memóriával, nagyobb számítási kapacitással, illetve jobb perifériákkal (ADC) rendelkező mikrokontroller vagy DSP (digital signal processor) felkonfigurálható az algoritmus nagyobb pontosságú verziójával. Esetleg egy jobb minőségű, illetve irányított mikrofon is felhasználható lenne az alkalmazáshoz.
3. Az algoritmus mobiltelefonos applikációba is átültethető, így egy különálló beágyazott rendszer helyett a dobos saját telefonján is futtathatná a tempódetektáló szoftvert.
4. Bár nagy hangsúlyt helyeztem a C nyelvű kód memóriahasználat és futási idő szempontjából történő optimalizálására, biztos vagyok benne, hogy nem értem el tökéletes hatásfokot, a program további optimalizálásával akár kisebb erőforrásokkal rendelkező rendszereken is képes lehet futni.
5. Az eszköz stabilabb és esztétikusabb formába önthető, azaz a külső áramkör NYÁK-ra (nyomtatott áramkör) integrálható; az eszközök pedig egy dobozba, vagy valamilyen szebb külsőbe helyezhetőek. Csak arra kell figyelni, hogy a mikrofon egy hangszigeteléstől mentes helyet kapjon. Így egy olyan eszköz állna elő, aminél már egyáltalán nem kell félni, hogy hordozás közben valahol kontakthiba lép fel, kiesik egy alkatrész, vagy hogy a felhasználó kárt tesz benne.

Irodalomjegyzék

[1] J. Laroche, *Estimating tempo, swing and beat locations in audio recordings,* Proceedings of the IEEE Workshop on the Applications of Signal Processing to Audio and Acoustics, New Platz, NY, USA, 2001, oldalak: 135-138

[2] Stark, A. & Plumbley, Mark & Davies, Matthew, *Real-time beat-synchronous audio effects*, Proceedings of New Interfaces for Musical Expression (NIME), NY, USA, 2007, oldalak: 344-345

[3] Siska Ádám, *Zeneelméleti alapok*, https://apocalypse.rulez.org/kozos/Zeneelm%C3%A9letBlog/2007-02-20 (letöltve: 2020. nov.)

[4] Horváth Gergely, *Ritmusérzék-fejlesztő alkalmazás fejlesztése*, Budapesti Műszaki és Gazdaságtudományi Egyetem, BSc szakdolgozat, 2017

[5] Pirkó Balázs, *Akkordfelismerés és gépi improvizáció*, Budapesti Műszaki és Gazdaságtudományi Egyetem, MSc diplomaterv, 2016

[6] Balázs János, *Beat detection and correction for djing applications*, Budapest University of Technology and Economics, Msc thesis, 2013

[7] Gainza, Mikel & Coyle, Eugene, *Tempo Detection Using a Hybrid Multiband Approach*, Transactions on Audio, Speech, and Language Processing, IEEE, 2011, 19. 57-68.

[8] Alonso, Miguel & David, Bertrand & Richard, Gaël, *Tempo and Beat Estimation of Musical Signals*, ENST-GET, D´epartement TSI, 2004

[9] TAMA Rhythm Watch Mini "RW30" gyártói termékoldala, https://www.tama.com/usa/products/detail/rw30.html (letöltve: 2020. nov.)

[10] Pioneer SVM-1000 keverőpult hivatalos termékoldala, https://www.pioneerdj.com/en-gb/product/mixer/archive/svm-1000/black/overview/ (letöltve: 2020. nov.)

[11] Weboldal: Tap BPM - Online Beats Per Minute Calculator and Counter, http://www.beatsperminuteonline.com/ (letöltve: 2020. nov.)

[12] Simple BPM Detector applikáció Google Play oldala, https://play.google.com/store/apps/details?id=lelloman.com.simplebpmdetector (letöltve: 2020. nov.)

[13] MATLAB hivatalos, angol nyelvű weboldala, https://uk.mathworks.com/products/matlab.html (letöltve: 2020. nov.)

[14] Miskolci Egyetem, *A MATLAB (for Windows) programcsomag rövid ismertetése*, http://ait.iit.uni-miskolc.hu/~ait/segedlet/matlab/matlab-1.htm

[15] Előadó: Blur, zeneszám címe: *Song 2*
Youtube-on elérhető: https://youtu.be/SSbBvKaM6sk
(letöltve: 2020. nov.)

[16] T. W. Parks and C. S. Burrus. 1987. *Digital filter design.* Wiley-Interscience, USA, oldalak: 153-158

[17] Kép forrása: Végtelen impulzusválasz Wikipedia oldala,
https://en.wikipedia.org/wiki/Infinite_impulse_response
(letöltve: 2020. nov.)

[18] T. W. Parks and C. S. Burrus. 1987. *Digital filter design*. Wiley-Interscience, USA, oldalak: 162-169

[19] Kép forrása: Butterworth szűrő Wikipedia oldala,
https://en.wikipedia.org/wiki/Butterworth_filter
(letöltve: 2020. nov.)

[20] Prabhu, K.M.M., *Window Functions and Their Applications in Signal Processing*, Taylor & Francis Group, 2013, oldalak: 91-94

[21] Silicon Labs publikus weboldala,
https://www.silabs.com/
(letöltve: 2020. nov.)

[22] Az Arm Ltd. Cortex-M3 processzorcsaládot bemutató weboldala,
https://developer.arm.com/ip-products/processors/cortex-m/cortex-m3
(letöltve: 2020. nov.)

[23] USER MANUAL, Starter Kit EFM32GG-STK3700, Silicon Labs,
online elérhető: http://www.farnell.com/datasheets/1886679.pdf
(letöltve: 2020. nov.)

[24] Kép forrása: Silicon Labs EFM32 Giant Gecko 32-bites mikrokontrollerét bemutató weboldala, https://www.silabs.com/mcu/32-bit/efm32-giant-gecko
(letöltve: 2020. nov.)

[25] A Silicon Labs, Simplicity Studio szoftvert bemutató weboldala,
https://www.silabs.com/developers/simplicity-studio
(letöltve: 2020. nov.)

[26] EM-60031 elektret mikrofon előerősítő gyártói weboldala,
http://www.elmodules.hu/products/22
(letöltve: 2020. nov.)

[27] Budapesti Műszaki és Gazdaságtudományi Egyetem, Beágyazott és Ambiens Rendszerek c. tárgy, 5-7. laborgyakorlathoz kiadott PDF,
szemeszter: 2019/2020/1

[28] Az elkészült eszközöm működésének demonstrációja,
Youtube: https://youtu.be/A3vfBg_k_Gc
(letöltve: 2020. dec.)