

Kajdácsi Szabolcs - Kajdácsi Balázs

Audió alapú mérések

Közúti járműforgalom mérése audió segítségével

Műszaki tudományok témakör

Témavezetők: Dr. Drenyovszki Rajmund, Dr. Kovács Lóránt

Tartalomjegyzék

1. Bevezetés	2
1.1. A közúti forgalom mérése	2
Célja, alkalmazása	2
Mérési módszerek és azok működése	3
Induktív hurokdetektor (sebesség mérése)	3
Piezoelektromos érzékelő (súly, sebesség mérése)	4
Kamerás képfeldolgozás (forgalomsűrűség, sebesség, rendszám ellenőrzés)	5
Fénymegszakításos detektor (sebesség, járműhossz)	6
Floating car data FCD	6
1.2. Mikrofonos (hang alapú) mérés	7
Hang terjedése, karakterisztikája és felépítése	7
Mikrofon hang rögzítési technikája	8
Analóg jel digitalizálása	8
Fourier-transzformáció	8
2. A mi kutatási területünk	10
2.1. Kezdetek	10
2.2. Utólagos (off-line) feldolgozó algoritmusok	12
Különböző forgalmi körülmények	12
Hagyományos algoritmusok	13
Küszöbölés (Threshold)	16
Burkoló (Sound-Envelope)	19
Frekvencia (Spektrum)	21
Gépi tanulás	24
Adatok gyűjtése, osztályozása	24
Modell feltanítása	24
Modell tesztelése	27
Modell alkalmazása sikeres tesztelés után	28
2.3. Valós idejű (Real-time, on-line) feldolgozás mikrovezérlőn	29
Felhasznált eszközök	29
Hagyományos algoritmus	30
Küszöbérték és Spektrum használata Real-Time	30
Gépi tanulás	31
Modell átépítése mikrovezérlős rendszerre	31
3. Összefoglalás	37
3.1. Eredmények és forgalmi körülmények	37
3.2. Kérdések, Fejlesztési lehetőségek	38
3.3. Felhasznált Szakirodalom	39

1. Bevezetés

1.1. A közúti forgalom mérése

Célja, alkalmazása

A forgalom mérése és megfigyelése jelentős szerepet játszik napjainkban a forgalomirányítási rendszerekben, mind hatékonyság mind a biztonság növelése érdekében.

Míg lakott területen elsősorban a forgalomirányítási jelzőlámpák forgalomhoz alkalmazkodó irányítása az elsődleges szempont, addig a gyorsforgalmi utakon és autópályákon a tájékoztatás változtatható digitális jelző- és információs táblákkal és az ellenőrzés súly- és sebességmérő, valamint rendszámfelismerő rendszerekkel.[\[1\]](#)

Korábbi kutatásokban találhatunk eredményeket hang alapú mérésekre, azonban a téma nem olyan elterjedt forgalommérés tekintetében, mivel csak alacsony forgalomsűrűség mellett biztosít elfogadható pontosságú mérést. Ezen kutatásokban is a cél, hogy egy minél egyszerűbb és olcsóbb rendszerrel egy elfogadható pontosságú mérést/becslést lehessen adni a forgalomra.

Egy, a lengyelek által készített tudományos cikkben a hangsúly a különböző időjárási körülmények (vizes, havas út) és kiemelkedő közúti esetek (balesetek, megkülönböztető jelzésű járművek) felismerésén van. Ezt a zajszint alkalmazkodó küszöbértékével valósítják meg, és utána a detektált tevékenységet kategorizálják gépi tanulás segítségével.[\[2\]](#)

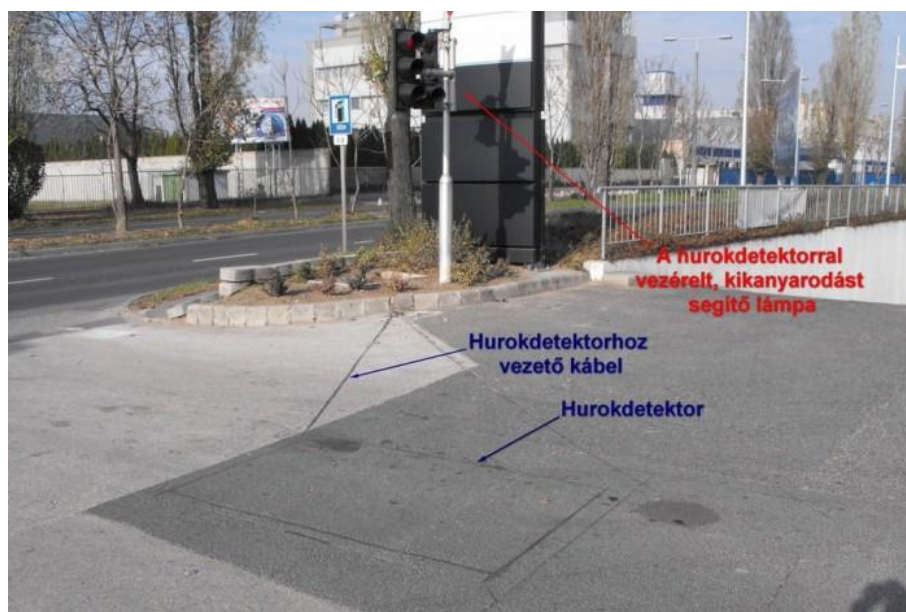
Egy másik publikációban a mi általunk is használt eszközökhöz hasonlóval történt a megvalósítás. Egy Arduino mikrovezérlőt használtak, melyen a beépített mikrofont hívták segítségül, amire egy [Edge Impulse](#) gépi tanulási modellt feltöltve osztályozták a járműveket biciklire, autóra, furgonra valamint egyéb zajforrásra. Mindezt egy saját, általuk készített telefonos alkalmazás segítségével monitorozták valós időben.[\[3\]](#)

Mérési módszerek és azok működése

A következőkben bemutatjuk a forgalom mérésére (járművek detektálására) általában alkalmazott módszereket és eszközöket.

Induktív hurokdetektor (sebesség mérése)

A legrégebb óta használt és a legelterjedtebb mérési módszer. Működése egy úttestbe épített hurokra kapcsolt mérőáramkör elektromágneses terének megváltozásán alapul. Amikor egy nagyobb fémtömeg (jármű) halad el felette, akkor megváltozik ezzel a tekercs rezonancia frekvenciája.



1. ábra Hurokdetektor alkalmazása jelzőlámpás kereszteződésnél

Induktivitás kiszámítása: $L = 1.1 \cdot K \cdot n^2$

L - hurok induktivitása μH -ben (mikrohenry)

K - hurok kerülete (1 menet hossza méterben)

n - menetek száma a tekercsben

1,1 - tapasztalati szorzószám ($\mu\text{H}/\text{m}$) [4]

Az átlagsebesség meghatározása az alábbi képlet alapján lehetséges:

$$\underline{v} = \frac{1}{N} \sum_{j=1}^N \frac{\underline{L} + L_{det}}{t_j}$$

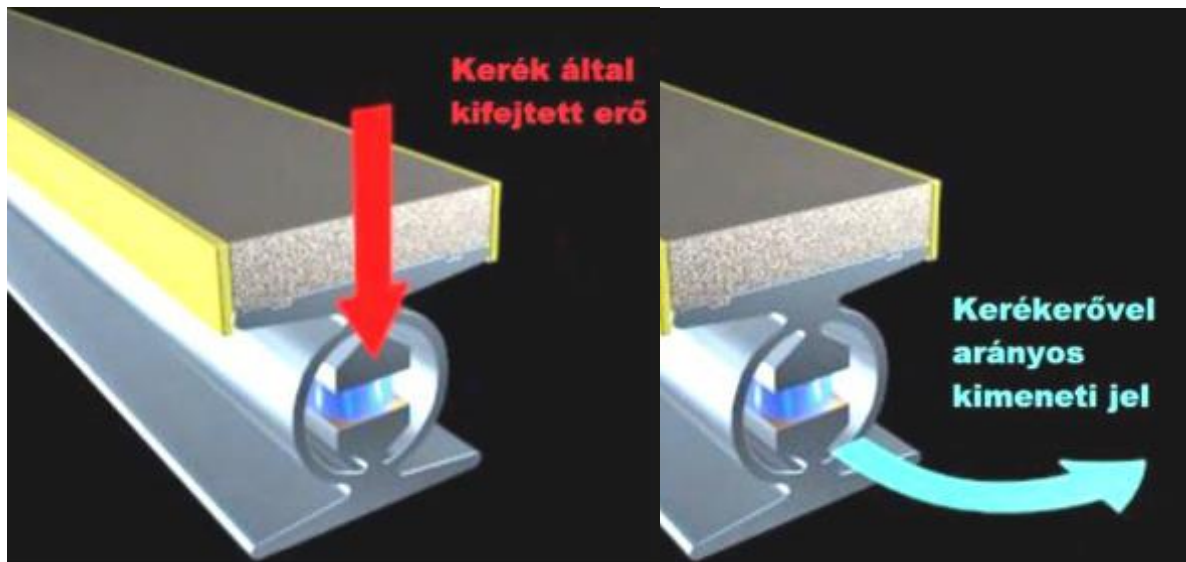
t_j - hurokdetektor foglaltsági idő

$\underline{L}$ - átlagos járműhossz

L_{det} - detektor hossz

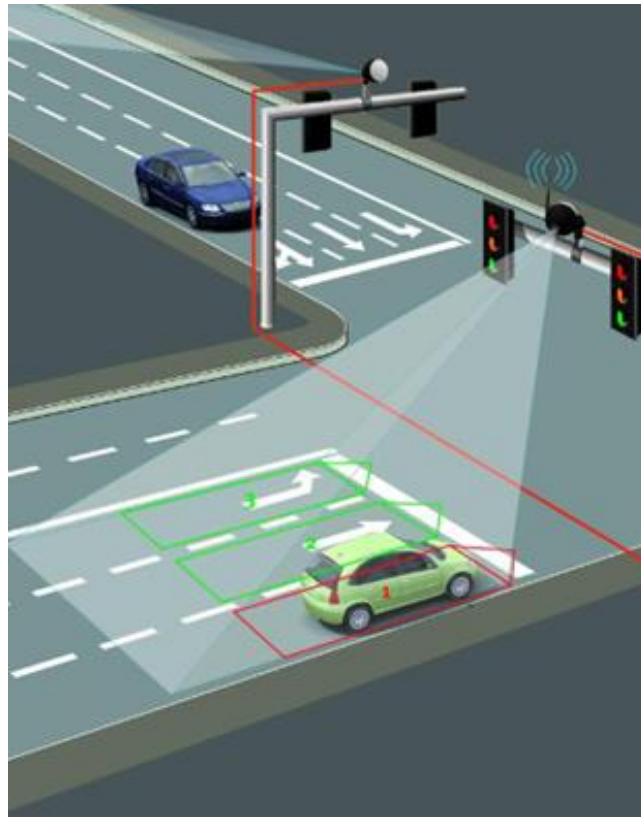
N - detektoron áthaladt járműszám T mintavételi idő alatt

Piezelektromos érzékelő (súly, sebesség mérése)



2. ábra Piezelektromos útburkolatba építhető súlymérő

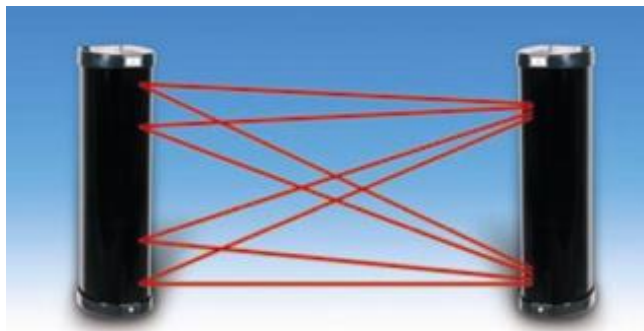
A piezokristályok tulajdonságát használja ki, melyek elektromos feszültséget indukálnak amikor fizikai hatás éri őket, jelen esetben függőleges terhelést kapnak. Az indukált feszültség egyenesen arányos az terhelés mértékével, ezáltal tökéletesen alkalmazhatóak a rajtuk áthaladó jármű súlyának mérésére. A haladási irányra merőlegesen beépített érzékelő alkalmas még a tengelyek számát mérni és ezzel egy viszonylag pontos tengelyterhelést számolni. Sebesség mérésére is alkalmas, amennyiben több van egymás után az útburkolatba építve. A piezelektromos érzékelők használata gyakoribb lett a tengelymérő rendszerek megjelenése óta.^[1]

Kamerás képfeldolgozás (forgalomsűrűség, sebesség, rendszám ellenőrzés)

3. ábra Kereszteződésbe telepített kamerás mérés és feldolgozás.

Az egyik legelterjedtebb forgalom mérési módszer, amit az útburkolat megbontása nélkül is lehet telepíteni. A rendszer egy kamerából és egy képfeldolgozó egységből áll (*videodetektor*). A képfeldolgozás az elkészült kép képmátrixán dolgozik és két felvétel közötti jelentős intenzitású különbségeket keresi, amint ez bekövetkezett ott valószínűleg valamilyen mozgás történt. Azután ezeket összehasonlítja a betanított mintákkal és egyezéseket keres. Előnye hogy bármilyen forgalmi viszonyok mellett képes 95%-nál pontosabb mérési eredményre. A hátrányokhoz a látást zavaró tényezők tartoznak, pl. időjárás, sötét / árnyékok, illetve azon autók színei melyek nagyon hasonlítanak az útburkolat színéhez, valamint karbantartást igényel ha a lencse elkoszolódik vagy valami eltakarja azt.^[1]

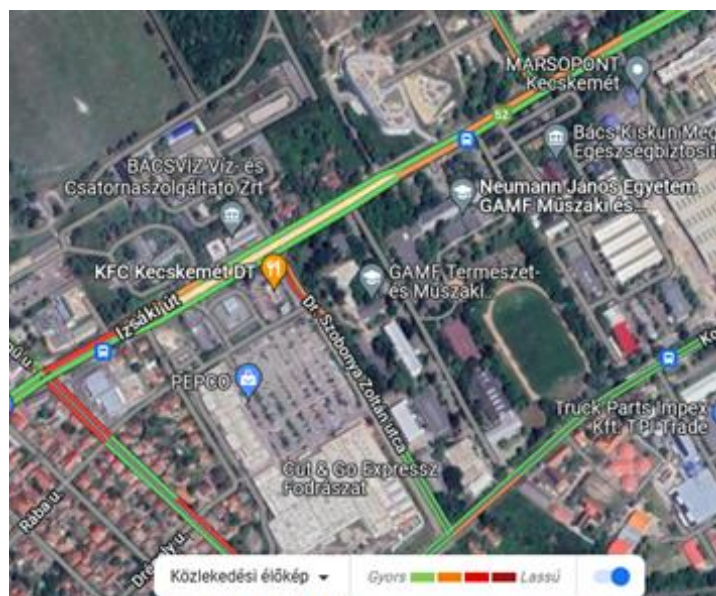
Fénymegszakításos detektor (sebesség, járműhossz)



4. ábra Fénymegszakításos érzékelő

A fénymegszakításon alapuló detektor egy adó- és vevőegységből áll, melyeket az út két oldalán egymással szemben helyeznek el, általában alacsony sebességű területeken alkalmazzák. Ezzel a módszerrel a megszakítás időtartama mérhető ($t = \text{járműhossz} / \text{átlagos sebesség}$), amelyből következtetni lehet a jármű típusára, így kategorizálni lehet azt. Hátránya az időjárási viszonyokra való érzékenysége.^[1]

Floating car data FCD



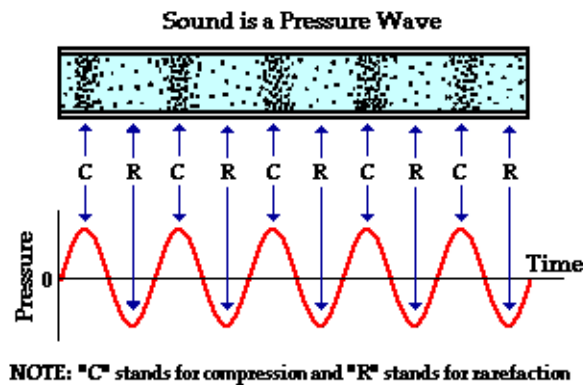
5. ábra FCD-n alapuló mérés a google-on

A floating szó jelentése a forgalommal együtt haladó járműre utal, amely a mérést végzi. A mérés során az autó útja követett, így egy adott útszakaszra megállapítható az átlagsebesség és a forgalomsűrűség. A mérés előnye, hogy elegendő csupán, ha a forgalomban résztvevő

járművek 2%-a adatot szolgáltat ahhoz, hogy a mérési pontosság 85%-os legyen. Erre a mérési módszerre egy nagyon jó és szinte bárki által elérhető példa a Google Maps Traffic funkciója.^[1]

1.2. Mikrofonos (hang alapú) mérés

Hang terjedése, karakterisztikája és felépítése



6. ábra Nyomásváltozás, azaz hanghullám megvalósulása

A hang, mint fizikai jelenséget mindig valamilyen rezgés váltja ki. Beszéd esetén a hangszálak, zenénél a hangszerek húrja / membránja, egyéb zajforrásoknál pedig magának a tárgynak a rezgése. De hogyan is halljuk mi ezeket a rezgéseket? A rezgésbe jött tárgy továbbadja ezt az energiát a levegő részecskéinek, amely hanghullámok formájában terjed amíg akadályba nem ütközik. A hang terjedése bármilyen folyékony vagy gáz halmazállapotú anyagban lehetséges, csupán a mértéke és minősége változik.

Egy egyszerű hang karakterisztikáját tekintve harmonikus rezgés (*szinuszos jel*), ami megfelel egy r sugarú körpályán egyenletes ω szögsebességgel keringő pont vetülete által leírt mozgásnak^[5]. A hang amplitúdója a hanghullám pillanatnyi részecske sűrűségét adja meg (*egyszerűbben a hang hangossága*).

$$y(t) = r \cdot \sin(\omega t + \varphi),$$

ahol r az amplitúdó értékek maximális értéke (csúcsértéke),

$$\omega = 2\pi f,$$

ahol f a rezgés frekvenciája, aminek reciproka T periódusidő,

φ pedig a fázis, azaz a kezdeti szöge.

A valóságban viszont nagyon ritkán találkozunk teljesen tiszta egyszerű hanggal. Az esetek nagy részében összetett hangokról beszélünk, amik különálló harmonikus rezgésből állnak

össze, amelyeknek a frekvenciájuk, amplitúdójuk és fázisuk is eltérhet. Ezek a hangok tulajdonságaik alapján szétválaszthatóak, emiatt tudunk hangelemzésről beszélni, amihez össze kell gyűjteni az összetett hang adatait.

Mikrofon hang rögzítési technikája

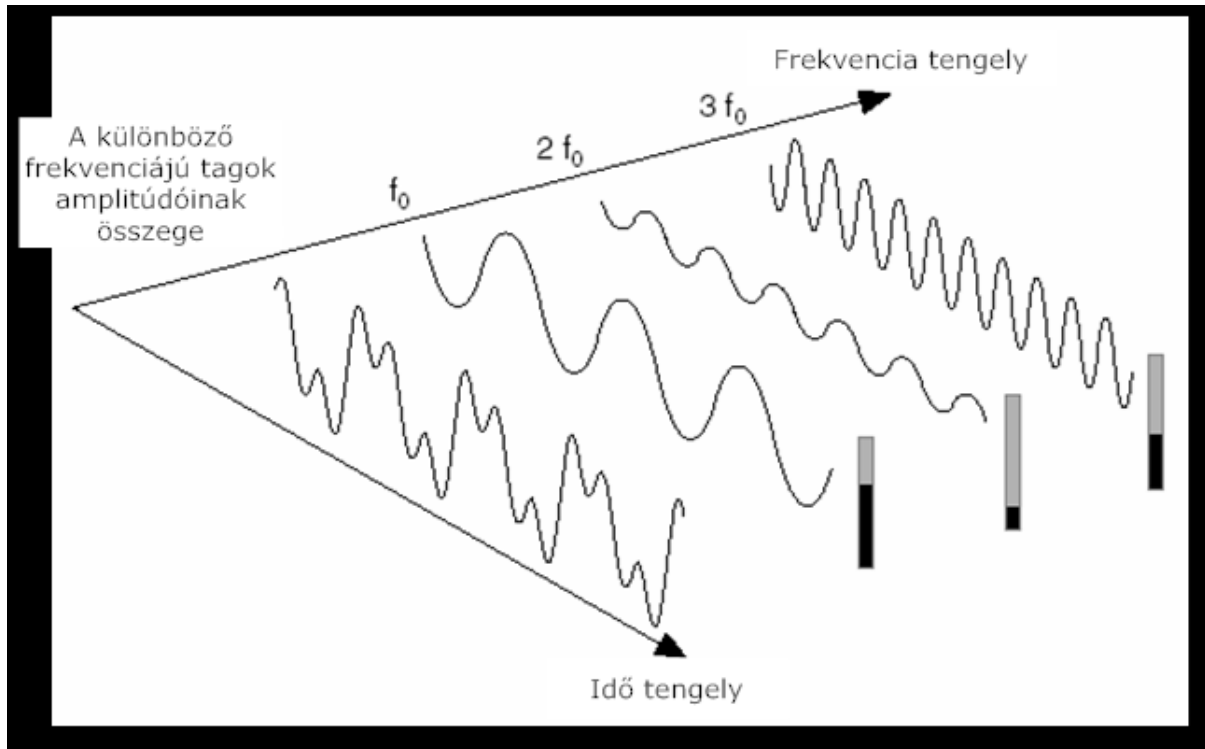
A mikrofonok technológia szerint nagyon változatosak, viszont működési elvük nagyon hasonló. A mikrofon hang érzékelésének lényege, hogy a hanghullámok rezgésbe hoznak valamilyen módon egy membránt, amit utána a mikrofon egy adott technikával elektromos jellé alakít át.

Analóg jel digitalizálása

A valóságban a jelforrások legtöbbször analóg formában jelentkeznek, azaz folytonosak. Mivel ezek az analóg jelek elméletben bármilyen értéket felvehetnek, amiket számítógépben nem tudunk ábrázolni, éppen ezért az analóg jeleket először digitalizálnunk kell, ami azt jelenti, hogy meghatározott időközönként mintákat veszünk az analóg jelből és ebből létrehozunk egy nem folytonos mintavételezett digitális jelet véges értékészlettel rendelkező számok formájában. Később ezekkel a számokkal végzünk különböző műveleteket a feldolgozási célunk érdekében. Minél sűrűbben veszünk mintát az időben, annál nagyobb a felvételünk pontossága, ezt mintavételi frekvenciának nevezzük, melyet hertz-ben $[Hz]$ adunk meg.

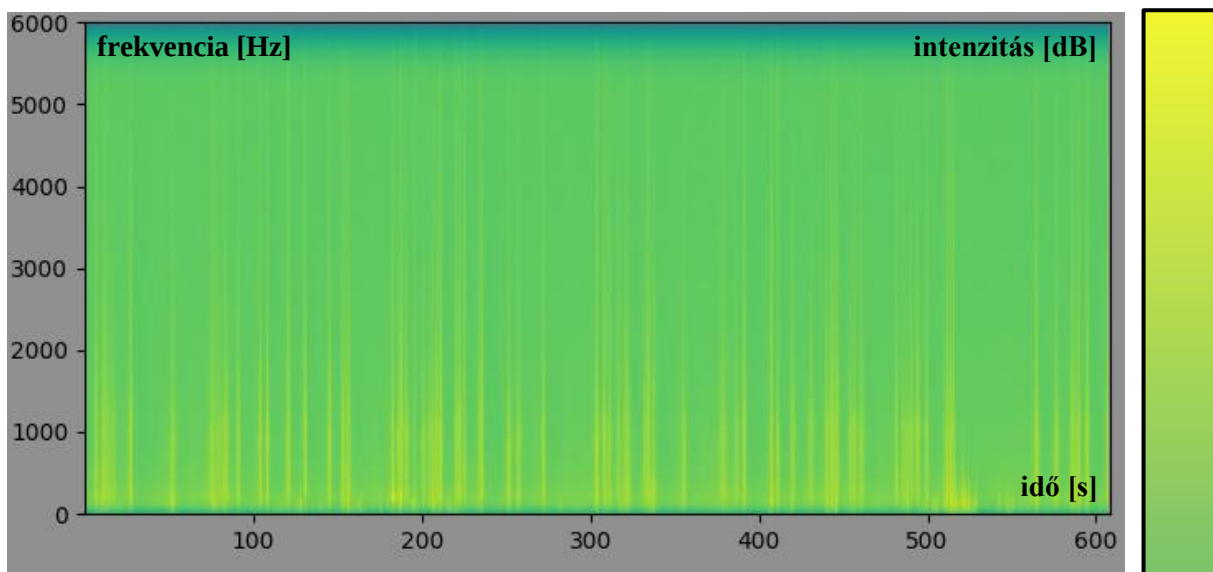
Fourier-transzformáció

A jelfeldolgozásban használt hasznos eszköz, melynek segítségével egy jel különböző tulajdonságainak elemzésére van lehetőségünk. Nevét Joseph Fourier-ről kapta.[\[6\]](#)



7. ábra A Fourier-transzformáció eredménye egy összetett hangon, idő és frekvencia tengelyen ábrázolva

A transzformáció célja az összetett jel szétbontása eltérő frekvenciájú komponenseire, amiből felépíthető egy adott pillanatban a hang spektruma, ezt idő függvényében nézve megkapjuk az idő/frekvencia alapú spektrogrammot.



8. ábra A Fourier-transzformáció spektrumon ábrázolva, idő/frekvencia tengely, a hőérték az egyes frekvenciaértékek intenzitását ábrázolja

A spektrogrammon a vízszintes tengelyen az idő, függőleges tengelyen a frekvencia található és az egy adott frekvencia értékhez egy adott időpillanatban tartozó pont koordináta színe határozza meg a frekvencia intenzitását. Az intenzitás azt mutatja meg, hogy az adott pillanatban az adott frekvenciaérték mennyire jelentős részét teszi ki az összetett hangnak, azaz a frekvenciaérték amplitúdóját.

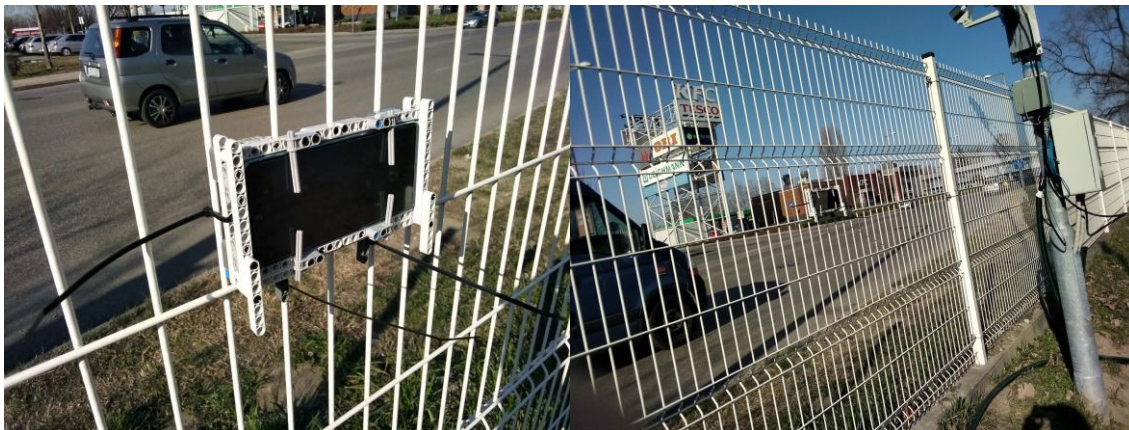
2. A mi kutatási területünk

2.1. Kezdetek

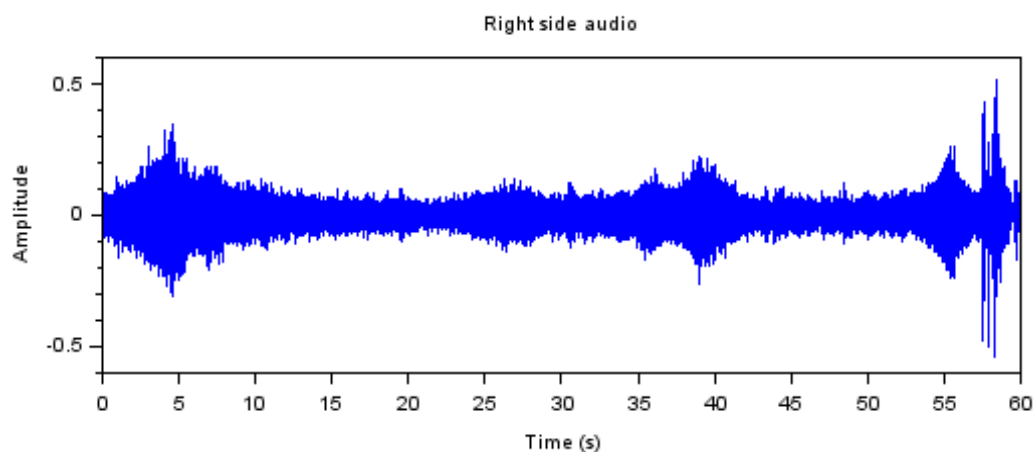
Projektünk egy egyetemi feladatból indult ki, melyben össze kellett hasonlítanunk különböző, kevésbé elterjedt, közúti forgalom mérésére alkalmas mérési módszereket. Elsőként a földben (útpadka) mérhető rezgésekkel próbálkoztunk, azonban a kezdeti mérések alapján kiderült számunkra, hogy a rezgés terjedés nem ideális az úttesten, szinte alig mérhető. Továbbgondolva a problémát a levegőben terjedő rezgések mérésével foglalkoztunk, ami valójában a hang mérését jelenti egy mikrofon segítségével. Mikrofonnal történő tesztméréseink során már több sikerrel jártunk és szemmel látható megfelelő eredményeket kaptunk. Ezután jött a következő kérdés, lehetséges-e automatizált mérést és jelfeldolgozást mikrofonos mérésekkel elvégezni? Dolgozatunk ezzel a témával foglalkozik, amiben részletesen leírjuk a mérési körülményeket, a mért adatokat, a feldolgozásukat és osztályozásukat. A mérési adatok feldolgozását számítógépen utólag, illetve valós idejű környezetben is elvégeztük a fejlesztett algoritmusok és eszközök segítségével.

Telefonba épített mikrofon

Először azt próbáltuk ki, hogy mennyire lenne egyszerű egy olyan készülékkel forgalmat mérni ami szinte mindenki kezében ott van, egy okostelefonnal és azon belül is a mikrofon használatával. Az ötlet eléggé eltér a manapság használatos forgalommérési módszerektől, de gondoltuk érdekességből érdemes ezzel kezdeni, és később kipróbálni esetleg más bevált és elterjedtebb módszereket. A méréssel kezdtük a feladatot, ahol 2 különböző kategóriás telefon mikrofonját hasonlítjuk össze.

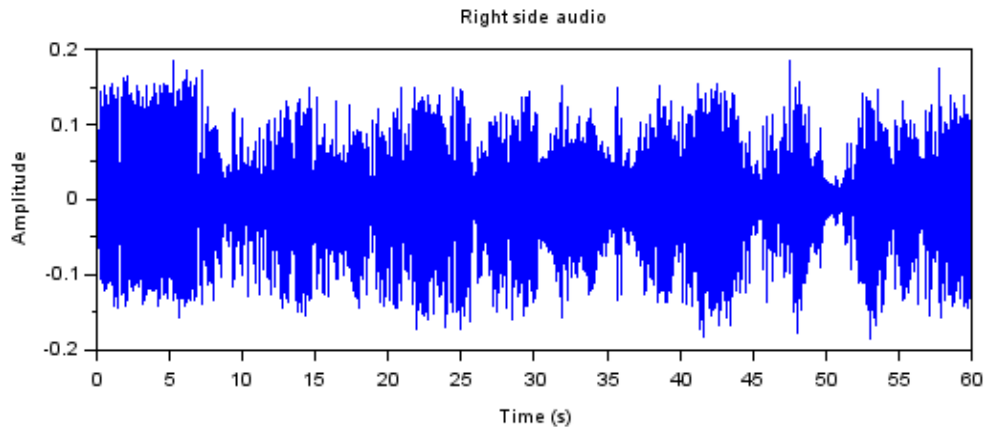


9. ábra Legelső “mérésünk” egy rögzített telefon segítségével



10. ábra Legelső mérés ábrázolása amplitúdó és idő függvényében

A legelső mérést egy Samsung Galaxy S20 FE készülékkel végeztük egy csütörtöki nap (2022.03.10) a délelőtti órákban (08:58:26 - 09:13:57). A tesztmérés 15 perc 31 másodperc hosszú, *(a telefonnal készíthető maximális mérés 20 percre van korlátozva)* az időjárás nem volt kedvező a széllelkések miatt, amik sajnos befolyásolták a mérés pontosságát. A mérésről egy rövid szakasz a 10. ábrán figyelhető meg.



11. ábra Legelső mérés egy alacsonyabb árkategóriás telefonnal (láthatóan több zaj)

A második mérést egy gyengébb minőségű Nokia 1.3-as telefonnal végeztük. A mérés ideje 2022.04.06 (13:42:05 - 13:44:32), hossza 2 perc 27 másodperc. Célunk a kétféle telefon mérési eredményének összehasonlítása volt. A különbség szemmel láthatóan is észrevehető (11. ábra).

2.2. Utólagos (off-line) feldolgozó algoritmusok

Különböző forgalmi körülmények

A mérési körülmények a legmeghatározóbbak egy mérés megtervezéséhez. Jelen esetünkben a forgalmi körülményeket kell jól megválasztani, ahol a mérést szabályozni tudjuk, azaz próbáljuk csak azt mérni amire szükségünk van, mindezt úgy, hogy a lehető legpontosabb legyen az. Ehhez figyelembe kell venni a forgalom sűrűségét, közlekedés rendjét, valamint az egyéb forrásokból érkező esetleges zavaró tényezőket, zajforrásokat.

Minden közúti mérésnek megvan a maga előnye és hátránya is. A hang alapú mérésnél nem befolyásolóak a látási és fényviszonyok, viszont az egyéb zajforrások annál inkább. A járművek hangja kiemelkedik a környezetéből, és a mérésünk erre alapszik, azonban ha a forgalom túl nagy, akkor a járművek hangja lesz a környezeti alapzaj, így mérésünk pontatlanná válik. Célszerű egy alacsonyabb forgalmú úton elvégezni a mérést.

Térjünk ki a közlekedésre. Elég sűrűn előfordul, hogy a forgalom nem csak egy sávon halad egyszerre, ez olyan bonyodalmakhoz vezethet, mint például több jármű halad el egyszerre egymással párhuzamosan vagy szemben, a mérést megnehezítve. Hasznos ilyen esetek

megelőzése érdekében minél kevesebb forgalmi sávból álló úton mérni, hiszen annál kevesebb lesz az esély az ilyen esetekre.

Említést kell tennünk továbbá a közlekedési csomópontokról is (keresztezések, le- és felhajtók, valamint körforgalmak). Mivel hang alapján nem egyértelműen meghatározható a járművek közeledési és távolodási iránya, így érdemes a csomópontokban való mérés elkerülése.

Hagyományos algoritmusok

Scilab

A rögzített hang feldolgozásához először Scilab-ban készítettünk egy programot (scriptet). A script a hangsáv amplitúdó értéke alapján határozza meg az elhaladó autók számát. Lényege, hogy ha talál értékben kiugró adatot, akkor növel egy számlálót. Ennek olyan hátránya van, ha esetleg több sávon halad ugyanabban az időben több jármű, akkor azt is csak 1-nek számolja és a más forrásból származó kiugró zaj forrásokat is beleszámolja, jelen esetben a széllelkéseket.

A script paraméterei:

- ampRange: két detektálás között mennyi másodpercrek kell eltelnie.
- ampThreshold: ha az amplitúdó érték átlépi ezt a szintet, az számít detektálásnak

ampRange	0.5	ampRange	0.5
ampThreshold	0.15	ampThreshold	0.15
db	91	db	11

12. ábra A kettő mérés futtatási paraméterei és eredményei

(Samsung Galaxy S20 FE) - A mérés időtartama alatt 99 elhaladó autót számoltunk.

(Nokia 1.3) - A mérés időtartama alatt 14 elhaladó autót számoltunk.

A magas háttérzaj és az algoritmus egyszerűsége mellett is jobb eredményt kaptunk mint amire számítottunk, viszont még sok a false positive és false negative találat is. A detektálás akkor false positive, amikor nem ment autó de mégis detektálás történt. A false negative pedig ha elhaladt egy autó, viszont nem volt detektálva.

A script második verziójának alapja hasonlít az előzőhöz, viszont itt már több állítható paraméter található:

```
targetSampling = 24000; //Hz
ampMergeRange = 0.4; //s
ampThreshold = 0.1; //amp-0-1
detectWidth = 0.65; //s
mozgoAtlagHossz = 3; //num-paratlan
```

13. ábra A futtatás paraméterei

targetSampling - mintavételezési frekvencia csökkentése, ezzel a program a felvett audioó mintavételét csökkenti le a kívánt értékre.

- Előny: a feldolgozás felgyorsítása.
- Hátrány: adatvesztés, ami pontatlanságot idéz elő.

ampMergeRange - a megadott értéket átváltja egy maximum index távolságra a sampling rate alapján és az ennél közelebb lévő threshold-ot átlépő (1-es) értékeket összevonja.

ampThreshold - ha egy amplitúdó érték átlépi ezt az értéket, akkor az adat indexéhez 1, ellenkező esetben 0 értéket rendel.

detectWidth - a megadott értéket szintén átváltja egy minimum index szélességre és az ennél nagyobb vagy egyenlő szélességű 1-es adatokat meghagyja, a többi törli (kinullázza).

A script mindezen lépések után összeszámolja a kiugrásokat, ami egyenlő az elhaladó autók számával.

mozgoAtlagHossz - a mozgó átlag ablak hosszát lehet vele állítani, ami az adathalmazt átalakítja.

Eredmények a fenti paraméterek használata esetén [mért (valós)]:

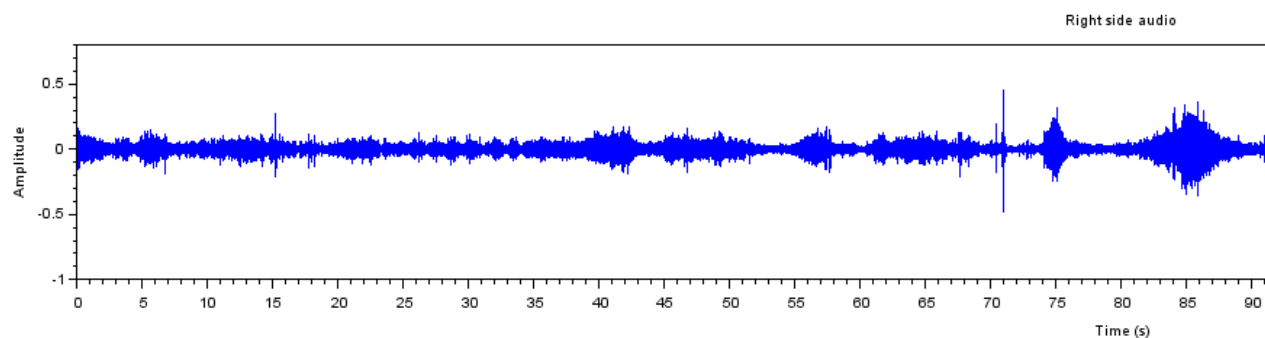
(Samsung Galaxy S20 FE) - 100 (99)

(Nokia 1.3) - 43 (14)

Következtetés:

Az első script 10 - 20 %-os pontatlansággal dolgozik a jó, és nem jó minőségű felvételeken is, míg a második script nagyon jó pontossággal dolgozik a nem zajos felvételen, ellentétben a zajossal, amin szinte használhatatlannak mondható a pontatlansága miatt.

Külső mikrofon

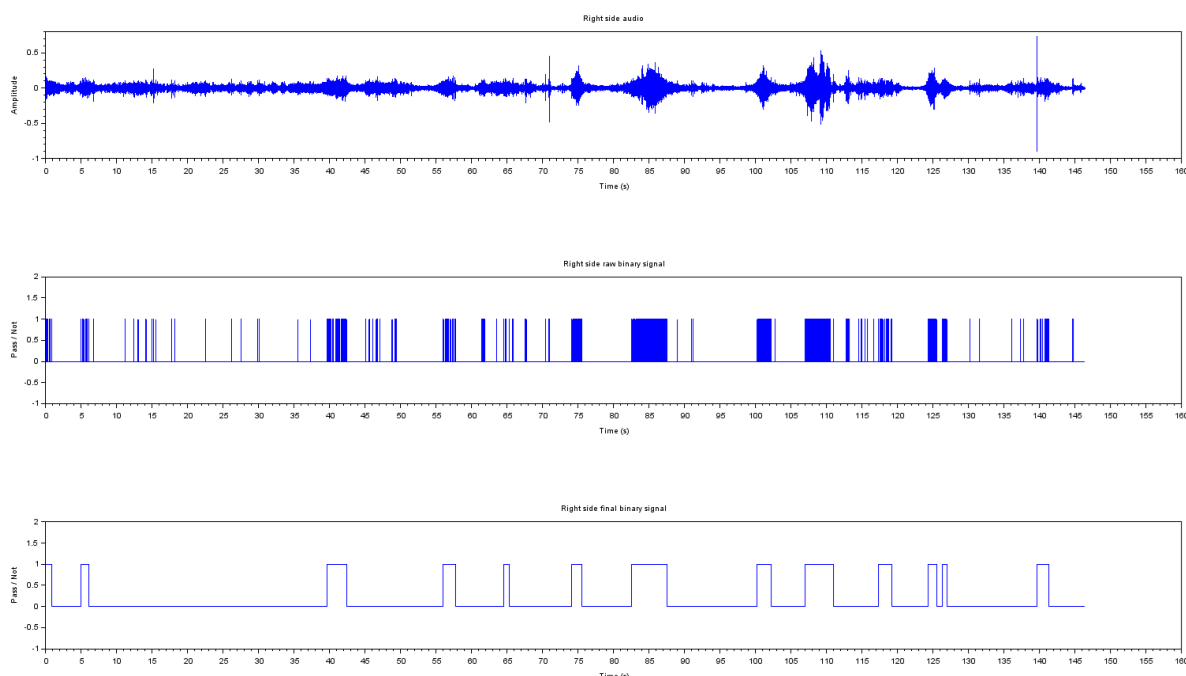


14. ábra Legelső mérés első része egy külső mikrofon segítségével

A telefonos méréseket követően egy irányított puskamikrofont használtunk, (*Audio-Technica ATR6550x*) azt feltételezve, hogy egy zajmentesebb mérés jobb eredménnyel fog szolgálni. A mérés ideje egybeesett a második telefonos méréssel (2022.04.06 | 13:42:05 - 13:44:32).

Első script eredmény: 12 (14)

Második script eredmény: 13 (14)

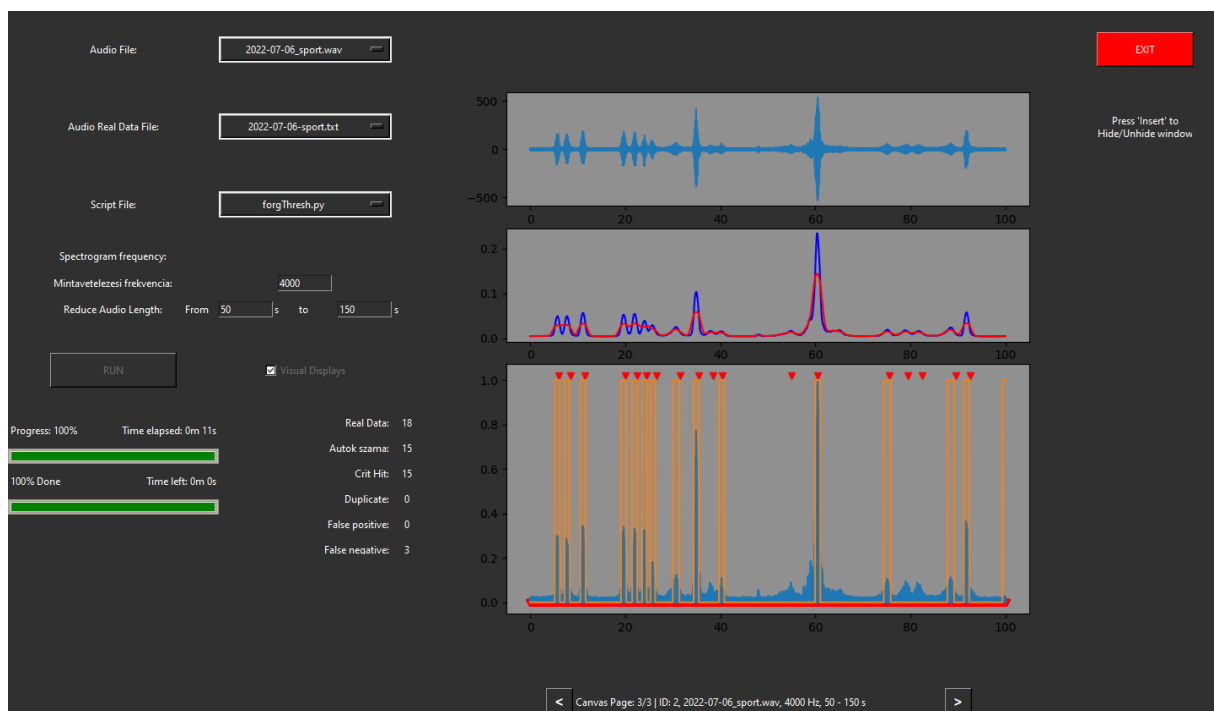


15. ábra A mikrofonos mérés feldolgozása a jelenlegi algoritmusokkal

A mikrofonos mérés hasonló pontosságú eredményt adott mint a jobb telefonnal történt mérés, a későbbiekben a viszont ugyanez a script nagyon eltérő eredményeket produkált a későbbi méréseken az egyszerűsége miatt, így ideje továbblépni egy másik megközelítés irányába.

Python alapú utólagos feldolgozás

Projektünkhöz készítettünk egy Python Tkinter grafikai felületet, melyben egy tetszőleges mérést tudunk futtatni állítható paraméterekkel, miután a végeredmény grafikus és szöveges formában is megjelenik az alkalmazáson belül, így segítve a tesztelést és az ellenőrzést. A mérési eredmények, valamint a grafikonok is mentésre kerülnek, így bármely futtatás visszanezhető később is.

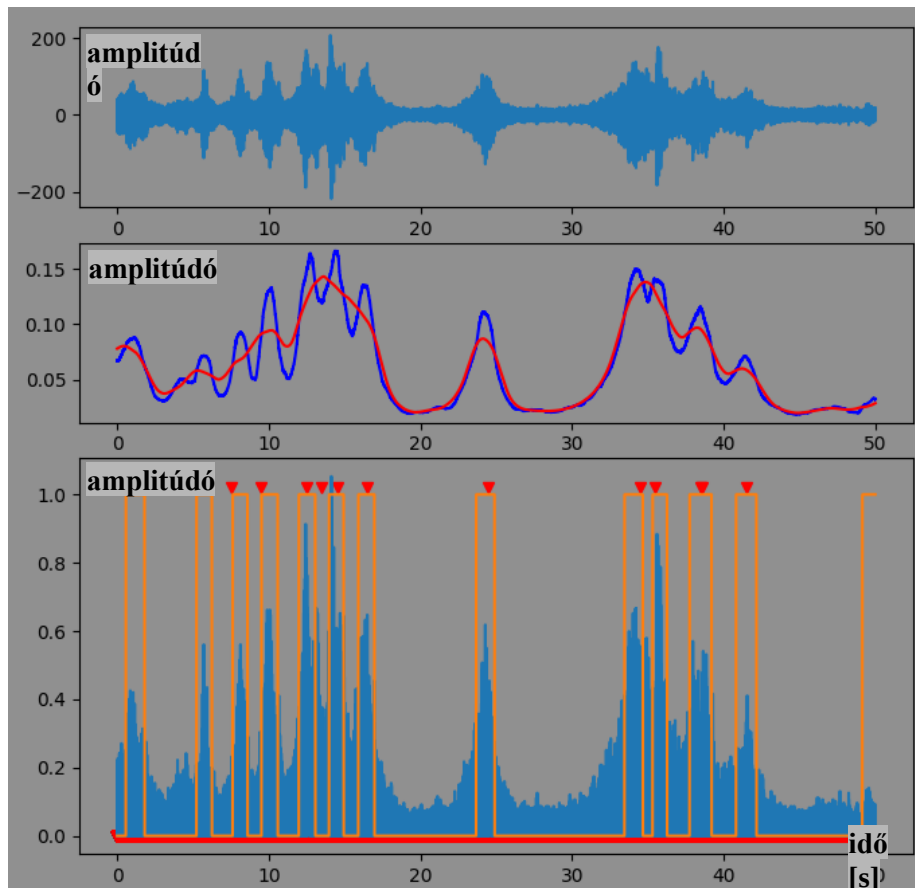


16. ábra Az utólagos feldolgozás Pythonban írt grafikus felülete

Küszöbölés (Threshold)

Amplitúdó maximum értékek kigyűjtése ez egy ezen adatokra alkalmazott automatikusan alkalmazkodó küszöbérték már elég jó eredményt produkált minden mérésre, viszont a meglévő problémák még mindig fennállnak: több egyszerre elhaladó autó megkülönböztetése és az egyéb forrásból származó zajok kiszűrése.

Amplitúdó értékek alapján nagy forgalomban a mérés nagymértékben pontatlan lesz, így egyelőre kisebb forgalomban kell majd a méréseket végezni.



17. ábra Küszöbölésen alapuló nagy forgalmú mérés eredményének diagrammja

A 17. ábrán egy nagy forgalmú mérés egy részlete látható; első grafikonon az eredeti audio látható kék színnel, a másodikon az adott pillanat környezetében lévő maximum amplitúdó értékek kékkel és az aktuális küszöbérték pirossal, valamint a harmadik grafikonon piros nyíl jelzi a valóságban elhaladó autókat (kézzel felvett értékek), kék színnel az eredeti audio amplitúdó abszolút értékeit, valamint narancs oszlopok a detektációt.

Az amplitúdó értékek meghatározása úgy történik, hogy egy adott ablak méretnyi (jelen esetben 1 másodperccel egyenlő mintavételek száma) adatot abszolút értékkel kiátlagol és aztán továbblép eggyel, ahol ugyanezt végrehajtja, így tulajdonképpen egy simított amplitúdó érték sorozatot kapunk, amivel a későbbiekben már könnyebb lesz dolgozni.

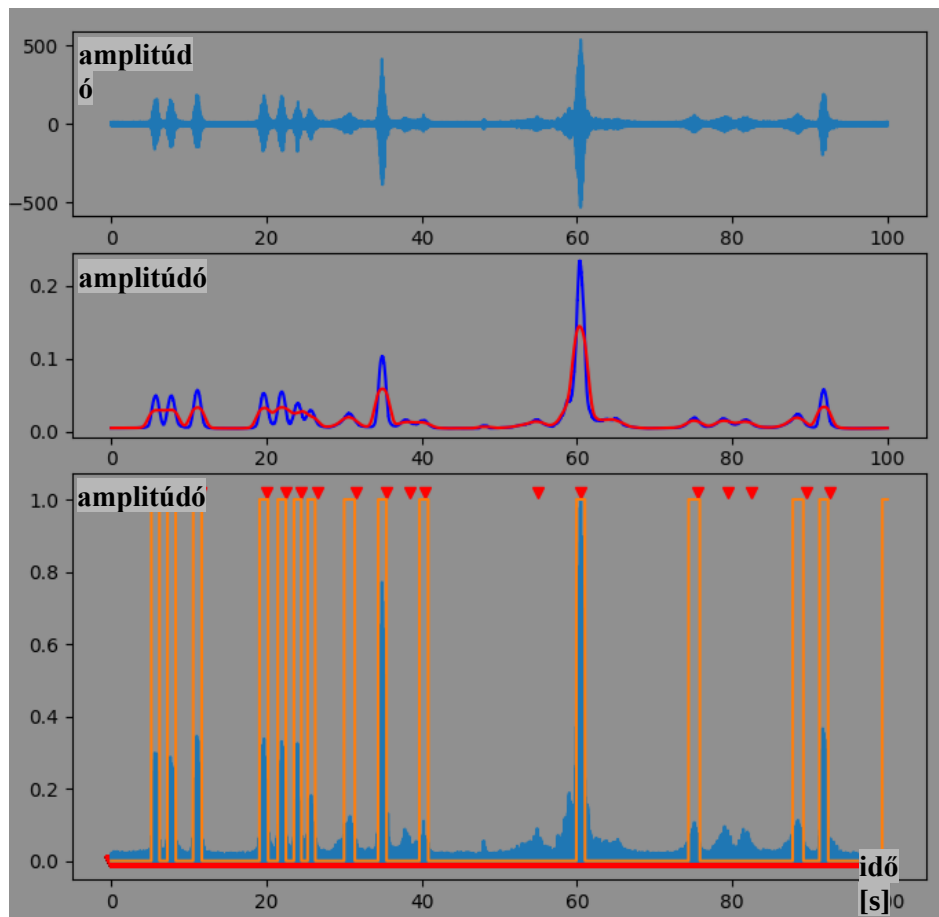
Küszöbölés alapú maximális amplitúdó vonal meghatározása

```
def getMaxLine(signalInput, Fs, ampLineWindow, window, pb, pbLabel, pb2, pbLabel2, isVisual, etaLabel, timeLabel, percent):
    firstFill = 0
    prevValue = 0
    outputSignal = np.zeros(len(signalInput))

    for i in range(len(signalInput)): # Egy ciklussal végigmegyünk minden elemen
        if i <= math.floor((ampLineWindow*Fs) / 2): # Hogyha az ablak eleje még kilóg az adatból
            if firstFill == 0: # Hogyha még nem töltöttük fel az első ablakot
                for j in range(math.floor((ampLineWindow*Fs) / 2) + i): # Akkor az első értékekkel töltjük fel az adatot
                    outputSignal[i] = outputSignal[i] + signalInput[j] # És összesítjük azokat
                firstFill = 1
            else:
                outputSignal[i] = prevValue + signalInput[i + int((ampLineWindow*Fs) / 2)] # Ha már feltöltöttük az első
                                                                                             ablakot, akkor csak az előző értéket kell hozzáadni
                prevValue = outputSignal[i] # Az előző értéket elmentjük, hogy később az átlátszónál használhassuk a gyorsabb
                                                                                             számítás érdekében
                outputSignal[i] = outputSignal[i] / (i + (ampLineWindow*Fs) / 2) # Végül kiszámoljuk az átlagot
        elif i > (len(signalInput) - math.floor((ampLineWindow*Fs) / 2)): # Hogyha az ablak vége már kilóg az adatból
            outputSignal[i] = prevValue - signalInput[i - int((ampLineWindow*Fs) / 2)] # Akkor csak az előző értékből kell
                                                                                             kivonni
            prevValue = outputSignal[i] # Az előző értéket elmentjük
            outputSignal[i] = outputSignal[i] / (len(signalInput) - (i - ((ampLineWindow*Fs) / 2))) # Végül kiszámoljuk az
                                                                                             átlagot
        else: # Ha az ablak eleje és vége is benne van az adatban
            outputSignal[i] = prevValue + signalInput[i-1 + int((ampLineWindow*Fs)/2)] - signalInput[i -
int((ampLineWindow*Fs)/2)]
            # Akkor az előző értékből kivonjuk az első értéket, és hozzáadjuk az utolsót
            prevValue = outputSignal[i] # Az előző értéket elmentjük
            outputSignal[i] = outputSignal[i] / (ampLineWindow*Fs) # Végül kiszámoljuk az átlagot
    if isVisual: # Ha a grafikus felületet használjuk akkor frissítjük a progress barokat
        if i % Fs == 0:
            percentPerCycle = (Fs/len(signalInput))
            update_progress_bar(window, pb, pbLabel, etaLabel, timeLabel, percentPerCycle * percent)
            update_progress_bar2(window, pb2, pbLabel2, percentPerCycle * 100, 'Preparing audio...  ')
    return outputSignal # Visszaadjuk a kiszámolt értékeket
```

A küszöbérték is hasonló elven működik annyi különbséggel, hogy az átlagolást a már meglévő átlagolt amplitúdóból számolja ki, és van egy állítható statikus értéke, ami segíti az alapzaj kiszűrését.

Utána ha az amplitúdó érték elég hosszan a küszöbnél magasabb értékű, akkor az detektálásnak minősül.

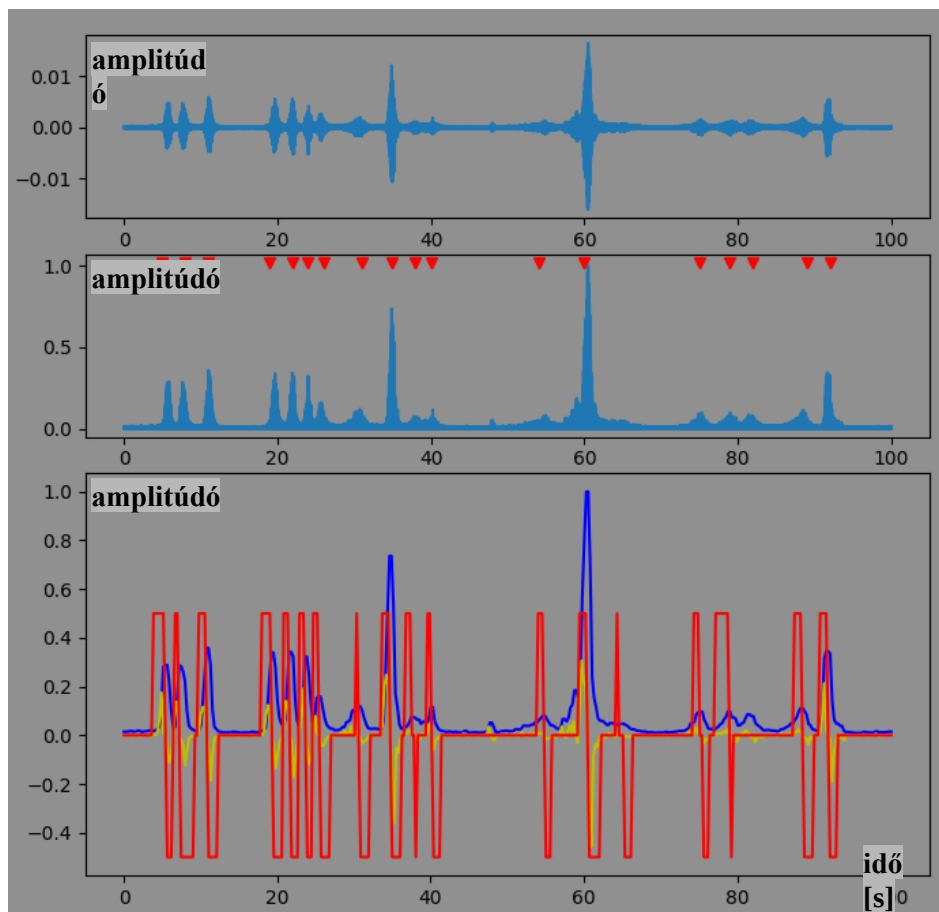


18. ábra Küszöbölésen alapuló kis forgalmú mérés eredményének diagrammja

A következő mérés már egy kisebb forgalmú úton történt, a 18. ábrán már sokkal jobban el lehet különíteni az autókat egymástól, viszont a halkabb járműveket a zajszűrés kiszűri

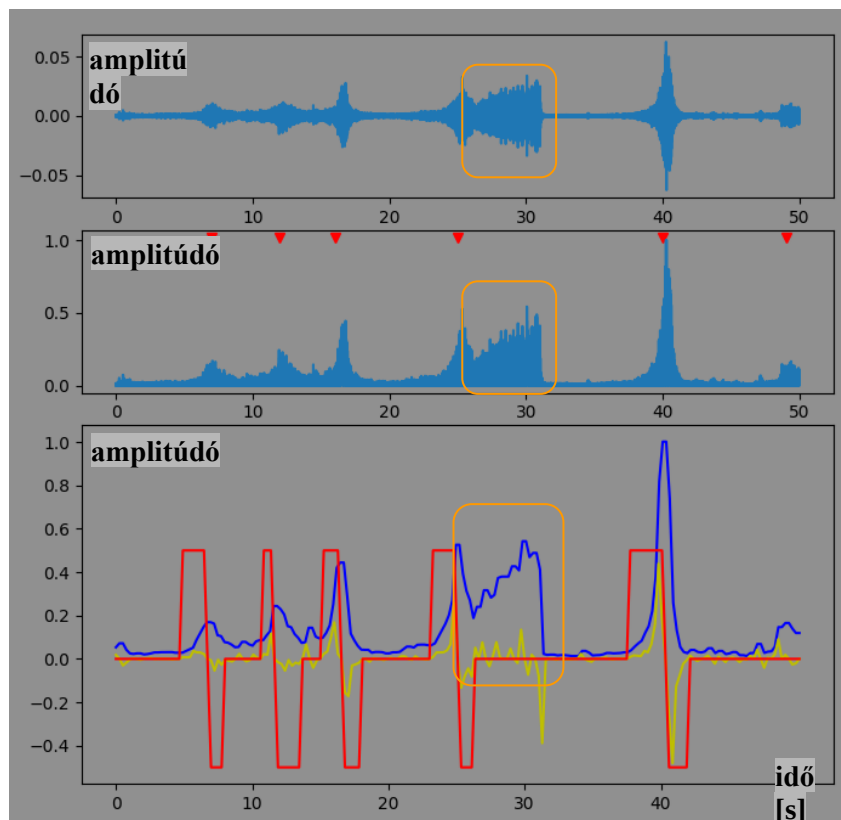
Burkoló (Sound-Envelope)

Egy másik algoritmus fejlesztésével meghatározható a járművek tipikus hangosodó / halkuló hangja, ezáltal el lehet ezeket különíteni egyéb zajforrásoktól. Az ötlet [Valerio Velardo](#)^{[12][13]} youtube csatornájáról származik, ahol a hang feldolgozásával foglalkozik különféle módszerekkel.



19. ábra Burkoló segítségével meghatározott hangosodó és halkuló típusú szakaszok kiemelése

A második diagramon háttérben lévő sárga adat mindig a differenciát mutatja az előző adathoz képest, így ha növekvő amplitúdó értékek vannak, akkor nagyobb mint 0, ha csökkenő, akkor pedig kisebb mint 0, a nullától való eltérés mértéke a meredekségtől függ. A piros adat egy egyszerűsített adathalmaz a differenciálok alapján, ha növekvő akkor 0.5, ha stagnál/nem egyértelmű akkor 0, ha pedig csökkenő akkor -0.5.

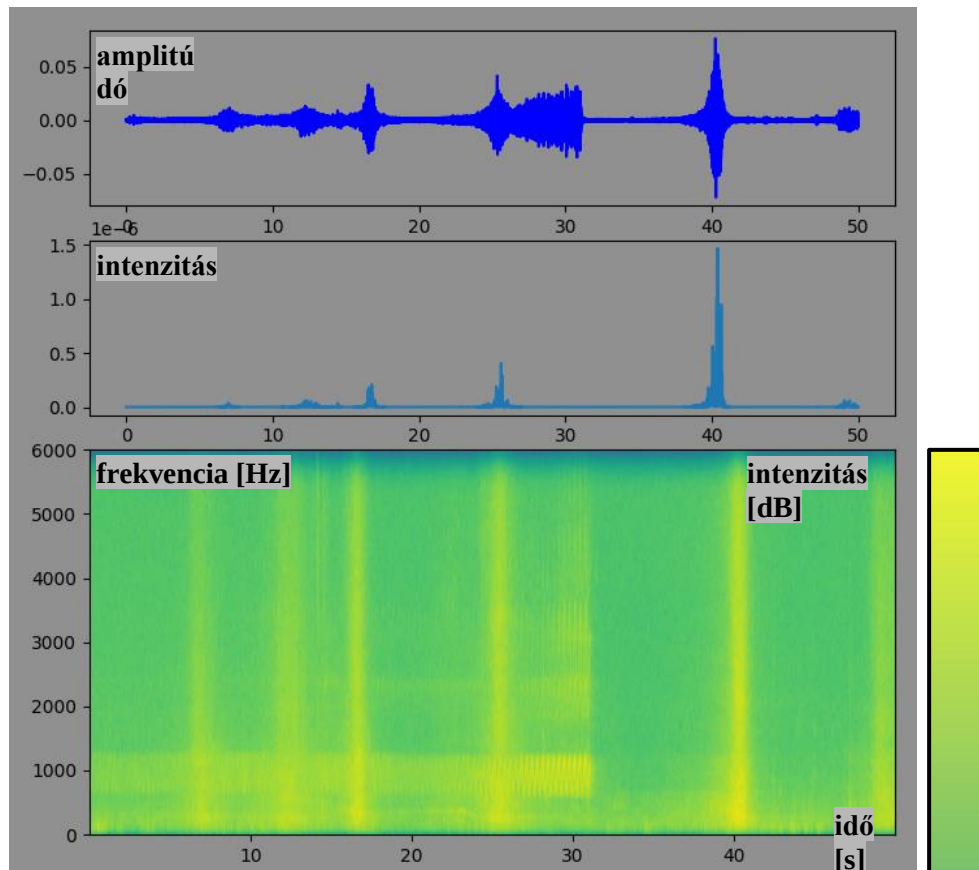


20. ábra Mentőautó szirénájának kiszűrése burkoló segítségével

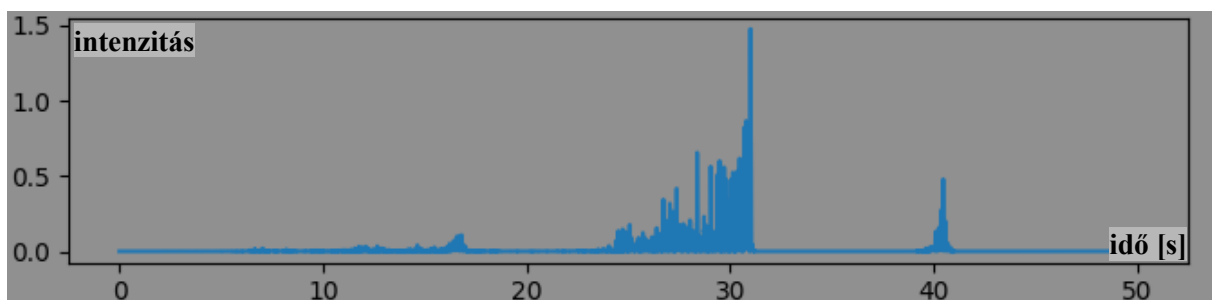
Ebben a példában egy mentőautó közeledett megkülönböztető jelzéssel (narancssárga keretezés), viszont amikor lekanyarodott a nagy forgalmú útról a kisebbre, kikapcsolta azt, ami a diagramon is észrevehető és a halkuló szakasz hiányában az algoritmus a magas amplitúdó értékek ellenében sem jelzett detektációt.

Frekvencia (Spektrum)

A spektrum vizsgálatával könnyen elkülöníthető a mentő szirénája az elhaladó autók zajától. A 21. ábra középső diagrammján 468.75 Hz intenzitása látható, amin egyáltalán nem vehető észre az.



21. ábra Mentőautó szirénájának kiszűrése spektrális szűrés segítségével



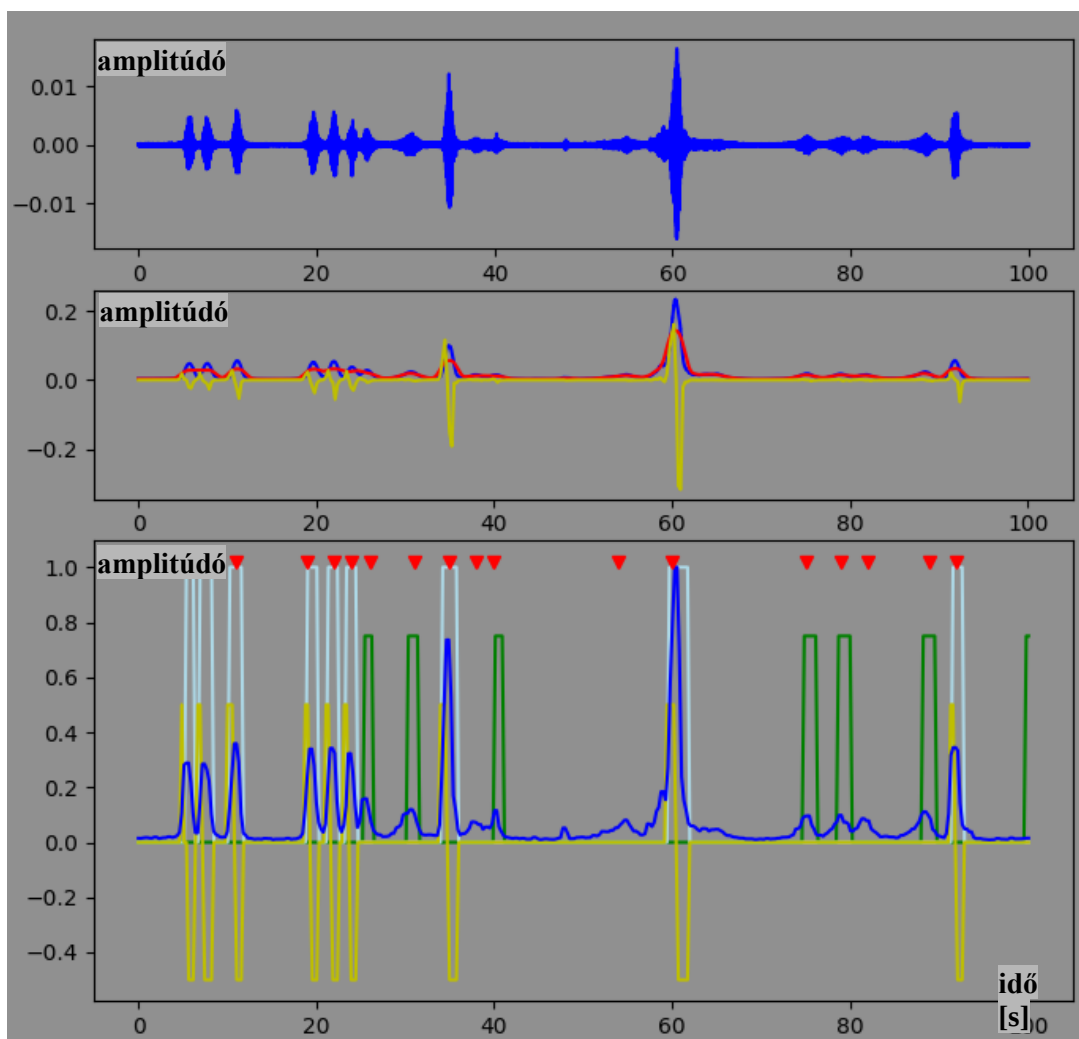
22. ábra Ellentétben az 1031.25 Hz intenzitását többségben csak az teszi ki.

Kombinált (Küszöb, Burkoló és Frekvencia figyelembe vétele)

A harmadik példa kombinálja az előzők fejlesztett tulajdonságait és összesíti őket a harmadik diagramon. Világoskékkel azok a detektációk vannak jelölve amiket mind a két algoritmus detektált, míg zölddel a csak valamelyik által detektált járművek vannak jelölve. Az elhaladó járművek száma úgy van meghatározva, hogy a biztosan detektáltak, valamint az összes egyéb detektált járművek számát veszi, melyek huzamosabb ideig átlépték az 1200 Hz-es frekvencia küszöböt. Ebben az esetben a 15 és 3 összegét, ami 18-al egyenlő és jelen esetünkben ez a szám megegyezik a valóságban elhaladó járművek számával.

Gyorsított algoritmus, ahol már csak a csúcserőtekeket vesszük figyelembe

```
def ampEnvelope(signal, frameSize, hopLength, window, pb, pbLabel, pb2, pbLabel2, isVisual, etaLabel, timeLabel, percent):
    ampEnv = []
    for i in range(0, len(signal), hopLength): # Végiglépkedünk az adatokon a hopLength lépésközzel
        curFrameEnv = max(signal[i:i+frameSize]) # Megkeressük a legnagyobb értéket a jelenlegi ablakban
        ampEnv.append(curFrameEnv) # Hozzáadjuk az eredmények listájához
    if isVisual: # Ha a grafikus felületet használjuk akkor frissítjük a progress barokat
        percentPerCycle = (hopLength/len(signal))
        update_progress_bar(window, pb, pbLabel, etaLabel, timeLabel, percentPerCycle * percent)
        update_progress_bar2(window, pb2, pbLabel2, percentPerCycle * 100, 'Creating envelope... ')
    return np.array(ampEnv) # Visszaadjuk a listát numpy tömbként
```



23. ábra Kombinált módszer alkalmazása

Gépi tanulás

Adatok gyűjtése, osztályozása

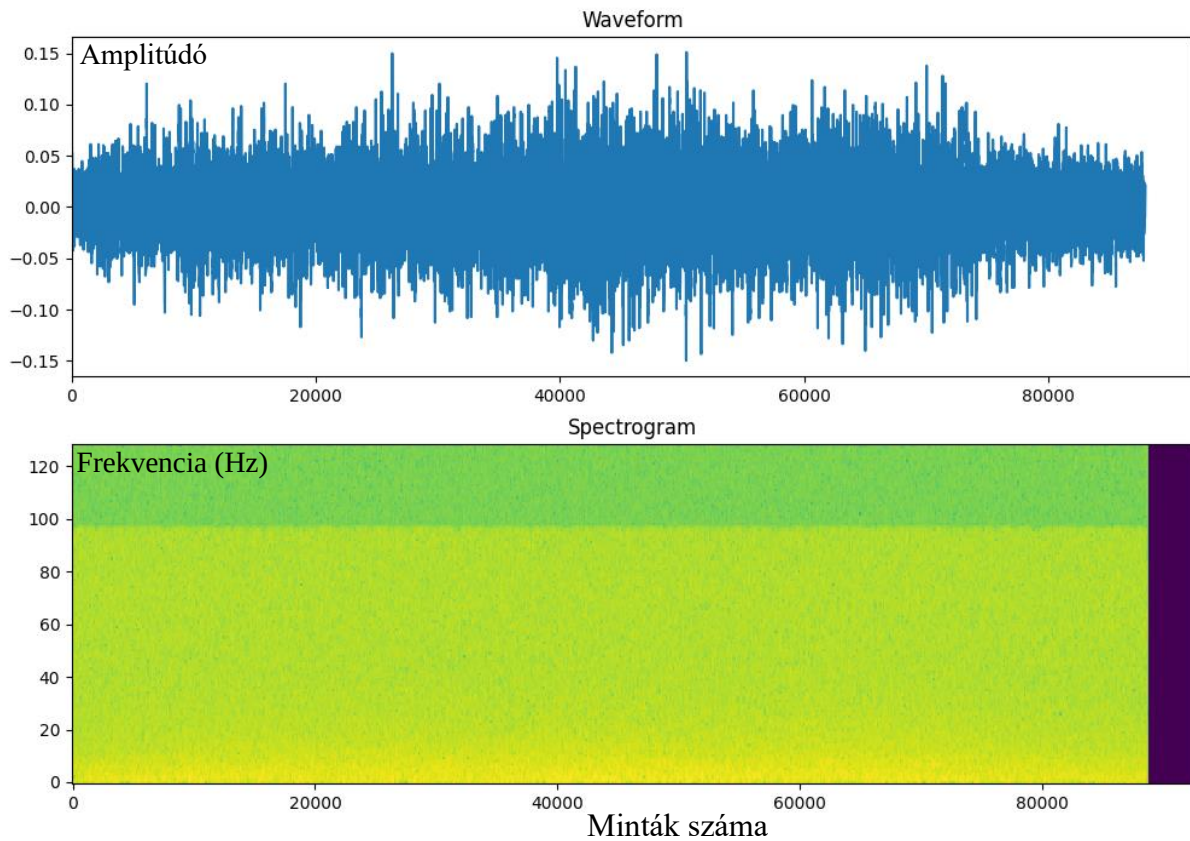
Az adatokat egy irányított mikrofon segítségével rögzítettük, egyszerre legalább 5 percet, majd manuálisan, Audacity használatával megváltuk és szétválogattuk autó hang, egyéb zaj és megkülönböztető jelzést használó jármű osztályokba. Az osztályozott hangfájlok külön mappákba lettek elhelyezve, ezzel segítve a gépi tanulás felcímkezését. Az adatok 48 kHz-en felvett .wav kiterjesztésű fájlok, a legjobb minőség érdekében.

Modell feltanítása

A modell elkészítéséhez a TensorFlow oldalán található “Simple audio recognition: Recognizing keywords” segédanyag kódja jelentette a kiindulási alapot.^[8] Ezt a kódot kicsit átalakítva az adathalmazhoz tudtuk igazítani a modellt, ez segített a hang alapú gépi tanulás feltérképezésében, megértésében.

A feltanítás megismételhetősége érdekében minden véletlen változót ki kell küszöbölni. A TensorFlow feltanítási módszerei minden futásnál más véletlen számot generálnak, így más lesz a kimenet eredménye is. Egy seed megadásával állandósítani lehet ezt a folyamatot, így a kimenetek két azonos paraméterekkel való futtatás között mindig azonosak lesznek.

Az adatokat spektrogrammá alakítva a legegyszerűbb a tulajdonságaikat feltanítani. Az audiót át kell egy csatornásra alakítani, különben a program hibára fut a két csatorna miatt. A hullám formátumot Short-Time Fourier Transform (STFT) segítségével lehet átalakítani.^[9] Ez annyiban különbözik a Fast Fourier Transform (FFT, a diszkrét Fourier-transzformáció számítógépes algoritmusokkal való megvalósítása)^[10] megoldástól, hogy a frekvenciák pontos helyét is meg



24. ábra Hullámformátum és spectrogram

lehet határozni csúszóablakos megoldással, nem csak a frekvenciák csúcsertékét.

Az átalakítás után a terjedelmét/magasságát kell meghatározni. Ezután hozzá kell adni egy új dimenziót, hogy képszerű bemeneti adat legyen belőle.

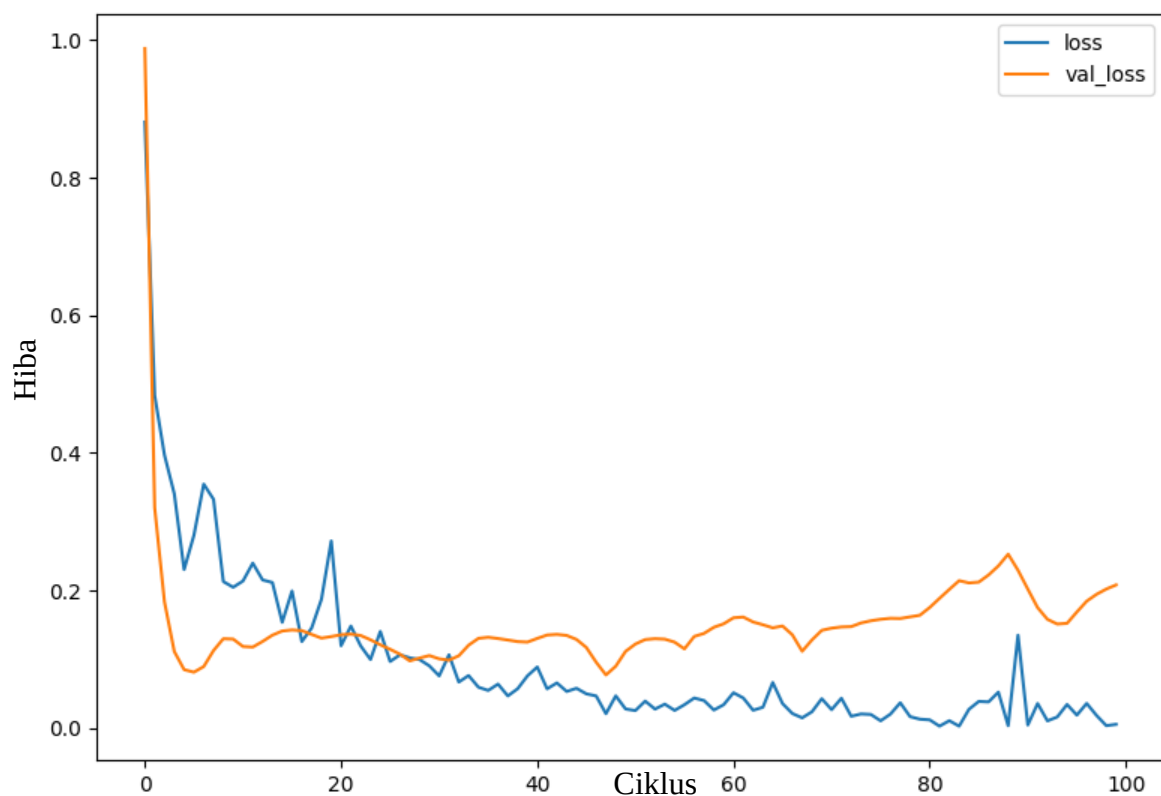
A címkéket a mappák nevéből lehet kinyerni, ezt egy tömbben elég tárolni. Az adatokat 2 részre kell osztani, tanítási és ellenőrzési osztályba, ez nagyjából 80% tanítás, 20% ellenőrzés arányban történik.

A feltanítás a TensorFlow Keras moduljával történt. Ehhez kell egy normalizációs réteg, ami hozzáilleszti a réteg állapotát a spektrogramhoz. Ezután egy szekvenciális Keras modell létrehozásával a modellt lehet elkészíteni. Ez tartalmazza:

- a normalizációs réteget valamint
- egy átméretező réteget (`Resizing()`), amely egyenlő méretűekre kicsinyíti az átalakított hangfájlokat
- 2 darab 2 dimenziós konvolúciós réteget (`Conv2D()`), egy konvolúciós réteget hoz létre, ami elvégzi a konvolúciót a bemeneti réteggel, így egy kimeneti tenzort hoznak létre
- egy 2 dimenziós leskálázó réteget (`MaxPooling2D()`), ez leskálázza a bemenetet a magassági és szélességi irányban úgy, hogy mindig az adott maximumot keresi meg egy bemeneti ablakon belül

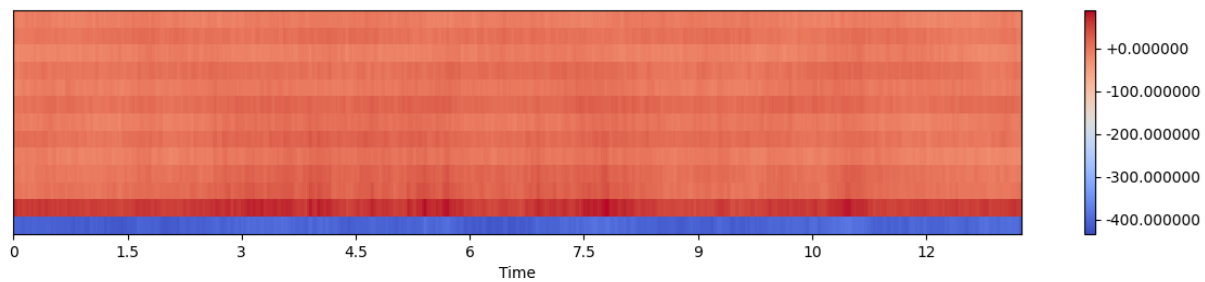
- 2 darab “kiejtő” (Dropout()) réteget a modell különböző pontjain, ez a réteg egy 0-1 közötti számot vár. Ez a réteg véletlenszerűen 0-ra állítja a bemeneteket a megadott valószínűséggel. Amelyik bemenet nem 0 lett, azt felskálázza $1/(1 - \text{valószínűség})$ értékkel, így a súlyok összege nem változik
- egy ellaposító réteget (Flatten()), amely a bemenet dimenzióit csökkenti 1-re, a többi dimenzióban lévő adatot a tömb végéhez fűzi hozzá
- 2 darab sűrű réteget (Dense()), amelyek közül az utolsó a címkék számát kapja meg paraméterként, ennyi neuronja lesz ennek a rétegnek

Ezután a modellt le kell fordítani, majd feltanítani több cikluson keresztül.



25. ábra Hiba (kék) és validációs hiba (narancs)

Másik megoldásként az adatokat át lehet alakítani MFCC (Mel-frequency cepstral coefficients)[\[11\]](#) értékekké (ebben Valerio Velardo Deep Learning (Audio) Application: From Design to Deployment című lejátsszási listája segített).[\[14\]](#) A Mel a Melody szó rövidítése, értéke: $m = 2595 \log_{10}(1 + \frac{f}{700})$, ahol f a frekvencia (Hz-ben), m pedig a Mel-érték.[\[15\]](#) A cepstral a spectral szó első négy betűjének megfordításából keletkezett, ez az érték a becsült jel spektrumának logaritmusára elvégzett inverz Fourier transzformáció eredménye.[\[16\]](#)



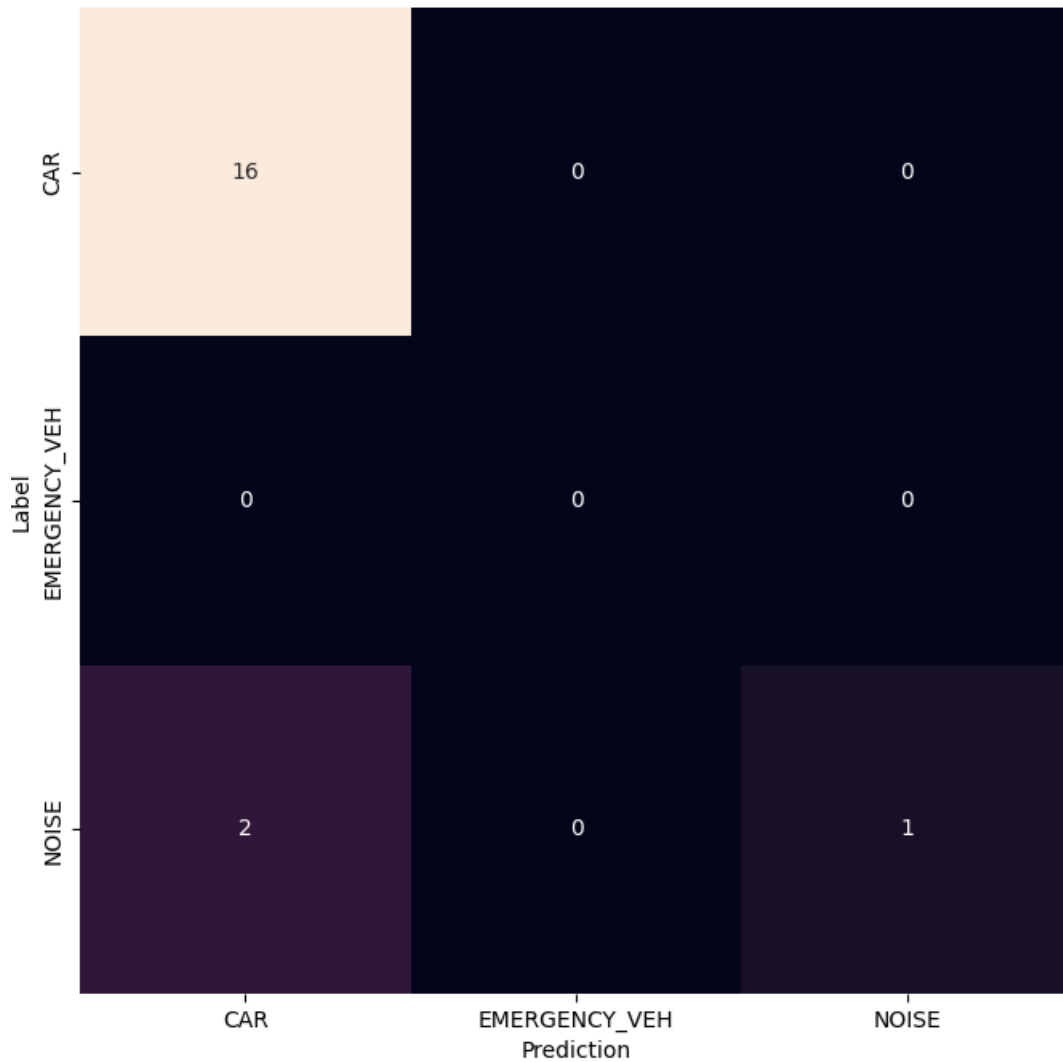
26. ábra MFCC értékek

Azokat az adatokat, amelyek meghaladják az egy másodperc hosszúságot, le kell rövidíteni egy másodpercre, amelyek nem érik el ezt a hosszúságot, azokat feltölteni 0-kal. Hosszabb felvételhossz esetén mindig statikus és rossz értékeket ad vissza, a 0-val való feltöltés miatt. Az MFCC kinyeréséhez a Librosa nevű Python könyvtár nyújt segítséget, ennek segítségével egy sornyi kóddal meg lehet határozni az értékeket. A meghatározott értékeket, a címkéket és a fájlok elérési útját egy .json kiterjesztésű fájlba kiírva lehet a továbbiakban felhasználni. Ezt a fájlt betöltve egy másik programban lehet létrehozni a modellt. Az adatok itt is tanítási és teszt csoportokban vannak. A modell egy szekvenciális Keras modell, három konvolúciós, egy ellaposító és egy softmax réteget tartalmaz. A konvolúciós rétegek egy Conv2D(), egy BatchNormalization() és egy MaxPooling2D() réteget tartalmaznak, az ellaposító réteg egy Flatten(), egy sűrű (Dense()) és egy kiejtő (Dropout()) réteget tartalmaz, a softmax réteg pedig egy sűrű réteget softmax aktivációval. Ezek után a modellt le kell fordítani és feltanítani több cikluson keresztül, majd elmenteni egy .h5 kiterjesztésű fájlba.

Modell tesztelése

A feltanított modell a teljes adathalmaz 20%-ával tesztelésre kerül, ezek a modell számára új, sosem “látott” adatok, ezzel pontosabb képet lehet kapni a valós pontosságról. Minden tanítási ciklus végén megjeleníti konzolon az adott ciklus tanítási hibáját, pontosságát, validációs/teszt hibáját és pontosságát. A pontosság pár cikluson belül eléri a maximumot, ezért ez nem feltétlen ad pontos információt. A hiba viszont folyamatosan változik, ha konzisztensen csökken, akkor jó a modell komplexitása és mérete. Amikor a hiba csökken, majd egy pont után elkezd emelkedni, az overfitting jelensége figyelhető meg. Itt a modell a tanítási adatokat jól megtanulja, de az új adatokat nem tudja általánosítani. Ennek ellenkezője az underfitting, amikor a hiba értéke magas marad. Ez akkor jelentkezik, amikor a modell nem tudja az adathalmaz jellemzőit modellezni, ezáltal nagy hibákat generálva. Amikor a modellre sem overfitting, sem underfitting nem jellemző, a modell várhatóan jól fog teljesíteni, el lehet menteni későbbi használatra. Ez után megjelenik egy confusion matrix, amivel a modell

teljesítményét lehet szemléltetni. Az x tengelyen található a címke predikciója, az y tengelyen a tényleges címke. Ebben az esetben a legjobb, ha a bal felső sarokból a jobb alsóba tart az átló és nincs benne nagy eltérés.



27. ábra Hiba mátrix

Modell alkalmazása sikeres tesztelés után

Az elkészült modellt egy új programban lehet felhasználni. A mentett modell betöltése után a modell alkalmazása 2 külön szálon történik^[20], előre felvett^[19], adott hosszúságú fájlakon. Az első szál a felvételért felelős. Jelenleg 10 db 10 másodperc hosszú felvételt készít, majd előről kezdi és felülírja a már meglévő fájlokat, mivel ennyi idő elég a másik szálnak, hogy feldolgozza az adatokat. A másik szál a feldolgozásért felelős, megvárja, amíg a felvétel befejeződik, utána kezd el dolgozni az éppen felvett fájlon. Ez a felvett audiót konvertálja

spektrogrammá, mivel a modell is ezzel a módszerrel készült el. A konvertálás után pedig ad egy predikciót. A predikció egy egész szám, 0 és 2 közötti értékkel, a címkéket reprezentálja betűrendben. Ezzel a program jelenleg nem ad vissza jó értékeket.

Az MFCC alapú modellt betöltve ebbe a programba egy új algoritmusra is szükség van. Ez betölti az éppen felvett hangfájlt, végez rajta egy előfeldolgozást, amivel át tudja alakítani MFCC értékekké, majd tesz egy predikciót és kiírja konzolra az eredményt.

```
Recording started at: 12:44:09.303262
1/1 [=====] - 0s 96ms/step
NOISE
prog time: 0:00:00.322613
File infered: Audio/test/test_2.wav

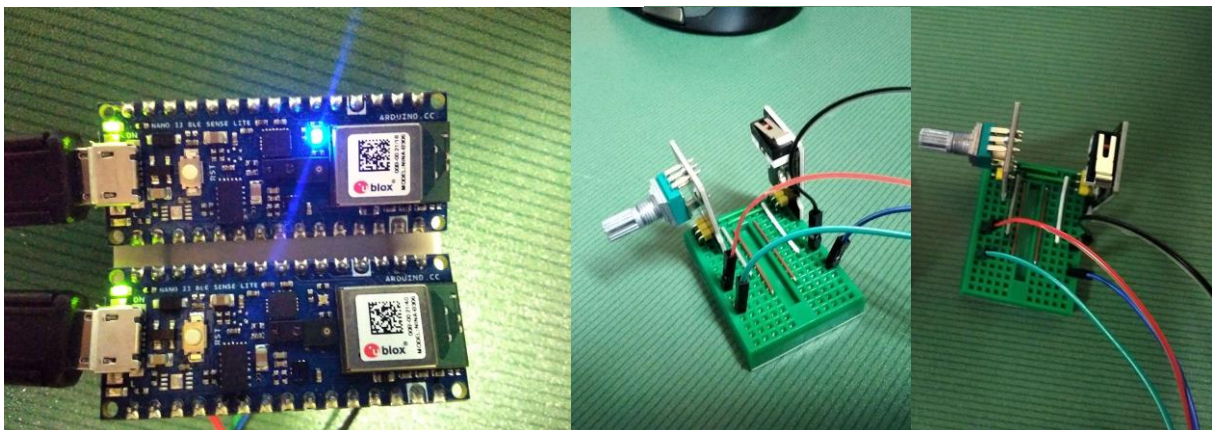
Recording finished, written: test_3.wav, at: 12:44:10.458359
```

28. ábra A konzolos megjelenítés eredménye

2.3. Valós idejű (Real-time, on-line) feldolgozás mikrovezérlőn

Felhasznált eszközök

A mikrovezérlő, amit a mérésre és feldolgozásra fogunk használni egy Arduino Nano 33 BLE Sense, melyen található egy beépített MP34DT05 mindenirányú mikrofon. Ennek segítségével a vezérlő képes hang felvételére, amit eltárol egy bufferben és később ezeket az adatokat dolgozza fel. Egy Keystudio Crash Sensor annak érdekében, hogy a kézzel történő számlálás is egyszerű legyen. Valamint egy Keystudio Analog Rotation Sensor a küszöbérték statikus zajszint értékének kézi állítása érdekében.



29. ábra Arduino eszközök, kézi számláló és egy potenciometer a zajszint manuális állításához

Hagyományos algoritmus

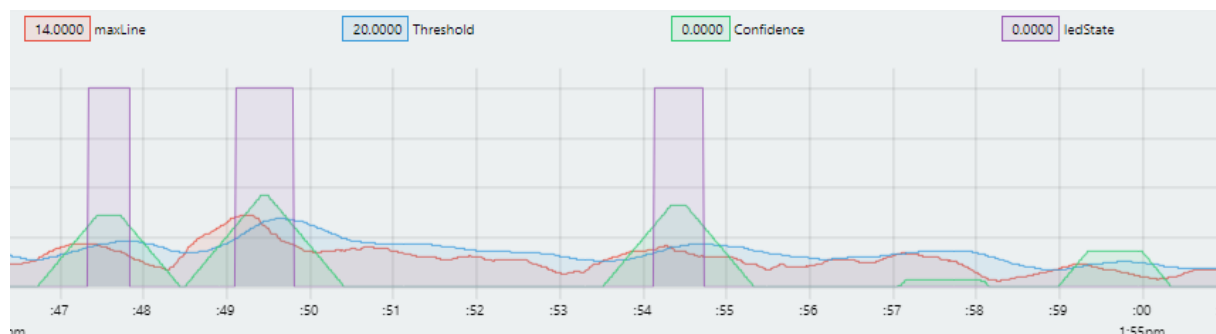
Küszöbérték és Spektrum használata Real-Time

A mikrovezérlős feldolgozás elveiben hasonlít az eddigi utólagos feldolgozáshoz, viszont a nehézséget itt a mikrovezérlő limitált teljesítménye jelentette. A felvétel 16 kHz-es mintavételi frekvenciával történik.

A mérés és a feldolgozás nem párhuzamosan történik, ezért figyelembe kellett venni azt, hogy a feldolgozás minél rövidebb idő alatt megtörténjen. Minél több idő jut a mérésre, annál pontosabb eredményhez vezet. A mérés egyszerre egy mintavételezési tömböt tölt fel, majd ezt dolgozza fel.

Küszöbölés

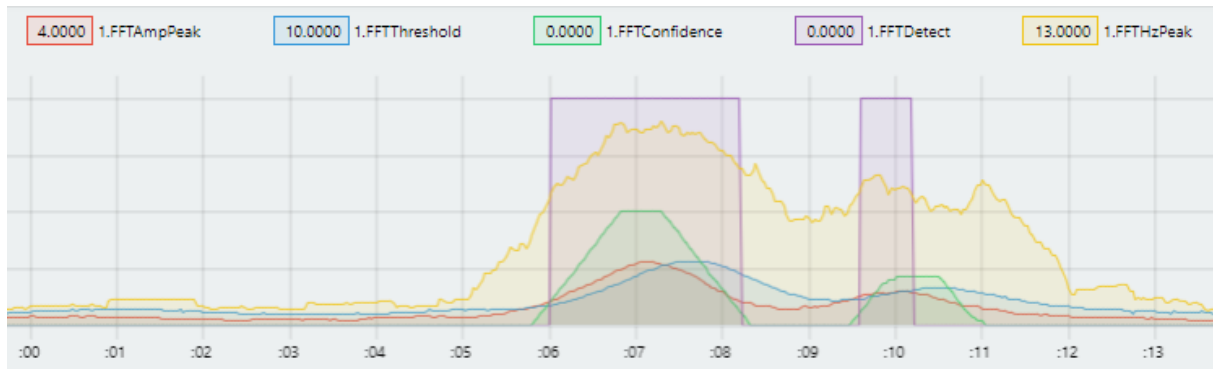
A feldolgozás tömbönként kiveszi az amplitúdó csúcsertékét, csak ezt az adatot lementve dolgozik tovább, a többit eldobja, így csökkentve a további számítások időkölségét a megfelelő mérési pontosság megtartása mellett. Utána ezeknek a csúcsertékeknek veszi az elmúlt egy másodperc ablakméretnyi átlagát. A küszöbértéket ezeknek az adatoknak szintén az átlagából számolja, aminek eredményeképpen a küszöbérték fázisa eltérő lesz a csúcsertékekétől, valamint simítottabb az átlagolás miatt. Ezután ha az elmúlt másodperc minimum 60 százalékában a maximum érték a küszöbérték felett volt, akkor az egy detektálásnak minősült. Ezzel a módszerrel a környezeti zajtól eltérő/kiugró hangjelenségeket számlálja, ami ideális esetben járműveket jelen, azonban nincs teljesen tökéletes mérési körülmény, így ideje volt egy fejlesztésnek.



30. ábra Valós idejű grafikon soros monitorról leolvasva

Küszöbölés és frekvencia

A második módszer teljes egészében az előző annyival kiegészítve, hogy az adott rögzített mintára alkalmaz egy FFT-t és a legjellemzőbb frekvencia értékét elmenti. Ha ez a frekvenciaérték egy (állítható) intervallumon belül van a detektálás időpontjában, akkor tényleges a detektálás, máskülönben figyelmen kívül hagyja.



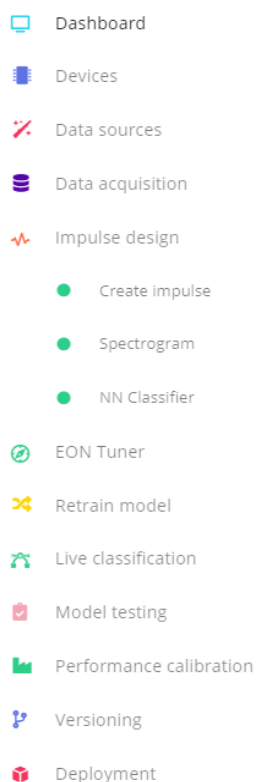
31. ábra Valós idejű grafikon soros monitorról leolvasva, frekvencia értékekkel

Gépi tanulás

Modell átépítése mikrovezérlős rendszerre

Az elmentett modellt át lehet alakítani mikrovezérlős használatra is, viszont így túl nagy lesz a mérete, ezért a neuronokat csökkenteni kell úgy, hogy a modell pontossága és hibája ne sokat romoljon. Ez a neuronok negyedére csökkentésével érhető el minden rétegben. A TensorFlow Lite nevű Python modul segítségével .tflite kiterjesztésűvé lehet alakítani a csökkentett neuronokkal rendelkező modellt, aminek méretét ez az átalakítás még tovább csökkenti. Ezt a konvertált modellt így még nem lehet használni, át kell alakítani egy C fejléccé (C header), ami a modellt tartalmazza hexadecimális formában egy model nevű tömbben. Ehhez a Google Colab nevű fejlesztői környezete volt segítségemre. Ez egy Linuxos környezet, böngészőben futtatható Python Notebook szerkesztő, minden kód fordítása és futtatása a Google szerverein történik, nagyban felgyorsítva a futási időket és megkönnyítve a gépi tanulás folyamatát. Itt inkább a Linux környezetre volt szükség, hogy a TensorFlow Lite modellt át lehessen alakítani egy C állományba, amit a mikrovezérlő is felismer és fel tud dolgozni. Az elkészült C állományt az Arduino kódot tartalmazó mappába kell másolni, majd a kódban is el kell helyezni az elérési útvonalát, különben nem lesz használható.

A program elején be kell állítani pár konstans makrót, amit a program nem tud megváltoztatni futásidőben. Ezek írják le a mikrofon érzékenységet, a felvételi puffer méretét, a minták méretét, a mintavételi időközöket, a címkék számát és nevét, a jósolt eredményben való biztonság küszöbét, a bemenetek és kimenetek méretét, valamint a tenzor aréna méretét (ez a maximális méret amire szükség lehet bármely időpillanatban a tenzorok eltárolásához). A probléma megoldásához használható az Eloquent TinyML^[17] Arduino könyvtár, amely egyszerűsíti a TensorFlow Lite használatát. A program folyamatosan fut, felvesz egy rövid hanganyagot, amit eltárol a pufferben. Ezt a hanganyagot elemzi, majd ad egy valószínűségi/magabiztossági értéket minden címkére 0 és 1 között. A felvétel/predikció után vár 1 másodpercet majd újból felvesz. Az eredményeket soros porton jeleníti meg, a program futásához szükséges idővel együtt.



32. ábra Az EdgeImpulse oldal oldalsó menüje

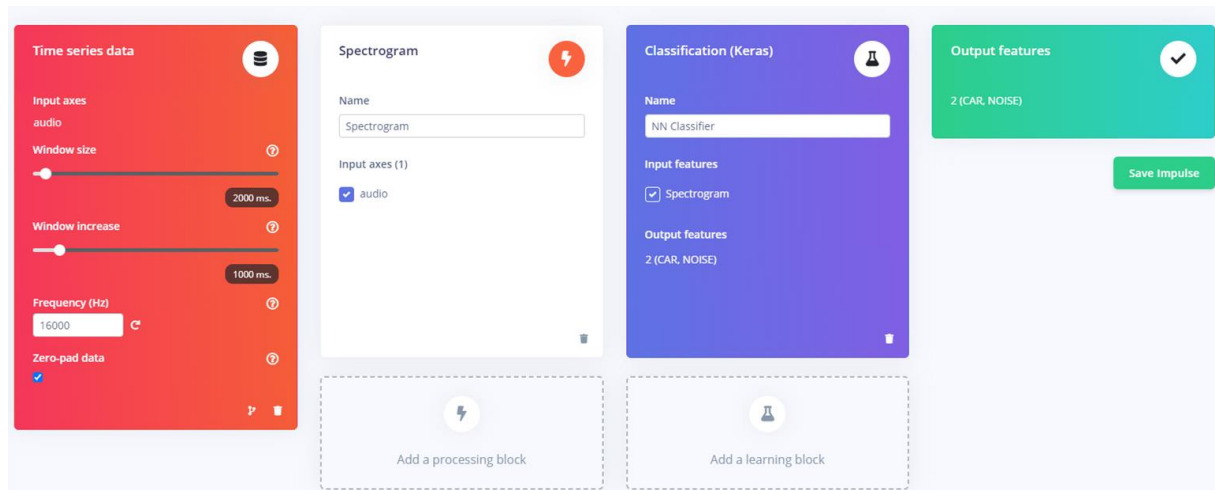
Egy másik lehetséges megoldás az EdgeImpulse használata.^[17] Ez egy webes felület, amely megkönnyíti a gépi tanulást és a modellek elkészítését, kifejezetten mikrovezérlőkhöz tervezve. Az oldal használatához felhasználói fiók szükséges. A Data acquisition menü fejlécében lévő Upload data ponton belül megtalálható az adatok feltöltéséhez szükséges oldal. Az oldal alapértelmezetten a fájlok neveiből készíti el a megfelelő címkéket, ez nem megfelelő az általunk összegyűjtött fájlok neveihez. Az Upload into category pontnál az alapértelmezett beállítás az adatok tanítási és tesztelési csoportokba való osztása, ez 80% tanítási, 20% tesztelési elosztást jelent, ez megfelelő. A Label pontnál be kellett állítani a manuális címke megadást, így a hangfájlokat név szerint lehet feltölteni, nem fog a nevekből címkét készíteni. Az adatok és a címke megadása után a Begin upload gombra kattintáskor elkezdődik a címkézett adatok feltöltése. Néhány fájlt lehet nem tud feltölteni, de van elég adat ahhoz, hogy ez ne legyen probléma.

Az Impulse design pontban lehet az “impulzust”, a programot összeállítani, a fő feldolgozó komponenseit megadni. Itt lehet

beállítani:

- a feldolgozandó adatok hosszát, a csúszó ablakot, ha az audió meghaladja a megadott hosszúságot, a 0-val való kitöltést, ha az audió nem éri el a megadott hosszúságot, a frekvenciát, amelyen az adatok mintavételezése történik

- a feldolgozó blokkokat
- a tanítást/tanulást végző blokkokat



33. ábra Az impulzus elkészítése

Az impulzus elmentése után a feldolgozó és tanító blokkokat lehet konfigurálni. A tanító blokkon belül be lehet állítani:

- a tanítási ciklusokat
- a tanulás sebességét
- a validációs adathalmaz méretét
- az adatok növelését zaj hozzáadásával, idő- és frekvenciasávok maszkolásával valamint az idő tengely “elhajlításával”
- előre beállított neurális hálózati architektúrákat vagy saját, egyedi hálózatot

A feltanítás az EdgeImpulse szerverein történik, ezáltal felgyorsítva a tanulást és erőforrásokat spórolva a felhasználónak. A tanulás végén ad egy mátrixot a pontosságról és egy grafikont az adatok szórásáról.

Neural Network settings

Training settings

Number of training cycles

Learning rate

Validation set size %

Auto-balance dataset ☐

Audio training options

Data augmentation ☒

Add noise

Mask time bands

Mask frequency bands

Warp time axis ☐

Neural network architecture

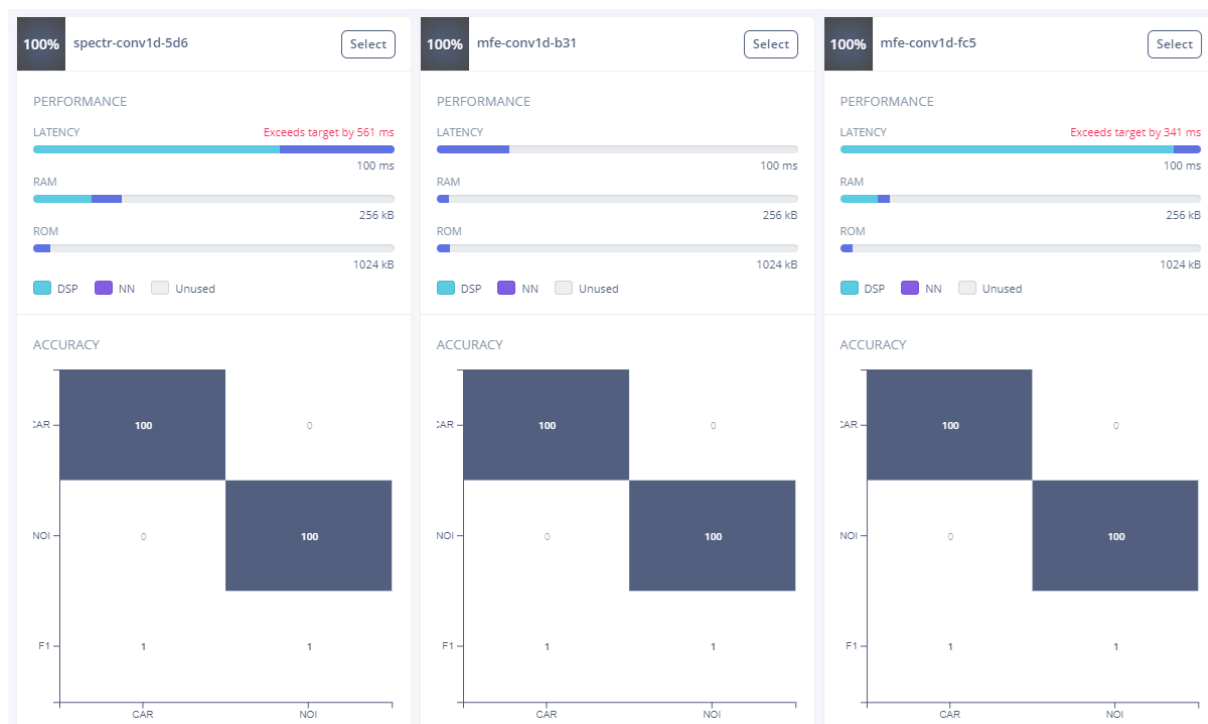
Architecture presets

34. ábra A feltanítási menüpont



35. ábra A tanítás utáni hiba mátrix és az adatok szórása

Az EON Tuner pontban lehet a már feltanított modellt tovább pontosítani vagy a legoptimálisabb modellt megtalálni, ami sebességben, pontosságban és tárhely használatban is megfelelő. Megmutatja a használt beállításokat, a legjobbnak talált modell kiválasztása után frissíti a modell paramétereit, amelyet újra kell tanítani a Retrain model pontban. Erre az adathalmazra viszonylag gyorsan lefut és megtalálja a legjobb beállításokat, de ha több ezer fájl és több címke van, akár órákig is eltarthat ez a folyamat.

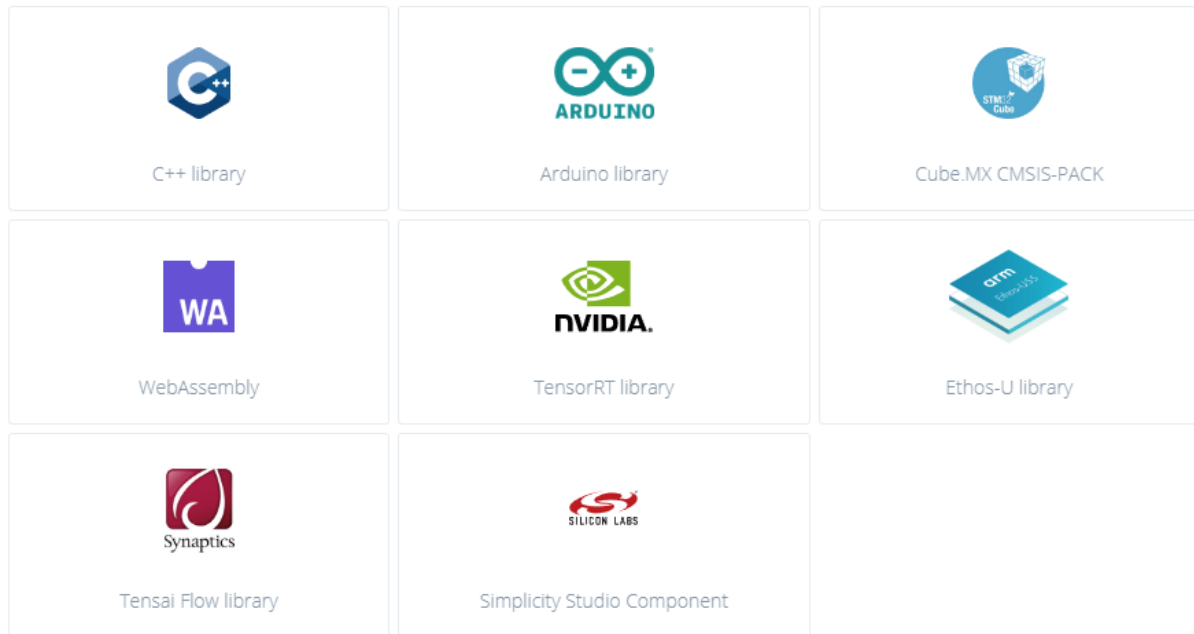


36. ábra Néhány opció az EON Tuner menüpontból

A Deployment pontban az elkészített modellből lehet különböző könyvtárakat létrehozni, különböző mikrovezérlőkhöz firmware-t készíteni vagy az impulzust közvetlenül futtatni okostelefonon vagy laptopon/asztali számítógépen.

Create library

Turn your impulse into optimized source code that you can run on any device.



37. ábra A Deployment menüpont néhány lehetősége

A firmware-t fel kell tölteni a mikrovezérlőre, jelen esetben a “flash_windows.bat” fájl segítségével. Ahhoz, hogy a firmware-t lehessen használni, kell még néhány kiegészítő program:

- Python 3
- Node.js v14, Windowson a telepítés közben be kell jelölni a “Tools for Native Modules” pontot

A telepítés hosszabb ideig tarthat a telepítési adathordozótól függően. Miután a telepítés befejeződött, az “npm install -g edge-impulse-cli --force” parancs segítségével lehet az EdgeImpulse konzolos felületét telepíteni.^[18] Ezen a felületen az “edge-impulse-run-impulse” paranccsal lehet a modellt futtatni, amely a konzolon jeleníti meg az egyes címkékre adott magabiztosságot.

```

Inferencing settings:
  Interval: 0.06 ms.
  Frame size: 32000
  Sample length: 2000 ms.
  No. of classes: 2
Starting inferencing, press 'b' to break
Starting inferencing in 2 seconds...
Recording...
Recording done
Predictions (DSP: 52 ms., Classification: 17 ms., Anomaly: 0 ms.):
  CAR: 0.00000
  NOISE: 0.99609
Starting inferencing in 2 seconds...
Recording...
Recording done
Predictions (DSP: 52 ms., Classification: 17 ms., Anomaly: 0 ms.):
  CAR: 0.01953
  NOISE: 0.98047

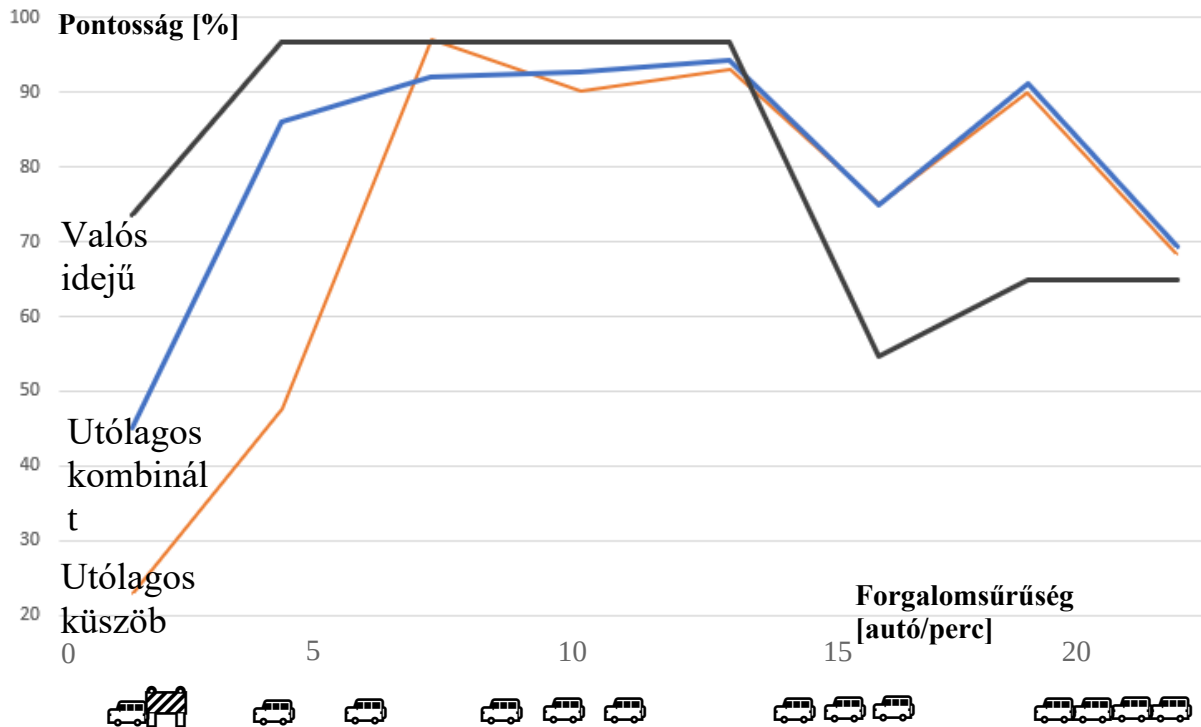
```

38. ábra A modell konzolos futtatása

3. Összefoglalás

Egyáltalán nem gondoltuk volna, hogy lehet forgalmat mérni ilyen “olcsó” eszközök segítségével, viszont a várt eredményekhez képest meglepően pontos méréseket sikerült kivitelezni. Az audio alapú mérésnek is megvannak a hátrányai, mivel nem lehet kiszámítani, hogy éppen milyen forgalmi/időjárási körülményekben fog a mérés történni, melyek pontatlanságot idéznek elő. Egy mérést szinte lehetetlen feladat felkészíteni minden körülményre, de kutatásunk és fejlesztésük itt még nem ér véget, mivel ez a témakör még eléggé kiaknázatlan a forgalmi mérés szempontjából. Megteszünk minden tőlünk telhetőt, hogy tovább haladjunk ezzel a projekttel és elérjünk pontosabb mérési eredményeket a többféle és olykor szélsőséges zavaró tényezők ellenére is.

3.1. Eredmények és forgalmi körülmények



39. ábra Feldolgozási és mérési eljárások pontosságának összehasonlítása

A legpontosabb mérést a valós idejű arduino mikrovezérlővel sikerült elérnünk, ami 96,7% volt, viszont ezt csak zajmentes és ideális forgalmi körülmények között sikerült elérni. A legalacsonyabb forgalom egy útépitésnél volt, ahol tisztán látszik az algoritmusok közti különbség zajszűrés tekintetében.

Mérési helyszínek voltak:

Dr. Szobonya Zoltán utca

Izsáki út

Sport utca

Csabay Géza körút

3.2. Kérdések, Fejlesztési lehetőségek

Irány és sebesség mérése - A megvalósítása ennek a legegyszerűbb módon kettő mikrofon segítségével lenne lehetséges, melyeket távolabb helyezünk el egymástól ugyanazon az útszakaszon. A 2 mikrofon segítségével a zajok kiszűrése is sokkal hatékonyabban tudna történni, ugyanis feltételezzük, hogy egy jármű mindkét mikrofon előtt elhalad, amit más zajforrások nem feltétlenül tesznek meg.

Analóg rezonáns körök alkalmazása a jellemző frekvenciatartományokra, valamint egyéb jelfeldolgozásban bevált módszerek használata.

Csupán paraméter állítással más közlekedési járművek detektálása és csoportosítása, például reptereken repülő, kikötőkben hajók.

Az autók műszaki hibáinak kijelzése/megjósítása hang segítségével.

Gépi tanulás

A program jelenleg folyamatosan felvesz, értelmez és felülír, ami nem optimális az adathordozók élettartamának szempontjából. Ezt át lehet alakítani úgy, hogy ne készüljön felvétel, hanem az adatokat gyűjtse egy pufferbe és ezeket az adatokat csak akkor gyűjtse, ha a hang erőssége átlép egy küszöbértéket. Ezután ebből a pufferből kiolvassa az adatokat értelmezzen és adjon predikciót a program.

3.3. Felhasznált Szakirodalom

[1] - Tettamanti Tamás - Varga István - Csikós Alfréd. Közúti mérések. Eszközök és módszerek a közúti járműforgalom megfigyelésére. Typotex, Budapest, 2016.

[2] - [Acoustic Road Monitoring](#)

[3] - <https://www.hackster.io/463990/traffic-monitor-iot-ai-74c2a9>

1.ábra - <https://www.vezess.hu/magazin/2010/12/14/miert-nem-kapunk-zoldet/>

[4] - [induktív hurok érzékelő](#)

2.Ábra - [Közúti forgalomtechnikai paraméterek mérése és becslése](#)

3.Ábra - [Közúti forgalomtechnikai paraméterek mérése és becslése](#)

4.Ábra - [Közúti forgalomtechnikai paraméterek mérése és becslése](#)

5.Ábra - <https://www.google.com/maps>

6.Ábra - <https://www.physicsclassroom.com/class/sound/u11l1c.cfm>

[5] - <https://hu.wikipedia.org/wiki/Hang>

[6] - <https://hu.wikipedia.org/wiki/Fourier-transzform%C3%A1ci%C3%B3>

7.Ábra - <https://mogi.bme.hu/TAMOP/mereselmelet/ch11.html>

[7] - [Not So Fast! Building an ML-Powered Speed Trap](#)

[8] - https://www.tensorflow.org/tutorials/audio/simple_audio

[9] - https://en.wikipedia.org/wiki/Short-time_Fourier_transform

[10] - https://en.wikipedia.org/wiki/Fast_Fourier_transform

[11] - https://en.wikipedia.org/wiki/Mel-frequency_cepstrum

[12] - <https://www.youtube.com/playlist?list=PL-wATfeyAMNqIee7cH3q1bh4QJFAaeNv0>

[13] - <https://www.youtube.com/playlist?list=PL-wATfeyAMNpEyENTc-tVH5tfLGKtSWPp>

[14] - https://www.youtube.com/playlist?list=PL-wATfeyAMNpCRQkKgtOZU_ykXc63oyzp

[15] - https://en.wikipedia.org/wiki/Mel_scale

[16] - <https://en.wikipedia.org/wiki/Cepstrum>

[17] -

<https://create.arduino.cc/projecthub/alankrantas/eloquenttinymml-easier-voice-classifier-on-nano-33-ble-sense-ebb81e>

[18] - <https://docs.edgeimpulse.com/docs/edge-impulse-cli/cli-installation>

[19] - <https://realpython.com/playing-and-recording-sound-python/#recording-audio>

[20] - <https://www.geeksforgeeks.org/multithreading-in-python-set-2-synchronization/>