

# **TDK Dolgozat**

Készítette:  
Koszteczky Bence

*Koszteczky Bence*

# **MÁGNESES ÉRZÉKELÉSEN ALAPULÓ JÁRMŰDETEKCIÓ**

Témavezető: Dr. Simon Gyula

Műszaki Informatikai Kar, Rendszer- és Számítástudományi tanszék

**Pannon Egyetem**

**2014**



|       |                                         |    |
|-------|-----------------------------------------|----|
| 1     | Bevezetés .....                         | 5  |
| 1.1   | Motiváció .....                         | 5  |
| 1.2   | Járműdetekció .....                     | 6  |
| 1.3   | Mágneses érzékelők .....                | 9  |
| 2     | Felhasznált eszközök.....               | 11 |
| 2.1   | MicaZ - TinyOS .....                    | 11 |
| 2.2   | HMC1002 - Mag3110 .....                 | 13 |
| 2.3   | Matlab .....                            | 14 |
| 3     | Fejlesztési folyamat.....               | 15 |
| 3.1   | Megvalósítás HMC1002 szenzorral.....    | 15 |
| 3.1.1 | Offset konfigurációs algoritmusok ..... | 15 |
| 3.1.2 | Eseményazonosítás .....                 | 18 |
| 3.1.3 | Sebességbecslés .....                   | 19 |
| 3.1.4 | Mérési összeállítás .....               | 20 |
| 3.2   | Megvalósítás Mag3110 szenzorral.....    | 22 |
| 3.2.1 | Digitális átállás.....                  | 22 |
| 4     | Szoftver dokumentáció.....              | 23 |
| 4.1   | Implementáció.....                      | 23 |
| 4.1.1 | A rendszer felépítése.....              | 23 |
| 4.1.2 | AppP.nc .....                           | 23 |
| 4.1.3 | BinarySearch .....                      | 24 |
| 4.1.4 | SlowCompensation .....                  | 26 |
| 4.1.5 | findVehicle .....                       | 26 |
| 4.1.6 | setConfig .....                         | 28 |
| 4.1.7 | sendMessage.....                        | 29 |
| 4.1.8 | sendMessageX.....                       | 30 |
| 4.1.9 | initArray.....                          | 31 |

|        |                                        |    |
|--------|----------------------------------------|----|
| 4.1.10 | movingCalc .....                       | 31 |
| 4.1.11 | Boot.booted .....                      | 32 |
| 4.1.12 | RadioControl.startDone .....           | 32 |
| 4.2    | Alkalmazás használata .....            | 33 |
| 4.3    | A sebességbecslő implementálása.....   | 34 |
| 5      | Teszt eredmények.....                  | 36 |
| 5.1    | Összehasonlítás más rendszerekkel..... | 41 |
| 6      | Összefoglalás .....                    | 43 |
| 6.1    | Konklúzió .....                        | 43 |
|        | Irodalomjegyzék.....                   | 45 |

# 1 Bevezetés

## 1.1 Motiváció

Napjainkban komoly gondot jelent erdeink állományának megóvása. A folyamatosan terjeszkedő mezőgazdasági területek és fakitermelések, tarvágások nagymértékben csökkentik hazánk erdeinek méretét. Ezekhez a negatív hatásokhoz jön hozzá a fakitermelésekkel egyidejűleg jelentkező falopások száma, amely nagyon nehezen követhető nyomon, hiszen a kitermelés során a szállításra váró farakások a munkaidő végeztével és a munkaszüneti napokon elhagyatottan várnak arra, hogy az erdészet elszállítsa.

A fakitermelések többnyire olyan helyen zajlanak, amit magánúton lehet megközelíteni, de ezeknek a használata közforgalom előtt nem elzárt, így azon bárki közlekedhet. A gyakorlati példa azt mutatja, hogy a tolvajok célzottan, akár teherautókkal is útnak indulva fosztják ki a már korábban kitermelt és szállításra váró alapanyagot. Az erdészetek a személyautóval történő falopást nem is nehezményezik, annak csekély kárértéke miatt.

A feladat egy olyan biztonsági rendszer készítése, amely nehezebb telepítési munkálatok nélkül, könnyen kihelyezhető a fakitermelési területek határába. Az alkalmazásnak hosszú rendelkezésre-állási idővel kell rendelkeznie. A folyamatos rendelkezésre állás mellett a rendszernek alkalmasnak kell lennie járművek detektálására szélsőséges körülmények között is. Álcázhatósága pedig az illegális behatolók által okozott károk elkerülése miatt fontos, a rendszer épségét megőrizendő. Az elkészült rendszernek fontos a robosztussága, hiszen a kitermelés egy adott helyen akár hónapokig is tarthat, ez alatt az idő alatt folyamatosan készenlétben kell állnia, hogy a központ felé továbbíthassa a szenzorhálózat az esetleges behatolásokat. Figyelni kell az időjárás viszonyaira is: az erdei utak folyamatos „mozgásban” vannak, hiszen egy jól felázott talajon a keresztülhaladó megrakott több tonnás teherautó méteres árkot képes sülyeszteni a talajba, amelynek nyomán a kisebb terepjárók már nem képesek közlekedni ezért új utat járnak ki maguknak. Ezzel a folyamatos változással számolva szükséges, hogy a rendszer több méteres távolságból is képes legyen megbízható eredménnyel detektálni az elhaladó járműveket.

Megvizsgálva a jelenlegi lehetőségeket arra a megállapításra jutottunk, hogy jelenleg az adott feladat ellátására nem létezik gyakorlati megvalósítás, így egy hiánypótló rendszer kerülhet kifejlesztésre. Tehát egy olyan rendszer megvalósításába kezdtünk, amely képes több méterről járműveket detektálni, akár viszonytárgyos időjárási körülmények között is és jól elrejthetőnek kell lennie, hogy a véletlen vagy szándékos rongálástól meg tudjuk óvni. A dolgozatban az általam kifejlesztett és megvalósított jelfeldolgozó és detekciós algoritmusokat mutatom be a többlépcsős iteratív fejlesztési folyamat állomásain keresztül. Bemutatom a fejlesztési folyamat során választott irányokat (zsákutcákat és végleges fejlesztési irányokat is beleértve). Végül bemutatom az elkészült rendszert és annak laboratóriumi és éles körülmények között végzett tesztelése során elért eredményeimet.

**Köszönetnyilvánítás:** munkámat a GOP-1.1.1-11-2011-0070 projekt támogatásával végeztem.

## 1.2 Járműdetekció

A mai világban a járművek detektálására több területen is szükség van, a forgalomban résztvevő személygépjárművek számának jelentős megnövekedése miatt. Autókból az 1960-as években még csak 30.000 volt, de 1980-ra több mint 1 millió jármű volt az utakon, amely 2005-ben már 2.5 millióra nötte ki magát [1].

A detekció a forgalom számlálása mellett a forgalom szabályozásában játszik nagy szerepet napjainkban [2]. Legfőbb célja, hogy a jelentős forgalmi dugók elkerülhetőek legyenek. Ezzel a céllal több megoldás is létezik, amik napi használatban vannak és folyamatosan segítik a közlekedők akadálymentes és optimális célba jutását. A fentiek mellett fontos megemlíteni a biztonsági feladatot ellátó járműdetektáló alkalmazásokat. Több olyan megvalósítás létezik már, amelyek a teherautók figyelésével foglalkoznak, hogy megállapítsák a rakodótér foglaltságát. Jellemzően ezek az alkalmazások mind csak a közeli járművek detektálására és észlelésére szolgálnak, leginkább egy adott közeli területen (1 méteres távolság) jelen lévő személy- vagy teherjárműveket kell észlelni. Számunkra egy nagyobb területen átmenő jármű forgalmat fontos megfigyelni, az elvárt távolság legalább 2-3 méteres lenne.

A járművek detektálását telepítés szempontjából két külön csoportra lehet bontani. Az egyik csoport azokat a módszereket tartalmazza, amelyek a burkolat megsértése nélkül kihelyezhetőek, egyszerűen villanyoszlopokra, hidakra, út mellé. A másik csoport pedig olyan eszközöket tartalmaz, amelyeket a burkolat alá építve alkalmaznak. Ezeknek a rendszereknek a telepítéséhez a kész útburkolatot kell felmarni és megfelelő méretű rést készíteni a szenzor vagy annak bizonyos részeinek behelyezése érdekében.

Azokat a szenzorokat, amelyekhez nem szükséges az út felületének megsértése könnyebb és gyorsabb telepíteni, továbbá nincs szükség a forgalom jelentős korlátozására mindamellett, hogy kevesebb veszélyt rejt magában ezzel a módszerrel a kihelyezés folyamata. Ezek közé a szenzorok közé tartozik jelenleg például a videóval történő forgalomfigyelés, az ultrahangos radarok, a lézer radarok és a passzív infravörös Doppler érzékelők [3].

A képfeldolgozó eljárások az egyik legjobb és legpontosabb megoldásokat adják a járműdetekcióban, de sajnos anyagi vonzatuk igen magas, így nem ajánlott minden helyre azokat kihelyezni. Képesek akár több sáv egyidejű megfigyelésére is és későbbiekben elég a szoftveres javítás ahhoz, hogy megváltoztassuk a mérés helyét és célját (szoftveres detekciós zónák átállításával). A megfigyeléshez hozzá tartozik, hogy későbbiekben felhasználható logok születnek a mérésről, amelyeket máshol is belehet mutatni, ha esetleg biztonsági eljárásokat kell érvénybe léptetni. A rendszer hátrányát az időjárás viszonyosságai jelentik, gyakran kerülhet szennyeződés a kamera lencséjére, ami csökkenti az értékes információ tartalommal rendelkező kép tartalmát, hiszen a feldolgozáshoz szükséges hasznos adat előfordul, hogy nem elegendő a detektáláshoz. További nehézségeket okozhat az éjszakák-nappalok váltakozása, amit a rendszernek kezelnie kell. Előfordulhat, hogy gyenge a kontraszt, így nem vehetőek jól ki az elhaladó járművek körvonalai. A kamerákat a jó látószög és belátható terület érdekében több méterrel az út fölé kell helyezni, és minden esetben figyelni kell arra, hogy a nagy széllökések azt ne mozdítsák ki a helyéről.

A mikrohullámú radarok előnye mindamellett, hogy teljesen érzéketlenek a zord időjárásra az, hogy képesek az elhaladó járművek sebességét is detektálni. Lehetőség

van az egyszerre történő több sáv megfigyelésére is. A szenzor hátránya, hogy az álló autókat nem képes érzékelni a doppler effektus használata miatt.

Az ultrahang alapú radar szenzorok legnagyobb előnye a több sávós megfigyelés. Alkalmazásában nehézségeket a hőmérsékletváltozás és a komolyabb széllekedések tudnak okozni. Több olyan eszköz is létezik már, amelyek a hőmérséklet változásához képesek alkalmazkodni.

Az infravörös szenzorok olyan eszközök, amelyek több mérést használnak ahhoz, hogy megállapítsák a járművek sebességét, helyét és osztályozni tudják a mérete alapján. Lehetőség nyílik passzív szenzorok használatára is, amelyek kevesebb információval szolgálnak, mint az aktív párjuk, de több helyen elhelyezve képesek az elhaladó autó sebességét meghatározni. A hátrányok közé sorolható, hogy a nagy ködben és havazásban a szenzorok érzékelése nagyban csökken. A passzív szenzorok számára a köd mellett az eső is komoly problémát tud okozni.

Útfelületi munkát igénylő alkalmazások közé tartozik az induktív hurok, a jelenlegi mágneses érzékelők, a pneumatikus csövek és a piezoelektromos kábelek [3]. Ezek közül leggyakrabban az induktív hurkokkal találkozhatunk Magyarországon. A közös ezekben az alkalmazásokban, hogy mindegyik érzékelő az útburkolat alatt vagy annak szintjén helyezkedik el és az elhelyezéshez egy kis lyukban, vagy vékony hurokban az aszfalt felszedését igényli. A feldolgozó egységeket általában az út szélén kapnak helyet és odáig vezetik ki az eszközök jeleit.

Az induktív hurok rugalmasságának köszönhetően sok területen használható, mindemellett könnyen megérthető technológián alapszik. Az induktív hurkot egy rezgőkörben használják; amikor egy jármű áthalad vagy megáll a szenzor felett, annak induktivitása lecsökken, így a rezgőkör frekvenciája megnő, amit a feldolgozó egység érzékel. Az eszköz alapvetően jelenlétet tud detektálni, de nagyobb mintavételezési sebesség mellett akár járműosztályozásra is alkalmas lehet, amelynek során a változások hosszát vizsgálják. Az eszköz hátrányai közé tartozik, hogy a burkolat helyenkénti felszedése szükséges a telepítéséhez, ami egyben rongálja az út élettartamát is. További kellemetlenséget okoz az utak szükségszerű lezárása a telepítés és karbantartás időtartamára.

A pneumatikus csövek működésüket tekintve olyan szerkezetek, amelyek a jármű áthaladását úgy jelzik, hogy a beépített csöveken keresztül magasnyomású levegő áramlik a mérőműszer felé az elhaladás következtében. A légnyomás okozta változás egy légkapcsolót lezár, ami elektromos jelet generál és ezt továbbítja a feldolgozó egységnek. A pneumatikus alkalmazások nagy előnye, hogy olcsón előállíthatóak és kihelyezhetők, továbbá a telepítés gyors. A hátrányai többek között az időjárás érzékenysége, és a könnyebben sérülő csövek.

A piezoelektromos eszközök alapvető tulajdonsága, hogy olyan anyagokból épülnek fel, amelyek képesek kinetikus energiából elektromos energiát előállítani. Felépítésük tekintetében egy fém alsó lapra kerül a piezoelektromos tulajdonsággal rendelkező anyag, amire egy külső fém réteget helyeznek. Az így készült eszköz összenyomásának hatására a két fém réteg töltötté válik, így köztük elektromos feszültség jön létre. Járműdetektálás során ez a feszültség arányos a rajta áthaladó jármű méretével. Mivel a szenzorok a rajtuk áthaladó súlyra érzékenyek, így járműosztályozásra is kiválóan alkalmas ez az eljárás. Hátránya lehet, hogy hőmérséklet-érzékeny, hiszen az anyagok tulajdonságai megváltoznak a hőmérséklettel együtt, így esetenként más tulajdonságú jelek keletkeznek.

Természetesen léteznek olyan alkalmazások, ahol a fentiekben megnevezett technológiákat egymással kombinálva használják a pontosabb mérések érdekében, de

ezek növelik az egyes alkalmazások költségeit, mindemellett több információt lehet megtudni az egyes forgalmi helyzetekről. Az ilyen kombinált rendszerek segítségével a forgalmi dugók, vagy a sorban állók mérete is megállapítható.

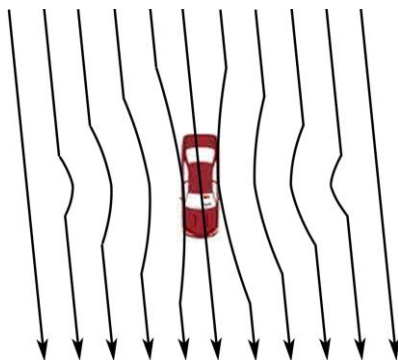
A szenzorok vizsgálata költség oldalról is lehetséges, de itt komoly megszorításokat tud okozni a felhasználás célja. Ha csak járműveket szeretnénk detektálni, gyorsan és olcsón felszerelhető infravörös vagy ultrahangos megoldásokat tudunk igénybe venni, de amint több funkcionalításra van szükségünk, célszerűbb megoldást tud nyújtani egy drágább megoldás, mint például a képfeldolgozó megoldások. Jelentősen megnöveli az árát a képfeldolgozó eszközök beszerzése és kiszerezése, de gyakran olyan többlet információ nyerhető ki az ehhez hasonló drágább megoldásokból, amelyeket további fejlesztésekre is fel lehet használni.

Fontos szempont a rendszer álcázhatósága: olyan szenzorokra van szükség, amelyek jól elrejtethők, ezáltal a rongálás kockázata minimálisra csökkenthető. Ez például egy kamerás rendszer esetén nagyon nehezen teljesíthető követelmény.

A megvizsgált lehetőségek tükrében arra a megállapításra jutottunk, hogy olyan megvalósítás jelenleg nincs a piacon, amely a civilizációtól távolabb eső területek alacsony áramfogyasztási követelményéhez illeszkedve a kívánt detekciós távolságot tudná biztosítani, ráadásul jól elrejtethető eszközökkel. Fejlesztésünk ezért ilyen eszközök létrehozását tűzte ki célul. Munkám a projekten belül egy mágneses elven működő szenzoralkalmazás tervezése és elkészítése volt, ezen fejlesztés eredményeit tartalmazza a dolgozat.

### 1.3 Mágneses érzékelők

A mágneses erő a mozgó elektromágneses töltésű objektumok között ható vonzó vagy taszító erő. Egymáshoz képest mozdulatlan töltések között csak elektromos erő hat, míg egymáshoz képest mozgó töltések között elektromos és mágneses erő is hat. A mágneses tér változó elektromos tér vagy elektromos áram környezetében olyan tartomány, amelyben mágneses erő figyelhető meg [4]. A mágneses térnek jellemzője a fluxussűrűség, amelynek mértékegysége a *Tesla*, de gyakran használják a *Gauss*-t is ( $1 \text{ Tesla} = 10\,000 \text{ Gauss}$ ). A *Gauss* az  $1 \text{ cm}^2$  áthaladó mágneses erővonalak számát jelenti. A Föld felszínén a mágneses mező  $0.5 \text{ Gauss}$  körül alakul.



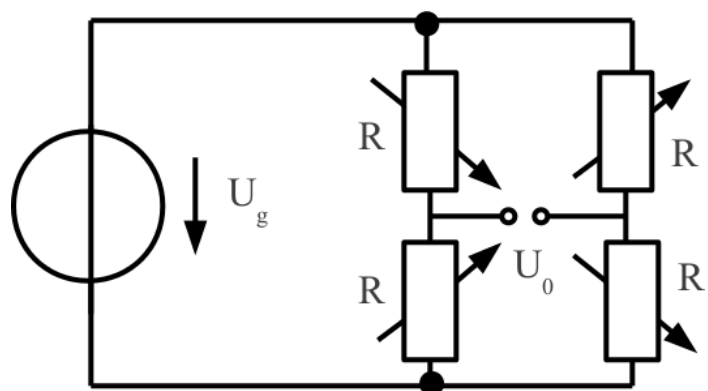
1. ábra: A Föld mágneses mezejét  
megzavaró autó

Az általunk használt mágneses szenzorok a föld állandó mágneses mezőjét érzékelik. Ebben a mágneses mezőben okoznak a ferromágneses anyagok eltérő nagyságrendű változásokat azoknak mérete és iránya alapján. Ezt szemlélteti az 1. ábra, amin azt láthatjuk, hogy egy jármű milyen módon torzítja a Föld mágneses mezőjét.

Mágneses érzékelőkből léteznek Hall- és Reed elven működő, valamint magnetorezisztív szenzorok. A Hall szenzoroknál a Hall-effektus kerül kihasználásra, amely szerint ha a vezetőn áram folyik keresztül és azt mágneses térbe helyezzük, akkor az áramot hordozó részecskékre Lorentz-erő hat, amiből az következik, hogy a vezető két oldalán potenciálkülönbség jön létre. Ez a Hall-feszültség.

A Reed-relé egy inert gázzal töltött üvegcsébe helyezett eszköz, ami védi a szennyeződéstől és a korróziótól a szenzort. Érintkezőket helyeznek el benne, amelyek ferromágneses anyagból készülnek, továbbá az érintkezés helyén arannyal vonják be őket. Mágneses tér hatására az érintkező nyelvek átmágneseződnek és így egymáshoz kapcsolódnak.

Magnetorezisztív elven működő mágneses érzékelők olyan fémekből készülnek, amelyeknek ellenállása a mágneses tér hatására megváltozik. Ezekből az anyagokból általában egy Wheatstone-híd kerül megvalósításra, amelynek felépítése a 2. ábrán látható. Az általunk használt szenzorokban a magnetorezisztív érzékelőt nikkel-vas ötvözetből készült Wheatstone-híd valósítja meg. A híd elemeinek ellenállás változása a kimeneti jel feszültségében okoz változásokat.



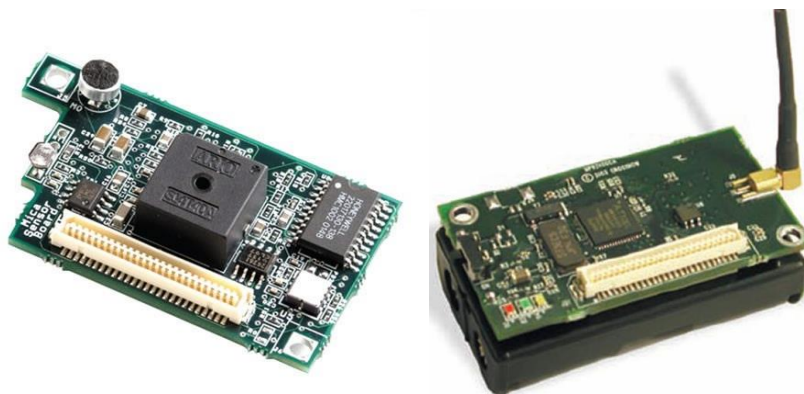
2. ábra: Whitstone-híd magnetorezisztív anyagokból felépítve

## 2 Felhasznált eszközök

### 2.1 MicaZ - TinyOS

A MicaZ (lásd a 3. ábrát) szenzorok a Crossbow által fejlesztett alacsony fogyasztású eszközök (későbbiekben: mote), amelyek 2.4 GHz-es rádióval vannak felszerelve az adatok vezeték nélküli továbbításához. További bővítési lehetőséget biztosító szenzorbordok illeszthetők a rendszerhez, amik fény-, hőmérséklet-, nyomás-, gyorsulás-, mágneses-, akusztikus szenzorral vannak ellátva. Ezeket a mote-okat több eszközt tartalmazó szenzorhálózatok kiépítéséhez tervezték. Ezt mutatja az is, hogy léteznek előre definiált vezeték nélküli kommunikációs sémák, mint például a „dissemination”, ahol minden node-hoz (node jelentése: a szenzorhálózat 1 eleme) eljut az adott információ, vagy a „collection”, ahol egy kiválasztott node-hoz jut el bármilyen úton a kért adat [5].

A TinyOS alacsony fogyasztású szenzorhálózatokra tervezett beágyazott operációs rendszer, programozási nyelve a NesC (ami a Network Embedded System C rövidítése), ami technikailag egy eseményvezérelt, komponens alapú programozási nyelv. Az operációs rendszer Java és shell script front-endet is biztosít. A TinyOS olyan mikrokontroller alapú eszközökre tervezett operációs rendszer, amely eszközök képesek a vezeték nélküli hálózati kommunikációra és használatukhoz széleskörű szenzoros támogatással rendelkeznek. Összevetve más hardver közeli rendszerrel – például az Arduinoval – elmondható, hogy az Arduinora való C nyelvű fejlesztés egyszerű és könnyen használható megoldásokat tartalmaz, míg a TinyOS egy minden eszközzel felvértezett operációs rendszer a fejlesztésekhez. A TinyOS mélyebb ismereteket követel meg, és szélesebb körű fejlesztési lehetőségeket biztosít más eszközzel szemben [6, 7].



3. ábra: Az mts300 szenzorboard és a MicaZ szenzor

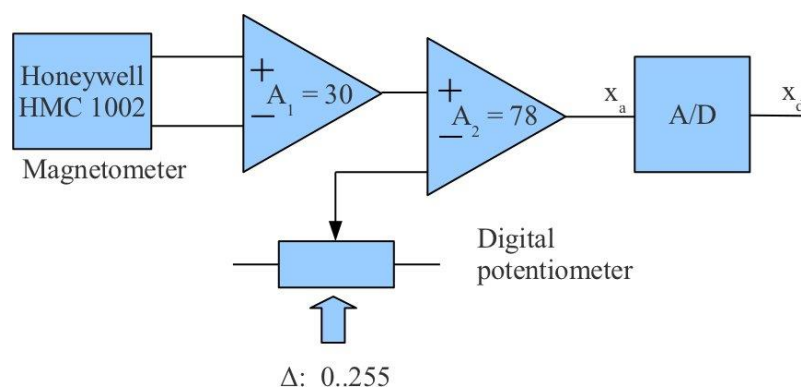
Alacsony fogyasztású eszközökre optimalizálva könnyen lehet olyan alkalmazásokat írni, amelyek akár hónapokig vagy évekig képesek működni. A fenti pozitívumokon kívül bármilyen hálózat- méret mellett rendszerszinten képes biztosítani idősinkronizációt. Természetesen léteznek hátrányai is a TinyOS-nek. Ezekből mindenképpen foglalkozni kell a beágyazott rendszerek alacsony számítási kapacitásával, amit komoly tervező munkával lehet csak jól megvalósítani korlátozott erőforrások mellett. További nehézségeket jelenthet, hogy az operációs rendszer

programozásához több időbe telik beletanulni, mint azt egy már ismertebb programozási nyelvet használó alkalmazásnál elvárjuk.

## 2.2 HMC1002 - Mag3110

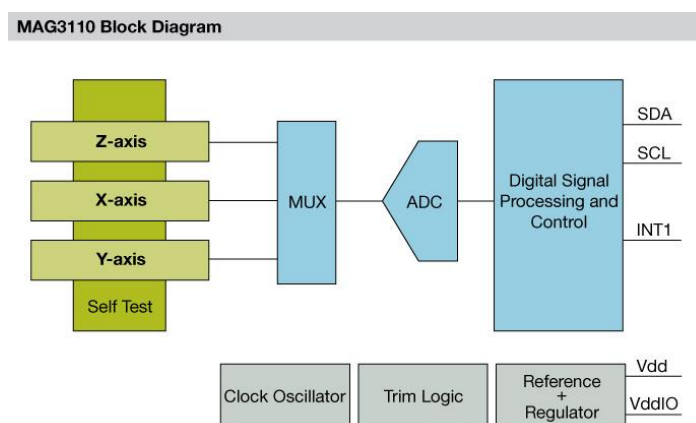
A feladatok elvégzéséhez első körben a MicaZ szenzorokhoz kapható mts300-as szenzorboardot vettük elő. Ez az eszköz mikrofonnal, hangszóróval, 2-tengelyű gyorsulásmérővel, és 2-tengelyű mágneses érzékelővel van felszerelve [8].

A magnetorezisztív elven működő mágneses szenzor a Honeywell HMC1002-es modellje. Működési elvét tekintve egyszerű híd alapú megvalósítás, amelyre 0 – 10 V közötti feszültséget kapcsolva a szenzor működésbe lép. Nikkel és vas ötvözetű anyagból építették fel a szenzort, amely a jelen lévő mágneses mező hatására megváltoztatja a kimeneti feszültséget. Mielőtt az analóg/digitális (későbbiekben A/D) konvertálás megtörténne, a kimeneti jel két erősítő segítségével erősítésre kerül. Az erősítő második fokozatában offset kompenzációra is lehetőség van egy digitális potenciométer segítségével. Működési tulajdonságainál fontos megjegyezni a szélsőséges értékeket is, így képes működni -55 – 150 C°-on. Mérési tartománya -2 – 2 Gauss-ig terjed. Az mts300-as szenzorboardra épített eszköz blokkdiagramját a 4. ábra mutatja be [9].



4. ábra: Az mts300 szenzorboard blokkdiagramja

Digitális mágneses érzékelőkből a Mag3110-es szenzort használtuk, amelyet a Fressscale gyárt. Blokkdiagramját az 5. ábrán lehet látni. Alacsony fogyasztású 3-tengelyű szenzorról beszélünk, amely támogatja az I<sup>2</sup>C szabványt. Analóg párjához képest a szenzor csak a -40 – 85 C° tartományban képes operálni. Mérési tartománya pedig -10 – 10 Gauss-ig terjed [10].



5. ábra: Mag3110 szenzor blokk diagramja

## 2.3 *Matlab*

Fejlesztési és tesztelési célzattal is egyaránt a Matlab szoftvercsomagot vettük igénybe, amely egy magas szintű programozási nyelv. Adatfeldolgozás szempontjából gyorsan lehet látványos eredményeket elérni, és a program segítségével továbbá lehetőségünk nyílt valósidejű monitorozást is megvalósítani [11].

A MathWorks fejlesztésében álló szoftver technikai számításokra optimalizált programozási környezet, alkalmas numerikus számításokra, adat feldolgozásra, vizualizációra és algoritmusok fejlesztésre. Az iparban és a kutatásban is széles körben használt eszköz, amelyet az autóiparban, telekommunikációban és űrrepülésben is alkalmaznak.

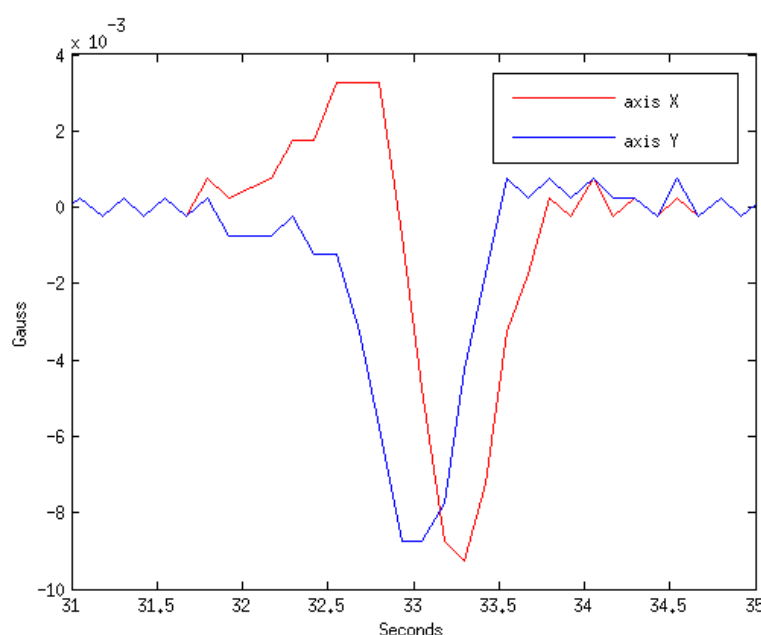
Teszteléseink során a rengeteg eredmény feldolgozásában nagyban segített minket a Matlab használata, pár soros szkriptek segítségével jól vizualizálható eredményeket sikerült felmutatnunk, továbbá a fejlesztés során a nyers adatokon el tudtuk végezni azokat a számításokat, amiket még nem implementáltunk a mote-okra. Az előzetes tesztekben kapott eredményeket pedig összevetve a mote-ra programozott algoritmus eredményeivel kimutathatóak azok a hibák, amik az implementáció során kerültek a kódba. Így sok eszköz felprogramozási nehézségtől sikerült megszabadulnunk, hiszen egy-egy felprogramozáshoz a szenzor telepeinek eltávolítása szükséges, szenzorboard eltávolítása és USB csatlakozó össze csatlakoztatása, továbbá a programozó csatlakoztatása és a programozás elindítása szükséges. Ebben a folyamatban az eszköz amortizációja is könnyen előfordulhat és a sokadik teszt algoritmusra meglehetősen kellemetlen folyamat lehet.

## 3 Fejlesztési folyamat

### 3.1 Megvalósítás HMC1002 szenzorral

#### 3.1.1 Offset konfigurációs algoritmusok

Analóg szenzorunk felhasználásával kezdtük el a projekt megvalósítását, amely során több problémával is szembe kellett néznünk. Végighaladva a fejlesztés folyamatán, első lépésben bemutatom a szenzor nyers jelét, amelyet a 6. ábrán láthatunk.



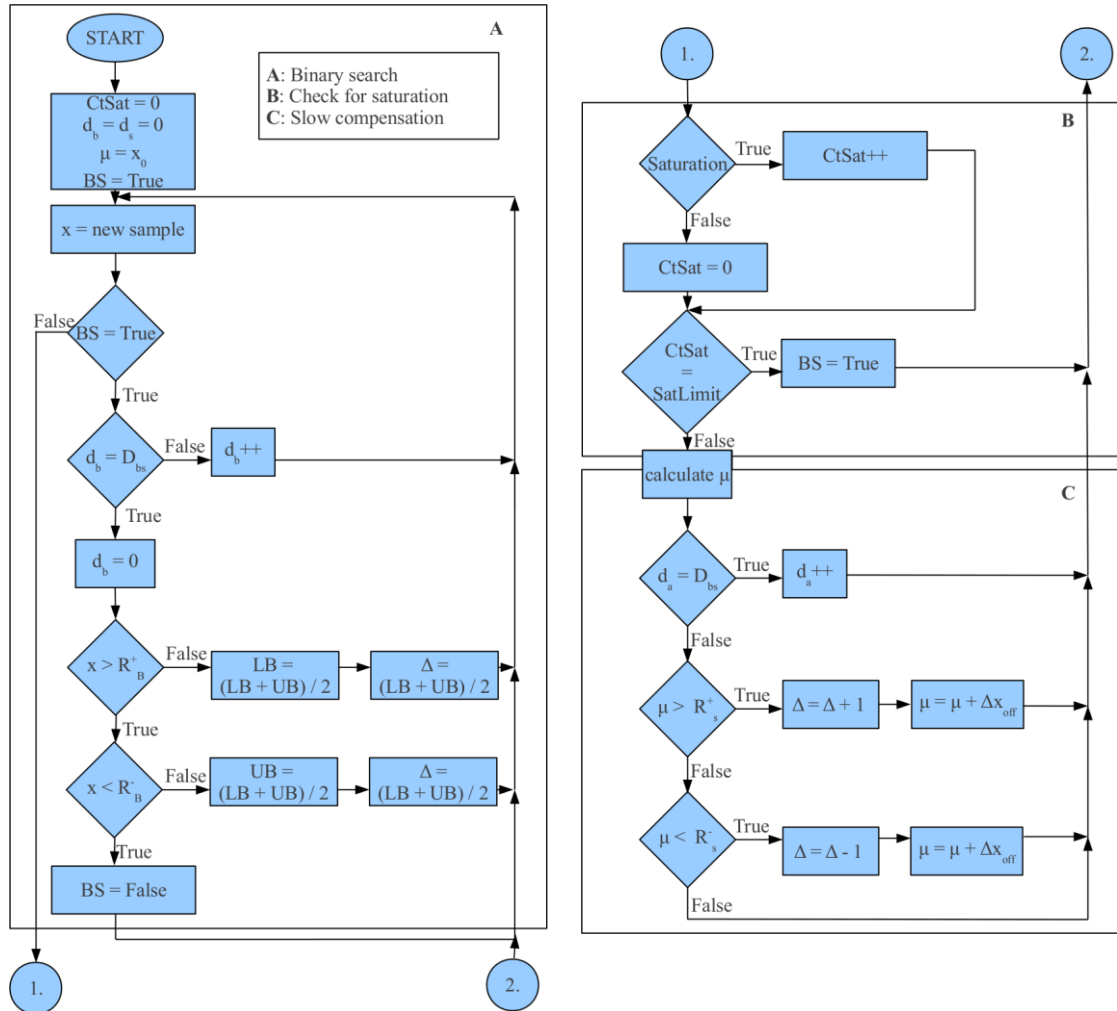
6. ábra: A mágneses mező változása a szenzortól 3 méterre elhaladó autó hatására

A szenzorunk érzékenysége 7.7 mV/Gauss-ra volt állítva és 2340-es erősítéssel dolgoztunk, ahogy a 4. ábra is mutatja. Látható, hogy egy elhaladó jármű 0.01 gauss változást okoz a Föld mágneses mezejében. Természetesen ez az érték függ az elhaladó jármű méretétől és távolságától is. A 4. ábrán látható mérés egy 3 méterre elhaladó járműről készült. Ahhoz, hogy ezt a kis változást akár 2-3 méteres távolságról is mérni tudjuk, nagy érzékenységre van szükségünk. Mivel a Föld 0.5 Gauss-os mért mágneses mezőjének iránya és ereje függ az adott pontban elhelyezett szenzorok lehelyezési irányától az eszközök könnyen telítődésbe vezérlődhetnek valamelyik irányba. Ennek a leküzdésére automatikus offset-kompenzációra van szükség a szenzor kitelepítése és áthelyezése után, illetve az esetleges véletlen elmozdulások kezelésére.

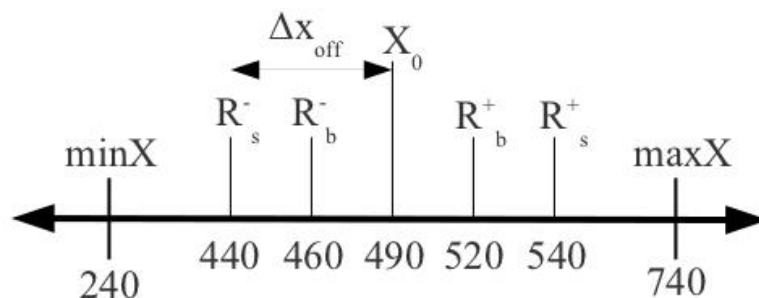
A 4. ábrán bemutatott mágneses érzékelő blokkdiagram esetén az A/D konverter után kapott eredmény  $x_{min} = 240$  és  $x_{max} = 740$  között van. Ahhoz, hogy sikeresen tudjunk járműveket detektálni, a nyugalmi állapotban valahol  $x_0 = 490$  körül kell beállni a szenzornak. Az offset a digitális potenciométer segítségével állítható be, ahol az LSB (Legkevésbé Szignifikáns Bit) változtatásával, amit  $\Delta$  jelöl, körülbelül  $\Delta x_{off} = 54$  LSB eltérést okoz az A/D konverter kimenetén. Mivel az offsetet nem tudjuk pontosabban

beállítani, mint a  $\Delta x_{off}$  értéke, ezért a beállítási algoritmusunk eredményeként az  $x_0$ -nak valahol  $R_b^+ = 480$  and  $R_b^- = 420$  között kell lennie. További követelménye az offset kompenzációs algoritmussal szemben, hogy gyorsan kell reagálnia a fellépő nyugalmi állapotbeli változásokra, akár elmozgatás okozza a kitérést, akár a rendszer első elindításáról van szó, de ezek mellett figyelni kell arra is, hogy az elhaladó járművek mágneses mezőre gyakorolt hatása ne eredményezze a kompenzációs eljárás futását.

A 7. ábrán láthatjuk az algoritmus folyamatábráját, amely két fő részre van bontva. A két működési eljárás külön felel a gyors beállításhoz szükséges algoritmus futtatásáért és külön a mágneses mező lassabb változásai által okozott eltérések kiigazításáért. Amennyiben a jel ( $x_d$ ) telített, azaz  $x_d = 240$  vagy  $x_d = 740$  legalább CtSat számú külön mérés idejéig, akkor az eljárás átvált a gyors offset beállítási fázisba (BS = True) és amíg  $x_d$ -t nem sikerült  $R_b^+$  és  $R_b^-$  közé állítani ebben az állapotban marad. A bináris kereső algoritmus frissíti a  $\Delta$  bementi értékét a digitális potenciométernek minden  $D_{bs}$  minta után, amit azért kell késleltetni, hogy a beállási időt biztosítani tudjuk az eszköznek. Jelen esetben a  $D_{bs} = 2$  értéket használtuk.



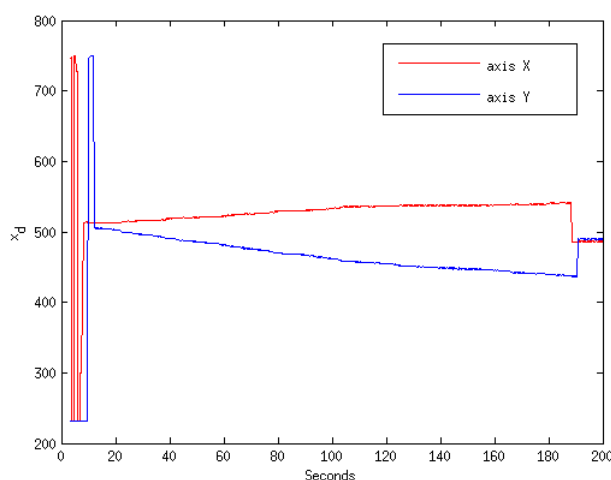
7. ábra: Offset kompenzációs algoritmusok folyamatábrája



8. ábra: A konstansok és a határok az offset kompenzációs algoritmusban.

A lassú kompenzációs algoritmus (*slow compensation mode*) feladata a lassú változások nyomon követése és korrigálása a kimeneti jelben. A problémakör vizsgálatához itt már nem az  $x_d$  kimeneti jelet fogjuk használni, hanem az  $x_d$  átlagát, amit  $\mu$ -vel jelölünk. A lassú kompenzációs algoritmus elvégzi a  $\mu$  értékelését minden  $D_{sl}$  minta után, kellő időt hagyva az átlag beállítására. A megvalósított alkalmazásban a  $D_{sl} = 8$  beállítást alkalmaztuk. Ha az átlagos  $\mu$  a  $[R_s^-, R_s^+]$  határokon kívül van, akkor a  $\Delta$  értékét az algoritmus megváltoztatja eggyel, ami az  $x_d$ -ben fog változást eredményezni. Mivel ilyenkor a nyugalmi állapot értéke megváltozik  $\Delta x_{off}$ -al, ezért a számított átlag értékét is meg kell változtatni. Azonos értéket használunk azzal, amit egy átlagos változás okoz, hiszen előfordulhat, hogy az átlag visszatérése lassabb, mint a következő ellenőrzés időpontja, így az túlírányítottá válhat (ezt az értékadást mutatja a 7. ábrán a  $\eta = \eta \pm \Delta x_{off}$ ). Ezzel a módszerrel az  $x_d$  és a  $\mu$  együtt változnak, amivel megoldottuk azt a problémát, hogy a detektor nem érzékeli a jelben bekövetkezett változást. A 8. ábra mutatja a szenzornál használt határokat és konstansokat. A tervezés során  $R_b^+$  és  $R_b^-$  tudatosan lett nagyobbra választva, mint a  $\Delta x_{off}$  hiszen így tudjuk elérni, hogy minden esetben egy közepes értéket találjon a bináris kereső algoritmusunk. Az  $R_s^+$  és  $R_s^-$  távolsága  $x_0$ -tól megegyezik a  $\Delta x_{off}$  méretével azért, hogy az  $\mu$  a lassú beállítás segítségével mindig a középpontban legyen.

A szenzor elindítása vagy a szenzor nagyobb elmozgatása után a bináris kereső algoritmus gyors beállítási protokollt biztosít. Amikor az offset sikeresen beállításra került az algoritmus normál üzemmódba vált, ahol a lassú kompenzációs algoritmus figyeli a szenzor működését. A 9. ábra mutatja az algoritmus egy kezdeti gyors beállítás utáni működését, a végére egy lassú kompenzációs lépést elhelyezve.



9. ábra: Automatikus gyors és lassú offset kompenzáció.

A lassú változás miatt a nyugalmi állapot nem áll be a szenzornál, de ezt a lassú kompenzáló algoritmus sikeresen kiigazítja az offset helyes beállításával. Természetesen a 9. ábrán egy mesterséges változást idéztünk elő a szenzor lassú mozgásával, de éles környezetben is jelen van ez a lassú változás, amely több óra alatt éri el azt a szintet, hogy az algoritmus újra állítsa az offsetet.

### 3.1.2 Eseményazonosítás

A 10. ábra mutatja az energia-alapú esemény detekciós algoritmus folyamatábráját. A  $\mu[k]$  átlag a nyers jel értékeiből számolt, ezek után a jel négyzetes hibáját számoljuk, ami az  $sx[k]$  jelölést kapta. Ezt az értéket egy aluláteresztő szűrővel szűrjük, hogy megkapjuk az  $M_{sx}[k]$ -t, amit az események detekciójához fogunk felhasználni. Az átlagot az alábbi képlet szerint számoljuk ki:

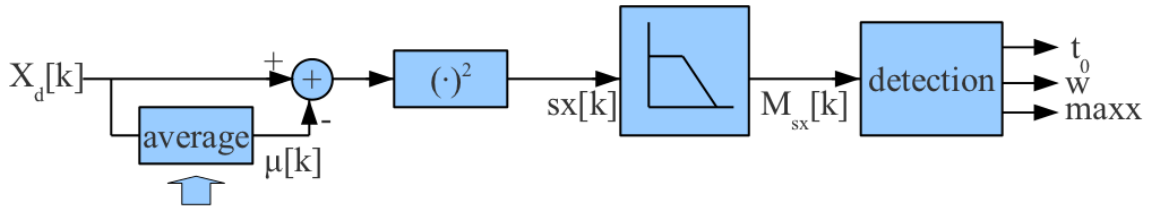
$$\mu[k] = \mu[k-1] + \alpha \cdot \text{sign}(e[k]) \cdot \min(\text{abs}(e[k]), \gamma),$$

ahol az  $e[k] = x[k] - \mu[k-1]$ ,  $\gamma$  pedig az ideiglenesen fellépő rendellenességek minimalizálására alkalmazott konstans, ilyenek az autók okozta elmozdulások is,  $\alpha$  pedig a felejtési tényező. A négyzetes hibátág számítása az alábbi módon történik:

$$sx[k] = (sx[k] - \mu[k])^2$$

$$M_{sx}[k] = \frac{1}{M} \sum_{i=k-M+1}^k sx[i]$$

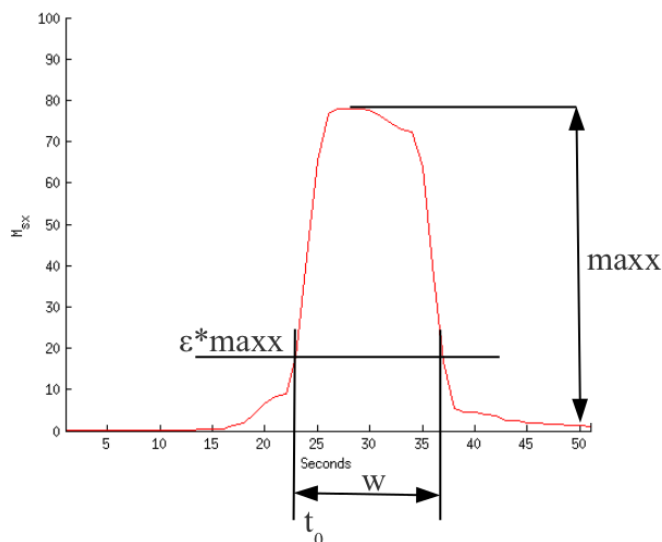
Az aluláteresztő szűrő hossza jelen esetben  $M$  hosszú átlagoló.  $M_{sx}[k]$  értékén több tulajdonságot tudunk vizsgálni, meg lehet határozni az esemény idejét ( $t_0$ ), a szélességét ( $w$ ) és a maximum amplitúdóját ( $maxx$ ). Az esemény azonosítására szolgáló adatok számításának folyamatábrája a 10. ábrán látható.



Update  $\mu$  (slow offset compensation)

10. ábra: Energia-alapú detekció folyamatábrája

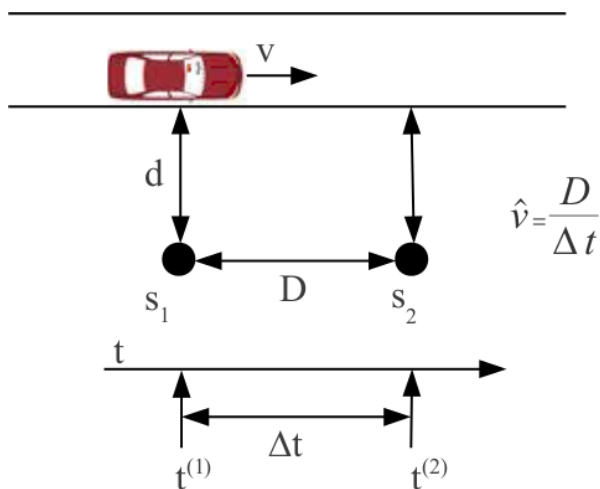
Az eseményazonosításnál használt információkat a 11. ábra mutatja be.



11. ábra: A megfigyelt tulajdonságok

### 3.1.3 Sebességbecslés

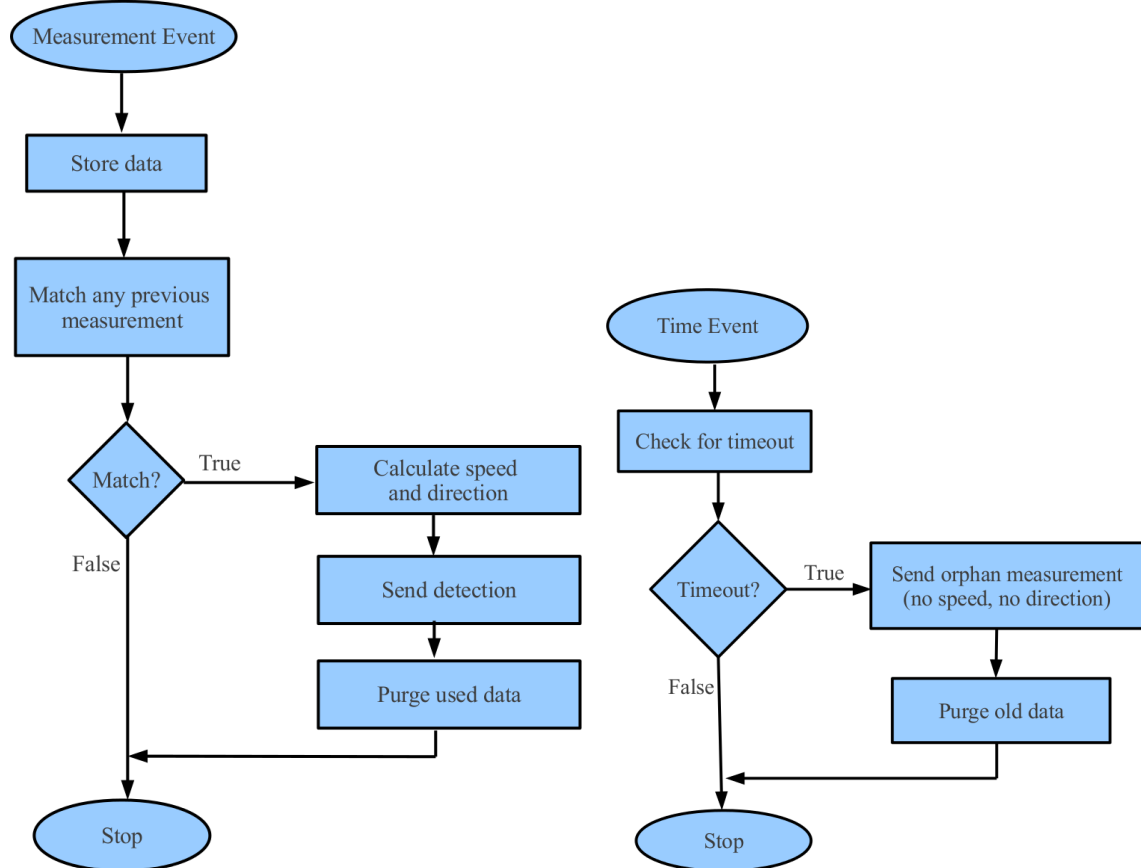
Több szenzor használatával lehetőségünk van az elhaladó járművek sebességének becslésére. Két szenzorból álló mérést mutatunk be a 12. ábrán, ahol az elhaladó járművet az  $s_1$  és  $s_2$  szenzorok  $t_0^{(1)}$  és  $t_0^{(2)}$  időben detektálják.



12. ábra: Sebesség kalkulálás két szenzor segítségével

A járművek sebességet megkaphatjuk a  $\hat{v} = \frac{D}{\Delta t}$  egyenlőségből, ahol a  $\Delta t = t_0^{(2)} - t_0^{(1)}$  és a  $D$  a két szenzor közötti távolságot reprezentálja. Meg kell jegyezni, hogy a szenzorok közötti időszinkronizáció elengedhetetlen. A sebességbecslő tartalmaz egy korrelátor egységet, amely eldönti, hogy a két mérés  $(t_0^{(1)}, w^{(1)}, maxx^{(1)})$  és  $(t_0^{(2)}, w^{(2)}, maxx^{(2)})$ , amit az  $s_1$  és  $s_2$  szenzorok küldtek, ugyanahhoz az autóhoz tartoznak-

e vagy sem. A detektor felhasználja a mérés időpontját, a legnagyobb értéket, és a szélességét az okozott kitérésnek. Abban az esetben, ha a detektor talál két összeillő párt, a sebesség becselő elvégzi a számításokat.

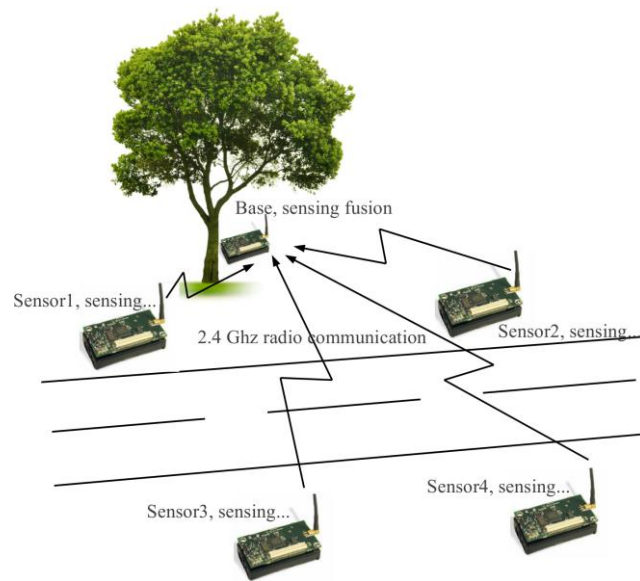


13. ábra: a, Esemény azonosítás és b, időzített eldobás folyamatábrája

A detektor algoritmus aktiválódhat egy jármű elhaladásától is, de egy külső időzítő is futásra kényszerítheti. Amikor egy eseményt detektálunk, a hármasunk ( $t_0$ ,  $w$ ,  $maxx$ ) eltárolásra kerül és egy esemény párt keresünk hozzá a többi eltárolt detektálás közül. Ennek a módszernek a folyamatábráját a 13. ábra a, része mutatja. Ha sikerül találni párt a detektált eseményhez, akkor a jármű sebességére és elhaladásának irányára végzett következtetés rádión keresztül elküldésre kerül. Egy külső időzítő periodikusan jelzi a detektornak, hogy a memóriában tárolt még pár nélküli eseményeket törölje a rendszerből. Erről az 13. ábra b, része mutatja a folyamatábrát.

### 3.1.4 Mérési összeállítás

Egy szenzorhálózati alkalmazás ábrát láthatunk a 14. ábrán. A szenzorok az út mellé vannak elhelyezve, 3-5 méteres távolságokra. Jelen alkalmazásnál a szenzorok az alacsony szintű detekciót végzik el, tehát a detektált járművekből elvégzik a hármas értékek ( $t_0$ ,  $w$ ,  $maxx$ ) kiszámítását. A mért eredményeket pedig a 2.4 GHz-es szabad ISM csatornán egy bázisállomás felé továbbítják, ahol megtörténik az adatok összesítése és a mért adatok függvényében kiszámításra kerül az elhaladó jármű sebessége és iránya. Az eszközök közötti közös időalap megteremtéséhez utólagos időszinkronizációt alkalmazunk a bázisállomás és a szenzor node-ok között.



**14. ábra: Szenzorhálózat sematikus ábrája**

## 3.2 Megvalósítás Mag3110 szenzorral

### 3.2.1 Digitális átállás

Miután a *MicaZ* szenzorokkal sikerült megvalósítani a kitűzött célt, elkezdtek a rendszer hosszú távú tesztjeit végezni. Ehhez az egyedi fejlesztésű eszközökbe is beépített HMC1002-es és HMC1001-es szenzorokat használtuk. Így sikerült megvalósítani egy 3 tengelyű szenzort, amely fogyasztási eredményei sajnos nem voltak kielégítőek a hosszabb távú használatra. Ezeknél a teszteknel figyeltük meg, hogy az analóg szenzorok fogyasztása nehezen csökkenthető az általunk elvárt körülmények között, ezért a továbbfejlesztéshez további tanulmányokat végeztünk. Arra a megállapításra jutottunk, hogy a célnak esetleg egy digitális szenzor érzékenysége is megfelelő lenne, áram felvétel szempontjából pedig egyenesen kiváló, hiszen a digitális szenzorok által nyújtott áramfogyasztási adatok kimagaslóan jók voltak. Az új szenzorokat a 15. ábra mutatja.



15. ábra: Egyedi fejlesztésű mágneses szenzor

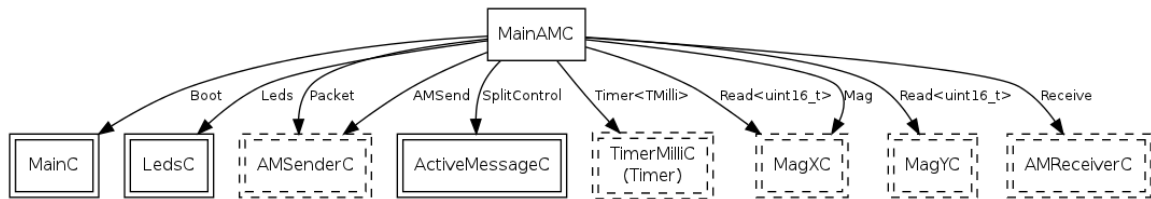
A kiválasztási folyamatnál a Mag3110-es szenzorra esett a választás, amelynek tulajdonságai a 2.2 fejezetben korábban leírásra kerültek. Az analóg szenzornál kifejlesztett kódot sikerült átültetni erre az eszközre. A digitális szenzoroknál a Föld mágneses mezője nem okoz túlvezérlést, így ezeknél sem offset állításra, sem a kompenzációs algoritmusokra nincs szükség.

## 4 Szoftver dokumentáció

### 4.1 Implementáció

#### 4.1.1 A rendszer felépítése

A *TinyOS* programozás eseményvezért környezetében megvalósított rendszernek robusztusnak, és folyamatosan elérhetőnek kell lennie. A követelmények közé tartozik a nagy érzékenység és hosszú rendelkezésre állási idő is. Ehhez az alábbiakban szeretném bemutatni a megvalósított rendszerünk tervét.



16. ábra: TinyOS alkalmazás blokk diagramja

A *TinyOS*-ben készült alkalmazásunk felhasznált objektumait és interfészeit láthatjuk a 16. ábrán, ez az alkalmazásnak azon része, amelyet a szenzorainkra töltünk fel. Felépítését tekintve egy olyan alkalmazás, amely alapvetően két fájlból áll, az egyik jelen esetben az *AppP.nc* a másik pedig az *AppC.nc*. Az *AppC.nc* fájlban adjuk meg azokat az információkat, hogy milyen interfészeket szeretnénk használni, és azokat milyen módon kötjük össze az alkalmazásunkban. Az *AppP.nc* fájl tartalmazza a forráskód általam implementált részét, amelybe tartoznak a függvények, az események kezelései. A harmadik fájl, amit fontos megemlíteni a *DataMsg* néven található meg az alkalmazásunkban. Ez tartalmazza a *struct* típusú mezők kifejtését és az üzenetküldéshez használt üzenetek definícióját.

Az implementálás oldaláról másik fontos elem a *Matlab*-ban megvalósított rész, amely a detektált események azonosításáért felel, továbbá a nyers mérési eredmények feldolgozásában játszott nagy szerepet a fejlesztési folyamatok során.

#### 4.1.2 AppP.nc

A fejlesztés során 2-, 3-tengelyű szenzorokat használtunk így a következő részben bemutatott implementáció során a változók és konstansok elnevezésében megjelenő *X,Y,Z* jelöli, hogy melyik tengely megvalósítását mutatom be, de a többi tengelyre is azonos a megvalósítás menete. Esetleges eltérés esetén arra külön felhívom a figyelmet.

A szoftverhez használt interfészek betöltése történik meg az *AppP.nc* fájlban, a modul részben, a feladathoz a *Mag3110* szenzor interfészét hívja be. A *Boot* a *TinyOS*-hez szükséges interfész. A szenzor értékeinek a kiolvasásához a *Read* és a *GeneralIO* interfészt használjuk. A rádióhoz tartozó interfészek a *SplitControl*, *AMSend*, *Packet*. A feladathoz szükséges egy *Timer* is, továbbá tesztelési funkciókhoz a *Led*-ek használata.

```
uses interface Mag3110;  
uses interface Boot;
```

```

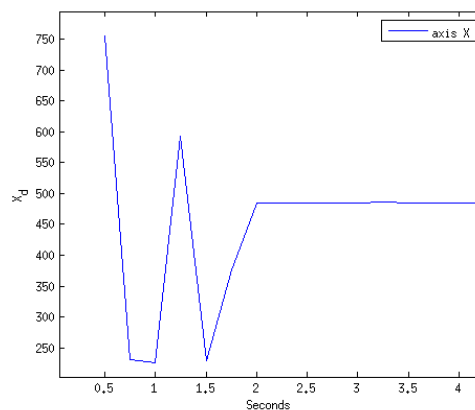
uses interface Read<reg_t>;
uses interface GeneralIO;
uses interface SplitControl as RadioControl;
uses interface AMSend;
uses interface Packet;
uses interface Timer<TMilli>;
uses interface Leds;
uses interface Timer<TMilli> as currentTime;

```

Az *AppP.nc* fájlban található a kód implementációja is (TinyOS: implementation szekciója). Ebben először a szükséges változók deklarását végeztem el. Itt került definiálásra pár statikus változó is.

### 4.1.3 BinarySearch

A gyors beállítási algoritmus implementálása található ebben a részben. Ehhez elsőként olyan konstansok definiálására volt szükség, amely az inicializálás során számunkra megfelelő állapotra hozza a szenzort. Első lépésként oly módon állítjuk be a szenzorunk érzékenységet, hogy az az egyik irányba túlvezérlődjön. Ezek után történik meg lépésenként a gyors beállítás. Minden lépésben az offset értékét bináris kereséssel



17. ábra: Gyors beállításra példa

változtatjuk, és azt vizsgáljuk, hogy a szenzor melyik szélső értékre vezérlődött túl, majd ennek megfelelően változtatjuk az offset értékét. Addig folytatjuk ezt a keresést, amíg a jel a kívánt tartományba nem esik. A 17. ábra mutat egy példát a bináris keresés alapú gyors beállításra, ebben az esetben 6 lépésből találta meg az algoritmus a helyes értéket (a kezdeti érték a 0. lépés). Látszik, hogy a szenzor 0.5 másodpercenként végez el egy-egy próbát, gyorsabb beállításra nincs lehetőség, mivel a digitális potenciométernek szüksége van türelmi időre, amíg felveszi a kívánt értéket.

A *binarySearchTreeX* segítségével állapítjuk meg, hogy a gyors beállításnak az aktuális lefutás esetén végre kell-e hajtódnia.

```

if(!binarySearchTreeX || !binarySearchTreeY){
    vizsgalCounterX = 0;
    vizsgalCounterY = 0;
}

```

```

if(elseAllitasX){
    elseAllitasX = FALSE;
    setBinaryValue = ((upperBoundX - lowerBoundX)
        / 2) + lowerBoundX;
    call MagX.gainAdjustX(setBinaryValue);
    return;
}

```

Az első beállítás után már van választási lehetőségünk. Az aktuális intervallumra vonatkozó adatokat a *setBinaryValue*-ban tároljuk. Megvizsgáljuk, hogy melyik irányba szükséges állítani a szenzorunk offsetjét és ennek megfelelően a *lowerBoundX* vagy *upperBoundX* változóknak megadjuk a szükséges értéket.

```

if(!binarySearchTreeX && !elseAllitasX){
    setBinaryValue = ((upperBoundX - lowerBoundX)
        / 2) + lowerBoundX;
    if(dataX > FELSOKORLAT_RB){
        lowerBoundX = setBinaryValue;
        setBinaryValue = ((upperBoundX -
            lowerBoundX) / 2) + lowerBoundX;
        call MagX.gainAdjustX(setBinaryValue);
    }else if(dataX < ALSOKORLAT_RB){
        upperBoundX = setBinaryValue;
        setBinaryValue = ((upperBoundX -
            lowerBoundX) / 2) + lowerBoundX;
        call MagX.gainAdjustX(setBinaryValue);
    }
}

```

Ha a beállítás sikeres, tehát az elvárt határértékeken belülre sikerült állítani a digitális potenciométerrel a jelet, akkor elkezdhetjük az átlagok számítását a többi funkció megvalósításához. Ehhez az *atlagX* és *atlagY* változóknak az optimális értéket adjuk, ami a 490. Mivel a későbbiek során szükség van az offset további állítására (lassú kompenzációs algoritmus során) eltároljuk az *allitX* változóban az utolsó digitális potenciométer állást.

```

else if((dataX < FELSOKORLAT_RB)&&(dataX >
    ALSOKORLAT_RB)){
    binarySearchTreeX = TRUE;
    atlagX = 490.0;
    atlagY = 490.0;
    allitX = setBinaryValue;
}
return;
}

```

A szenzorunkkal előfordulhat, hogy valamilyen módon elmozdítják a helyéről, vagy csak egyszerűen mi helyezzük odébb kikapcsolás és újraindítás nélkül, ezért szükséges egy ellenőrző rutint elhelyezni a kódban, amely figyel, hogy a szenzor tartósan ne legyen a skála határértékén beragadva. Ehhez az alábbi kódrészlet biztosít megoldást, ami 8 Hz-es mintavételezést figyelembe véve 2 másodpercig engedi a

szenzornak, hogy esetleges maximális vagy minimális értéket vegyen fel, azután beavatkozik és újraállítja a szenzort. Ez a 2 másodperces határ több teszt eredményeként is beigazolódott, hogy járművek detektálásában nem fog problémát okozni, ugyanis azok túlszordulást okozó eltérése 1 másodpercnél minden esetben rövidebb.

```

if((dataX < 300) || (dataX > 700)){
    saturationCounterX++;
}else{
    saturationCounterX = 0;
}
if(saturationCounterX == 16){
    binarySearchTreeX = FALSE;
    elsoAllitasX = TRUE;
    lowerBoundX = 0;
    upperBoundX = 255;
    return;
}else{
    atlagX = q*dataX + ( 1 - q ) * atlagX;
}

```

#### 4.1.4 SlowCompensation

A lassú kompenzációs algoritmus megfelelő időközönként megvizsgálja a jel átlagának értékét, és ha az a korábban megállapított  $R_s$  határértéken túl lép, automatikus kompenzáció történik. Ezt a *MagX.gainAdjustX()* függvény végzi el, amelynek paramétere az *allitX* tartalmazza a beállítási értéket.

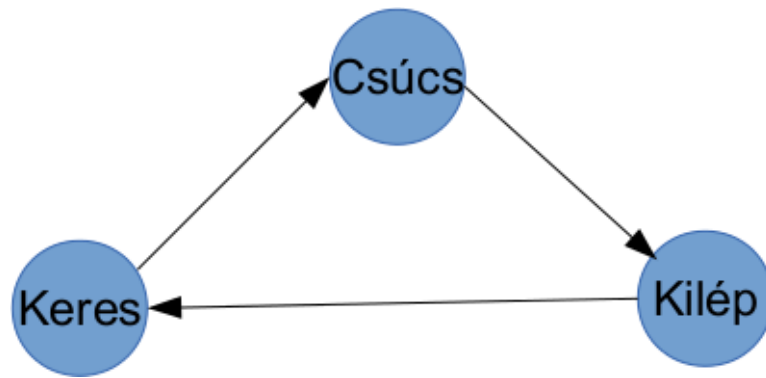
```

vizsgalCounterX++;
if(vizsgalCounterX == FINOMSAG){
    if(atlagX < ALSOKORLAT_RS){
        atlagX += 50;
        allitX -= 1;
        call MagX.gainAdjustX(allitX);
    }else if(atlagX > FELSORLAT_RS){
        atlagX -= 50;
        allitX += 1;
        call MagX.gainAdjustX(allitX);
    }
    vizsgalCounterX = 0;
}

```

#### 4.1.5 findVehicle

A *findVehicleX* függvényünk végzi el az adott adatsoron történő járműdetektálást. Az algoritmus egy állapotgépet valósít meg, amelynek egyes állapotaiban különböző vizsgálatokat végez el. Ennek az állapotgépnek a pontjai a 18. ábrán láthatóak.



18. ábra: A járműdetekció állapotgépe

Az algoritmus kezdeti lefutáskor a *Keres* állapotban van, ahol a kritériumoknak megfelelően egy statikus *SZORZO* tag és az algoritmus által számolt *atlagKonst* változó szorzataként kapott határérték átlépését vizsgáljuk. Amennyiben az algoritmusunk által számított jelek átlélik a szorzatként kapott átlagot az állapotgépünk átkerül a *Csúcs* állapotba.

```

if (KERESX) {
    if (arrayHistory[arrayHistoryCounter%50].avgX >
        (atlagKonstX*SZORZO)) {
        call Leds.led1On();
        KERESX = FALSE;
        CSUCSX = TRUE;
        StartHegyXTime = call currentTime.getNow();
        DeltaCountX = call currentTime.getNow();
        StartHegyXTime = StartHegyXTime/1000;
    }
}

```

A *Csúcs* állapotban az algoritmusunk folyamatosan keresi a mért értékek közötti maximumot, amely a detekcióhoz felhasználásra kerül majd. A maximális értéken kívül a maximális érték bekövetkezésének ideje is tárolásra kerül. A *Csúcs* állapotban a kilépési feltételeket is vizsgálja az algoritmus így a belépéshez hasonlóan a korábban ismertetett szorzat alá eső jelérték tovább lépteti az állapotgépet és a *Kilépés* fázisba váltja a programot.

```

else if (CSUCSX) {
    call Leds.led2On();
    axxisX = 1;
    if (arrayHistory[arrayHistoryCounter%50].avgX > MaxX) {
        MaxX = arrayHistory[arrayHistoryCounter%50].avgX;
        MaxXTime = call currentTime.getNow();
        MaxXTime = MaxXTime/1000;
    }
    if (arrayHistory[arrayHistoryCounter%50].avgX <
        (atlagKonstX*SZORZO)) {

```

```

        KILEPX = TRUE;
        CSUCSX = FALSE;
        EndHegyXTime = call currentTime.getNow();
        EndHegyXTime = EndHegyXTime/1000;
    }
}

```

*Kilépés* során a korábban megállapított maximális jelérték adat tagja és ideje, továbbá a kilépő jelérték adat tagja és időbélyege kerül elmentésre a későbbi használatok céljából. (Ezekből az adatokból lehet több szenzor által szolgáltatott értékeket összehasonlítani és következtetéseket levonni a jármű sebességére, és irányára.)

```

else if(KILEPX){
    call Leds.led2Off();
    call Leds.led1Off();
    if((EndHegyXTime - StartHegyXTime) > 1.0){
        sendMaxX = MaxX;
        sendWideX = EndHegyXTime - StartHegyXTime;
        sendStartX = StartHegyXTime;
        axxisX = 11;
        StartHegyXTime = 0;
        EndHegyXTime = 0;
    }else{
        axxisX = 0;
        StartHegyXTime = 0;
        EndHegyXTime = 0;
        KILEPX = FALSE;
        KERESX = TRUE;
        MaxX = 0;
    }
    if(arrayHistory[arrayHistoryCounter%50].avgX <
        (atlagKonstX*SZORZO) ){
        KILEPX = FALSE;
        KERESX = TRUE;
        MaxX = 0;
    }
}

```

#### 4.1.6 setConfig

Az algoritmusban a *setConfig* függvény segítségével végezzük el a mágneses szenzor életrekeltését. Elvégzi a *HMC* szenzor alapvető beállításait és felkészíti a műveletvégzésre.

```

if(conf == READ_REGISTER) todo = TO_READREG;
else if(conf == SET_OFFSET){
    todo = TO_SETOFF;
    offset[0] = 0;
}

```

```

        offset[1] = 0;
        offset[2] = 0;
    }else{
        todo = TO_MEASURE;
        mode.raw = conf & 0x04;
        mode.cont = conf & 0x02;
        mode.fast = conf & 0x01;
    }
}

```

#### 4.1.7 sendMessage

A *sendMessage* függvényünk segítségével állítjuk össze az üzenetküldéshez használt adatokat. Első lépésben egy feltétel vizsgálat történik meg, hogy a rádió foglalt állapotát leellenőrizzük. Amennyiben szabad, az üzenetküldésre létrehozuk az elengedhetetlen *data\_msg* változó típust, amelybe összeállítjuk az üzenet tartalmát. Az üzenetünknek alapvetően az aktuális mérési eredményeket kell tudnia elküldeni, amely a *magx*, *magy*, *magz* változókbba kerül. Az *x*, *y*, *z* változókbba a nyers mérési jeleket küldjük át így lehetőségünk van a számított eredményeket ellenőrizni, majd a detekcióhoz szükséges állapotgép feltételül szolgáló szorzatot is továbbítjuk az üzenetekben minden küldésnél.

```

if(!busy){
    data_msg* msg=(data_msg*)call Packet.getPayload(
        &dataMsg, sizeof(data_msg));
    BUGCOUNTER++;
    if(BUGCOUNTER == 100){BUGCOUNTER = 0;}
    msg->magx = arrayHistory...
        [arrayHistoryCounter%50].avgX;
    msg->magy = arrayHistory...
        [arrayHistoryCounter%50].avgY;
    msg->magz = arrayHistory...
        [arrayHistoryCounter%50].avgZ;
    msg->x = dataX;
    msg->y = dataY;
    msg->z = dataZ;
    msg->atlx = atlagKonstX*SZORZO;
    msg->atly = atlagKonstY*SZORZO;
    msg->atlz = atlagKonstZ*SZORZO;
    msg->counter = binarySat;
    msg->ch3 = 0;
    if(call AMSend.send(AM_BROADCAST_ADDR,
        &dataMsg, sizeof(data_msg))
        == SUCCESS){
        busy = TRUE;
    }
}
}

```

#### 4.1.8 sendMessageX

Az üzenetküldések során szükség van a detekciós üzenetek megkülönböztetésére, hiszen ezek az üzenetek másfajta adattagokat tartalmaznak, mint a korábban ismertetett általános mérési üzenetek. Minden tengelynek saját üzenetküldési metódusa van a detekció céljából történő üzenetküldésre. Először megállapítják az üzenet összeállításának kezdő pillanatát a későbbi time-stampelés céljából.

Az általános esetben már bemutatott rádió foglaltság ellenőrzés és üzenet deklarálás után az algoritmus megadja a detekció belépésének időpontját, amelyet a *sendStartX* változó tárol, ezt követi a detekció hossza, hogy pontosan mennyi ideig tartózkodott a jelzési szint felett a számított jelünk, a *sendWideX* változóban tároljuk az előbbi információt. Az utolsó előtti feladat megadni a legmagasabb mért értéket a detekció során, ezt a *sendMaxX* változóba adjuk meg. Végül pedig az érzékelést kiváltó tengely megjelölése történik meg, amelyet az *x*, *y* vagy *z* változók beállításával jelölünk. Üzenetküldéssel egy időben a rádió foglaltságát a *busy* változó értékének *true*-ra állításával jelezzük azt a rendszernek, hogy épp egy üzenet küldésének folyamata zajlik.

```
uint32_t temp;
temp = call currentTime.getNow();
temp = temp / 1000;

if(!busy){
    data_msg* msg=(data_msg*)
        call Packet.getPayload(
            &dataMsg, sizeof(data_msg));
    msg->magx = sendStartX;
    msg->magy = sendWideX;
    msg->magz = sendMaxX;
    msg->x = 1;
    if(binaryDoneX > 0){
        msg->y = temp - binaryDoneX;
    }else{
        msg->y = 0;
    }
    if(eventDetX > 0){
        msg->z = temp - eventDetX;
    }else{
        msg->z = 0;
    }
    msg->ch3 = 1;
    msg->atlx = 0.0;
    msg->atly = 0.0;
    msg->atlz = 0.0;
    msg->counter = TOS_NODE_ID;
    if(call AMSend.send(AM_BROADCAST_ADDR,
        &dataMsg, sizeof(data_msg))
        == SUCCESS){
        busy = TRUE;
    }
}
```

```
}
```

#### 4.1.9 initArray

Az algoritmusnak a feladat megvalósításához egy tömbre van szüksége, amely lehetőséget biztosít a korábbi adatok tárolására. A memória szemét által okozott kellemetlenségek megelőzése érdekében a mérési eredményeket tároló tömböt először inicializáljuk -1-es értékek feltöltésével. Ezzel a futás során tudjuk azt is ellenőrizni, hogy az algoritmusunk hol tart a tömb feltöltésével és így az átlagképzés során a még nem használt tömbhelyek figyelmen kívül hagyhatóak.

```
int idx = 0;
for(idx = 0; idx < 50; idx++){
    arrayHistory[idx].avgX = -1;
    arrayHistory[idx].avgY = -1;
    arrayHistory[idx].avgZ = -1;
    arrayHistory[idx].time = -1;
}
```

#### 4.1.10 movingCalc

Az algoritmus megköveteli az átlagok számítását, a *movingCalc* függvényünk végzi el a korábbi 12 elem átlagának kiszámítását. Ebben a függvényben mind a három (x,y,z) tengelyre elvégezzük az átlagok számítását. Az így kapott elmúlt 12 mérési eredmény átlaga bekerül az *arrayHistory* 50 elemű tömbbe, amely a naplózásra szolgál és a későbbi feladatvégzések is az ebben a tömbben eltárolt aktuális elemekkel történnek.

```
int idx = 0;
movingAvgX = 0.0;
movingAvgY = 0.0;
movingAvgZ = 0.0;
for(idx = 0; idx < 12; idx++){
    if(arrayX[idx] != -1 && arrayY[idx] != -1 &&
        arrayZ[idx] != -1){
        movingAvgX += arrayX[idx];
        movingAvgY += arrayY[idx];
        movingAvgZ += arrayZ[idx];
    }
}
movingAvgX /= 12;
movingAvgY /= 12;
movingAvgZ /= 12;
arrayHistory[arrayHistoryCounter%50].avgX
    = movingAvgX;
arrayHistory[arrayHistoryCounter%50].avgY
    = movingAvgY;
```

```

arrayHistory[arrayHistoryCounter%50].avgZ
    = movingAvgZ;
arrayHistory[arrayHistoryCounter%50].time
    = call currentTime.getNow();

```

#### 4.1.11 Boot.booted

A *TinyOS* alapvető függvénye a *Boot.booted()*, amely a bekapcsolás eseményére kerül végrehajtásra. A jelenlegi alkalmazásban egyszerűen beállítjuk a feladatvégzéshez szükséges Input/Output egységeket, elvégezzük a szenzor konfigurálását és végül a rádió elindításának parancsát adjuk ki.

```

call GeneralIO.makeOutput();
call GeneralIO.set();
setConfig(MEASURE_RON);
call Mag3110.setTask(todo, mode, offset);
call RadioControl.start();

```

#### 4.1.12 RadioControl.startDone

A *booted* függvényben elindításra került a *RadioControl.start()* függvény, amely a rádió bekapcsolásáért felel. A *start()* függvény a *startDone* eseménnyel tér vissza az alkalmazásban, amelynek egyetlen paramétere egy *error\_t* típusú változó, amelynek megvizsgálása után (*err == SUCCESS*) elindítjuk az időzítőnket, amely a *TIMER\_PERIOD* statikus változóban meghatározott időközönként tüzelésre készíti az alkalmazásunkat.

```

call Leds.led0On();
    if(err == SUCCESS) call
Timer.startPeriodic(TIMER_PERIOD);

```

## 4.2 Alkalmazás használata

Az alkalmazás elindításához először meg kell határoznunk, hogy a bázisállomás melyik portra van kötve, ehhez az alábbi parancsot használhatjuk UNIX operációs rendszerben.

```
ls /dev/ttyACM*
```

Miután az operációs rendszerből kinyertük a csatlakoztatott portot, el tudjuk indítani a *SerialForwarder* alkalmazást, amely a *TinyOS* által biztosított *Java API*. A parancsot a megfelelő porttal paraméterezve tudjuk elindítani, megadva a boudrátát is.

```
java net.tinyos.sf.SerialForwarder -comm  
serial@/dev/ttyACM0:57600 &
```

Miután a *Java* alkalmazás elindult az alábbi paranccsal tudjuk elindítani a mérési eredmények feldolgozását.

```
./doit.sh
```

A *doit* elvégzi egy ideiglenes fájlba a folyamat logolását, amelyet később archiválni kell, ehhez nyújt segítséget az alábbi parancs

```
./archive.sh
```

Az *archive* egy az aktuális időt tartalmazó mappába lementi a teljes logot, amely akár több szenzor eredményeit is tartalmazhatja. Ahhoz, hogy a logok szét legyenek választva, egy külön scriptet használunk, amely a lenti paranccsal futtatható:

```
./log.sh
```

A *log* script egyszerűen szétválasztja külön állományokra azokat az üzeneteket, amelyek az előre definiált *node\_id*-val rendelkeznek. Majd ezekből a fájlokból még külön levágja a mérés során bekövetkezett járműdetekciókat a könnyű visszakeresés érdekében.

```
#!/bin/bash  
  
cat log | grep "#00101" > log101  
cat log | grep "#00102" > log102  
cat log | grep "#00103" > log103  
  
cat log101 | grep "101 X:" > log101_jelzes  
cat log102 | grep "102 X:" > log102_jelzes  
cat log103 | grep "103 X:" > log103_jelzes  
  
cat log | grep "#00101" | grep -v "101 X:" > log101  
cat log | grep "#00102" | grep -v "102 X:" > log102  
cat log | grep "#00103" | grep -v "103 X:" > log103
```

### 4.3 A sebességbecslő implementálása

A sebességbecsléshez a *MathWorks* cég *Matlab* programját vettük igénybe, amellyel könnyen kapcsolatot lehet teremteni a *TinyOS* alkalmazásokkal a *Java* támogatásnak köszönhetően. A scriptünk megvalósítás szempontjából két részre bontható. Első lépésként minden üzenetérkezéshez feldolgozót társítunk.

```
function addElement0130(node_id, tipus, maxx, maxy,  
    widex, widey, startx, starty, java_time, delta)
```

A beérkezett üzenet után megvizsgáljuk, hogy annak tartalma melyik eszköztől származik. Jelen esetben az 1-es azonosítóval rendelkező által küldött adatokat dolgozzuk fel.

```
if node_id == 1  
    if tipus == 1  
        disp('====1Y====');  
        disp(maxy);  
        disp(widey);  
        tmpDelta = delta / 1000;  
        data1(end+1) = struct('maxx',-1.0,'maxy',maxy,  
            'widex',-1.0,'widey',widey,'startx',-1.0,  
            'starty',starty,'time',time,'in_time',1);  
        plot(time, maxy , 'r*');
```

A beérkezett adatoknál vizsgáljuk, hogy az adott szenzoron létezik-e olyan detekció, ami még nem lett társítva, amennyiben ilyen létezik, azt közöljük a felhasználóval.

```
if detection(3).matching == 0  
    disp('felulírt adat1');  
end  
detection(3) = struct('maxx',maxy,  
    'widex',widey,'startx',starty,'time',  
    time-tmpDelta,'node_id',1,'matching',0);  
tmpTime = detection(3).time - detection(4).time;
```

Abban az esetben, ha a beérkezés előtti 1 másodpercen volt másik szenzorról bejövő detekció, akkor azt úgy vesszük, hogy van párja a mérésnek, és elvégezzük a kalkulációkat az előre megadott adatok szerint.

```
if (tmpTime < 1.0) && (tmpTime > 0)  
    disp('*****DET1Y*****');  
    disp((METER/tmpTime)*3.6);  
    detection(3).matching = 1;  
    detection(4).matching = 1;  
end
```

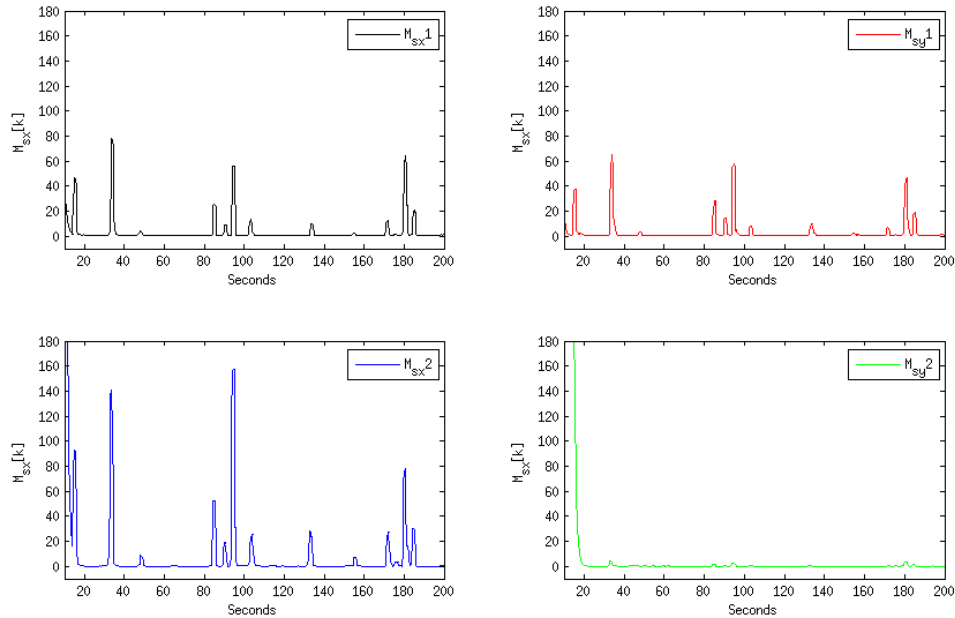
Azt az esetet, hogy egy jelzésnek nincs párja automatikusan küldött üzenettel is vizsgálunk. A szenzor egy figyelmeztetést küld, hogy az utolsó detekció már nem él, ha nem találtunk neki párt. Így újra inicializáljuk az algoritmust.

```
if tipus == 99
    tmpDelta = delta / 1000;
    time = time - tmpDelta;
    if (detection(1).matching == 0) &&
        (detection(1).time == time)
        if (time - detection(1).time) == 0.0
            disp('orphandlX');
            detection(1).matching = 3;
        end
    end
    if (detection(3).matching == 0) &&
        (detection(3).time == time)
        if (time - detection(3).time) == 0.0
            disp('orphandlY');
            detection(3).matching = 3;
        end
    end
end
end
```

## 5 Teszt eredmények

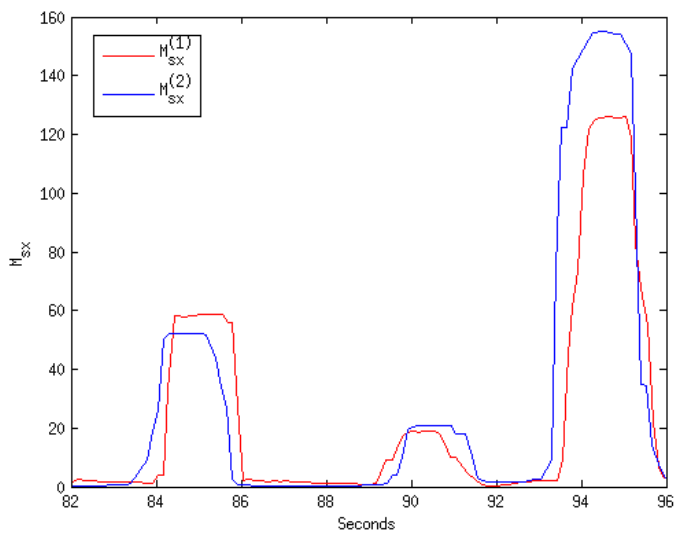
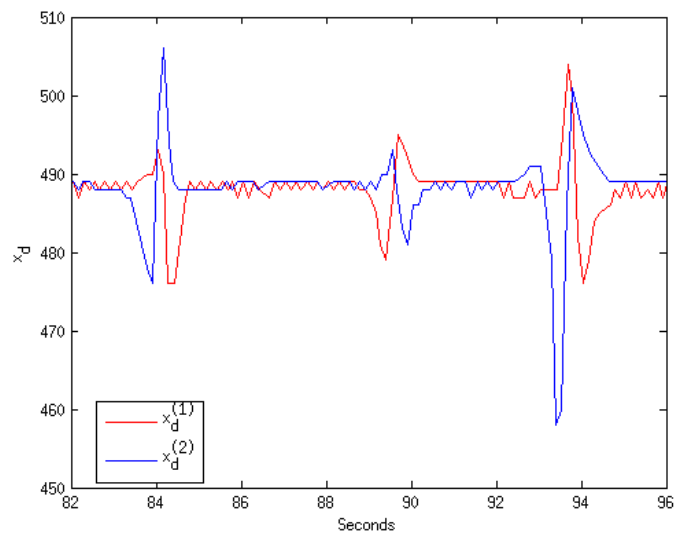
A fejlesztés során folyamatosan kellett a szenzorokon tesztek elvégznünk, amelyekről rengeteg feljegyzés készült. A tesztelési eredmények sokasága köszönhető annak is, hogy a szoftveres fejlesztéssel egy időben történt az eszköz hardveres fejlesztése is, így a tesztek több modellen keresztül készültek el, mire elértünk a végleges megoldásig. Ezekből az eredményekből csak néhányat kiemelve szeretném bemutatni az elkészített rendszer működését és tulajdonságait.

Egy analóg mérési folyamatot bemutatva látszik, hogy milyen eredmények tud a szenzorunk kimutatni egy jármű elhaladásakor. Jelen mérésnél még a sebesség következtető algoritmusunk nem volt implementálva. A teljes hosszú mérési eredményt a 19. ábra mutatja be. Az ábrán 2 szenzor párhuzamos mérési eredményei láthatóak. A baloldalon a két szenzor  $X$  tengelye, jobb oldalon pedig az  $Y$  tengelyen mért eredmények vannak. Jól látható, hogy a tengelyek érzékenysége nagyban különbözik, de az  $X$  tengely az útra merőleges beállításával, mind a két szenzoron kellően érzékeny volt.



19. ábra: Teljes mérés közúton

A 20. ábrán látható mérési eredmények egy olyan mérést mutatnak, ahol három jármű elhaladását figyeltük meg. Az út szélén elhelyezett két szenzor végezte a mérést, aminek az eredményeit a bázisállomás felé továbbítottak. A szenzorok egymástól 3 méteres távolságra voltak elhelyezve ( $D = 3\text{ m}$ ) és az úttól való távolságuk is 3 méter volt ( $d = 3\text{ m}$ ) (lásd a 12. ábrát). Ebben a kísérletben a *MicaZ* szenzorokat használtuk, az mts300-as szenzorboardal, amin a HMC1002-es magnetorezisztív érzékelők találhatók. A mérés során a szenzorok 8 Hz-es mintavételezési frekvenciára voltak állítva. A mérési eredményeket mutató első ábrán a nyers jelek kimenetele látható ( $x_d$ ) a második ábra mutatja a detekcióhoz használt már feldolgozott jelet ( $M_{sx}$ ). A táblázatban pedig a már általunk kalkulált eredmények láthatóak, ahol ( $t_0$ ,  $w$ ,  $maxx$ ) rendre az esemény kezdetének időpontja, az esemény hossza, és az esemény maximális kitérése.



|              |         |           |         |
|--------------|---------|-----------|---------|
| $t_0^{(1)}$  | 84,2 s  | 89.2 s    | 93.6 s  |
| $w^{(1)}$    | 1.76 s  | 2.25 s    | 2.12 s  |
| $maxx^{(1)}$ | 58.9    | 18.7      | 126.1   |
| $t_0^{(2)}$  | 83.3 s  | 89.5 s    | 93.4 s  |
| $w^{(2)}$    | 1.98 s  | 2.12 s    | 2.26 s  |
| $maxx^{(2)}$ | 52.1    | 20.8      | 154.9   |
| $\hat{v}$    | 22 km/h | - 41 km/h | 40 km/h |

**20. ábra: 3 elhaladó jármű detektálása**

**a, nyers jelek**

**b, kalkulált eredmények**

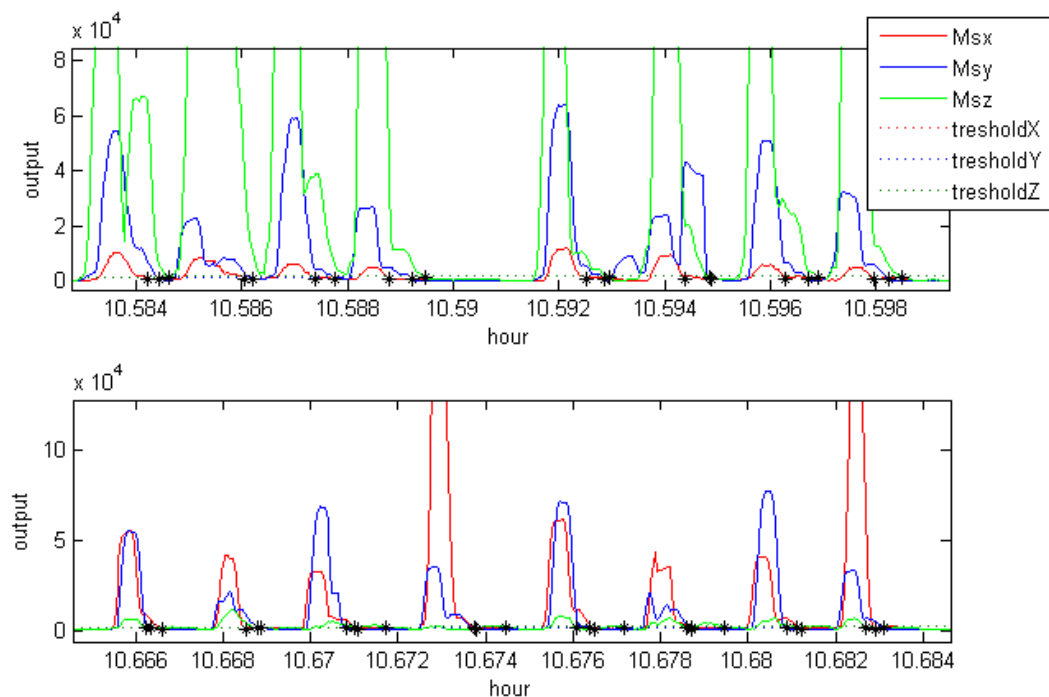
**c, mérési eredmények**

Látható, hogy a különböző elhaladó járművek ugyanazon a távolságon más mintát hagynak maguk után a mágneses térben, de az általunk megvalósított algoritmus képes széles skálán detektálni a járműveket.

Az analóg 3-tengelyes mérésekhez készítettünk érzékenységi teszteket is, amelyek azt mutatják meg, hogy a mágneses szenzor milyen irányú mozgásokat képesek a legjobban érzékelni. Ehhez a 21. ábra mutat egy mérési elrendezést, majd a 22. ábrán látható a mérés eredménye.



21. ábra: Tengelyek érzékenység vizsgálata

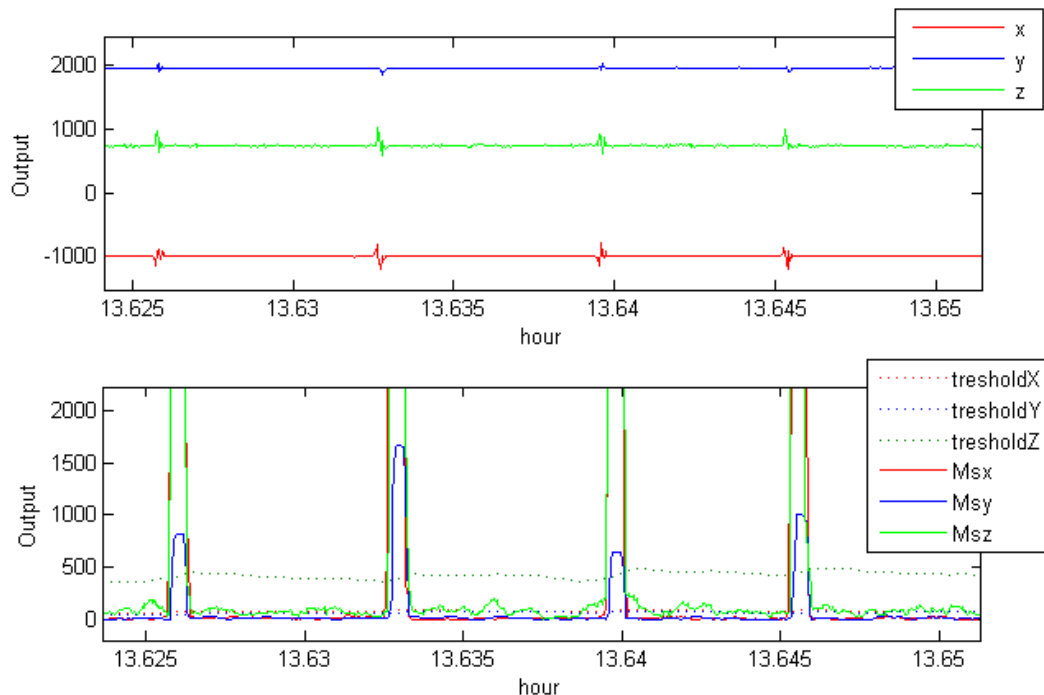


22. ábra: Tengelyek érzékenység vizsgálatának eredménye

A 22. ábrán bemutatott grafikon felső ábráján az állított szenzort gerjesztettük, amíg az alsó grafikonon a fektetett szenzort, így megkaptuk, hogy azok a tengelyek érzékenyek a változásra, amelyek merőlegesek a gerjesztés irányára. A mérési

elrendezés képén látható bicskával végeztük a gerjesztést és a külön felrajzolt íven haladtunk vele, ezzel biztosítva az egyenlő távolságokat.

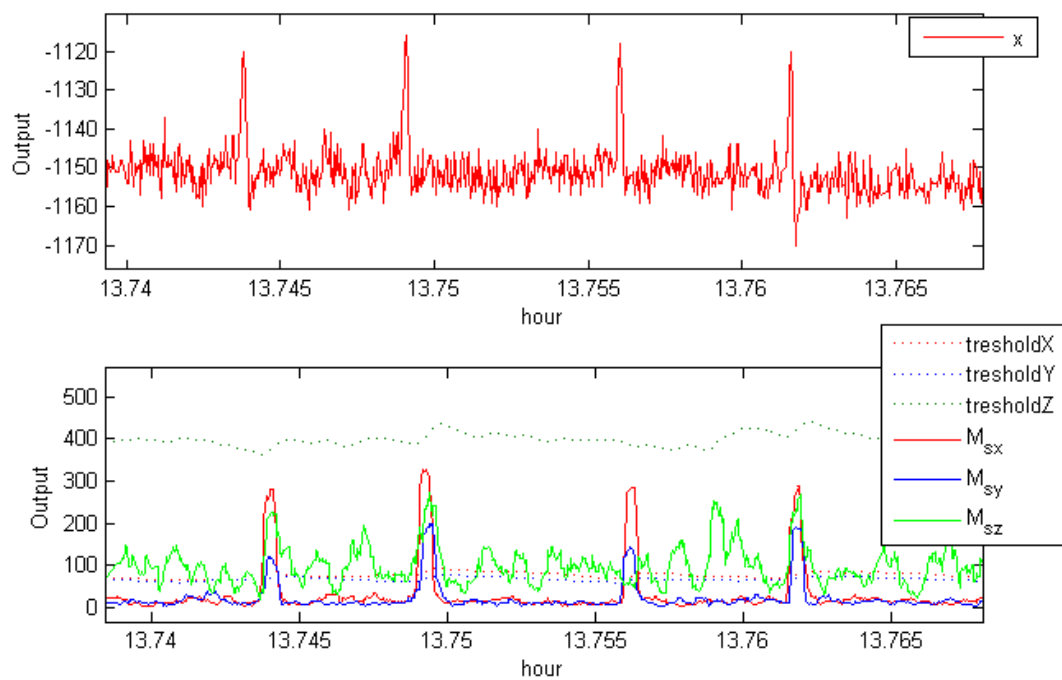
A digitális mérésekre példaként szeretném bemutatni az alábbi mérést, ahol jól látszik a távolságokban az érzékenység okozta nagymértékű csökkenés. A 23. ábra felső része mutatja a nyers jeleket három tengelyre levetítve, az alsó kép pedig a számolt értékünket, amelyet az adaptív küszöbértékhez viszonyítva (threshold jelöli ezt az értéket) tudjuk detektálni az elhaladó járműveket.



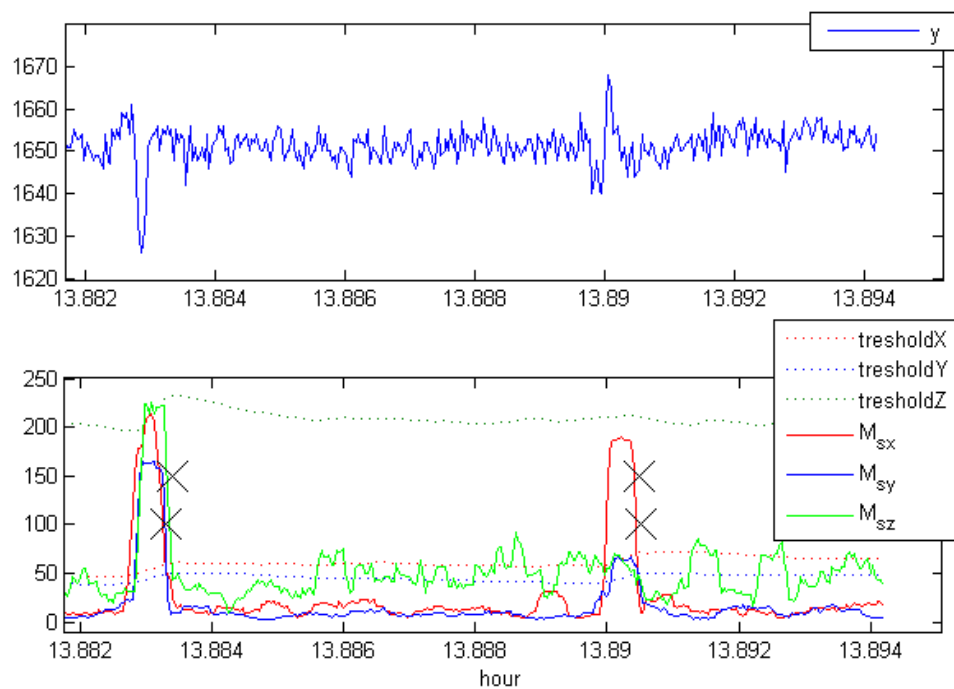
**23. ábra: Közvetlen az út melletti mérés eredményei**

A 24. ábra mutatja az 1 méteres távolságról készült képet. Láthatjuk, hogy a *zöld* tengelyünkön mért zaj jóval nagyobb, mint a *piros* vagy *kék* tengely zaja. Látható, hogy az 1 méteres távolságon a *zöld* tengely meg sem közelíti a detekcióhoz szükséges szintet, a *piros* és *kék* tengelyek jól láthatóan meghaladták mind a négy jármű elhaladásánál a detekciós szintet.

A 25. ábra a nyers jelet mutatja be (csak a kék tengelyre), alatta pedig láthatjuk, hogy az általunk számított jelből az algoritmus megállapította, hogy mikor haladt el jármű (A *zöld* tengelyen az elhaladást nem érzékelte). A detektált elhaladást az X-ek jelölik, minden tengely egy saját konstans értéket vesz fel, így a *piros* a 100-on jelzi, a *kék* pedig a 150-et veszi fel a grafikonon, ha elhaladást érzékel.



24. ábra: 1 méteres mérés eredménye



25. ábra: 1.5 méteres mérések detekcióval

## 5.1 Összehasonlítás más rendszerekkel

Az elkészített rendszer több szempontból is különleges a korábban elkészült mágneses szenzor alapú detekciós eljárásokhoz képest. Az eddigi alkalmazások többsége parkolást segítő rendszerek felépítéséhez használták, olyan eljárással, amely nem a járművek mozgás közbeni detekciójáról szól. Nagy távolságból érzékelő mágneses szenzorhálózattal nem találkoztunk.

*International Journal of Distributed Sensor Networks*-ben publikált cikk is egy olyan szenzorhálózattal foglalkozik, amely a parkolás segítésére lett létrehozva [12]. A cikkben tervezett rendszer szenzoregységek, bázisállomások és jelzőtáblák kombinációját valósítja meg. Céljuk a parkolás segítése szabad járműhelyek megfigyelésével és folyamatos visszajelzésével az érkező autósok számára. Alacsonyfogyasztású mágneses szenzort használva akár 5 éves rendelkezésre állási időt is tudnak biztosítani a parkolóba ágyazott szenzorjaiknak. Minden különálló parkolóhely alá elhelyeznek egy-egy szenzort és az általa küldött adatokból tudják, hogy az adott parkolóhely szabad-e. A feladathoz a HMC5883L típusú eszközt használják, amely egy digitális 3-tengelyű mágneses szenzor és a Honeywell terméke.

A Shenzhen WanQiaoHong Technology Co., Ltd cég által kínált rendszerek szintén intelligens parkolást segítő szenzorhálózatok, amelyek mágneses szenzoron alapuló, vezeték nélküli hálózatok [13]. Magas rendelkezésre állási idejét tekintve digitális szenzor használatát várnánk, amelyről sajnos nem kaptunk információt az oldalon (5 éves üzemidőt ígérnek). A rendszer célja ebben a formában, hogy az autósokat a legrövidebb idő alatt az első rendelkezésre álló szabad parkolóhelyhez irányítsák.

Hasonló célra RFID felhasználású parkolást segítő rendszereket is alkalmaznak. A Magnetic Autocontrol Group fejlesztésében olyan intelligens kapuk vannak a piacon, amelyek az elektromágneses tulajdonságot kihasználó RFID technológiára építenek [14]. A járművek a kapuhoz érkezéskor könnyen azonosíthatóak a kiadott egyedi kulcsú kártyák segítségével. Sajnos az általunk vizsgált területen ennek a fajta megoldásnak nincs létjogosultsága, hiszen ismeretlen járművek detektálására kell koncentrálnunk.

A Streetline fejlesztésében elkészült parkolást segítő rendszerhez bemutatásra kerültek a szenzor elhelyezéséhez szükséges munkálatok [15]. Ez mindössze három lépésből áll és akár 40 perc alatt is elkészülhet a kihelyezés. Első lépésben fel kell rajzolni a szenzor helyét (a parkolóhely közepére kell helyezni), majd az útburkolat, adott helyen való felmarása után a szenzor elhelyezhető a kivágott lyukban. Ezek után egy burkoló elem kerül a tetejére, amely védi az amúgy is masszív tokozású eszközt. Végül lépésként pedig egy bázisállomás elhelyezését végzik el egy magaslati ponton (például villanyoszlop). A használt szenzorokról a cég oldalán nem sikerült információkat találni.

Jól látható, hogy az általam megvalósított algoritmus és alkalmazás a mágneses szenzorokat más szempontból használja fel. Azok a szenzorhálózatok, amik a piaci szegmensben jelen vannak, a járművek jelenlétét vizsgálják és nem az elhaladásukat. A általam megvalósított rendszerben több szenzor segítségével a jelenlétén kívül a jármű haladási iránya és sebessége is becsülhető. Másik fontos különbség a szenzorok elhelyezésében van. A piaci rendszerekben a szenzorok a járművek alatt helyezkednek el, nagyon jó detekciót biztosító, mintegy fél méteres távolságon belül. Az általunk megcélzott alkalmazásban a szenzorokat az út mellett, biztonságos távolságra kell elhelyezni, védendő a szenzorokat a puha földutakon bekövetkező sérülésektől. A távoli

elhelyezés miatt a jel-zaj viszony rendszerünkben lényegesen rosszabb lett, így kifinomultabb detekciós algoritmusok alkalmazására volt szükség.

## 6 Összefoglalás

### 6.1 Konklúzió

A dolgozat elkészítése közben egy olyan olcsó mágneses érzékelésen alapuló szenzorhálózatot sikerült fejleszteni, amely képes érzékelni az elhaladó járműveket és képes azok sebességére becslést adni, illetve irányának meghatározására. A kidolgozott megoldásban kétfajta szenzort is alkalmaztunk. Az egyik nagyobb fogyasztással rendelkezik, így az hosszabb időre nem alkalmas az önálló működésre, de egy nagy érzékenységu mágneses szenzorral ellátott eszköz, így akár 3-5 méteres távolságból is alkalmas megfigyelésre. A másik megoldás egy alacsonyabb fogyasztású digitális szenzorral készült el, amely már kisebb távolságról képes csak érzékelni az elhaladó autókat, de rendelkezésre állási ideje jóval nagyobb, mint analóg párjának. Ezzel a szenzorral lehetőség van hosszú távon megfigyelni a területet és biztosítani a falopások elleni védelmet. A megfelelően elrejtett szenzorok egy magasban elhelyezett bázis állomás felé továbbítják a detektált járművek kiszámított adatait, amely riasztást tud adni egy központ felé a behatolásról.

A feladat megkezdéséhez használt 2-tengelyu mágneses szenzoregységgel ellátott *MicaZ* szenzorok jó szolgálatot tettek a magnetorezisztív elven felépülő szenzorok megismeréséhez és kitapasztalásához. Gyors offset beállításra alkalmas bináris keresés algoritmus fejlesztettem, amivel a szenzor bekapcsolás után gyors rendelkezésre állást biztosítja számunkra. Ezen kívül elkészítettem a lassú mágneses tér változásokból származó kimeneti jel elcsúszása okozta problémát orvosoló algoritmust, amelyet egy lassú kompenzációs logikával valósítottam meg. Lehetőségem nyílt így ezzel a szenzorral egy olyan jármű elhaladás felismerő alkalmazást készíteni, ami 3 méteres távolságról biztosan jelzi a közelben elhaladó járműveket és azok sebességére és irányára becslést tudok végezni a segítségével.

Miután elkészültek az első analóg szenzorral ellátott feladat specifikus szenzorok, amely már 3-tengelyu magnetorezisztív elemekből felépülő szenzorral volt ellátva, lehetőségem nyílt az eszköz terepen való tesztelésére. Természetesen mielőtt az új eszközt éles környezetben használtuk volna széleskörű teszteseteket definiáltunk, aminek meg kellett felelnie a szenzornak éles bevetés előtt. Ezek között a tesztek között szerepelt a tengelyek érzékenységeinek vizsgálata, a hosszú távú készenléti idő és mérési eredmény feldolgozás, továbbá a korábbi szenzor elkészített gyors beállási és lassú kompenzációs algoritmusok helyes futásának bemutatása. A hosszúra tervezett teszt mérések során és a terepmérésen mutatott eredményekből az analóg szenzorok létjogosultságát a magas fogyasztás miatt kétségbe kellett vonnunk, így azokról digitális szenzorra tértünk át.

A kisfogyasztású Mag3110 digitális szenzorokra elkészítettem a módosított detekciós algoritmusokat, amelyek tesztelése közben a várt érzékenység-csökkenés bebizonyosodott. Ezzel a megoldással a megbízható detekció mintegy 1.5-2m távolságról biztosítható, ami a kitűzött alkalmazási területhez még éppen megfelelő (a szenzor az út mellé telepíthető). A digitális szenzor által biztosítható energiatakarékos működés lehetővé teszi az eszközök karbantartás nélküli üzemeltetését akár egy éven keresztül is.

Jelenleg az általam készített szenzor hosszú távú terepi tesztüzemben vesz részt. A sikeres tesztek után remélhetőleg hamarosan valós környezetben tényleges falopás elleni védőeszközökben kerülhet alkalmazásra.

## Irodalomjegyzék

- [1] Köponti Statisztikai Hivatal. (dátum nélk.). Forrás: [https://www.ksh.hu/szamlap/eletunk\\_szg.html](https://www.ksh.hu/szamlap/eletunk_szg.html)
- [2] M. Shuai, K. X. (2008). An on-road wireless sensor network approach for urban traffic state monitoring. *ITSC2008*, (old.: 1195-1200).
- [3] Klein, L. E. (2000). Summary of vehicle detection and surveillance technologies used in intelligent transportation systems. Federal Highway Administration's Intelligent Transportation Systems.
- [4] Magyar Világ Kiadó. (1998). *Britannica Hungarica*. Budapest: Magyar Világ Kiadó.
- [5] Crossbow. (dátum nélk.). *MICAz wireless measurement system*. Forrás: <http://www.memsic.com>
- [6] TinyOS Documentation, <http://tinyos.com>
- [7] Gay, P. L. (2009). *TinyOS Programming*. New York.
- [8] MEMSIC. (dátum nélk.). *MTS/MDA Sensor Board USER MANUAL*. Forrás: <http://www.memsic.com>
- [9] Honeywell. *1- and 2-Axis Magnetic Sensors*. Forrás: <http://honeywell.com>
- [10] Freescale. (dátum nélk.). *Mag3110*. Forrás: <http://freescale.com>
- [11] MathWorks, <http://matlab.com>
- [12] Zusheng Zhang, X. L. (dátum nélk.). *Hindawi*. Forrás: <http://www.hindawi.com/journals/ijdsn/2013/107975/>.
- [13] WanQiaoHong. (dátum nélk.). *Magnetic Sensor Intelligent Car Parking Guidance System For Indoor , Airports and Hotels*. Forrás: [http://www.szwqh.cn/en/e\\_products\\_show/?id=112](http://www.szwqh.cn/en/e_products_show/?id=112)
- [14] Magnetic Autocontrol Group. (dátum nélk.). *Magnetic*. Forrás: <http://www.ac-magnetic.com/>
- [15] Streetline. (dátum nélk.). *Streetline parking sensor*. Forrás: <http://www.streetline.com/parking-analytics/parking-sensors-mesh-network/>