Gulyás Péter

Inerciális navigáció egy modelljárművön

Széchenyi István Egyetem

Villamosmérnöki BSc – Automatizálási Szakirány

2014/15

Konzulens: Dr. Ballagi Áron, tanszékvezető

Győr, 2018

Absztrakt

A TMDK dolgozat fő témája egy készülő autonóm járműben használt inerciális navigációs rendszer kialakítása és mérése. Ennek a rendszernek a segítségével pozíció meghatározást végezhetünk és a jármű által megtett útvonalról kaphatunk információt. Ennek összeállításához, felhasználok egy odometria rendszert, amit pontosítani tudok egy kilenc szabadságfokú MEMS szenzorral. Az inerciamérő szenzor a BNO055, melyet a Bosch már szenzorfúziós algoritmusokkal együtt szállít. Ennek köszönhetően a beépített gyorsulásmérők, giroszkópok, és magnetométer alapján már egy szűrt adathalmazzal kell dolgoznom, valamint relatív orientációt is kinyerhetek. A szenzorral kapcsolatban végeztem már korábban kalibrációs méréseket, de most egy fuzionált rendszer részeként végzem a méréseket, ahol valójában az odometria pontosítását végzi. A rendszer számítástechnikai alapja egy Raspberry Pi 3 mikroszámítógép és egy azon futó Robot Operating System, mely egy Linuxra épülő middleware. Az ROS segítségével valósítom meg az eszközök integrációját szoftveres szinten, valamint 8 bites mikrokontrollereket használok az odometria, és az inerciamérő kommunikációjának megvalósításában. A méréseket egy 1:6 méretarányú járműmodellen végzem, mely kinematikai modelljét tekintve egyezik a SZEnergy Team versenyautójával. Ez a csapat a Széchenyi István Egyetem elektromos járművek fejlesztésével és építésével foglalkozó hallgatói csapata, ahol távlati célunk, hogy a Shell Eco-marathon új -önvezető- kategóriájában induljunk. A tervek szerint a koncepció autónk önvezetést segítő rendszerében a 2019-es versenyen már egy inerciamérő szenzor is rendelkezésünkre áll a navigáció megvalósításához. Az előzetes algoritmus tesztelések, lehetőséget biztosítanak arra, hogy az elkészülő autót, rövid időn belül felvértezhessük a szükséges eszközökkel, és szoftverrel, annak érdekében, hogy az önvezető feladatokat is el tudja végezni a verseny során.

Tartalom

Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani témavezetőmnek, Dr. Ballagi Áronnak, aki szakértelmével, szemléletes magyarázataival és hasznos konzultációival segítséget nyújtott dolgozatom elkészítéséhez.

Köszönöm a segítséget Szeli Zoltánnak, aki a mérések és a modelljármű előkészítésénél a teljesítményelektronika összeállításában segítséget nyújtott, valamint Hajdu Csabának, aki a C++ programozásában volt nagy segítségemre a Robot Operációs Rendszer kapcsán.

Hálával tartozom továbbá szüleimnek és testvéremnek, hogy tanulmányaim során türelemmel és megértéssel támogattak, és minden helyzetben mellettem álltak.

A TMDK feladat elkészítését a GINOP-2.3.4-15-2016-00003 Felsőoktatási és Ipari Együttműködési Központ a Széchenyi István Egyetemen projekt támogatta.

1. Bevezetés

1.1. A fejlesztés célja

Elsődleges célom, egy olyan navigációs eszköz rendszerillesztése, mely lehetővé teszi a pontos navigációt GPS használata nélkül. Az önvezetés témakörében ez egy rendkívül fontos kérdés, mert a GPS jelek földalatti, vagy alagútban történő vétele a rálátás hiányából következően lehetetlen. Ennek ma már vannak részleges megoldásai, mert más szenzorok segítségével, valamint offline térképek segítségével közeli becslésekkel kiegészíthetők a navigációs információk, míg éppen alagútban haladunk. A legnagyobb ellenfél a mélygarázsok világa, ahol erre nincs opció a sok, és kis területen végzett manőver miatt. Feltételezve egy nagymértékben önvezetést támogató infrastruktúrát, valamint az „okosparkolást”, a föld alatti navigációra is precíz módon van szükségünk, melyre egy inerciális navigációs rendszer szolgálhat megoldásként.

1.2. Shell Eco-marathon: Autonomous Urban Concept

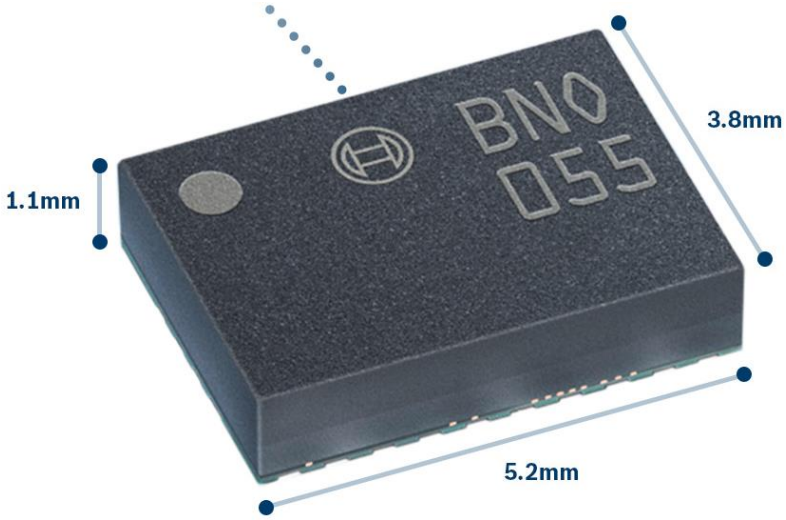
A Shell Eco-marathon története 1939-re nyúlik vissza az Egyesült Államokba, amikor a Shell kutatói fogadást kötöttek, hogy ki tud több utat megtenni egy gallon benzinnel. Akkoriban a legjobb eredmény 4,73 L / 100 km-re jött ki, ami ma már egy jól hangolt kis városi benzines autóval is kivitelezhető. A versenyt azóta is megrendezik, és ma már az egyetemi „kis” kutatók dolgoznak, fejlesztenek és építenek több kategóriában is, hogy elnyerjék a legenergiahatékonyabb címet az évben. Ezen verseny sorozat egy új kategóriája indult el az idei évtől, melyben az önvezetés, és a hozzá kapcsolódó feladatok megvalósítása a cél. Rendkívül bonyolult és összetett rendszerekre, valamint nagy teljesítményű számítógépekre van ehhez szükség, ezért nem csak embert, hanem csapatot próbáló versenyszámról van szó. Ebben a kategóriában kíván indulni a Széchenyi István Egyetemen működő SZEnergy Team fejlesztő csapatunk, ahol a navigációs rendszerünk alapja egy összetett inerciamérő chip lesz, mely nagyban segítheti majd munkánkat, és sikerre vezetheti a csapatot.

2. A fejlesztés elméleti háttere

2.1. Inerciális navigációs rendszerek

Inerciális navigációs rendszernek hívjuk azokat az útmutatást segítő rendszereket, melyek inercia, azaz tehetetlenség mérésen alapuló eszközök segítségével valósítanak meg. [1] Erre a célra manapság Micro Electromechanical System (MEMS), azaz micro-elektromechanikus rendszereket használnak. Ezt elképzelni legkönnyebben egy mikrochipként lehet. Apró szilícium kristályon megvalósított integrált áramköri elemek ezek.

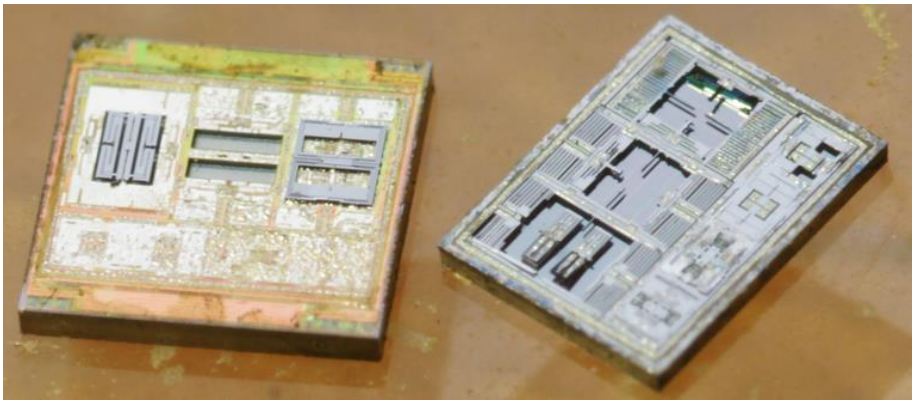
Az inerciális navigációs rendszereket a rendszer szabadságfokával szokás jellemezni. A szabadságfok alatt azt értjük, hogy a különböző tengelyeken végzett mérések mennyire függenek egymástól. Annál pontosabb egy szenzorrendszer minél függetlenebb méréseket vagyunk képesek végezni. Jelen esetben a dolgozatban használt BNO055-ös szenzor (1. ábra) minden mérést egyéni tengelyen végez, tehát a gyorsulásmérő, a giroszkóp és a magnetométer együttesen, egy 9 szabadságfokú szenzorcsomópontot valósít meg. A több szabadságfokú micro-elektromechanikus rendszerek belső felépítése rendkívül bonyolult. Egy ilyen eszköz felhasználásával megvalósítható az úgynevezett infrastruktúra független navigáció, melynek nincs szüksége GPS/GNSS jelekre, a lokalizációhoz. Ennek következtében a járművek sokkal rugalmasabbak és olcsóbbak lehetnek a jövőben, mert csak a fedélzeti eszközeikre és belső jelfeldolgozásukra fognak támaszkodni.



1. ábra. A Bosch BNO055 chip és annak fizikai méretei. [A1]

2.2.A gyorsulásmérő

A gyorsulásmérő felépítését tekintve egy rezgésbe hozható tömegből áll, mely gyorsulás hatására ellenállás- vagy kapacitás változást produkál. Ennek mérésével határozható meg a pillanatnyi gyorsulás. Ennek időszerinti integrálja pedig megadja a sebességet. A 2. ábrán látható egy 3 tengelyes gyorsulásmérő MEMS kialakítása.



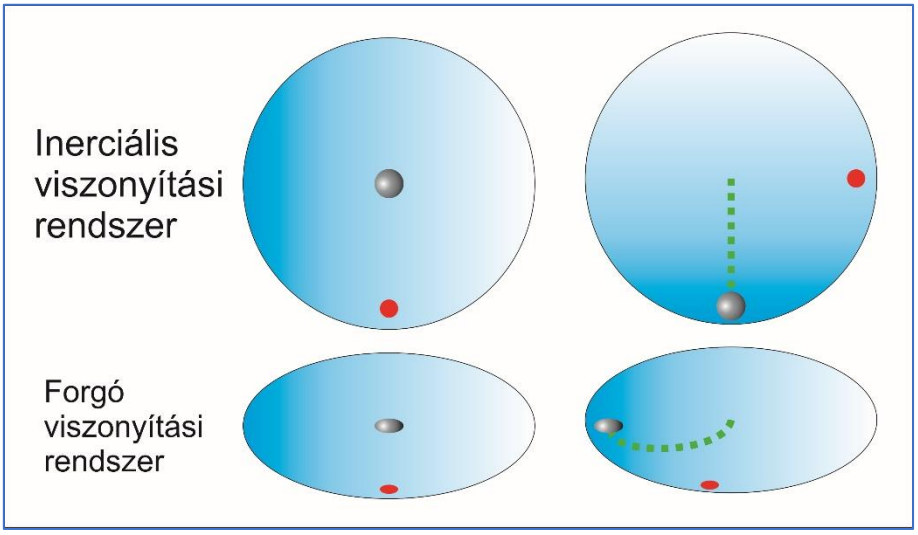
2. ábra. Egy összetett MEMS szenzor metszete. [A2]

2.3.A giroszkóp

A giroszkóp a perdületmegmaradás törvényét demonstrálja. A Newton-törvények egy test mozgását inerciarendszerben vizsgálják, melyről tudjuk, hogy nyugalomban van, vagy egyenes vonalú egyenletes mozgást végez. Amikor a Newton-törvényeket transzformáljuk egy egyenletesen, $\boldsymbol{\omega}$ szögsebességgel forgó vonatkoztatási rendszerbe, megjelenik a Coriolis-erő és a centrifugális erő. A Coriolis-erő egy ilyen forgó koordinátarendszerben ható tehetetlenségi erő, mely abból adódik, hogy egy **v** sebességgel mozgó **m** tömegű testre plusz gyorsulás hat a forgás tengelyére merőleges mozgás esetén. (3. ábra) Ezt az erőt az alábbi egyenletből számíthatjuk.

$$F_c = 2 \cdot m \cdot \mathbf{v} \cdot \boldsymbol{\omega} \tag{1}$$

Egy micro-elektromechanikus szenzor esetén a giroszkóp megvalósítására egy rezgő rendszer van megvalósítva a szilícium lapkán, amelyre annak forgatása közben Coriolis-erő hat [2, 3]. Ebben az esetben is a hatást ellenállás vagy kapacitás változáson keresztül mérhetjük.



3. ábra. A Coriolis-hatás illusztrációja.

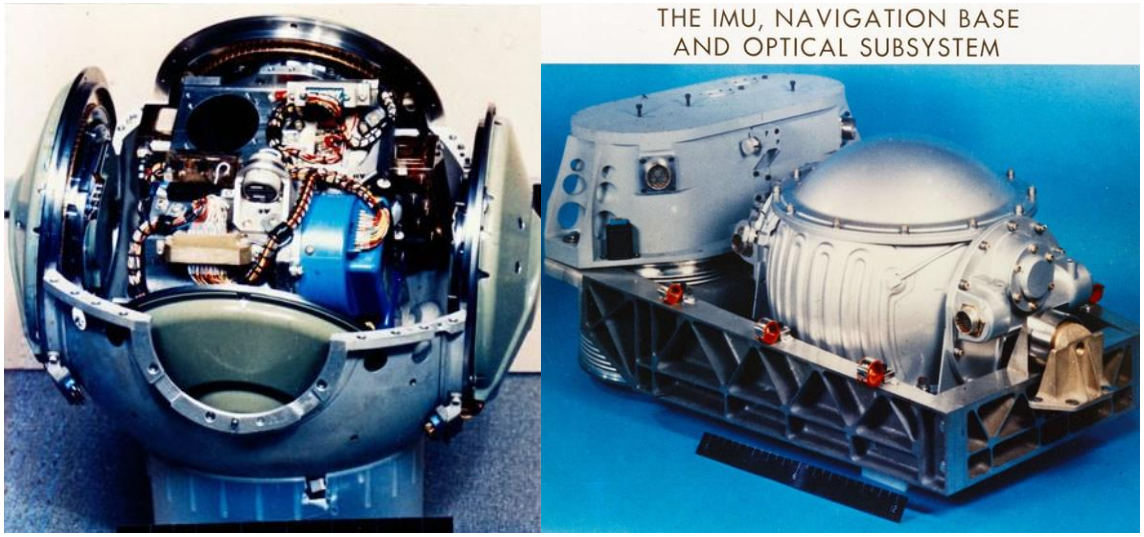
2.4. A magnetométer

A magnetométer a föld mágneses terét mérve segít az orientáció pontosabb meghatározásában. Ezen mérések is három tengelyen zajlanak, annak érdekében, hogy a chip bármilyen pozícióba kerül a mozgás során, képes legyen pontosan irányultságot meghatározni a föld északi pontjához képest. A különböző szenzor típusok ezen értékek alapján számolják az abszolút vagy relatív orientációjukat.

MEMS eszközöknél kulcsfontosságú a kalibráció, főleg a magnetométer kapcsán, hiszen az elektronikai eszközök elektromágneses kisugárzása is megzavarhatja annak működését. Ofszet eltolódás, mágneses tér torzulás, ortogonális hibák és zaj, ezek mind-mind negatívan ható effektek a működést tekintve [4]. A gyártó ajánlására fontos, hogy a kalibrációt olyan környezetben végezzük el, ahol később az üzemelni fog. Ezzel elkerülhető, hogy egy elektromágnesesen tiszta környezetben kalibrált szenzor, egy erősáramú környezetben hibás adatokat szolgáltasson majd. Természetesen figyelni kell a szenzor rendszer olyasfajta kialakítására, hogy minél kevesebb zavart gerjesszünk körülötte, és ezután az adott környezetben bekalibrálni. A helyi hibalehetőségeket a kalibráció során a szenzorfúziós algoritmus figyelembe veszi, így az adott környezethez lesz illesztve a magnetométerünk.

2.5. Történelmi visszatekintés: Apollo 8

Az 1961-ben kezdődött Apollo projekt az „űrverseny” kiemelkedő fejlesztését indította el, amit a benne dolgozók ma is úgy emlegetnek, mint a legnagyobb előrelépés a számítástechnika és az elektronika világában. A világűrbe feljutni önmagában nagy feladat, de ott manőverezni és hazatérni a Földre sokkal nehezebb munka.
Habár a csillagok szerint lehet tájékozódni, de ezt egy szűk kapszulában rövid idő alatt precízen nem lehet megvalósítani. Ennek okán kezdődött meg az inerciamérőkre épített navigációs rendszer fejlesztése a NASA-nál is [5].



4. ábra. balra: Az Apollo 8 projekt során használt inerciamérő egység központi egysége. jobbra: A beszerelésre került komplett navigációs egység, az optikai stabilizálóval. [A3]

Az Apollo 8 repülésének idejére már annyira kiforrta magát ez a mérési összeállítás (4. ábra), hogy a jegyzőkönyvek szerint, pontosabb volt a szenzor, mint amire szükség lett volna a fellövések alkalmával. Ezt bizonyítják az adatnaplózás kiértékelései is, ahol alább (5. ábra) látható, hogy a 147 órás repülés alatt a szenzorrendszer gyorsulásmérőinek legnagyobb ofszet hibája 0,02 cm/s^2 volt. A BNO055-ös szenzor esetében nincs is lehetőség a cm/s^2-es mértékegységre. Ennek oka maga a gyártástechnológia. A MEMS szenzorok ugyan nem képesek ilyen pontos mérésekre, de valójában erre nincs szükség a nagytömegű felhasználás körében, ellenben ezek a szenzorok rendkívül költséghatékonyan előállíthatók, nagy mennyiségben gyorsan gyárthatók.

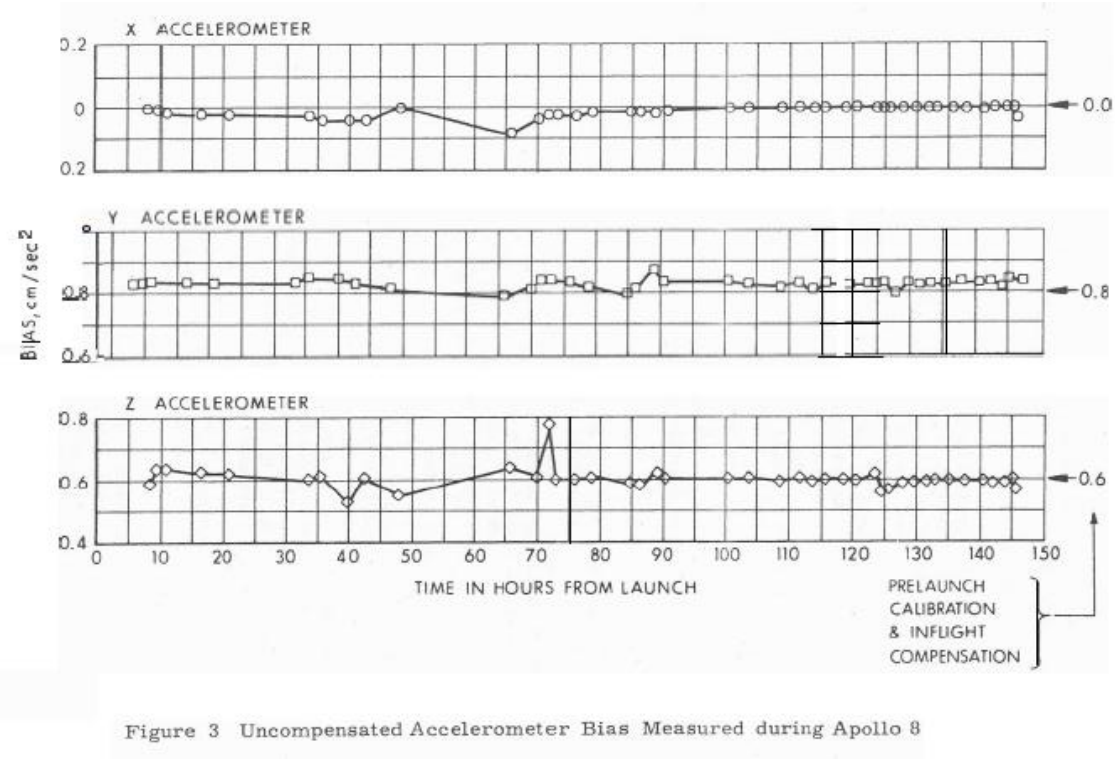


Figure 3 Uncompensated Accelerometer Bias Measured during Apollo 8

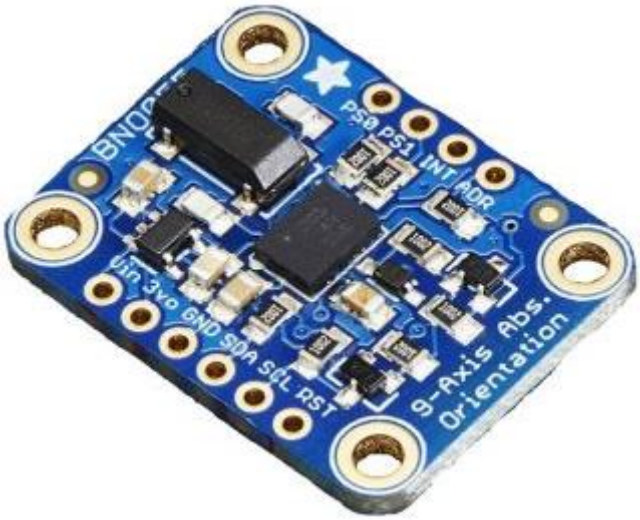
5. ábra. Apollo 8 repülési jegyzőkönyv részlet a gyorsulásmérők ofszet hibájáról. [A4]

3. A felhasznált szenzor: Bosch BNO055

A fenti típusnév egy Bosch által gyártott okos-szenzort, Applikáció Specifikus Szenzor Csomópontot takar (ASSN) , mely képes I^2C és UART kommunikációra is [6]. Ez az eszköz egy 28 lábú System in Package (SiP), ami tartalmaz:

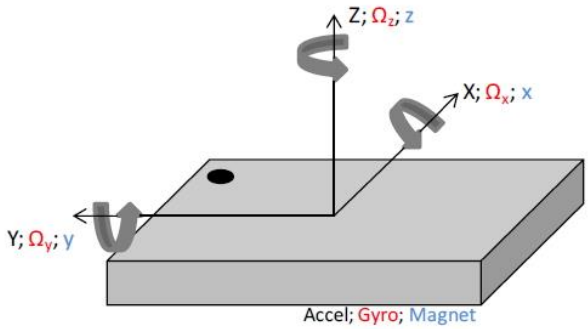
- háromtengelyes 14 bites gyorsulásmérőt, (m/s^2)
- háromtengelyes 16 bites giroszkópot, (deg/s)
- háromtengelyes magnetométert, (µT)
- 32 bites cortex M0+ mikrokontrollert a Bosch Sensortec fúziós szoftverrel.

A két éve piacra került ASSN különleges tulajdonsága, hogy nem csak gyors belső adatfeldolgozással rendelkezik, hanem a hozzá írt szenzorfúziós szoftver is elvégez olyan adatfeldolgozási feladatokat, melyre a kiolvasást követően már nem kell számítási kapacitást pazarolni a rendszerben. A chip egy Adafruit által gyártott lapkán (6. ábra) érkezett, mellyel I^2C kommunikációs protokollon keresztül kommunikáltam a feladatok elvégzéséhez.



6. ábra. Az Adafruit BNO055-ös lapkája. [A5]

A BNO055 chip esetén a mértékegységek programozhatók, melyre egy külön regiszter szolgál, ennek részletes leírása az 1. számú mellékletben található. Ennek a regiszternek az alapbeállítása megegyezik a fentebb említett mértékegységekkel, és ezt megfelelőnek találtam a projekthez, valamint az orientációs értékek alapbeállítása is a fok, így nem írtam felül ezen regisztert. A szenzor a beépítés variálhatóságát elősegítve tartalmaz egy külön regisztert, mely a tengelyek konfigurálására szolgál. (7. ábra) A mérésekhez én a gyári értékeket használtam, de így az X-tengelyen utólag az értékek inverzét kellett képezni, így ennek a regiszternek a felülírása mindenképp fontos lesz a későbbiekben.



7. ábra: A szenzor tengelyeinek gyári beállítása. [A6]

Az ASSN legnagyobb előnye korábbi versenytársaival szemben, hogy szenzorfúziót alkalmaz már a tokozáson belül. Külső adatfeldolgozás esetén ez a folyamat sokkal több időt és számítási kapacitást emésztene fel, valamint a mérési pontosság romlását eredményezné.

Szoftverében több üzemmód választható, melyek lehetőséget biztosítanak az önálló szenzorok használatára és fúziós használatra is.
Én az NDOF (Nine Degrees of Freedom) funkciót választottam, mely kilenc szabadságfokos vizsgálatokra ad lehetőséget, valamint itt működik a beépített szenzorfúziós algoritmus is.

- Abszolút orientáció 3 tengelyen (Heading-Roll-Pitch) 100Hz-es frissítésre képes
- Szögsebesség/Gyorsulás 3 tengelyen 100Hz-es frissítésre képes
- Mágneses térerősség 3 tengelyen 20Hz-es frissítésre képes

A szenzorfúzió hatására az eszközünk robusztus kimenetet ad, és nagy kimeneti adatsebesség érhető el, melyre nagy szükség van egy valós idejű rendszer kialakításakor. Minél több információ áll rendelkezésre egy másodpercen belül, annál jobban kiküszöbölhető a hibás mérés, és annál jobban pontosítható a lokalizáció.
A dolgozat és egyben a projekt célja az eszköz pontos kalibrálása, és beüzemelése, annak érdekében, hogy az önvezető modellünk számára biztosíthassuk a pontos helyzetmeghatározás eszközét beltérben is, ahol a GPS jel erőssége a kívánt pontossághoz nem elegendő. Ennek a későbbiekben is, egy valós méretű autóban is nagy szerepe lesz, például a már említett parkolóházakban és alagutakban.

4. Fejlesztési lépések

4.1. Háttér rendszer inicializálás

4.1.1. Ubuntu Linux

A fejlesztés alapja egy moduláris rendszer felépítése, melynek alapvető kritériumai:

- Real-Time közeli működés:
  Annak érdekében, hogy gyors reakciója legyen a később összeépítésre kerülő járművünknek, és értelmezni tudja a másodpercek alatt történő változásokat, szükséges a valós idejű feldolgozó képesség.
- Megbízható operáció:
  Nem megengedhető, hogy egy önvezető autó, bármely funkciójának megvalósítása során, olyan állapotba kerüljön, melyben megragad, és nem képes folytatni a küldetését.
- Magas fokú rendszerbiztonság:
  Fontos, hogy a rendszeren belül működő részegységeink, csak azokhoz a perifériákhoz és rendszer eszközökhöz férjenek hozzá, amelyekkel szorosan együttműködnek.

- Minimális memória igény

  Az óriási számítási kapacitás és a Real-Time működés kapcsán fontos kritérium, egy olyan operációs rendszer kiválasztása, mely nagyon kevés memóriát használ fel a rendszermemóriából, saját működésének fenntartására.

- Nyílt forráskódú:

  A teljes fejlesztést nyílt forrásokból, és minél alacsonyabb költségekkel kívánjuk megvalósítani. Ennek érdekében, ahol lehetőségünk van, nem használunk olyan szoftvereket, melyeket meg kell vásárolni.

Ezen fenti paramétereknek megfelelően egy Linux disztribúció lett kiválasztva, mely 2021-ig hosszútávon nyújt támogatást a ráfejlesztett rendszerünk számára. Egy Windows rendszerhez képest, már a követelmények is 50%-kal kevesebb memóriát írnak elő egy rendszer működtetéséhez a Linux javára. Ez annak köszönhető, hogy a Linux alatt működő grafikus megjelenítő jóval egyszerűbb és kevesebb felhasználói effekttel dolgozik, valamint a rendszer mögött működő dinamikus könyvtárakkal lehetőség nyílik arra, hogy több program egyszerre férjen hozzá megosztott könyvtárakhoz, ezzel kiküszöbölve azt, hogy minden program betöltse a teljes saját könyvtárát használat közben [7].

A rendszerfrissítések teljes mértékben a felhasználók által kontrollálhatók, és a rendszeren belüli adat- és eszközbiztonság is magasfokú, mert nagyon szigorú jogosultsági protokollokkal épül fel a rendszer. A fejlesztett szoftverek jogosultságai egyenként is könnyen definiálhatók, így kialakítható egy magas biztonsági fokú rendszer.

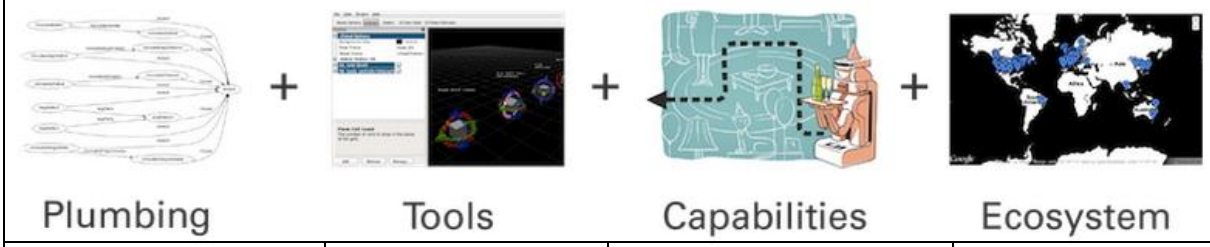
A választott operációs rendszer az Ubuntu Linux 16.04 LTS (Long Term Support). A rendszer előnye, hogy a fejlesztési időszak alatt grafikus rendszerként működtethetjük azt, majd, amikor beépítésre kerül a versenyautóba, akkor ezt könnyen eltávolíthatjuk, ezzel nagy mennyiségű tárhelyet, memóriát és számítási kapacitást szabadítunk fel a rendszerünkben. Parancssori működése rendkívül felhasználóbarát. Már a fejlesztés közben is sokat dolgoztam benne, így biztos vagyok benne, hogy ez az éles fedélzeti üzem alatt, később sem fog problémát okozni.

4.2. ROS Kinetic Kame

Az ROS rövidítés a Robot Operating System-et takarja, mely egy meta-operációs rendszer, elsősorban robotokhoz fejlesztve [8]. 2007 óta van jelen a robotika világában az ROS keretrendszere, és egyre növekvő tábora még jobban felgyorsítja a nyílt forráskódú rendszer terjedését. Felépítése nagyban hasonlít az Unix rendszerekhez, és ezért megköveteli egy Linux

disztribúció alapkörnyezetét. A szolgáltatásai lefedik, amit egy operációs rendszertől elvárunk, de egy másik operációs rendszerre épül, ami esetünkben Ubuntu Linux 16.04 LTS. Fontosnak tartottuk a projekt során egy olyan bázis rendszer kiválasztását mely minél tovább támogatást élvez, ezért nem a legfrissebb verziót választottuk az ROS rendszerek közül sem, hanem a 'Kinetic Kame' verziót, mely szintén 2021-ig biztosít támogatást, és ezzel garantálja a fejlesztések lehetőségét a további pár évben. Alább látható (1. táblázat) a rendszer áttekintése és funkciói.

1. táblázat. Az ROS áttekintése.



Folyamat menedzsment	Szimuláció	Szabályzás	Csomag rendszer
Kommunikáció a folyamatok közt	Vizualizáció	Tervezés	Szoftver disztribúció
Eszköz driver-ek	GUI	Érzékelés - Megfigyelés	Dokumentáció
	Naplózás	Térképépítés	Példák - Bemutatók
		Adat kezelés	

Az ROS biztosítja számunkra az alsó szintű driver kezelést, közvetett rálátást a hardverekre, általánosan használt függvények implementálását, csomag menedzsmentet és nem utolsó sorban a folyamatok közti üzenet közlést. Az ROS futásának folyamatábrája egy Peer-to-Peer (P2P) hálózathoz hasonlítható, ahol nincs kitüntetett szerver, mindenki egyenrangú fél, és így jelenik meg a rendszerben az összes folyamat, akár több számítógép között is egységesen. A kommunikáció során szinkron és aszinkron adatkapcsolatra is képes. A rendszer a Linuxhoz hasonlóan csomagokból épül fel, és ez adja a rendszer egyik legjobb tulajdonságát, mert ettől modulárissá válik, mely fejlesztés szempontjából elsőrendű a jelenlegi projektben. Az ROS a *publish-subscribe* kommunikációs modellt alkalmazza, ahol a programok valójában *node*-okként jelennek meg, és *topic*-okon keresztül megy az üzenetközlés, melyekre bárki feliratkozhat (*subscribe*) illetve bárki publikálhat (*publish*).

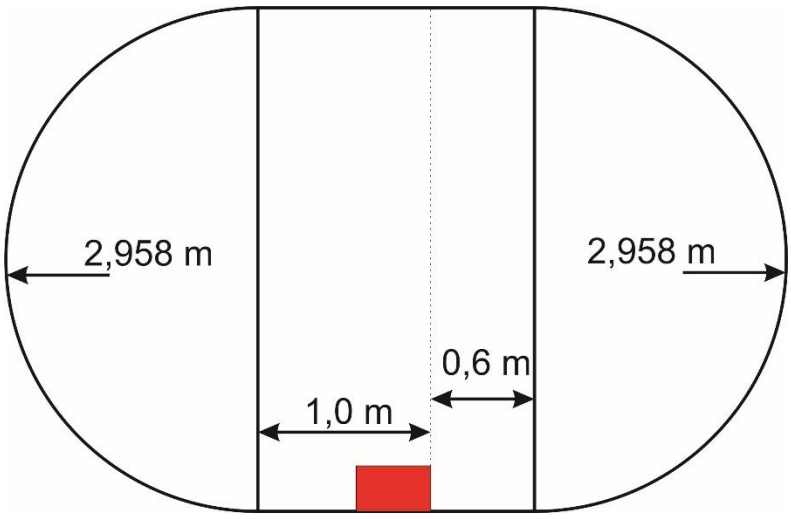
5. Mérési összeállítás

5.1. Vezérlés tesztelése

Mielőtt a navigációs szenzort a modelljárművön alkalmaznám, szükséges volt annak dinamikai problémáit felmérni. Ugyan kinematikailag méretarányosan lett megtervezve, de a modell dinamikai szempontból egyáltalán nem arányos annak eredeti megfelelőjéhez. A hibák feltételezettek, de ezek feltárása szükséges annak érdekében, hogy a későbbi mérések ne csak önmagukban feleljenek a hibákért. Fontos megjegyezni az alábbi rendszer problémákat:

- Nincs Ackermann kormányzás
- Tömegét tekintve a modell nem arányos
- Egy oldalon hajt a jármű
- Analóg kormányszervó visszajelzés egész értékek formájában

A modellautó dinamikájáról többet megtudhattam, ha egy idő alapú programot írva, teszteltem annak viselkedését, egy előre definiált mozgással. Ahogy az az ipari robotoknál is alapfeltétel, és a korábbi dolgozat mérései közben is meghatározó volt, az egyik alapfeladat, hogy egy valamilyen módon definiált útvonal után a robot, vagy jármű visszatérjen a kiinduló pontba. Az ipari robotok bemérése során egy ISO-kockában szokás elvégezni ezen pozíció és ismétlési méréseket [9].
Számomra adottak voltak a szakkörben formálható pályaelemek, amelyekből két egyenesből és két félkörívből álló szakaszt valósítottam meg (8. ábra), melyben az autónak vissza kell érnie a kiindulópontjába.

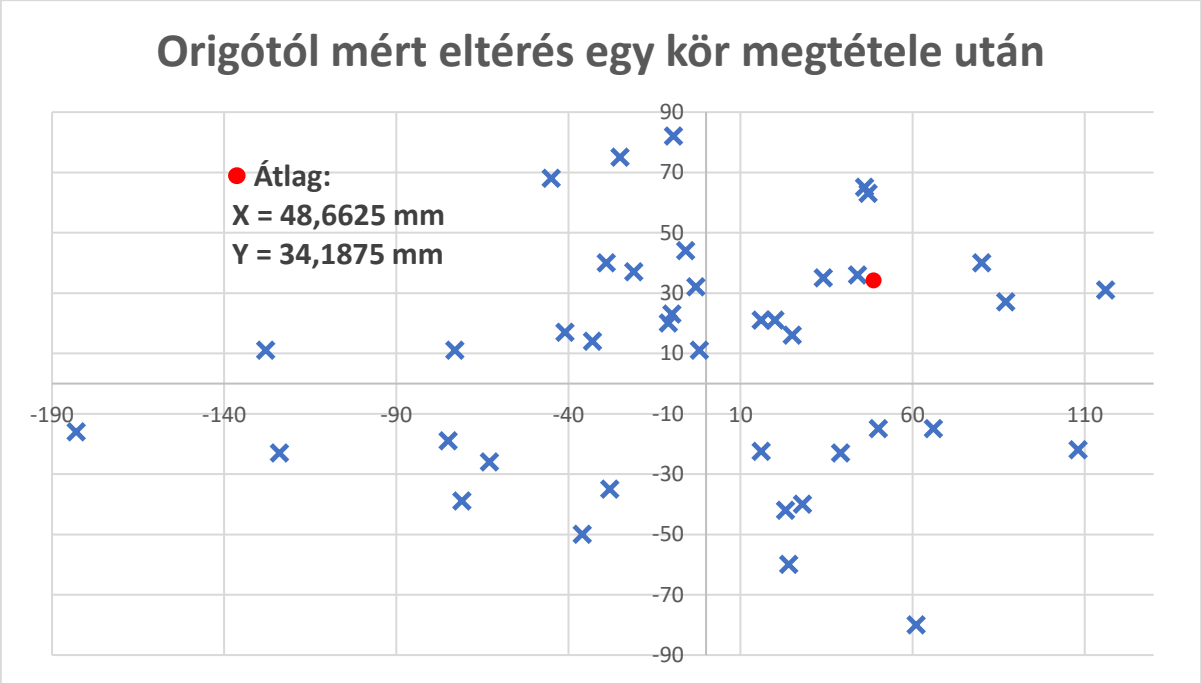


8. ábra. A megtenni kívánt tesztkör.

A vezérlést Python nyelven írtam, melyben soros kommunikációval küldök, megadott időközönként vezérlő jeleket a motor- és szervovezérlőnek, ezzel megvalósítva a pálya útvonalát. A soros kommunikáció, valamint a várakozásra használt időfüggvény először kimérésre kerültek, melyekből megállapítottam, hogy a függvények alkalmasak egy hiteles mérés kivitelezésére. Alább látható a kód futása során mért rendszeridő, valamint a különbségekből adódó végrehajtási idő is.

```
##3219.429703301   Time before serial_write
##3219.430038246   Time after serial_write, before sleep(1)
##  serial_write = 0.334945 ms
##3220.431092828   Time after sleep(1), before serial_write
##  sleep(1)     = 1001.054582 ms
```

Látható, hogy a soros írás alkalmával kevesebb, mint 1 ezredmásodpercre van szüksége a rendszernek, valamint a várakozás is 1 ezredmásodperc pontosságú másodpercenként. A teljes teszt kód a 3. számú mellékletben található meg. A tesztkódban több vezérlő jel is megtalálható egy adott feladatra, ennek oka, hogy az egy oldalon hajtott rendszer már egy ilyen kis tesztkör esetén is nagy oldalcsúszást generál. Így, a párhuzamosságra figyelemmel, kissé módosított értékek szükségesek az elviekben szimmetrikus tesztkör futásához mindkét egyenes szakaszon. A tesztet állandó feszültség mellett végeztem egy, az akkumulátor és a motorvezérlő közé kötött buck konverter segítségével, annak érdekében, hogy a merülő akkumulátor okozta feszültség esés a motoron ne jelentkezzen, ezzel kiküszöbölve a mérési eltéréseket negyven kör alatt.



9. ábra. Vezérlés mérésének eredményei.

A vezérlő kód módosítása szükséges volt annak érdekében, hogy aktív szabályzás nélkül megtehessen a modell egy egésznek tekinthető „kört”, mely az inerciamérő szenzor vizsgálatához megfelelő alapot ad, és elvárható értékekkel összehasonlítható legyen a szenzor naplózás eredménye. A grafikonon (9.ábra) látható, átlagosan **0,04866 m** -es eltérés, a nyílt hurkú vezérlésben teljesített tesztkörre vetítve **0,533 %** -os pozíció hibát eredményezett.

5.2. Szenzor adatok elemzése

5.2.1. ROS – Odometria

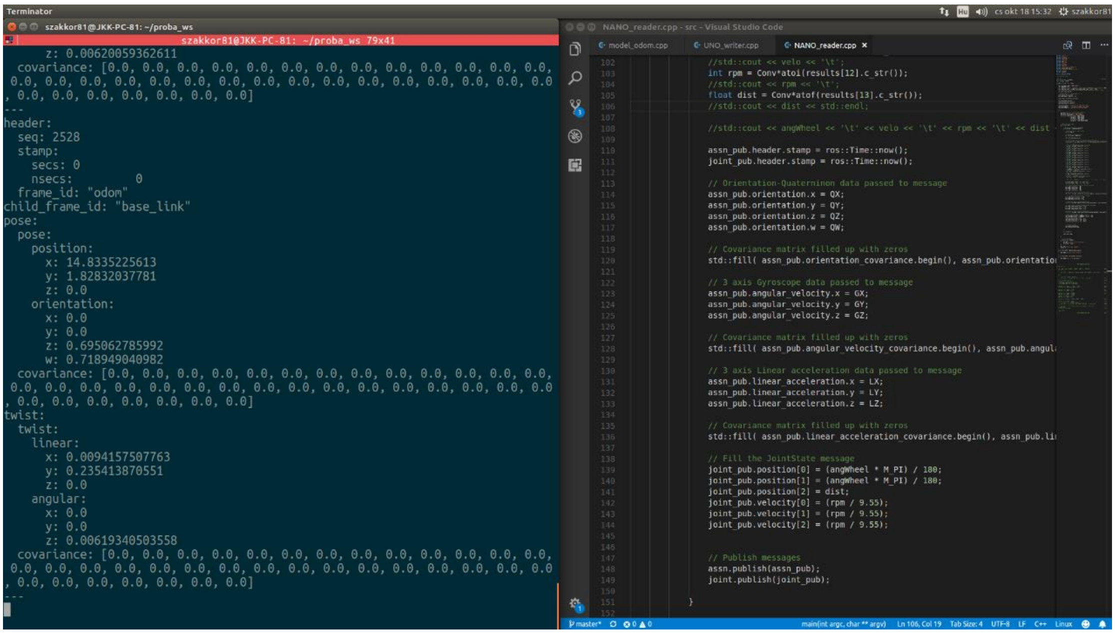
Az odometria az enkóder adatokból, és a kormányszögből hivatott pozíció meghatározást végezni. Erre a feladatra több *node* került megírásra az ROS-en belül, melyek egyesével kezelik a szenzor olvasását, a mikrokontrollerek vezérlését és a számítási műveleteket. Jelenleg még bonyolult a rendszer, és minden egyesével indítandó, de később ez összeintegrálható lesz, annak érdekében, hogy gördülékenyen lehessen méréseket végezni, valamint az önvezető funkciók számára is könnyen elérhető legyen a szoftvercsomag. A soros kommunikáció felépítésére a POSIX függvényeket [10], és egy elérhető „SerialPort” header fájlt használtam. A program futtatásához szükséges csomagok:

- UNO_writing
- NANO_reading
- model_odom

A program működése közben a NANO_reading csomag valósítja meg a mikrokontrollerek olvasását, és standardizált üzenetekbe rendezi őket, majd publikálja azokat.

- JointState üzenet típus
- IMU üzenet típus

Ezt követően a model_odom csomag elvégzi a szükséges számításokat, és amennyiben megtekintjük az aktuálisan publikált üzenetet (10. ábra), abból látható a jármű aktuális pozíció és orientáció értéke X és Y tengely szerint.



10. ábra. ROS „odometry" üzenet működés közben.

Az visszatéréses vizsgálatokat követően, ugyan azon kód használatával, de már soros kommunikációra épített olvasással együtt végeztem el az útvonal méréseket, amelyek során folyamatosan adatnaplózást végeztem. Ennek során rögzítésre került a mikrokontroller rendszerideje, ami időbélyegként szolgált az adatfeldolgozás során. Ezt követően az inerciamérő orientációs adatait mentettem ki, valamint a lineáris gyorsulás értékét. Ezen szenzor értékek mind átestek a Bosch által írt belső feldolgozáson és szenzorfúzión, de további szűrést nem tartalmaznak. Ezt követően kiolvasásra került a kormányszervó állapotának értéke, majd az enkóderből számolt sebesség, fordulatszám, és a megtett út. Az adattípusok feldolgozása során az alább látható 2. táblázat szerint kerültek felvételre a különböző adatok.

2. táblázat. Feldolgozott adatok típusa és felbontása.

Név	Típus	Mértékegység	Felbontás
Időbélyeg	Egész	ms	
Orientáció (X-Y-Z)	Tört	deg	4 tizedes pontossággal
Lineáris gyorsulás (X-Y-Z)	Tört	m/s^2	2 tizedes pontossággal
Kormány szög	Egész	deg	[-10° ; 10°]
Motor kimeneti tengely sebesség	Tört	rad/s	2 tizedes pontossággal
Fordulatszám	Egész	rpm	
Megtett távolság	Tört	m	2 tizedes pontossággal

A szenzoros mérések során 12 alkalommal egy-egy kört tettem meg a modellautóval, melyek végén minden alkalommal újra pozícionáltam az autót a helyes kiindulási helyzetbe, mert a vezérlés pontatlanságából adódóan nem volt tökéletes a visszaérkező modellautó helyzete.

5.2.2. Orientáció - Heading

A szenzoros adatfeldolgozás során az elsődlegesen vizsgált elemem az orientáció volt. Egy kör megtétele 360° változást kell, hogy adjon. A félkörívek elvárt változása 180°. Az orientáció pontatlansága az ismételt mérések együttes összehasonlításával, a 3. táblázatban látható.

3. táblázat. Összesített orientációs hibák.

Eltérés	Átlag	Minimum	Maximum	Szórás
Egyenes - 1	0,99°	0,63°	1,38°	0,75°
Kanyar -1	2,05°	0,31°	3,69°	3,38°
Egyenes - 2	1,89°	1,44°	2,19°	0,75°
Kanyar - 2	2,49°	0,25°	5,19°	4,94°
Egyenes - 3	0,47°	0,19°	0,75°	0,56°
Kezdő - végpont	5,54°	1,31°	10,00°	8,69°
	Összesített átlag	Min. átlag	Max. átlag	Szórás átlag
	1,58°	0,19°	5,19°	5,00°

Az értékekből jól látható, hogy az egyes műveletek elvégzése még **1 %**-os hibahatáron belül mozog, de a kezdő és végpontok között már **1,538 %**-ra nő. A teljes rendszert vizsgálva az átlagolt eltérés 1,58 fok, ami **0,438 %**-nak felel meg. Amennyiben figyelembe veszem a vezérlés **0,533 %** hibáját, annak tekintetében a BNO055 jól teljesített. A hiba oka visszavezethető mind a vezérlés pontatlanságára, mind pedig a jármű dinamikai anomáliáinak következményeire. Az alábbi táblázatból az is jól kitűnik, hogy a kanyarodás esetében jóval nagyobb eltéréseket mérünk, mely kiemeli a jármű kormányzási elégtelenségét.

5.2.3. Orientáció – Roll - Pitch

Az orientációs vizsgálatok további két paramétere a dőlés és billenés, angol kifejezéssel „roll” és „pitch”. Ezen értékek ilyen lassú sebességnél, és a modelljármű által megtett tesztkörben

elhanyagolhatók, de fontos megvizsgálni ezen tengelyek hibáját is, mert van elvárt értékünk (0°), és ezáltal összehasonlítható a kimenet.

4. táblázat. Dőlés és billenés hibaértékei.

	|Roll|	|Pitch|
Összesített átlag	1,7928°	0,5823°
Minimum átlag	1,1927°	1,4375°
Maximum átlag	2,5885°	0,6042°
Szórás	1,3958°	2,0417°

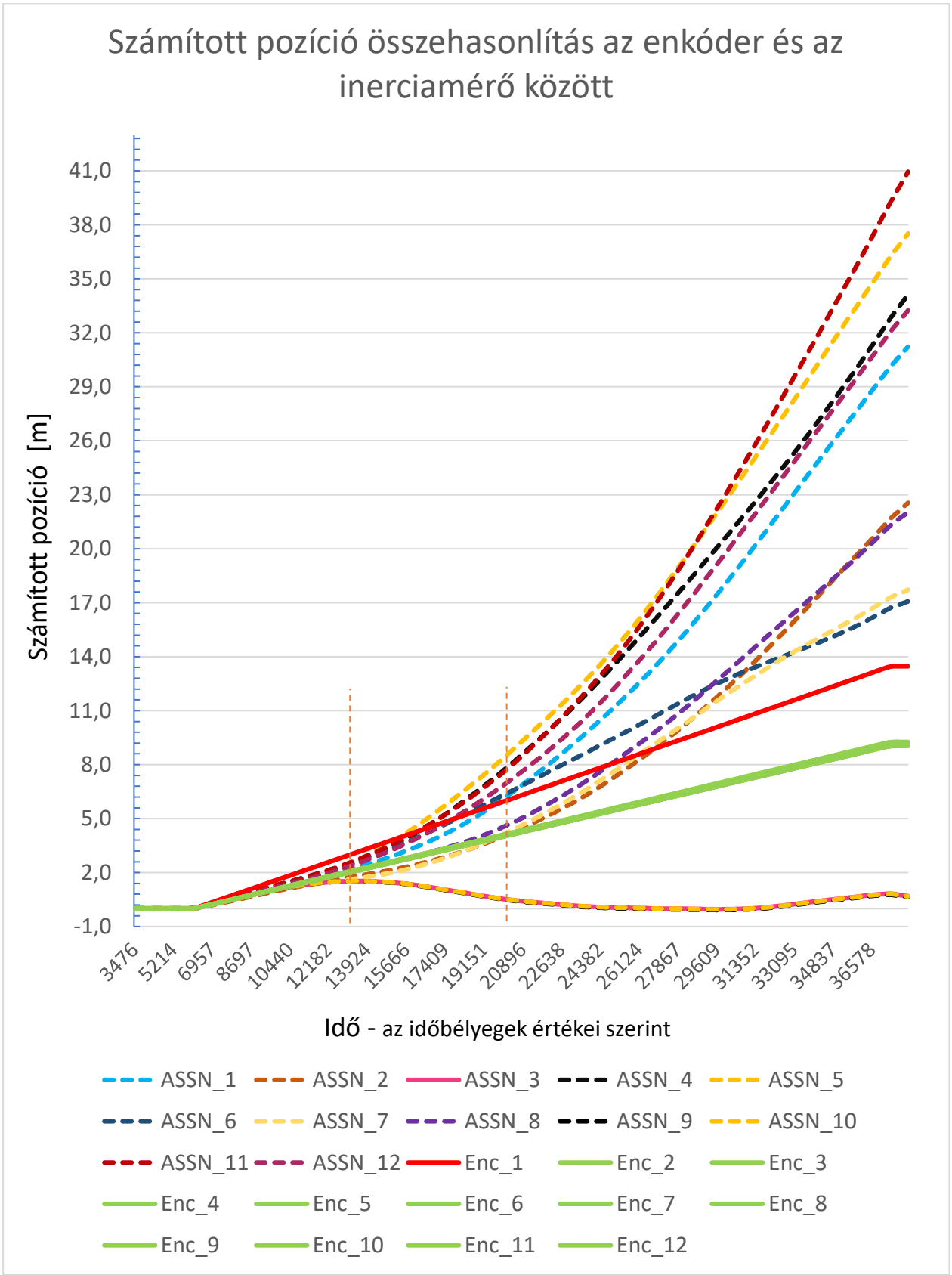
A fenti, 4. táblázatból jól látható, hogy azok a tengelyek melyek szándékosan nem kerülnek mozgatásra a mérés során, minimális eltérést mutatnak. Roll esetében átlagosan **0,996 %**-os a hiba, míg Pitch esetén ez csak **0,3235 %**. Ebből leszűrhető, hogy a szenzor **1 %**-os hibahatáron belül pontosan visszaadta a dőlés és billenés értékét.

5.2.4. Számított pozíció

A másodlagos vizsgálati pont, a már jelen lévő gyorsulás adatokból a sebesség majd a pozíció értékek meghatározása. Ennek hibája előre várható volt, de mértéke és szórása kíváncsiságra adott okot, hiszen a BNO055-ös chip belső szenzorfúziójáról nem tudni pontos adatokat, illetve nem ismertek az algoritmusok, amiket használ. A számított távolságot a gyorsulás, majd a sebesség integrálásával számítottam. Ehhez a művelethez a Newton-Cotes formulák közül a trapéz formulát [11] használtam fel:

$$\int_a^b f(x)dx \approx \frac{b-a}{2}\left(f(a)+f(b)\right) \tag{2}$$

Az idő teltével a hiba összeadódik, és erőteljes eltérés figyelhető meg a grafikonon (11. ábra) is.



11. ábra. Távolság számítás összehasonlítása.

Ezzel összevetve az enkóder stabil jelét, és az abból kiszámolt távolságot, világosan látszik, hogy a gyorsulásmérőkre támaszkodni önállóan az indulástól számítva körülbelül 8 másodpercig lehet. Az enkóder adataiból számolt út átlagban 9,4915 m, ami 0,3755 m-rel több, mint a teszt kör hossza. Ez a **4,119 %**-os eltérés arra enged következtetni, hogy az egész számú fordulatszám kijelzés pontosítást igényel, és mindenképpen tizedes törtként kell kezelni ezt is a későbbiekben. Három olyan mérés volt melyet érdemes volt időben tovább vizsgálni. Ezek 17 másodpercig tudták jól közelíteni az enkóder útszámítását de ekkor már 0,3 m-rel túllőttek azon, amely azért kritikus, mert ennyit az enkóder számítások is átlagban túllőttek. Végértékük szerint ez a három mérés is kritikusnak bizonyult, mert legalább **87,14 %**-kal haladták meg az enkóderből számított megtett távolságot.

A grafikonon a tizenkét enkóder mérésből egy volt eltérő, ami hibásnak tekinthető ez piros színnel látszik, míg a többi szinte egyként jelenik meg zöld színnel. A mérések közti szórás **0,2222** m volt, eltekintve az egy hibás méréstől, mely **4,03** méteres különbséget adott. Ettől eltérően az integrálást követően az inerciamérőből számított pozíció végső szórása **40,31** m, melyből a negatívba tartó mérések eltávolítása, is csak **23,87** m-re redukálta a szórást. Ezen adatok további szűrést igényelnek, mely után várhatóan jobban közelítik majd az enkóder alapú számításokat.

Összegzés

A vizsgált inerciamérő szenzor további méréseket és szűrést követően alkalmas lesz a navigálás megalapozására egy önvezető járműben, mert orientációs szempontból már jelenleg is gyors és pontos működést mutat. Ennek további finomításával, és ROS integrálásával megfelelhet a GPS nélküli navigációhoz, mely célkitűzésem a SZEnergy jövőben megépülő autójában. Fontosnak tartom az odometria és inerciamérő kettős összehangolását, hogy kiküszöbölhető legyen az az óriási eltérés mely jelenleg felmerült a számított távolságok esetén. A fejlesztése során igyekszem minimalizálni a mérések során felmerülő megoldatlan problémákat, ez azonban további feladatokat jelent a jövőre nézve.

A mérések és a program fejlesztése során számos új területtel ismerkedtem meg, mely mindig rejtett valami új kihívást, amire nem gondoltam volna. A felmerülő hibákat sikerült gyorsan orvosolni, és a mérések, valamint a kiértékelés során, olyan apróságokat felfedezni, mely a munka folytatására ösztönöz, hogy az még pontosabb legyen.

Rengeteget fejlődtem az alábbi témákkal kapcsolatban:

- Soros írás-olvasás szinkronizációja,
- C++ programozási nyelv,
- Járművek kinematikai és dinamikai modellje,
- Odometria számítás,
- Python nyelvű soros kommunikáció, és adatnaplózás,
- Programozás ROS környezetben,
- Dokumentálás.

További célok

Több körös mérések végzése szabályozó algoritmusokkal

A mérések során felmerülő dinamikai problémák javítása és (vagy) kiküszöbölése szabályozás segítségével, annak érdekében, hogy a szenzort több körös mérésekben is tesztelni lehessen. Ez a valósághoz köthető távlati tervek miatt elengedhetetlen, és a modelljármű funkcionalitásához is hozzá tartozik.

Navigáció az autonóm funkciók során

Az autonóm jármű szakkör keretein belül fejlesztett ROS alapú rendszerben, az odometria kiegészítése, valamint pontosítása az inerciális szenzorfúzió segítségével. Célom, hogy a jelenlegi navigáció és telemetria alapjaként szolgáló GPS-t, csak kiegészítő eszközként használja a jövőre tervezett autó, és annak helyében egy ilyen inerciamérő szolgálhasson navigációs alapszenzorként.

Mérési összeállítás fejlesztése

A modelljármű dinamikájának további elemzését követően az autó esetleges átépítése és újabb mérések elvégzése, ahol pontosabb referencia mérést valósítok meg lézeres eszközökkel.

Tesztelés-mérés más ASSN üzemmódban

A dolgozatban felhasznált BNO055-ös chip képes több más üzemmódra is, melyben nem működik a belső szenzorfúzió, de jobban paraméterezhetők az egyes szenzorelemek, ami pontosíthatja a méréseket. Ennek vizsgálata is a későbbi tervek közt szerepel, mert elképzelhető, hogy a belsőleg működő algoritmusokra jobb külső ROS alapú megoldásokat tudunk megvalósítani, és ezzel a rendszert pontosíthatjuk.

Irodalomjegyzék

[1] NovAtel Inc.: *An Introduction to GNSS*
NovAtel Inc., Calgary, Canada (2015), ISBN: 978-0-9813754-0-3, p. 62-65.

[2] A. D. King: *Inertial Navigation – Forty Years of Evolution*
GEC REVIEW, VOL. 13, NO. 3, 1998, pp. 140-149.

[3] Makan Gergely: *Versenykajak mozgásának mérése és a mért jelalakok analízise*
MSc Diplomamunka: Szegedi Tudományegyetem TTIK Kísérleti Fizika Tanszék, 2012

[4] NovAtel Inc.: *IMU Errors and Their Effects*
NovAtel Inc., Calgary, Canada (2014), ISBN: 978-0-9813754-0-3, p. 62-65.

[5] *Apollo 8 Report:*
http://web.mit.edu/digitalapollo/Documents/Chapter6/hoagprogreport.pdf
utolsó látogatás: 2018. október 22.

[6] *BNO055 Chip Datasheet:*
https://ae-bst.resource.bosch.com/media/_tech/media/datasheets/BST_BNO055_DS000_14.pdf
utolsó látogatás: 2018. október 22.

[7] Avi Silberschatz, Peter Baer Galvin, Greg Gagne: *Operating System Concepts*
John Wiley & Sons, Inc, 2012, ISBN: 978-1-118-06333-0, p. 283-287; 357-360; 781-827.

[8] Lentin Joseph: *ROS Robotics Projects*
Packt Publishing, 2017, ISBN: 978-1-78355-471-3, p 1-20.

[9] Hongyi Liu: *Environment for Industrial Robot Performance Evaluation*
MSc Diplomamunka: KTH Royal Institute of Technology CSC, 2015

[10] *POSIX programozás:*
https://www.cmrr.umn.edu/~strupp/serial.html
utolsó látogatás: 2018. október 22.

[11] Bárczy Barnabás: *Integrálszámítás*
Műszaki Könyvkiadó, Budapest (1969), ISBN: 963-10-9731-5, p. 320.

Ábrajegyzék

[A1] https://ae-bst.resource.bosch.com/media/_tech/bilder/products/motion/hubsnodes/bno055/pro_BNO055_m.jpg
(utolsó látogatás időpontja: 2018. október 22.)

[A2] https://s.zeptobars.com/mpu6050-HD.jpg
(utolsó látogatás időpontja: 2018. október 22.)

[A4] https://i2.wp.com/www.jetsonhacks.com/wp-content/uploads/2015/04/P02_F09_625.jpg?ssl=1 és https://i0.wp.com/www.jetsonhacks.com/wp-content/uploads/2015/04/P02_F10_625.jpg?ssl=1
(utolsó látogatás időpontja: 2018. október 22.)

[A5] http://web.mit.edu/digitalapollo/Documents/Chapter6/hoagprogreport.pdf
(utolsó látogatás időpontja: 2018. október 22.)

[A6] https://www.gotronic.fr/ori-module-9-dof-ada2472-23896.jpg
(utolsó látogatás időpontja: 2018. október 22.)

[A7] https://ae-bst.resource.bosch.com/media/_tech/media/datasheets/BST_BNO055_DS000_14.pdf
(utolsó látogatás időpontja: 2018. október 22.)

Mellékletek

1. BNO055 mértékegység regisztere

4.3.60 UNIT_SEL 0x3B

	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
Access	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w
Reset	0			0	0	0	0	0
Content	ORI Android Windows	reserved		TEMP Unit	reserved	EUL_Unit	GYR_Unit	ACC_Unit

DATA	bits	Description
ORI Android Windows	7	Read: Current selected orientation mode Write: Select orientation mode 0: Windows orientation 1: Android orientation See section 3.6.2 for more details
TEMP_Unit	5	Read: Current selected temperature units Write: Select temperature units 0: Celsius 1: Fahrenheit See section 3.6.1 for more details
EUL_Unit	3	Read: Current selected Euler units Write: Select Euler units 0: Degrees 1: Radians See section 3.6.1 for more details
GYR_Unit	2	Read: Current selected angular rate units Write: Select angular rate units 0: dps 1: rps See section 3.6.1 for more details
ACC_Unit	1	Read: Current selected acceleration units Write: Select acceleration units 0: m/s^2 1: mg See section 3.6.1 for more details

2. melléklet: ASSN adatolvasás C kódja

```
//Deep-pilot project SZE
//Gulyás Péter
//BNO055 ASSN, Encoder and servo data read

#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BNO055.h>
#include <utility/imumaths.h>
#include <EEPROM.h>

/* This driver reads Orientation-Linacc-Encoder-Servo data from the BNO055
   Connections:
   ===========
   Connect SCL to analog 5
   Connect SDA to analog 4
   Connect VDD to 5V DC
   Connect GROUND to GND
   D2 - Encoder A
   D8 - Encoder B
*/

// Set the delay between fresh samples and define PIN numbers
#define BNO055_SAMPLERATE_DELAY_MS (13)
#define SrvFb A0
#define ENC_A 2
#define ENC_B 8

//Define BNO-ID
Adafruit_BNO055 bno = Adafruit_BNO055(29);

//Define variables

const int interrupt0 = 0; // (PIN 2)  //ENCODER READING//

long sum_pulse_num = 0;
int pulse_num = 0;
double rpm = 0;
```

```
float QX = 0, QY = 0, QZ = 0, QW = 0;
float GX = 0, GY = 0, GZ = 0;
float LX = 0, LY = 0, LZ = 0;

struct MotFb
{
  double velo;
  double dist;
  double rpm;
};
typedef struct MotFb Motor_FB;

//---------------------------------------------------------------------

int AngFbInt;
String VeloFeedback;
String RpmFeedback;
String DistFeedback;                    //Variables for feedback//
String buf1;
String buf2;
String buf3;
String mystring;

//---------------------------------------------------------------------

void displaySensorOffsets();
void Enc_Calc();
void Pulse_counter();                   //Function list//
int ServFb();

//---------------------------------------------------------------------
// Initialization - Check if BNO is detected -->
// EEPROM reading for calibration offset values --> Set Crystal use

void setup(void)
    {
        pinMode (SrvFb, INPUT);
        pinMode (ENC_A, INPUT_PULLUP);
        pinMode (ENC_B, INPUT_PULLUP);
```

```
attachInterrupt(interrupt0, Pulse_counter, FALLING);

Serial.begin(115200);
delay(500);
//Serial.println("Orientation Sensor Test"); Serial.println("");
// Initialise the sensor
if (!bno.begin())
{
    // There was a problem detecting the BNO055. Check your connections!
    //Serial.print("No BNO055 detected. Check your wiring or I2C ADDR!");
    while (1);
}

int eeAddress = 0;
long bnoID;
bool foundCalib = false;

EEPROM.get(eeAddress, bnoID);

adafruit_bno055_offsets_t calibrationData;
sensor_t sensor;
bno.getSensor(&sensor);
if (bnoID != sensor.sensor_id)
{
    Serial.println("\nNo Calibration Data for this sensor exists in EEPROM");
    delay(500);
}
else
{
    //Serial.println("\nFound Calibration for this sensor in EEPROM.");
    eeAddress += sizeof(long);
    EEPROM.get(eeAddress, calibrationData);

    //Serial.println("\n\nRestoring Calibration data to the BNO055...");
    bno.setSensorOffsets(calibrationData);

    //Serial.println("\n\nCalibration data loaded into BNO055");
    delay(2000);
    bno.setExtCrystalUse(false);
```

```
            delay(100);
        }
    }


//---------------------------------------------------------------------
// Reading sensor and encoder-servo values

void loop(void)
{
// Get a new sensor event
   sensors_event_t event;
   bno.getEvent(&event);

// Display timestamp and Heading-Roll-Pitch sensor data
    Serial.print(millis());
    Serial.print(",");
    Serial.print(event.orientation.x, 4);
    Serial.print(",");
    Serial.print(event.orientation.y, 4);
    Serial.print(",");
    Serial.print(event.orientation.z, 4);
// Display the Linearaccelerometer data
    imu::Vector<3> eulerlin = bno.getVector(Adafruit_BNO055::VECTOR_LINEARACCEL);
    Serial.print(",");
    Serial.print(eulerlin.x());
    Serial.print(",");
    Serial.print(eulerlin.y());
    Serial.print(",");
    Serial.print(eulerlin.z());
    Serial.print(",");

// Display Encoder and Servo data
   Enc_Calc();
   AngFbInt = ServFb();
   Serial.print(AngFbInt);
   Serial.print(",");
   Serial.print(buf1);
   Serial.print(",");
   Serial.print(buf2);
```

```
    Serial.print(",");
    Serial.print(buf3);

// New line for the next sample
    Serial.println("");

// Wait the specified delay before requesting nex data
    delay(BNO055_SAMPLERATE_DELAY_MS);
}

//------------------------------------------------------------------------
// Functio to display Sensor offset values:

void displaySensorOffsets(const adafruit_bno055_offsets_t &calibData)
    {
        Serial.print("Accelerometer: ");
        Serial.print(calibData.accel_offset_x); Serial.print(" ");
        Serial.print(calibData.accel_offset_y); Serial.print(" ");
        Serial.print(calibData.accel_offset_z); Serial.print(" ");

        Serial.print("\nGyro: ");
        Serial.print(calibData.gyro_offset_x); Serial.print(" ");
        Serial.print(calibData.gyro_offset_y); Serial.print(" ");
        Serial.print(calibData.gyro_offset_z); Serial.print(" ");

        Serial.print("\nMag: ");
        Serial.print(calibData.mag_offset_x); Serial.print(" ");
        Serial.print(calibData.mag_offset_y); Serial.print(" ");
        Serial.print(calibData.mag_offset_z); Serial.print(" ");

        Serial.print("\nAccel Radius: ");
        Serial.print(calibData.accel_radius);

        Serial.print("\nMag Radius: ");
        Serial.print(calibData.mag_radius);
    }

//------------------------------------------------------------------------

void Enc_Calc(){
```

```
  Motor_FB FB_1;                // 48 Encoder pulse = 1 round
  rpm = pulse_num * 60 / 24;    // RPM calculating ratio for 1 ch
  //Serial.print(rpm);
  //Serial.print("\t");
  FB_1.rpm = rpm;
  FB_1.velo = rpm / 9.5492965964254;
  //Serial.print("Speed = ");
  //Serial.print(FB_1.velo, 4);
  //Serial.print(" rad/s");
  //Serial.print("\t");
  sum_pulse_num = sum_pulse_num + pulse_num;
  FB_1.dist = sum_pulse_num / 2370.72 * 0.282743;
  //Serial.print("Distance: ");
  //Serial.print(FB_1.dist, 4);
  //Serial.println(" m");
  //Serial.print("\t");
  VeloFeedback = String(FB_1.velo);
  RpmFeedback = String(FB_1.rpm);
  DistFeedback = String(FB_1.dist);
  buf1 = VeloFeedback;
  buf2 = RpmFeedback;
  buf3 = DistFeedback;
  pulse_num = 0x00;
}

//---------------------------------------------------------------------

void Pulse_counter()
{
 if(digitalRead(ENC_B) == HIGH) pulse_num++;
 else pulse_num--;
}

//---------------------------------------------------------------------

int ServFb(){
  int val = analogRead(SrvFb);              // Analog value reading to know the position
  //Serial.print(val);
  int valMap = map(val, 132, 413, -10, 10); // Remap the analog interval to get the angle of the car
  return valMap; }
```

3. melléklet: Python teszt kód

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
"""
Created on Mon Oct 4 11:27:10 2018
@author: Potex (github.com/Potex)
Python version: 3.4.2
Control script for Szenergy-model.
"""

# Raspberry Pi script which controls the motor and servo
# also creats log files to save sensor data

import serial
import time
import threading
import csv

t = time.localtime()
timestamp = time.strftime('%Y%m%d%H%M%S', t)
ser=serial.Serial("/dev/ttyACM0", 115200)
ser2=serial.Serial("/dev/ttyUSB0", 115200)
time.sleep(1)

sens_reading = True
log = []

with open('log_sensor'+timestamp+'.csv', 'w') as csvfile:
    Sens_log = csv.writer(csvfile, delimiter=' ', quotechar='|', quoting=csv.QUOTE_MINIMAL)
    Sens_log.writerow(['Sensor values form Encoder, Servo and ASSN'])
print('Sensor CSV Done. Start logging')

def sensor_read():
    while sens_reading:
        log1 = ser2.readline()
        data = str(log1)
        data2 = data[2:-5]
        with open('log_sensor'+timestamp+'.csv', 'a') as csvfile:
```

```
            Sens_log = csv.writer(csvfile, delimiter=' ', quotechar='|', quoting=csv.QUOTE_MINIMAL)
            Sens_log.writerow(data2)

s = threading.Thread(target=sensor_read)
s.start()
time.sleep(4)

nullStr  =  "1.0,1.0a"
forwStr  =  "0.0,0.95a"
rightStr =  "0.0,2.0a"
right2Str =  "0.0,1.88a"
leftStr =  "0.2,0.0a"
forw2Str = "0.0,0.93a"

with open('log_jarmumodell'+timestamp+'.csv', 'w') as csvfile:
    Modell_log = csv.writer(csvfile, delimiter=' ', quotechar='|', quoting=csv.QUOTE_MINIMAL)
    Modell_log.writerow(['Systime between performed tasks'])
print('Modell CSV Done. Start logging')

##3219.429703301   Time before serial_write
##3219.430038246   Time after serial_write, before sleep(1)
##  serial_write = 0.334945 ms
##3220.431092828   Time after sleep(1), before serial_write
##  sleep(1)     = 1001.054582 ms
##3220.431360638   Time after serial_write, before sleep(2)
##  serial_write = 0.267810 ms
##3222.433491833   Time after sleep(2), before serial_write
##  sleep(2)     = 2002.125453 ms
##3222.433730789   Time after serial_write, before sleep(11)
##  serial_write = 0.238956 ms
##3233.439386459   Time after sleep(11)
##  sleep(11)    = 11005.65567 ms

##-------------1. Lap ---------------
#log.append(time.perf_counter())
ser.write(nullStr.encode('ascii'))
#log.append(time.perf_counter())
time.sleep(1.5)
#log.append(time.perf_counter())
ser.write(forw2Str.encode('ascii'))
```

```
    #log.append(time.perf_counter())
    time.sleep(3.5)
    #log.append(time.perf_counter())
    ser.write(rightStr.encode('ascii'))
    #log.append(time.perf_counter())
    time.sleep(10.2)
    #log.append(time.perf_counter())
    ser.write(forwStr.encode('ascii'))
    #log.append(time.perf_counter())
    time.sleep(5.6)
    #log.append(time.perf_counter())
    ser.write(right2Str.encode('ascii'))
    #log.append(time.perf_counter())
    time.sleep(10.2)
    #log.append(time.perf_counter())
    ser.write(forwStr.encode('ascii'))
    #log.append(time.perf_counter())
    time.sleep(1.65)
    #log.append(time.perf_counter())
    ser.write(nullStr.encode('ascii'))
    #log.append(time.perf_counter())
    time.sleep(2)
    #log.append(time.perf_counter())
    ##-------------1. Lap ----END--------
    sens_reading = False
    time.sleep(1)
    ##---------Write to log file---------
    with open('log_jarmumodell'+timestamp+'.csv', 'a') as csvfile:
        Modell_log = csv.writer(csvfile, delimiter=' ', quotechar='|', quoting=csv.QUOTE_MINIMAL)
        Modell_log.writerow(log)
    print('Log written.')
    ##------Write to log file---END------

    print('Sensor logging thread stopped.')
    ser.close()
    ser2.close()
    print("Test round comlpeted. Serial connection closed, motor stopped, steering set to middle position. ")
```

4. melléklet: Odometria ROS kód

```
#include <ros/ros.h>
#include <std_msgs/String.h>
#include <sensor_msgs/JointState.h>
#include <nav_msgs/Odometry.h>
#include <std_msgs/Float64.h>
#include <szelectricity_common/VehicleParameters.hpp>
#include <szelectricity_common/MathGeometry.hpp>

const std::string DEFAULT_BASE_FRAME = "base_footprint";
const std::string DEFAULT_ODOM_FRAME = "odom";

struct joints {
  double p1;
  double p2;
  double velo1;
} olvasas;

sensor_msgs::JointState msg_joint;

nav_msgs::Odometry current_state;         // Current odometry state (pose, twist, covariance)
double theta = 0.0;                       // Current yaw related to COG
double dtheta = 0.0;                      // Change in theta
double dx = 0.0;                          // Change in x
double dy = 0.0;                          // Change in y

ros::Time prev_time, curr_time;

void jointCallback(const sensor_msgs::JointState::ConstPtr& msg)
{
  //std::cout << *msg << std::endl;
  msg_joint.position = msg->position;
  msg_joint.velocity = msg->velocity;
  //std::cout << msg_joint.position[0] << std::endl;
  //std::cout << msg_joint.velocity[2] << std::endl;
  olvasas.p1 = msg_joint.position[0];
  //std::cout << olvasas.p1 << std::endl;
  olvasas.p2 = msg_joint.position[1];
```

```
  olvasas.velo1 = msg_joint.velocity[2];

}

void UpdateStep(double dt, const ros::Time stamp)
    {

        current_state.header.stamp = stamp;
        // Update the pose estimation
        theta = theta+dtheta*dt;
        current_state.twist.twist.linear.x = dx;
        current_state.twist.twist.linear.y = dy;
        // Angular speed is defined as RPY, only YAW is considered
        current_state.twist.twist.angular.z = dtheta*dt;
        // Pose update
        current_state.pose.pose.position.x += dx*dt;
        current_state.pose.pose.position.y += dy*dt;

        current_state.pose.pose.orientation = szenergy::YawToQuaternion(theta);
    }

int main(int argc, char **argv)
{

    ros::init(argc, argv, "joint_listener");
    ros::NodeHandle n;
    ros::Subscriber sub = n.subscribe("/joint_states", 1000, jointCallback);
    ros::Publisher odom = n.advertise <nav_msgs::Odometry> ("odometry", 1000);
    ros::Rate loop_rate(20);


    szenergy::VehicleParameters params("modelcar",
      0.045,
      0.267,
      0.188,
      0.147);

    current_state.pose.pose.position.z = 0.0;
    current_state.twist.twist.linear.z = 0.0;
    current_state.pose.pose.orientation.w = 1.0;
```

```
    current_state.twist.twist.angular.x = 0.0;
    current_state.twist.twist.angular.y = 0.0;

    current_state.header.frame_id = "odom";
    current_state.child_frame_id = "base_link";

    while(ros::ok())

    {
        olvasas.p1;
        olvasas.p2;
        olvasas.velo1;
        //std::cout << olvasas.p1 << '\t' << olvasas.p2 << '\t' << olvasas.velo1 << std::endl;

        ros::Duration dt;
        const ros::Time stamp;
        curr_time = ros::Time::now();

        dt = curr_time - prev_time;


        double linvel = olvasas.velo1*params.wheelradius;
        //std::cout << linvel << '\t';
        dtheta = linvel/(params.wheelbase)*tan(olvasas.p1);
        //std::cout << dtheta << '\t';
        dx = linvel*cos(theta);
        //std::cout << dx << '\t';
        dy = linvel*sin(theta);
        //std::cout << dt << std::endl;
        UpdateStep(dt.toSec(), stamp);

        odom.publish(current_state);

        prev_time = curr_time;

    ros::spinOnce();

            loop_rate.sleep();
    }  }
```

5. melléklet: NANO_reading ROS kód

```
#include "ros/ros.h"
#include <stdio.h>
#include "SerialPort.h"
#include <algorithm>
#include <stdlib.h>
#include <math.h>
#include <iostream>
#include <string>
#include <unistd.h>
#include <cstdlib>
#include <sensor_msgs/JointState.h>
#include <sensor_msgs/Imu.h>
//#include <geometry_msgs/Quaternion.h>
#include <sstream>
#include <iomanip>

#define _USE_MATH_DEFINES

//int open_port(void);

//int fd;                                                   // Declare file descriptor for the port.
const double Conv = 1.000000;
float cov_base = 0.00;

int main(int argc, char **argv)
{
	ros::init(argc, argv, "NANO_reader");
	ros::NodeHandle n;
	ros::Publisher joint = n.advertise <sensor_msgs::JointState> ("joint_states", 1000);
	ros::Publisher assn = n.advertise <sensor_msgs::Imu> ("assn", 1000);
	//ros::Publisher quat = n.advertise <geometry_msgs::Quaternion> ("quat", 1000);
	ros::Rate loop_rate(50);

	// ROS message init, and message fill up.
	sensor_msgs::Imu assn_pub;
	assn_pub.header.frame_id = "imu";
```

```
	sensor_msgs::JointState joint_pub;

	joint_pub.name.resize(3);
	joint_pub.position.resize(3);
	joint_pub.velocity.resize(3);

	joint_pub.name[0] = "front_axis_to_left_front_steer";
	joint_pub.name[1] = "front_axis_to_right_front_steer";
	joint_pub.name[2] = "rear_axis_to_left_rear_wheel";

	try
	{
		//Open serial port with defined parameters
		ROS_INFO("Opening serial connection.");
		SerialPort serial_port ("/dev/ttyUSB0");
		serial_port.Open(SerialPort::BAUD_115200,
					SerialPort::CHAR_SIZE_8,
					SerialPort::PARITY_NONE,
					SerialPort::STOP_BITS_1,
					SerialPort::FLOW_CONTROL_HARD);


		// Enter the main loop
		while (ros::ok())
		{
			// Wait until the data is available
			if(serial_port.IsDataAvailable())
		    {
				// Define string to read in data
				std::string ss;

				// Reading in from serial port
				ss = serial_port.ReadLine();

				std::istringstream iss(ss);

				// Create a vector of strings separated by "whitespace"
				std::vector<std::string> results(std::istream_iterator<std::string>{iss},
std::istream_iterator<std::string>());
```

```
                        //std::cout << results[10].c_str() << '\t' << results[11].c_str() << '\t' << results[12].c_str() <<
'\t' << results[13].c_str() << std::endl;

                        //Print all IMU data to console as a string
                        float QX = Conv*atof(results[0].c_str());
                        //std::cout << QX  << '\t';
                        float QY = Conv*atof(results[1].c_str());
                        //std::cout << QY  << '\t';
                        float QZ = Conv*atof(results[2].c_str());
                        //std::cout << QZ  << '\t';
                        float QW = Conv*atof(results[3].c_str());
                        //std::cout << QW  << '\t';
                        float GX = Conv*atof(results[4].c_str());
                        //std::cout << LX  << '\t';
                        float GY = Conv*atof(results[5].c_str());
                        //std::cout << LY  << '\t';
                        float GZ = Conv*atof(results[6].c_str());
                        //std::cout << LZ  << '\t';
                        float LX = Conv*atof(results[7].c_str());
                        //std::cout << GX  << '\t';
                        float LY = Conv*atof(results[8].c_str());
                        //std::cout << GY  << '\t';
                        float LZ = Conv*atof(results[9].c_str());
                        //std::cout << GZ << '\t';
                        int angWheel = Conv*atoi(results[10].c_str());
                        //std::cout << angWheel << '\t';
                        float velo = Conv*atof(results[11].c_str());
                        //std::cout << velo << '\t';
                        int rpm = Conv*atoi(results[12].c_str());
                        //std::cout << rpm << '\t';
                        float dist = Conv*atof(results[13].c_str());
                        //std::cout << dist << std::endl;

                        //std::cout << angWheel << '\t' << velo << '\t' << rpm << '\t' << dist << std::endl;

                        assn_pub.header.stamp = ros::Time::now();
                        joint_pub.header.stamp = ros::Time::now();

                        // Orientation-Quaterninon data passed to message         // IMU MESSAGE START
                        assn_pub.orientation.x = QX;
```

```
                        assn_pub.orientation.y = QY;
                        assn_pub.orientation.z = QZ;
                        assn_pub.orientation.w = QW;

                        // Covariance matrix filled up with zeros
                        std::fill( assn_pub.orientation_covariance.begin(), assn_pub.orientation_covariance.end(),  cov_base
);

                        // 3 axis Gyroscope data passed to message
                        assn_pub.angular_velocity.x = GX;
                        assn_pub.angular_velocity.y = GY;
                        assn_pub.angular_velocity.z = GZ;

                        // Covariance matrix filled up with zeros
                        std::fill( assn_pub.angular_velocity_covariance.begin(), assn_pub.angular_velocity_covariance.end(),
cov_base );

                        // 3 axis Linear acceleration data passed to message
                        assn_pub.linear_acceleration.x = LX;
                        assn_pub.linear_acceleration.y = LY;
                        assn_pub.linear_acceleration.z = LZ;

                        // Covariance matrix filled up with zeros
                        std::fill( assn_pub.linear_acceleration_covariance.begin(),
assn_pub.linear_acceleration_covariance.end(),  cov_base );

                        // Fill the JointState message        IMU MESSAGE END    JOINTSTATE MESSAGE START
                        joint_pub.position[0] = (angWheel * M_PI) / 180;
                        joint_pub.position[1] = (angWheel * M_PI) / 180;
                        joint_pub.position[2] = dist;
                        joint_pub.velocity[0] = (rpm / 9.55);
                        joint_pub.velocity[1] = (rpm / 9.55);
                        joint_pub.velocity[2] = (rpm / 9.55);


                        // Publish messages                                                    JOINTSTATE MESSAGE END
                        assn.publish(assn_pub);
                        joint.publish(joint_pub);

                }
```

```
                  ros::spinOnce();

                  loop_rate.sleep();

            }

            // Close connection
            if(serial_port.IsOpen())
            {
                  ROS_INFO("Closing serial connection.");
                  serial_port.Close();
            }

            ROS_INFO("Serial node closed.");
            return 0;
      }
      catch (SerialPort::OpenFailed exception)
      {
            ROS_ERROR("Failed to open the port");
      }

      catch (SerialPort::AlreadyOpen exception)
      {
            ROS_ERROR("Port is already open");
}

}

// ---------------------------------PORT OPENING FUNCTION ------------------------------------------------------
/*
int open_port()
{
      fd = open("/dev/ttyACM0", O_RDWR | O_NOCTTY | O_NDELAY);                    // Open port:
      if (fd == -1)
    {
            std::cout << "open_port: Unable to open /dev/ttyACM0" << std::endl;    // ERROR: Could not open port!
    }
      else
      {
```

```
	fcntl(fd, F_SETFL, FNDELAY);
		std::cout << "In Open port fd = " << fd << std::endl;			// Port opened successfully!
	}
	struct termios options_2;						// Read the configureation of the port.
	tcgetattr( fd, &options_2 );
	cfsetispeed( &options_2, B115200 );					// Set I/O speed of the port.
	cfsetospeed( &options_2, B115200 );


	options_2.c_cflag |= ( CLOCAL | CREAD );				// Enable the receiver and set local mode.
	tcsetattr(fd, TCSAFLUSH, &options_2);				// Set the new options for the port.

	options_2.c_cflag &= ~CSIZE;				// Set the Charactor size: Mask the character size bits
	options_2.c_cflag |= CS8;				// Select 8 data bits

	options_2.c_cflag &= ~PARENB;				// Set parity - No Parity (8N1)
	options_2.c_cflag &= ~CSTOPB;
	options_2.c_cflag &= ~CSIZE;
	options_2.c_cflag |= CS8;

	options_2.c_iflag &= ~(IXON | IXOFF | IXANY);			// Disable Software Flow control

	options_2.c_oflag &= ~OPOST;					// Chose raw (not processed) output
	if ( tcsetattr( fd, TCSANOW, &options_2 ) == -1 )
		std::cout << "Error with setting attributions." << std::endl;		// ERROR: set tcsetattr!
	else
		std::cout << "Attributes settings succeeded" << std::endl;		// tcsetattr OK!
	usleep(1500);
	tcflush(fd,TCIOFLUSH);							// Flush I/O queue.

	return(fd);
}
// ---------------------------------PORT OPENING FUNCTION -------------------------END-----------------------*/
```