Ágoston Máté – Gulyás Péter

Gyors szenzor validáló módszer fejlesztése inerciális navigációs rendszerhez

Villamosmérnöki BSc – Automatizálási Szakirány

2014/15

Konzulens: Dr. Ballagi Áron, tanszékvezető

Absztrakt

A TMDK dolgozat fő témája egy készülő autonóm járműben használt inerciamérő egység. Az inerciamérő rendszer segítségével pozíció meghatározást végezhetünk és a jármű dinamikai adatairól kaphatunk információt. Az önvezető járművek fejlesztése rohamosan felgyorsult az utóbbi években és az egyetemen is indult tavaly szeptemberben egy Autonóm Jármű szakkör. A szakkör távlati célja, hogy 2019-ben önvezető autóval indulhassunk a Shell Eco-marathon új kategóriájában. A rendszer kialakítását először egy kisméretű autón fejlesztjük ki, mely moduláris felépítéséből adódóan egy az egyben illeszthető lesz a versenyre készülő nagy autóba. A moduláris rendszer keretét egy Linux alapú úgynevezett ’middleware’ képezi, amit már évtizedek óta használnak a robotika világában. Ez a Robot Operating System. A fejlesztés alatt álló járműmodell 1:6 méretarányban készül és önvezetésre lesz képes. Ebben a járműben a Bosch által gyártott BNO055 típusú szenzort alkalmazzuk az inerciamérésre. Az eszköz megismerése és beállítása során több mérést is kellett végezni, nagy pontossággal. A precíz mérések elvégzéséhez egy ABB ipari robotkart használtunk, így programozott irányokban tudtuk mozgatni a szenzort, ezáltal pontosan meg volt, milyen adatokat várunk vissza az eszköztől. A robotkar programozásához az ABB RobotStudio-t használtuk, mely rendelkezik szimulációs környezettel is. További feladatunk volt egy olyan programkód megvalósítása mellyel az adatok real-time feldolgozása és vizualizálása, valamint, naplózása is egyszerre történhet.

Tartalom

A projekt felvezetése

A TDK dolgozatunk egy a Bosch által gyártott Application Specific Sensor Node-ot (továbbiakban ASSN) bemutatásával és mérésével foglalkozik. Az egyetemen működő autonóm jármű szakkör keretein belül ennek az eszköznek a segítségével egy önvezető modellautó navigációját valósítjuk majd meg. A projekt célja egy olyan modelljármű elkészítése, melyet teljes egészében saját magunk fejlesztünk. Az általunk tervezett és készített alvázra moduláris felépítéssel illesztjük az hardverjeinket és a hozzá tartozó rendszert, így ezt a későbbiekben már könnyebben tudjuk átültetni a Shell Eco-marathon versenyre készített járművünkbe. Az önvezetés megvalósítására különböző szenzorokat, hardvereket és szoftvereket használunk. Szükségünk van:

- A dolgozatban bemutatott chipre, mely a jármű mozgásáról ad információkat lehetővé téve az aktuális pozíció és orientáció számítását
- Lidarra, ami a környezet érzékelésért felelős
- Infra / ultrahangos szenzorokra, melyek a pontos parkoláshoz szükségesek
- Kamera, ami a gépi látáshoz és a neurális hálóhoz nélkülözhetetlen

Mivel az érzékelők eltérő működésűek, ezért az általuk küldött adatok további rendszerezést és feldolgozást igényelnek. Annak érdekében, hogy a fejlesztés később könnyen átültethető legyen a versenyautóra egy modularitást elősegítő middleware-t alkalmazunk, a Robot Operating System-et. A most bemutatásra kerülő méréseinket mind dinamikusan végeztük az Automatizálási Tanszék ÚT111-es Robotlaborjában. Ebben a teremben kettő, az ABB által gyártott robot található, melyből egyet alkalmaztunk a mérések során.

Mi az az IMU?

Egy inercia mérő egység (továbbiakban IMU – Inertial Measurement Unit) három-három gyorsulás mérőből és giroszkópból álló rendszer, ahol a szenzorok úgy vannak elhelyezve, hogy mérendő tengelyeik egymásra ortogonálisok legyenek. Az inercia mérő szenzorok három alapvető adatot adnak vissza a felhasználók számára:

- Giroszkóp: ⇒ szögsebesség deg/s
- Gyorsulásmérő ⇒ lineáris gyorsulás m/s^2
- Magnetométer ⇒ mágneses térerősség µT

BNO055

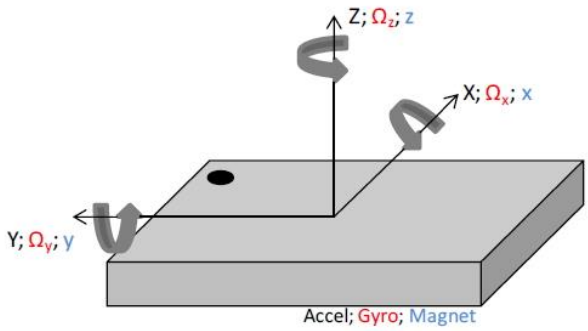
A fenti típusnév egy Bosch által gyártott okos-szenzort, ASSN-t takar, mely képes I^2C és UART kommunikációra is. Ez az eszköz egy 28 lábú System in Package (SiP), ami tartalmaz:

- háromtengelyes 14 bites gyorsulásmérőt
- háromtengelyes 16 bites giroszkópot, melynek tartománya ±2000 deg/s
- háromtengelyes magnetométert
- 32 bites cortex M0+ mikrokontrollert a Bosch Sensortec fúziós szoftverrel

A szenzor az Autonóm Jármű Szakkör keretén belül került a látóterünkbe, és egyben vált a projektmunkánkká. A piacon is újdonságként szereplő ASSN kecsegtető tulajdonságokkal rendelkezik, valamint a gyors belső adatfeldolgozása megalapozta a beszerzésére épülő bizalmat. A chip hozzánk egy Atmel által gyártott próbalapkán [1] került (1. ábra).

1. ábra : Atmel BNO055 Xplained Pro

A BNO055 chip esetén a mértékegységek programozhatók, melyre egy külön regiszter szolgál [2]. Ennek a regiszternek az alapbeállítása megegyezik a fentebb említett mértékegységekkel, és ezt megfelelőnek találtuk a projekthez, így nem írtuk felül. A szenzor a beépítés variálhatóságát elősegítve tartalmaz egy regisztert, mely a tengelyek konfigurálására szolgál. (2. ábra) A mérésekhez mi a gyári értékeknek (1.melléklet) megfelelően tudtuk elhelyezni a lapkát, így ennek a funkciónak az előnye egyelőre kihasználatlan maradt.



2. ábra: Tengelyek gyári beállítása

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 pontosságunk romlását eredményezné. Szoftverében több üzemmód választható, melyek lehetőséget biztosítanak önálló szenzorok használatára és fúziós használatra is. Ezekből mi az NDOF (Nine Degrees of Freedom) lehetőséget választottuk, mely kilenc szabadságfokos vizsgálatokra ad lehetőséget:

- 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 eszközünk pontos kalibrálása, és beüzemelése, annak érdekében, hogy az önvezető modellünk számára biztosíthassuk a pontos lokalizáció 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 parkolóházakban és alagutakban.

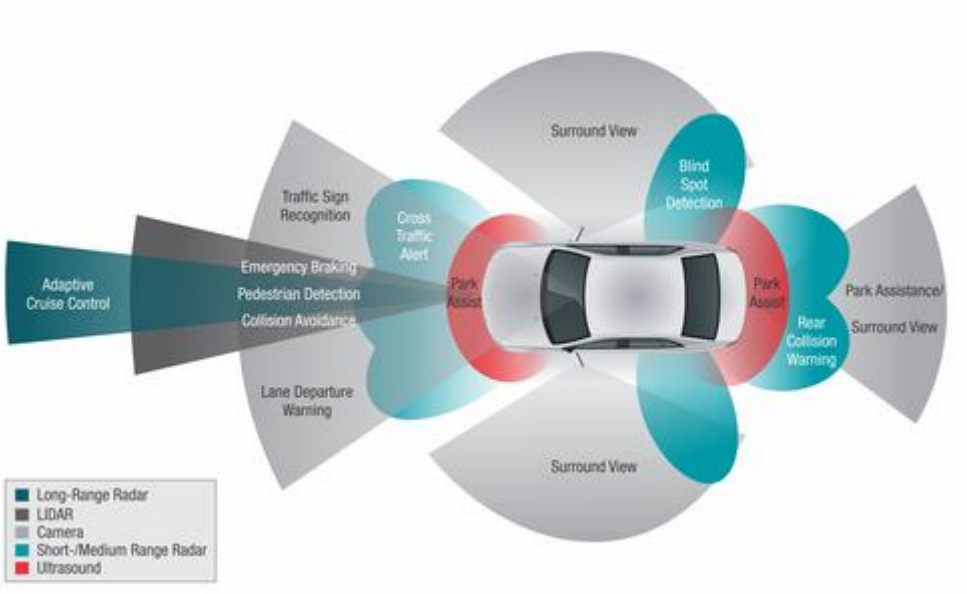
Autonóm járművek

Egy autonóm jármű képes önállóan, sofőr beavatkozása nélkül ’A’ pontból ’B’ pontba eljutni akár közúti forgalomban, köszönhetően a fejlett hardvereknek és szoftvereknek. Ez a technológia nem annyira új, mint azt elsőre gondolnánk. A mezőgazdaságban például már régóta fontos szerepet játszik, de a közúti járművek fejlesztésében csak az elmúlt években indult rohamos fejlődésnek. A jövő autói képesek érzékelni környezetüket, navigálni saját magukat, kommunikálni tudnak egymással, ami azt eredményezi, hogy egyes forgalmi szituációkat gyorsabban és biztonságosabban képesek megoldani, mint az emberek. Egy közúti jármű automatizáltsági szintjét 2014-ben a Society of Automotive Engineers (továbbiakban SAE) szabvány formájában definiálta [3]. A szintek közötti különbség abban mérhető, hogy a dinamikus vezetési műveletek milyen arányban oszlanak meg ember és a rendszer között.

Automatizáltsági szint	Definíció
0. szint	A humán vezető végez minden irányítási műveletet.
1. szint	A jármű emberi irányítás alatt áll, de a rendszer a kormányzási vagy fékezési/gyorsítási feladatokat átveheti.
2. szint	1. szinttől annyiban különbözik, hogy a rendszer egyszerre vezéreli a sebességet és a kormányzást.
3. szint	A rendszer irányítja a járművet, de feltételezi, hogy a sofőr szükség esetén visszaveszi az irányítást.
4. szint	A 3. szinttől annyiban különbözik, hogy a rendszer számol azzal, ha a sofőr nem képes megfelelően reagálni.
5. szint	A rendszer irányít minden műveletet. A környezetének változását ugyan úgy kezeli, mint egy ember. Felügyeletéhez semmilyen humán beavatkozás nem szükséges.

Navigáció

Az önvezető járművek irányítása szempontjából legfontosabb, hogy képesek legyenek a környezetüket minél részletesebben feltérképezni, valamint minél gyorsabban észleljék a közelükben bekövetkező változásokat. Ennek érdekében az autóra radart, lidart, GPS-t, szenzorokat és a gépi látást segítő kamerákat szerelnek fel. A szenzorok érzékelési tartományát a 3. ábra mutatja.

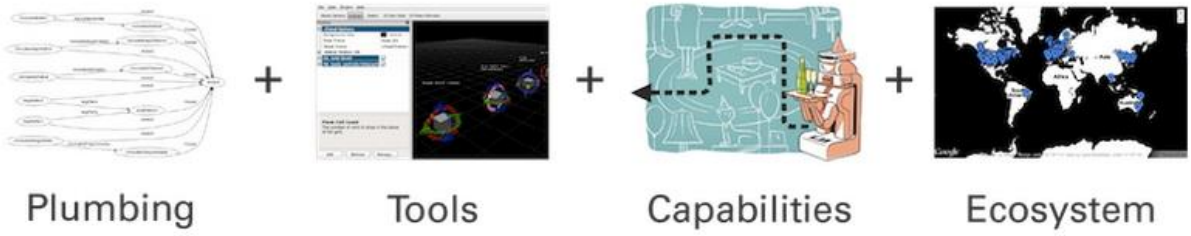


3. ábra: Autonóm jármű szenzorai

Az navigációért felelős rendszer ezen szenzorok adatait begyűjtve határozza meg a lehető legoptimálisabb útvonalat, figyelembe véve az úttesten található akadályokat és az útmenti jelzőtáblákat, valamint a közlekedési lámpákat, mindeközben képes saját, digitális térképet készíteni. A navigáció tulajdonképpen három fő komponensre bontható: lokalizáció, útvonal tervezés, jármű irányítás. Dolgozatunk ezen három alkotóelem közül a lokalizációval foglalkozik. Ennek feladata a jármű saját pozíciójának és orientációjának meghatározása. Az ASSN segítségével azonban ezen két komponens mellett meghatározza még, hogy a jármű hogyan, milyen sebességgel mozog [4]. Ezek az adatok elengedhetetlenek ahhoz, hogy az irányító rendszer a legmegbízhatóbban működhessen.

Robot Operating System

A Robot Operating System (továbbiakban ROS) egy nyílt forráskódú meta operációs rendszer elsősorban robotokhoz [5]. A szolgáltatásai lefedik, amit egy operációs rendszertől elvárunk, de egy másik operációs rendszerre épül, ami esetünkben Ubuntu Linux 16.04 LTS. Fontosnak tartottuk 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 2021-ig biztosít támogatást, és ezzel garantálja a fejlesztések lehetőségét a további pár évben. Alább látható (4. ábra, 1. táblázat) a rendszer áttekintése és funkciói.



4. ábra: ROS áttekintés

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	

1. táblázat: ROS funkciók

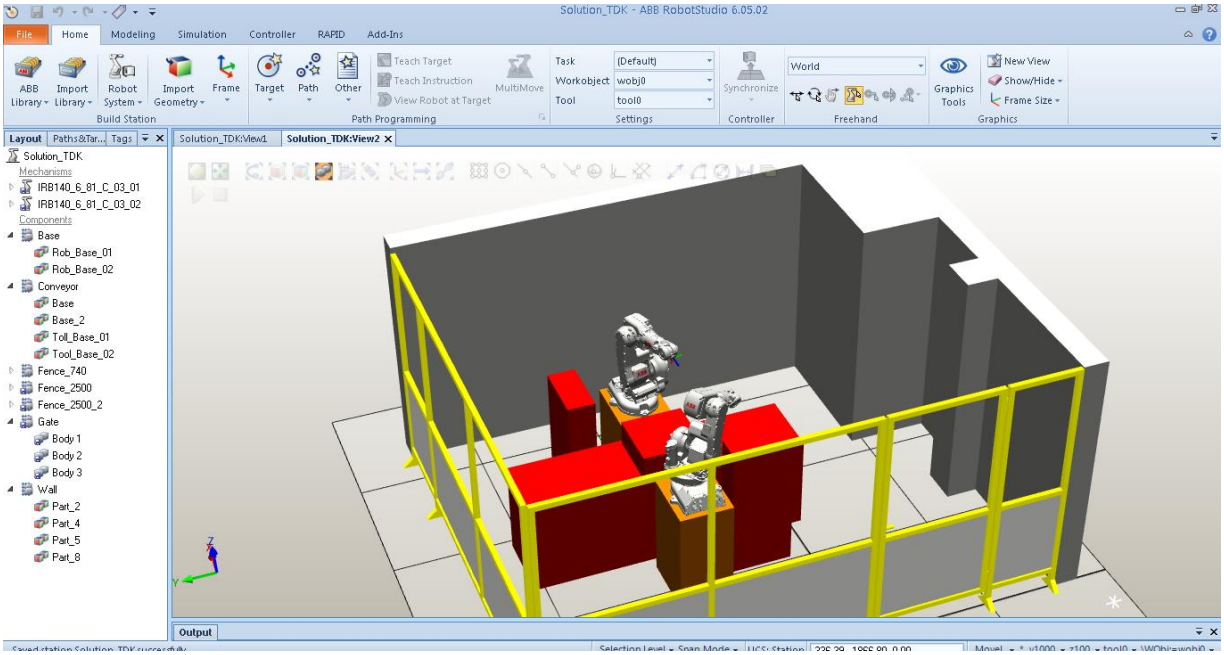
Az ROS biztosítja számunkra az alsó szintű driver kezelést, közvetett rálátást a hardverekre, általánosan használt függvények implementálását, csomag menedzsmentet és nem utolsó sorban a folyamatok közti üzenet közlést. Az ROS futásának folyamatábrája [6] egy Peer-to-Peer (P2P) hálózathoz hasonlítható, ahol nincs kitüntetett szerver, mindenki egyenrangú fél, és így jelenik meg a rendszerben az összes folyamat, akár több számítógép között is egységesen, valamint a kommunikáció során szinkron és aszinkron kommunikációra 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) ezekbe a Topic-okba.

ABB Ipari Robot

Az orientációs mérések elvégzéséhez olyan eszközt kellett választanunk, mely elég stabil, és nagy pontosságú, valamint a mérések ismételhetőségét biztosítja. Ezen felül az is fontos szempont volt, hogy képesek legyünk nem csupán összetett mozgásokat megvalósítani, hanem egyetlen tengely menti mozgását is befolyásolni. Ennek okán került előtérbe az automatizálási tanszék robot laboratóriuma és az itt található ABB - IRB 140-6/0.8 típusú robot [7]. Ez egy hat tengelyes többcélú ipari robot, amiből a laborban kettő darab áll rendelkezésünkre.

A robotokkal történő mérés további előnye, hogy dinamikus validációra ad lehetőséget, és biztosítja az ASSN szenzorjainak többtengelyes vizsgálatát egy teszt elvégzésén belül. Ennek jelentősége, hogy így az elvégzett manuális kalibráció ellenőrzése gyorsan kivitelezhetővé válik, és ipari minőséget biztosít a későbbi orientációs mérések számára, mely kulcsfontosságú a navigációban.

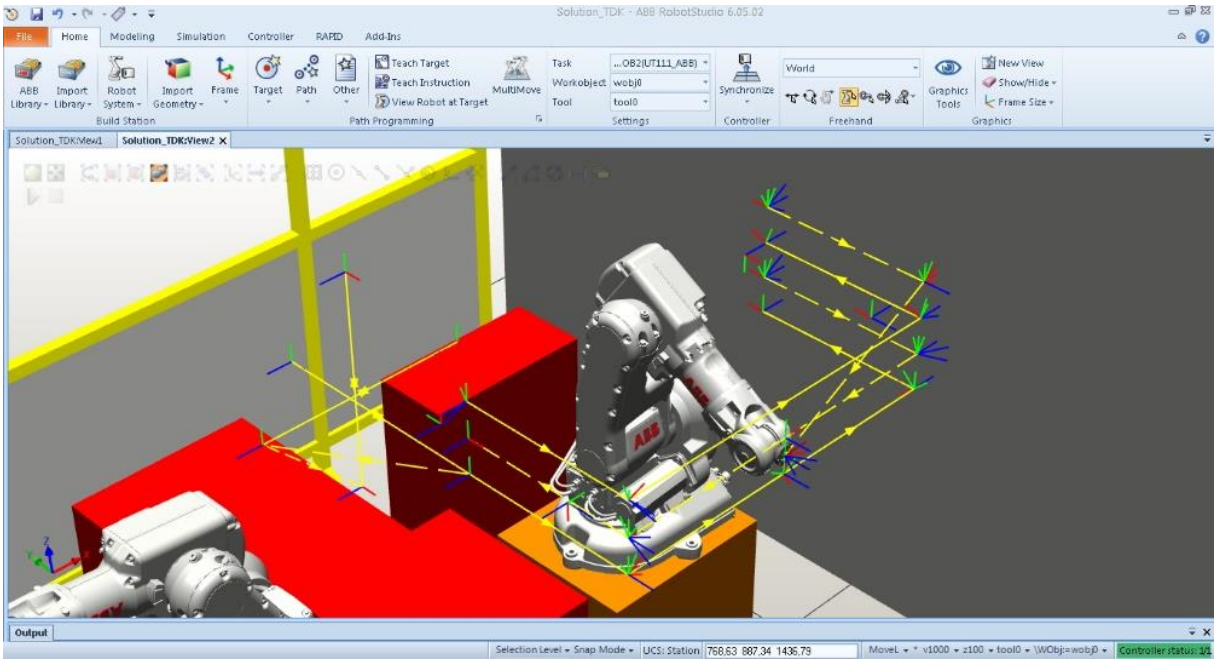
A bejárandó pálya és az orientáció váltások megszerkesztésére a RobotStudio-t használtuk, mely az ABB által biztosított szimulációs program a saját robotjaihoz. A program lehetőséget biztosít az ütközésvizsgálatra is, így biztosak lehettünk abban, hogy nem okozunk kárt a robotcellában. Az első feladatunk a robotcella méreteinek meghatározása volt, majd az egyes elemek megszerkesztése, és a cella összeállítása. Az 5. ábrán a robot laboratórium robotcelláját láthatjuk RobotStudio-ban megvalósítva.



5. ábra: Robotcella

A mérésekhez csak a 02-es jelzésű ABB robotot használtuk. A mérést több pályára osztottuk, ahol külön vizsgáltuk az ASSN tengelyeit. Az első három szinten egytengelyes orientációs méréseket végeztünk kitüntetett pontokban, majd a negyedik szinten komplex mozgásokat is megvalósítottunk. A mérési útvonal második szakaszában, kifejezetten a lineáris gyorsulást vizsgáltuk. Ezen a szakaszon tengelyirányú mozgást valósítottunk meg, több különböző robotsebesség mellett.

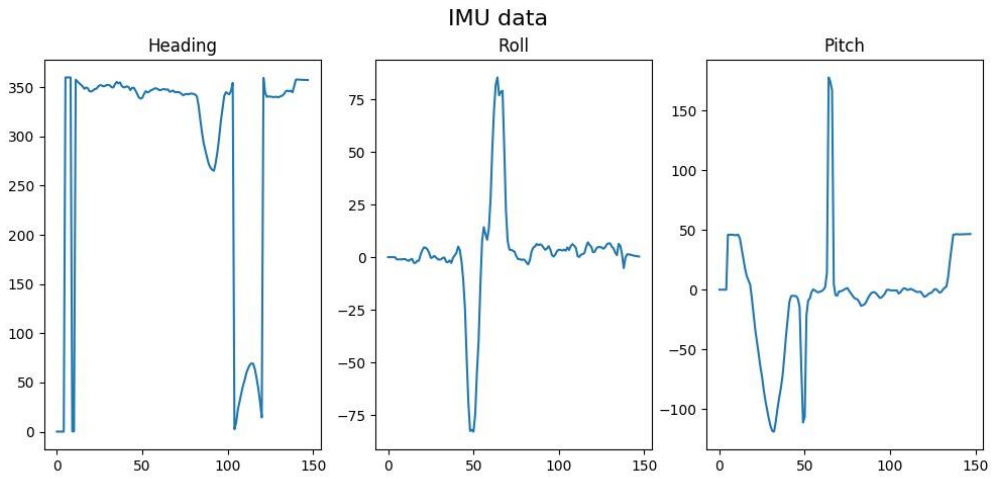
A robotra írt programunk a szimuláció alapján készült el RAPID nyelven [8] melyet részben a szoftver generált számunkra, részben magunk szerkesztettük. Ennek futtatása a robothoz kapcsolt IRC5-ös vezérlőn keresztül történt. A 6. ábrán az általunk programozott robotpálya látható, melyben jól elkülöníthető a négy szint, valamint a lineáris gyorsulás vizsgálatára írt szakasz. A robotprogram (5. melléklet) megírása során figyelmet fordítottunk arra, hogy minden mérendő tengelyt megmozgassunk, és átfogó képet kapjunk a szenzor méréseiről.



6. ábra: Szimulációs utak a RobotStudio-ban

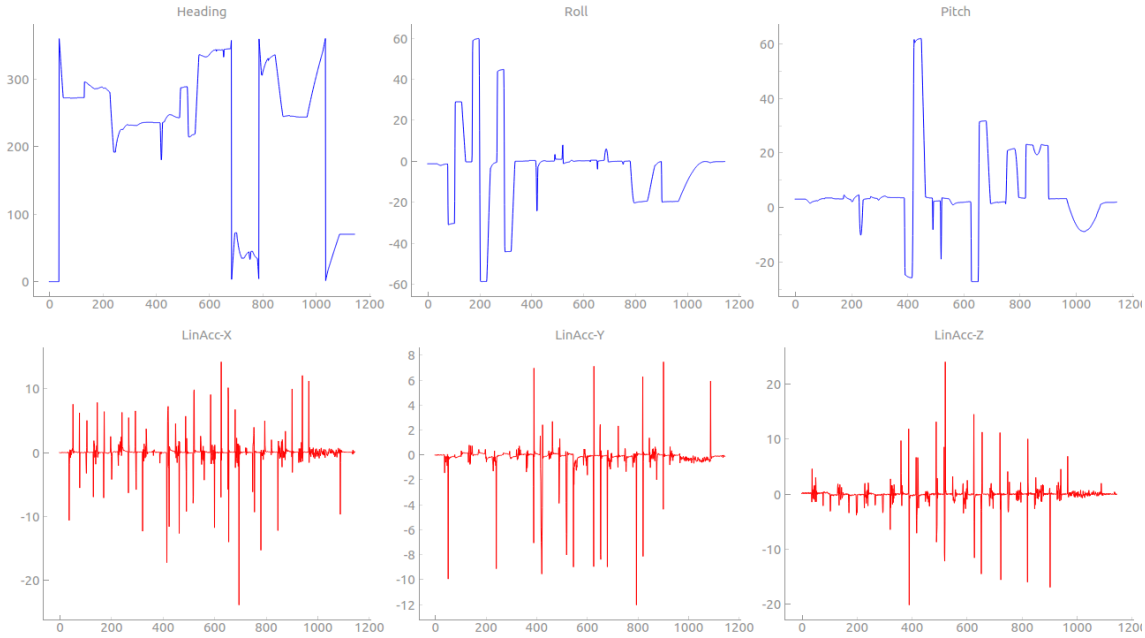
Vizualizáció és adatfeldolgozás

A sorosan érkező szenzor adatokat azonnal feldolgozva és naplózva kell megoldanunk a vizualizációt, hiszen a naplózott adatokból később fejleszthetjük rendszerünket, valamint visszamenőleg szimulálhatjuk az autó korábbi működését, mozgását. Az adatfeldolgozás és a vizualizáció mind ’python’ nyelven íródott, mert itt tudtuk legrugalmasabban megvalósítani a fejlesztés korai szakaszát. Az adatok olvasása során szét kellett választani a különböző típusú adatokat, ezért több dinamikusan növekvő tömb lett létrehozva, melyek később még optimalizálhatók lesznek. Első megközelítésünk alkalmával, a *matplotlib* könyvtárat használtuk az adatok valós idejű megjelenítésére (7. ábra), azonban háromnál több adat egyidejű vizsgálatánál az animáció kezelő nem boldogult a nagy adatmennyiség megjelenítésével.



7. ábra: MatPlotLib vizualizáció

A vizualizáció számára olyan megoldást kellett keresnünk, mely rendkívül gyorsan tudja frissíteni a képet, hiszen a szenzorból érkező 16 Hz-es információáradatból képzett tömbjeink rövid időn belül óriási méretűvé válnak. A feladatra kisebb keresés után a *PyQtGraph*-ot [9] találtuk a legalkalmasabbnak, mely önálló grafikus ablak megjelenítő motorral rendelkezik, és beváltotta a hozzáfűzött reményeket, mert ennek segítségével az összes szenzoradatunkat egyszerre, élőben láthattuk (8. ábra). A vizualizációs-adatfeldolgozó kód a mellékletben megtalálható (1. melléklet).



8. ábra: PqQtGraph vizualizáció

Mérési összeállítás

Raspberry Pi3 - Arduino Uno

A modelljármű alapja egy Raspberry Pi3 mikroszámítógép. A mérések során szerettük volna ezzel a mikroszámítógéppel megvalósítani a vizualizációt, mert így egy teljesen wireless, mobil mérőegységként állt volna össze a rendszer, melyet VNC-n keresztül kívülről elérünk. Két lehetséges kommunikáció közül választhattunk: UART vagy I^2C. Az UART interfészt egy másik eszköz, a lidar számára tartjuk fent a projekt keretein belül, ezért az I^2C-t kellett használnunk. A kezdeti méréseink bebizonyították, hogy ha a mikroszámítógépen található dedikált I^2C pineket szeretnénk használni, abban az esetben sajnos úgynevezett Clock Stretching[1] alkalmazása szükséges [10]. Erre azonban a Raspberry nem alkalmas a BNO szenzorunkkal kapcsolatban. Sajnos a vizualizációhoz a Raspberry teljesítménye kevésnek bizonyult, így USB-n keresztül kellett az Arduino Uno-t működtetni, és a Raspberry-t teljesen kivenni a mérő körből. Mivel a modelljárművön található, más feladatok elvégzésére is egy Arduino, és voltak szabad pinek még, adódott a lehetőség, hogy az Arduinoval valósítsuk meg a szenzor összeköttetést. Az Atmel-board és az Arduino közötti összeköttetés lábkiosztását az alábbi, 2. táblázat mutatja:

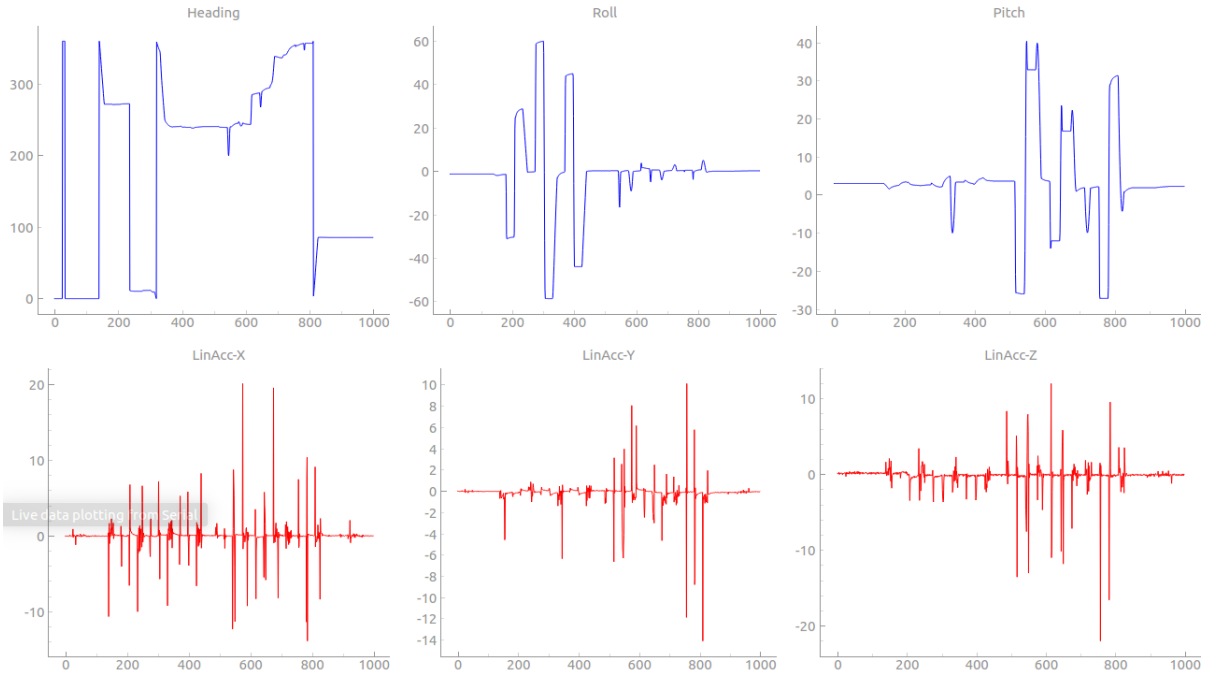
[1] Clock Stretching: Az I^2C kommunikáció során a master határozza meg az órajel sebességét, amihez a slave szinkronizál. Azonban előfordulhat olyan eset, ahol a slave nem alkalmas együttműködni a master által szolgáltatott órajellel. Erre szolgál megoldásként a Clock Stretching.

Atmel BNO055 Xplained Pro		Arduino Uno	
SCL	Pin 12	Analog 5	Pin A5
SDA	Pin 11	Analog 4	Pin A4
VCC	Pin 20	5V DC	Pin 5V
Ground	Pin 19	Ground	Pin GND

2. táblázat: Lábkiosztás

Az ASSN és az Arduino programkönyvtárait felhasználva, egy saját kódot írtunk a kalibráláshoz, és a méréseinkhez. Ennek a programnak a feladata, hogy az általunk kívánt szenzor adatokat kiolvassa, majd továbbítsa a számítógép felé, ahol további adatfeldolgozást valósítunk meg python nyelven. A kalibrálás folyamata az Arduinon keresztül történik. A chipnek vannak olyan üzemmódjai, ahol a kalibrációt saját maga elvégezi. Mi is ilyen üzemmódban mérünk, ellenben szeretnénk a projekt keretein beül más üzemmódokat is használni, ahol a kalibráció elvégzése szükséges. Erre a célra egy külön programkódot készítettünk. (2. melléklet) Ilyenkor a programkódunk 4 regiszter értékét figyeli, melyek 0-3 közötti értékeket adnak vissza. A 0 – nem kalibrált, 3 – teljesen kalibrált állapotot jelzi. A giroszkóp, a magnetométer, valamint a gyorsulásmérő is más-más kalibrálási eljárást igényel. A giroszkóp beállítása a legegyszerűbb, mert az eszközt elég csak pár másodperce nyugalomban hagyni, a giroszkóp kalibrálása elkészül. A magnetométert egy nyolcas alakú pályán kell mind a 3 irányban mozgatni, ahhoz, hogy a kalibrálási szintje elérje a maximumot. A gyorsulásmérő kalibrálásához az eszközt több dőlési szögben is, pár másodpercig tartani kell. Legegyszerűbb, ha egy képzeletbeli kocka formájú tárgy minden oldalára ráhelyezzük egy kis időre. A szenzor mérésekhez használt program kódja a 3.mellékletben látható. Az üzemmód választását jelenleg a ’header’ fájlunkban valósítjuk meg. A méréshez használt kódban, megtörténik a kalibrációs adatok kiolvasása az EEPROM-ból, ha léteznek és szükségesek, majd megkezdődik az aktuális szenzor adatok továbbítása, minden adathalmazhoz időbélyeget rendelve.

Mérési eredmények



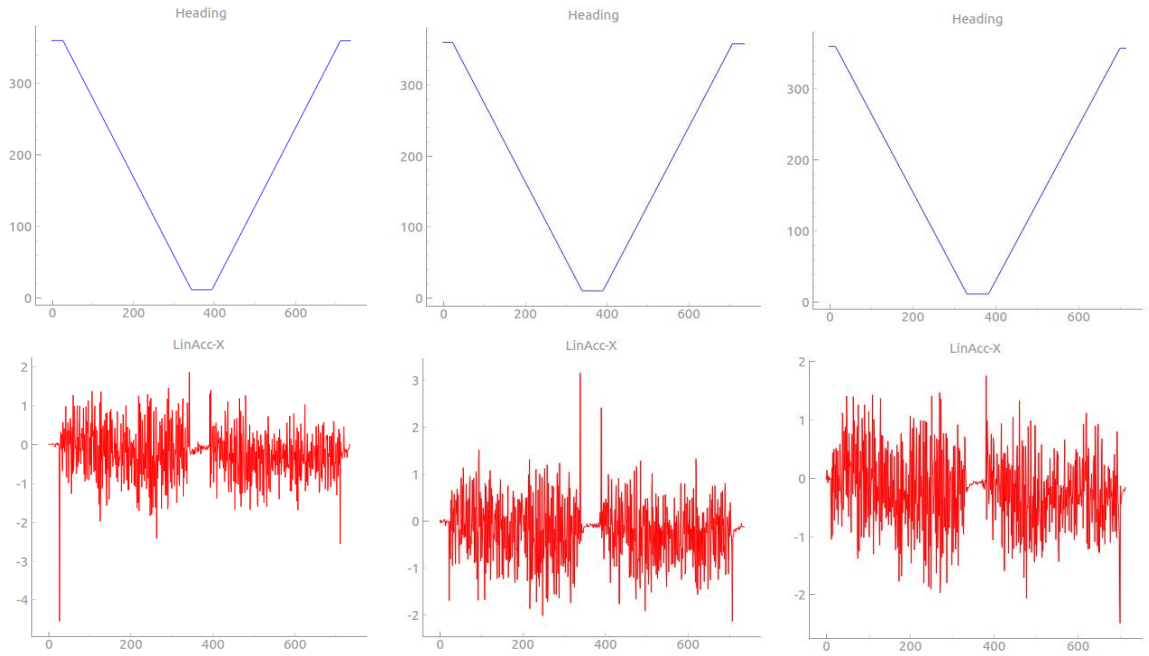
9. ábra Mérések monitorozása

A méréseinket folyamatosan, monitorozva (9. ábra) tudtuk végig kísérni, köszönhetően az erre a célra írt vizualizációs programkódnak. Ez a kód tartalmaz egy naplózást is így a későbbiekben elérhetőek a már mért eredményeink.

A méréshez használt robotprogramkódunk úgy lett kialakítva, hogy egyes részletei egyik esetben az orientációt vizsgálják, máskor pedig a gyorsulásmérést. Az egyes részleteket külön-külön is képesek voltunk vizsgálni, de ettől függetlenül ezek a szakaszok képesek egy ciklusban is lefutni. Az orientációhoz, valamint a gyorsulásmérőhöz írt programrészleteket is minden esetben háromszor, egymás után futtattuk, majd az így kapott eredményeket összehasonlítottuk. Mivel minden esetben ugyanazon pályán haladt végig a robot, így azonos eredményeket vártunk a mérések során és ez biztosította a lehetőséget az értékek egyeztetéséhez. A robot és a naplózás indítása között jelenleg nem sikerült fix kapcsolatot létesíteni, így a naplófájlok egymáshoz képest alkalmanként időben eltolva adják vissza az elvárt eredményeket, de manuális igazítással a naplófájlok eredményei kellőképpen összemérhetők.

Heading

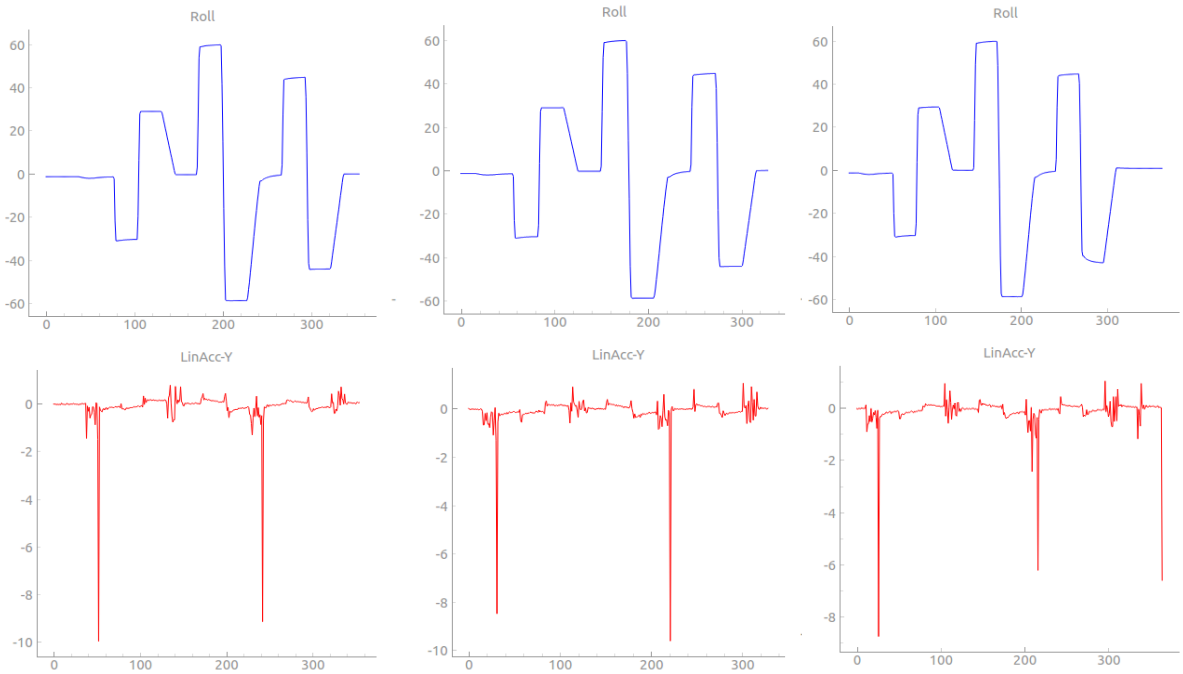
A projekt során használt modell aktuális irányát meghatározni kívánt szenzorértékünk a „heading”. Ez az érték a chip ’Z’ tengelye körüli elmozdulást vizsgálja. Tulajdonképpen megmutatja, hogy az eszköz milyen irányba néz. Ennek vizsgálata kiemelt szerepet játszott, mert a későbbi céljaink közt szereplő navigációhoz elengedhetetlen. A méréseket a robotkar J1 tengelyének egyenletes sebességű forgatásával végeztük, mely 360 fokos tartományban is képes volt a mérések futtatására. A három mérés elvégzését követően 360° megtétele után az első és az utolsó mért értékünk között átlagosan 1,6 fok eltérést tapasztaltunk. A vizualizált eredményeink (10. ábra) megmutatták, hogy valóban lineáris változás zajlott a folyamat során, és az ismételt mérés alakhűséget mutatott a várt ábrában.



10. ábra: Heading vizsgálata 360 fokos tartományban

Roll

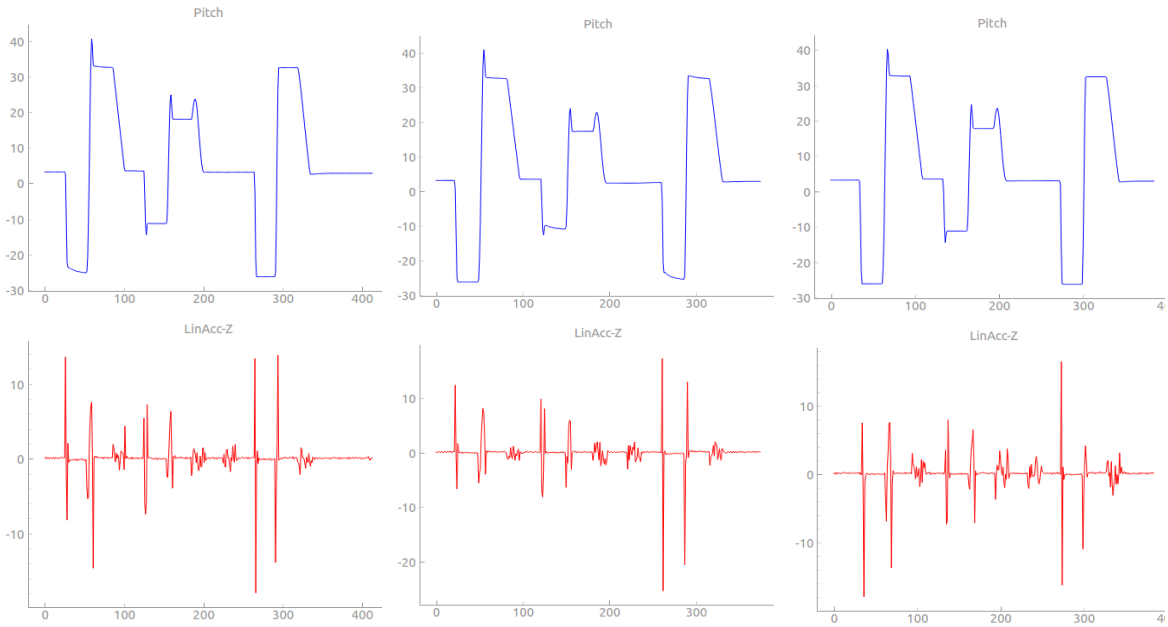
Ez az érték az ASSN ’Y’ tengely körüli változást méri. Megmutatja, hogy a chip a vízszinthez képest mennyire borult el jobb vagy bal oldalra. A roll értékeinek változását hat különböző szögben is vizsgáltuk. A program 0° dőlési szögből pozitív és negatív irányba is mozgatta a chipet. Az ismételten elvégzett méréseink alapján megállapítottuk, hogy a kapott értékeink egymáshoz viszonyítva maximálisan 1.19° különbséget mutattak. A egymást követő negatív és pozitív programozott állások, azonos amplitúdóval, de ellentétes irányban jelentek meg a vizualizáció során a bázisponthoz képest ezáltal a 11.ábránkon látható a nagymértékű szimmetria.



11. ábra: Roll eredményeinek vizualizációja

Pitch

Magyarul szokás bólogatásnak is nevezni. Az eszközünk ’X’ tengelyen való elfordulást méri. A pitch vizsgálata során a robotpálya miatt előfordultak olyan mérési pontok is, melyeket a robot csak úgy tud elérni, ha az előre beállított szöghöz képest, azon kis mértékben túlhaladt. Ennek oka, hogy voltak műveletek, ahol nem lehetett az adott pontban egyetlen tengely mozgatásával beállni a kívánt pozícióba, ezért többtengelyes mozgást kellett alkalmaznunk, mely sajnos túllövésekkel járt alkalmanként. Ezek a túllövések jól láthatók a 12. ábrán. Sajnos a kalibráció elvégzésétől eltekintve a „pitch” értékeink ofszetje nem volt megfelelő a robotkaros mérések során, ezért nullponteltolódást tapasztaltunk. Ezek a grafikus ábráinkon is jól láthatók. Ettől eltekintve a mérések bebizonyították, hogy a „pitch” tengelyre vonatkozóan a nullponthoz képest azonos arányban tértek ki az eredményeink, mind negatív, mind pozitív szögben elmozdítva.

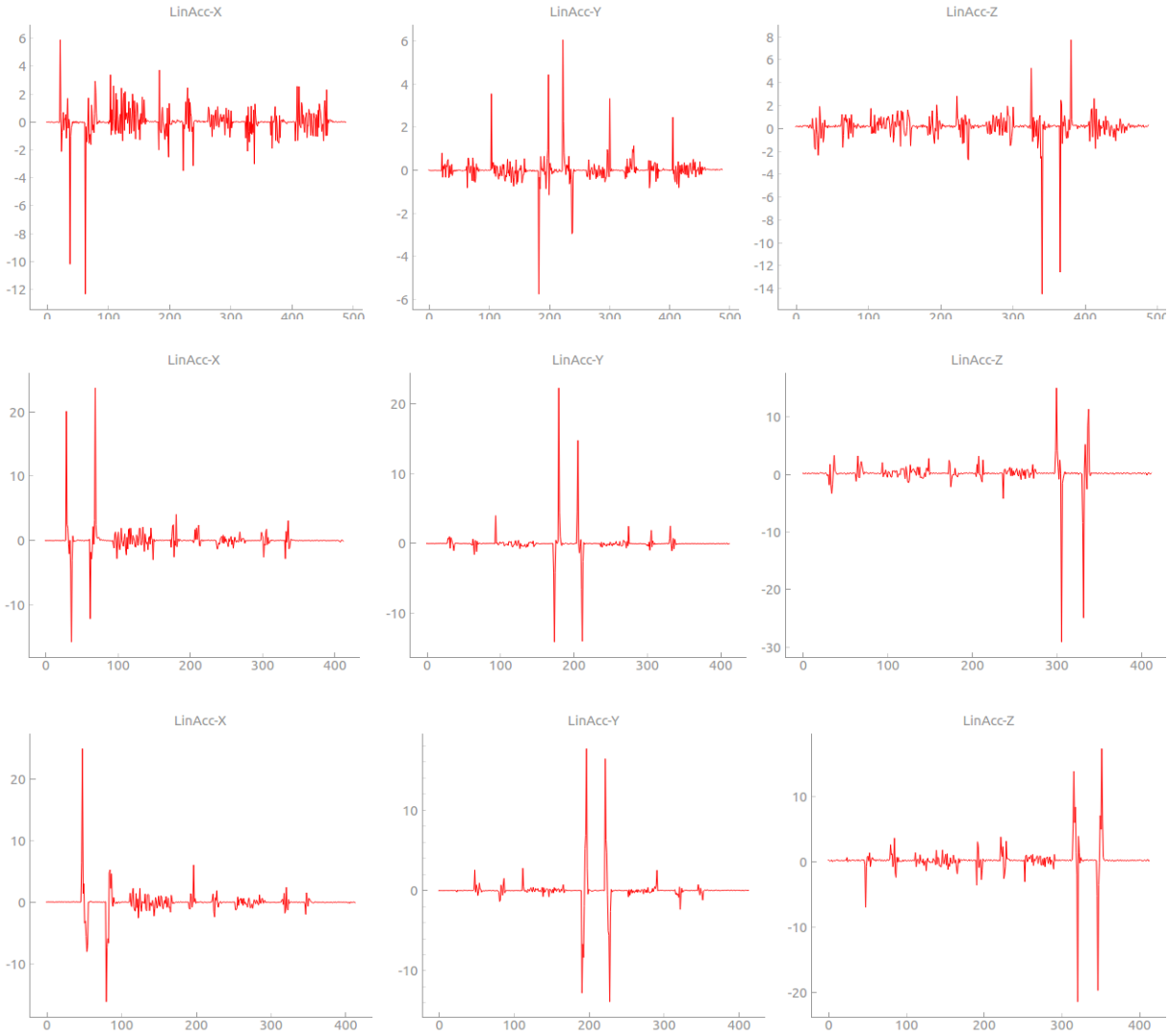


12. ábra: Pitch vizualizációja

Lineáris gyorsulás

Ezen mérésünk során az IRB 140-es robotunkat három különböző sebességre programoztuk, miáltal képesek voltunk három egymástól eltérő gyorsulást szimulálni teljesen azonos szakaszon, és teljesen azonos mérési körülmények között. A 13. ábrán a három mérés összehasonlítása látható, melyből az alábbi következtetések vonhatók le:

- A robotkar gyorsítása és lassítása nem azonos mértékű, ezért ezen funkció vizsgálata további kutatást igényel.
- Annak ellenére, hogy a gyorsulást közvetlenül nem tudjuk befolyásolni, a sebesség növelésével, melyet a robotnak el kell érnie, nő a gyorsulása is.
- Minél nagyobb gyorsulások mérése esetén a közbenső mozgások gyorsulása egyre elhanyagolhatóbb



13. ábra: Lineáris gyorsulás vizsgálata

Összegzés

A vizsgált ASSN további méréseket és kalibrálást követően kiválóan alkalmas lesz a navigálás megalapozására egy önvezető járműben, mert már jelenleg is gyors és pontos működést mutat, mely csak még kifinomultabbá válhat. A fejlesztése során igyekszünk minimalizálni a mérések során felmerülő megoldatlan problémákat, ez azonban további feladatokat jelent a jövőre nézve.

A projekt során számos új területtel ismerkedtünk meg, melyek egyesével is sok kihívást tartogatnak. Újabb és újabb megoldandó feladatokba ütközünk a kísérletek alkalmával, de ez egy fejlesztési ciklus során mindennapos. A felmerülő hibákból minden alkalommal sikerült tanulni, és megoldásuk egyre nagyobb motivációt adott a további feladatok elvégzéséhez.

Rengeteget fejlődtünk az alábbi témákkal kapcsolatban:

- Kommunikációs interfészek illesztése
- Robotok programozása – RAPID nyelv
- Robot műveletek szimulációja
- Python nyelvű vizualizációs szoftver készítés
- C nyelvű programozás
- Dokumentálás

Távlati célok

Mérések pontosítása

A mérések során felmerülő és megismert problémák javítása, annak érdekében, hogy a szenzort még pontosabban megismerhessük, és alkalmazásának profilját szélesíthessük. Az adatfeldolgozás és a rendszer optimalizálása is kisebb ráfordítást igényel még a jövőben.

Navigáció az autonóm modellben

Az autonóm jármű szakkör keretein belül, jelenleg fejlesztés alatt álló 1:6 arányú SZElectricity versenyautójának navigálása az ASSN segítségével. Célunk, hogy más szenzorokkal együttesen felszerelve az autó képes legyen önvezetésre, beltérben is, ahol GPS jelek nélkül kell megvalósítanunk a vezetési feladatokat.

Kanyarrészlet szimulálása

A későbbiekben szeretnénk egy fizikailag is létező kanyart, pályarészletet is szimulálni. Az ABB robotkarra egy olyan programkódot tervezünk, mely a chipet úgy mozgatná, mintha az valóban egy versenyautó, pályán történő haladását érzékelné. Ezáltal vizsgálni tudjuk majd, hogy egyforma pályarészleten többször is végig mozgatva az eszközt, ugyanazon értékeket kapjuk-e vissza minden esetben. Értelemszerűen minél több mérést végezünk, annál több adatot gyűjthetünk, melyeket átlagolva lehetőségünk van az adott szakaszhoz egy optimális értékkészletet meghatározni.

Neurális háló tanítása a szimulációs eredményekből

A neurális háló egy tanulási algoritmust futtatva tudja saját magát egyre pontosabbá és megbízhatóbbá fejleszteni. Ezenkívül rendelkezik egy olyan algoritmussal, mely a már megtanult információ visszahívására szolgál. Ezen két tulajdonsága miatt, a következőkben a neurális hálót nem csak képi bemenetekkel, hanem a saját mérési adatainkkal is szeretnénk bővíteni. Így biztosítva azt, hogy a hardverek által szolgáltatott információ ne csak a működést tegye lehetővé, hanem a saját rendszerünk fejlesztésében is szerepet játsszon.

Irodalomjegyzék

[1] BNO055 Design dokumentáció:
https://www.tme.eu/fr/Document/62ba86ae006faa8218db25920c9ed99f/ATBNO055-XPRO-DTE.pdf
utolsó látogatás: 2018. április 9.

[2] BNO055 Chip Datasheet:
https://ae-bst.resource.bosch.com/media/_tech/media/datasheets/BST_BNO055_DS000_14.pdf
utolsó látogatás: 2018. április 9.

[3] Society of Automotive Engineers (SAE) szintek:
http://cyberlaw.stanford.edu/blog/2013/12/sae-levels-driving-automation
utolsó látogatás: 2018. április 9.

[4] IMU integrálása:
http://www.oxts.com/technical-notes/why-it-is-necessary-to-integrate-an-inertial-measurement-unit-with-imaging-systems-on-an-autonomous-vehicle/
utolsó látogatás: 2018. április 9.

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

[6] ROS rendszer információ:
http://wiki.ros.org
utolsó látogatás: 2018. április 9.

[7] ABB robot datasheet:
https://search-ext.abb.com/library/Download.aspx?DocumentID=PR10031EN_R15&LanguageCode=en&DocumentPartId=&Action=Launch
utolsó látogatás: 2018. április 9.

[8] RAPID programozási leírás:
https://library.e.abb.com/public/688894b98123f87bc1257cc50044e809/Technical%20reference%20manual_RAPID_3HAC16581-1_revJ_en.pdf
utolsó látogatás: 2018. április 9.

[9] PyQtGraph információ
http://www.pyqtgraph.org
utolsó látogatás: 2018. április 9.

[10] Clock-stretching:
https://www.i2c-bus.org/clock-stretching/
utolsó látogatás: 2018. április 9.

Ábrajegyzék

[A1] https://atmelcorporation.files.wordpress.com/2015/02/xplained.png

(utolsó látogatás időpontja: 2018. március 30.)

[A2] https://ae-bst.resource.bosch.com/media/_tech/media/datasheets/BST_BNO055_DS000_14.pdf

(utolsó látogatás időpontja: 2018. március 30.)

[A3] http://img.deusm.com/designnews/2016/03/280168/01-autonomous-car-sensors_TI.png
(utolsó látogatás időpontja: 2018. március 30.)

[A4] http://www.ros.org/wp-content/uploads/2013/12/ros_equation.png
(utolsó látogatás időpontja: 2018. március 30.)

Mellékletek

1. melléklet:

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

```
###Live data plot for BNO055 IMU
###Deep-Pilot Project
###Peter Gulyas, Mate Agoston

from pyqtgraph.Qt import QtGui, QtCore
import numpy as np
import pyqtgraph as pg
from pyqtgraph.ptime import time
import serial
      import csv

      # Enable antialiasing for prettier plots
      pg.setConfigOptions(antialias=True)
      # Set background color to white
      pg.setConfigOption('background', 'w')

      ser=serial.Serial("/dev/ttyACM0",230400)
      ser.flushInput()

      tp = []
      hp = []
      rp = []
      pp = []
      lxp = []
      lyp = []
      lzp = []

      tv=None
      hv=None
      rv=None
      pv=None
      lxv=None
      lyv=None
      lzv=None

      H1=2018
      H2=2018
      H3=2018
```

```
39.      H4=2018
40.      datarecovery = 0
41.
42.      ser.flushInput()
43.      while H1 != 0:
44.          input1 = ser.readline()
45.          data1 = input1
46.          data1 = str(data1)
47.          data1 = data1[2:-5]
48.          H1 = len(data1)
49.          #print(H1)
50.          while H2 != 18:
51.              input2 = ser.readline()
52.              data2 = input2
53.              data2 = str(data2)
54.              data2 = data2[2:-5]
55.              H2 = len(data2)
56.              #print(H2)
57.              while H3 != 18:
58.                  input3 = ser.readline()
59.                  data3 = input3
60.                  data3 = str(data3)
61.                  data3 = data3[2:-5]
62.                  H3 = len(data3)
63.                  #print(H3)
64.                  while H4 != 18:
65.                      input4 = ser.readline()
66.                      data4 = input4
67.                      data4 = str(data4)
68.                      data4 = data4[2:-5]
69.                      H4 = len(data4)
70.                      #print(H4)
71.      ###Create basic CSV file, where log will be written, Writing a header, to define coulumns
72.      with open('log_t_tdk_mereshead2.csv', 'w') as csvfile:
73.          BNOlogw = csv.writer(csvfile, delimiter=' ', quotechar='|', quoting=csv.QUOTE_MINIMAL)
74.          BNOlogw.writerow(['Time,Heading,Roll,Pitch,LinAcc_X,LinAcc_Y,LinAcc_Z'])
75.      print('CSV Done. Start logging')
76.
77.      app = QtGui.QApplication([])
78.
79.      win = pg.GraphicsWindow(title="BNO055 live plotting")
80.      win.resize(1000,800)
```

```
81.       win.setWindowTitle('Live data plotting from Serial')
82.
83.       p1 = win.addPlot(title="Heading")
84.       curve1 = p1.plot(pen='b')
85.       p2 = win.addPlot(title="Roll")
86.       curve2 = p2.plot(pen='b')
87.       p3 = win.addPlot(title="Pitch")
88.       curve3 = p3.plot(pen='b')
89.       win.nextRow()
90.
91.       p4 = win.addPlot(title="LinAcc-X")
92.       curve4 = p4.plot(pen='r')
93.       p5 = win.addPlot(title="LinAcc-Y")
94.       curve5 = p5.plot(pen='r')
95.       p6 = win.addPlot(title="LinAcc-Z")
96.       curve6 = p6.plot(pen='r')
97.       win.nextRow()
98.
99.       def update():
100.          input5 = ser.readline()
101.          data5 = input5
102.          data5 = str(data5)
103.          data5 = data5[2:-5]
104.         ## print(data5)
105.          ###Create a recovery value from the last suitable serial read
106.          if len(data5) >= 41:
107.              datarecovery == data5
108.          ###Use the recovery value to continue plotting, while bad serial read
109.          if len(data5) <41:
110.              data5 = datarecovery
111.          t,h,r,p,lx,ly,lz, = data5.split(',')
112.          tp.append(t)
113.          hv=(float(h))
114.          hp.append(hv)
115.          rv=(float(r))
116.          rp.append(rv)
117.          pv=(float(p))
118.          pp.append(pv)
119.          lxv=(float(lx))
120.          lxp.append(lxv)
121.          lyv=(float(ly))
122.          lyp.append(lyv)
```

```
123.          lzv=(float(lz))
124.          lzp.append(lzv)
125.          log_data=t,',',hv,',',rv,',',pv,',',lxv,',',lyv,',',lzv
126.          with open('log_t_tdk_mereshead2.csv', 'a') as csvfile:
127.              BNOlogw = csv.writer(csvfile, delimiter=' ', quotechar='|', quoting=csv.QUOTE_MINIMAL)
128.              BNOlogw.writerow(log_data)
129.          curve1.setData(hp)
130.          curve2.setData(rp)
131.          curve3.setData(pp)
132.          curve4.setData(lxp)
133.          curve5.setData(lyp)
134.          curve6.setData(lzp)
135.
136.          app.processEvents()
137.
138.      timer = QtCore.QTimer()
139.      timer.timeout.connect(update)
140.      timer.start(0)
141.
142.      if __name__ == '__main__':
143.          import sys
144.          if (sys.flags.interactive != 1) or not hasattr(QtCore, 'PYQT_VERSION'):
145.              QtGui.QApplication.instance().exec_()
```

3. melléklet

```
//BNO055 ASSN calibration
//Deep-Pilot Project
//Peter Gulyas, Mate Agoston

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

/* Connections
   ===========
   Connect SCL to analog 5
   Connect SDA to analog 4
   Connect VCC to 3-5V DC
   Connect GROUND to GND
*/
// Set the delay between fresh samples
#define BNO055_SAMPLERATE_DELAY_MS (100)

void displaySensorDetails(void);
void displaySensorStatus(void);
void displayCalStatus(void);
void displaySensorOffsets(const adafruit_bno055_offsets_t &calibData);

Adafruit_BNO055 bno = Adafruit_BNO055(55);

//**************************************************************************
/*
  Arduino setup function (automatically called at startup)
*/
//**************************************************************************
void setup(void)
{
    Serial.begin(57600);
    delay(1000);
    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("Ooops, 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;

//  Look for the sensor's unique ID at the beginning oF EEPROM.
//  This isn't foolproof, but it's better than nothing.

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);

    displaySensorOffsets(calibrationData);
}
delay(1000);

// Display some basic information on this sensor
displaySensorDetails();

// Optional: Display current status
displaySensorStatus();
```

```
    bno.setExtCrystalUse(true);

    sensors_event_t event;
    bno.getEvent(&event);
    Serial.println("Please Calibrate Sensor: ");
    while (!bno.isFullyCalibrated())
    {
        bno.getEvent(&event);

        Serial.print("X: ");
        Serial.print(event.orientation.x, 4);
        Serial.print("\tY: ");
        Serial.print(event.orientation.y, 4);
        Serial.print("\tZ: ");
        Serial.print(event.orientation.z, 4);

        // Optional: Display calibration status
        displayCalStatus();

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

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

    Serial.println("\nFully calibrated!");

    /* Actual Calibration saved in EEPROM:
    Fully calibrated!
    --------------------------------
    Calibration Results:
    Accelerometer: 65521 65504 13
    Gyro: 65535 4 0
    Mag: 65499 65493 466
    Accel Radius: 1000
    Mag Radius: 773

    Storing calibration data to EEPROM...
    Data stored to EEPROM.

    --------------------------------
```

```
    */

    Serial.println("----------------------------------");
    Serial.println("Calibration Results: ");
    adafruit_bno055_offsets_t newCalib;
    bno.getSensorOffsets(newCalib);
    displaySensorOffsets(newCalib);

    Serial.println("\n\nStoring calibration data to EEPROM...");

    eeAddress = 0;
    bno.getSensor(&sensor);
    bnoID = sensor.sensor_id;

    EEPROM.put(eeAddress, bnoID);

    eeAddress += sizeof(long);
    EEPROM.put(eeAddress, newCalib);
    Serial.println("Data stored to EEPROM.");

    Serial.println("\n----------------------------------\n");
    delay(500);
}

void loop()
{}
//**************************************************************************
/*
  Displays some basic information on this sensor from the unified
  sensor API sensor_t type (see Adafruit_Sensor for more information)
*/
//**************************************************************************
void displaySensorDetails(void)
{
    sensor_t sensor;
    bno.getSensor(&sensor);
    Serial.println("------------------------------------");
    Serial.print("Sensor:       "); Serial.println(sensor.name);
    Serial.print("Driver Ver:   "); Serial.println(sensor.version);
    Serial.print("Unique ID:    "); Serial.println(sensor.sensor_id);
    Serial.print("Max Value:    "); Serial.print(sensor.max_value); Serial.println(" xxx");
    Serial.print("Min Value:    "); Serial.print(sensor.min_value); Serial.println(" xxx");
```

```
    Serial.print("Resolution:   "); Serial.print(sensor.resolution); Serial.println(" xxx");
    Serial.println("------------------------------------");
    Serial.println("");
    delay(500);
}
//**************************************************************************
/*
  Display some basic info about the sensor status
*/
//**************************************************************************
void displaySensorStatus(void)
{
// Get the system status values (mostly for debugging purposes)
    uint8_t system_status, self_test_results, system_error;
    system_status = self_test_results = system_error = 0;
    bno.getSystemStatus(&system_status, &self_test_results, &system_error);

// Display the results in the Serial Monitor
    Serial.println("");
    Serial.print("System Status: 0x");
    Serial.println(system_status, HEX);
    Serial.print("Self Test:     0x");
    Serial.println(self_test_results, HEX);
    Serial.print("System Error:  0x");
    Serial.println(system_error, HEX);
    Serial.println("");
    delay(500);
}
//**************************************************************************
/*
  Display sensor calibration status
*/
//**************************************************************************
void displayCalStatus(void)
{
// Get the four calibration values (0..3)
// Any sensor data reporting 0 should be ignored,
// 3 means 'fully calibrated"
    uint8_t system, gyro, accel, mag;
    system = gyro = accel = mag = 0;
    bno.getCalibration(&system, &gyro, &accel, &mag);
```

```
    // The data should be ignored until the system calibration is > 0 */
    Serial.print("\t");
    if (!system)
    {
        Serial.print("! ");
    }

    // Display the individual values
    Serial.print("Sys:");
    Serial.print(system, DEC);
    Serial.print(" G:");
    Serial.print(gyro, DEC);
    Serial.print(" A:");
    Serial.print(accel, DEC);
    Serial.print(" M:");
    Serial.print(mag, DEC);
}

//**************************************************************************
/*
  Display the raw calibration offset and radius data
*/
//**************************************************************************
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);
}
```

4. melléklet

```
//BNO055 ASSN sensor data reading
//Deep-Pilot Project
//Peter Gulyas, Mate Agoston

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

// Set the delay between fresh samples
#define BNO055_SAMPLERATE_DELAY_MS (60)

void displaySensorDetails(void);
void displaySensorStatus(void);
void displayCalStatus(void);
void displaySensorOffsets(const adafruit_bno055_offsets_t &calibData);

Adafruit_BNO055 bno = Adafruit_BNO055(55);

void setup(void)
{
    Serial.begin(230400);
    // 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("Ooops, 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;

    // Look for the sensor's unique ID at the beginning oF EEPROM.
    // This isn't foolproof, but it's better than nothing.

    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);

        displaySensorOffsets(calibrationData);
    }

    delay(1000);

    // Display some basic information on this sensor
    displaySensorDetails();

    // Optional: Display current status
    displaySensorStatus();

    bno.setExtCrystalUse(true);
}

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());
    // New line for the next sample
    Serial.println("");
    // Wait the specified delay before requesting nex data
    delay(BNO055_SAMPLERATE_DELAY_MS);
}

//**************************************************************************
/*
  Displays some basic information on this sensor from the unified
  sensor API sensor_t type (see Adafruit_Sensor for more information)
*/
//**************************************************************************
void displaySensorDetails(void)
{
    sensor_t sensor;
    bno.getSensor(&sensor);
    Serial.println("------------------------------------");
    Serial.print  ("Sensor:       "); Serial.println(sensor.name);
    Serial.print  ("Driver Ver:   "); Serial.println(sensor.version);
    Serial.print  ("Unique ID:    "); Serial.println(sensor.sensor_id);
    Serial.print  ("Max Value:    "); Serial.print(sensor.max_value); Serial.println(" xxx");
    Serial.print  ("Min Value:    "); Serial.print(sensor.min_value); Serial.println(" xxx");
    Serial.print  ("Resolution:   "); Serial.print(sensor.resolution); Serial.println(" xxx");
    Serial.println("------------------------------------");
    Serial.println("");
    delay(500);
}

//**************************************************************************
/*
```

```
  Display some basic info about the sensor status
*/
//**************************************************************************
void displaySensorStatus(void)
{
    // Get the system status values (mostly for debugging purposes)
    uint8_t system_status, self_test_results, system_error;
    system_status = self_test_results = system_error = 0;
    bno.getSystemStatus(&system_status, &self_test_results, &system_error);

    // Display the results in the Serial Monitor
    Serial.println("");
    Serial.print("System Status: 0x");
    Serial.println(system_status, HEX);
    Serial.print("Self Test:     0x");
    Serial.println(self_test_results, HEX);
    Serial.print("System Error:  0x");
    Serial.println(system_error, HEX);
    Serial.println("");
    delay(500);
}

//**************************************************************************
/*
  Display sensor calibration status
*/
//**************************************************************************
void displayCalStatus(void)
{
    // Get the four calibration values (0..3)
    // Any sensor data reporting 0 should be ignored,
    // 3 means 'fully calibrated"
    uint8_t system, gyro, accel, mag;
    system = gyro = accel = mag = 0;
    bno.getCalibration(&system, &gyro, &accel, &mag);

    // The data should be ignored until the system calibration is > 0
    Serial.print("\t");
    if (!system)
    {
        Serial.print("! ");
    }
```

```
    // Display the individual values
    Serial.print("Sys:");
    Serial.print(system, DEC);
    Serial.print(" G:");
    Serial.print(gyro, DEC);
    Serial.print(" A:");
    Serial.print(accel, DEC);
    Serial.print(" M:");
    Serial.print(mag, DEC);
}

//**************************************************************************
/*
  Display the raw calibration offset and radius data
*/
//**************************************************************************
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);
    Serial.println("");
}
```

5. melléklet

```
MODULE IMU_modul

    !   ABB IRB 140 sensor testing RAPID code
    !   Deep-Pilot Project - BNO055
    !   Peter Gulyas, Mate Agoston

    !   Target desciption:  Name:=[[Target world coordinates],[Terget orientation],[Target configuration],[External
joint configuration]]
    !   Acition description:    Move_type To_which_point,Velocity,Zone,Tool_used

    CONST extjoint EJ :=[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09];

    CONST robtarget L4P5:=[[1620,-20,740],[0.5,0.5,0.5,0.5],[1,0,0,0],EJ];
    CONST robtarget L4P5_p30:=[[1620,-20,740],[0.353553391,0.612372436,0.612372436,0.353553391],[1,0,0,0],EJ];
    CONST robtarget L4P5_pm30:=[[1620,-20,740],[0.612372436,0.353553391,0.353553391,0.612372436],[1,-1,1,1],EJ];
    CONST robtarget L4P4:=[[1150,-500,490],[0.707106781,0.707106781,0,0],[1,0,0,1],EJ];
    CONST robtarget L4P4_h30:=[[1150,-500,490],[0.683012702,0.683012702,0.183012702,0.183012702],[1,-1,1,1],EJ];
    CONST robtarget L4P4_hm30:=[[1150,-500,490],[0.683012702,0.683012702,-0.183012702,-0.183012702],[1,0,0,1],EJ];
    CONST robtarget L4P3:=[[1620,-500,740],[0.707106781,0.707106781,0,0],[1,1,-1,1],EJ];
    CONST robtarget L4P3_r30:=[[1620,-500,740],[0.683012702,0.683012702,-0.183012702,0.183012702],[1,1,0,1],EJ];
    CONST robtarget L4P3_rm30:=[[1620,-500,740],[0.683012702,0.683012702,0.183012702,-0.183012702],[1,1,-1,1],EJ];
    CONST robtarget L1P1:=[[650,0,440],[0.5,0.5,-0.5,-0.5],[-1,1,-1,0],EJ];
    CONST robtarget L1P2:=[[650,-500,440],[0.707106781,0.707106781,0,0],[0,1,-1,0],EJ];
    CONST robtarget L1P2_ym30:=[[650,-500,440],[0.683012702,0.683012702,-0.183012702,0.183012702],[0,1,-1,0],EJ];
    CONST robtarget L1P2_yp30:=[[650,-500,440],[0.683012702,0.683012702,0.183012702,-0.183012702],[0,1,-1,0],EJ];
    CONST robtarget L1P3:=[[1150,-500,440],[0.707106781,0.707106781,0,0],[1,2,-2,0],EJ];
    CONST robtarget L1P3_ym60:=[[1150,-500,440],[0.612372436,0.612372436,0.353553391,-0.353553391],[1,2,-2,0],EJ];
    CONST robtarget L1P3_yp60:=[[1150,-500,440],[0.612372436,0.612372436,-0.353553391,0.353553391],[1,2,-1,0],EJ];
    CONST robtarget L1P4:=[[1620,-500,440],[0.5,0.5,0.5,0.5],[1,1,-1,0],EJ];
    CONST robtarget L1P4_ym45:=[[1620,-500,440],[0.653281482,0.27059805,0.653281482,0.27059805],[1,1,-1,0],EJ];
    CONST robtarget L1P4_yp45:=[[1620,-500,440],[0.27059805,0.653281482,0.27059805,0.653281482],[1,1,0,0],EJ];
    CONST robtarget L1P5:=[[1620,-20,440],[0.5,0.5,0.5,0.5],[1,1,-1,0],EJ];
    CONST robtarget L1P1_LIN_Y:=[[650,640,440],[0.5,0.5,-0.5,-0.5],[-1,1,-1,0],EJ];
    CONST robtarget L1P1_LIN_X_Base:=[[390,420,440],[0.5,0.5,-0.5,-0.5],[-1,1,-1,0],EJ];
    CONST robtarget L1P1_LIN_X_End:=[[990,420,440],[0.5,0.5,-0.5,-0.5],[-1,1,-1,0],EJ];
    CONST robtarget L1P1_LIN_Z_base:=[[650,400,190],[0.5,0.5,-0.5,-0.5],[-1,1,-1,0],EJ];
    CONST robtarget L1P1_LIN_Z_top:=[[650,400,800],[0.5,0.5,-0.5,-0.5],[-1,0,0,0],EJ];
    CONST robtarget L3P1:=[[650,0,640],[0.5,0.5,-0.5,-0.5],[-1,1,-1,0],EJ];
```

```
    CONST robtarget L3P1_x20:=[[650,0,640],[0.405579788,0.579227965,-0.579227965,-0.405579788],[-1,0,0,0],EJ];
    CONST robtarget L3P2:=[[650,-500,640],[0.69636424,0.69636424,-0.122787804,0.122787804],[0,1,0,0],EJ];
    CONST robtarget L3P2_ym20:=[[650,-500,640],[0.564862521,0.806707284,-0.099600503,0.14224426],[0,0,0,0],EJ];
    CONST robtarget L3P4:=[[1620,-500,640],[0.405579788,0.579227965,0.579227965,0.405579788],[1,0,0,0],EJ];
    CONST robtarget L3P4_y30:=[[1620,-500,640],[0.405579788,0.579227965,0.405579788,0.579227965],[1,1,0,0],EJ];
    CONST robtarget L3P5:=[[1620,-20,640],[0.405579788,0.579227965,0.405579788,0.579227965],[1,1,-1,0],EJ];
    CONST robtarget L2P5:=[[1620,-20,540],[0.5,0.5,0.5,0.5],[1,1,-1,0],EJ];
    CONST robtarget L2P5_xm30:=[[1620,-20,540],[0.612372436,0.353553391,0.353553391,0.612372436],[1,1,-1,0],EJ];
    CONST robtarget L2P5_xp30:=[[1620,-20,540],[0.353553391,0.612372436,0.612372436,0.353553391],[1,0,0,0],EJ];
    CONST robtarget L2P4:=[[1620,-500,540],[0.5,0.5,0.5,0.5],[1,1,-1,0],EJ];
    CONST robtarget L2P4_xm15:=[[1620,-500,540],[0.732962913,0.562422224,0.232962913,0.303603179],[1,2,-2,0],EJ];
    CONST robtarget L2P4_xp15:=[[1620,-500,540],[0.562422225,0.732962914,0.303603178,0.232962912],[1,1,-1,1],EJ];
    CONST robtarget L2P3:=[[1150,-500,540],[0.707106781,0.707106781,0,0],[1,2,-2,0],EJ];
    CONST robtarget L2P2:=[[650,-500,540],[0.707106781,0.707106781,0,0],[0,1,-1,0],EJ];
    CONST robtarget L2P2_xm30:=[[650,-500,540],[0.866025404,0.5,0,0],[0,1,-1,0],EJ];
    CONST robtarget L2P2_xp30:=[[650,-500,540],[0.5,0.866025404,0,0],[0,0,0,0],EJ];
    CONST robtarget L2P1:=[[650,0,540],[0.5,0.5,-0.5,-0.5],[-1,1,-1,0],EJ];

PROC main()
        Roll_test;  !Process for roll testing course
        WaitTime 1.5;
        Pitch_test; !Process for pitch testing course
        WaitTime 1.5;
        Combined;   !Process for combined task course
        WaitTime 1.5;
        Slow_ori;   !Process for slowly moved orientation tests
        WaitTime 1.5;
        Lin_test_300;   !Process for testing linear acceleration with robot velocity 300 mm/s
        WaitTime 1.5;
        Lin_test_600;   !Process for testing linear acceleration with robot velocity 600 mm/s
        WaitTime 1.5;
        Lin_test_1000;  !Process for testing linear acceleration with robot velocity 1000 mm/s
        WaitTime 1.5;
        Lin_test_1500;  !Process for testing linear acceleration with robot velocity 1500 mm/s
        WaitTime 1.5;
        Lin_test_2000;  !Process for testing linear acceleration with robot velocity 2000 mm/s
        WaitTime 1.5;
ENDPROC

PROC Slow_ori()
        MoveJ L4P5,v100,fine,t_Gripper_Base;
        WaitTime 1.5;
```

```
        MoveL L4P5_p30,v100,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveJ L4P5_pm30,v100,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveJ L4P5,v100,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveJ L4P3,v100,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveL L4P3_r30,v100,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveL L4P3_rm30,v100,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveJ L4P3,v100,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveJ L4P4,v100,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveL L4P4_h30,v100,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveL L4P4_hm30,v100,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveJ L4P4,v100,fine,t_Gripper_Base;
        WaitTime 1.5;
ENDPROC

PROC Combined()
        MoveJ L3P1,v500,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveJ L3P1_x20,v500,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveL L3P2,v500,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveJ L3P2_ym20,v500,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveL L3P4,v500,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveJ L3P4_y30,v500,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveL L3P5,v500,fine,t_Gripper_Base;
        WaitTime 1.5;
ENDPROC

PROC Pitch_test()
```

```
        MoveJ L2P5,v500,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveJ L2P5_xm30,v500,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveJ L2P5_xp30,v500,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveJ L2P4,v500,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveJ L2P4_xm15,v500,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveJ L2P4_xp15,v500,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveJ L2P3,v500,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveJ L2P2,v500,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveJ L2P2_xm30,v500,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveJ L2P2_xp30,v500,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveJ L2P1,v500,fine,t_Gripper_Base;
        WaitTime 1.5;
ENDPROC

PROC Roll_test()
        MoveJ L1P1,v500,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveL L1P2,v500,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveJ L1P2_ym30,v500,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveJ L1P2_yp30,v500,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveL L1P3,v500,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveJ L1P3_ym60,v500,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveJ L1P3_yp60,v500,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveL L1P4,v500,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveJ L1P4_ym45,v500,fine,t_Gripper_Base;
```

```
        WaitTime 1.5;
        MoveJ L1P4_yp45,v500,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveL L1P5,v500,fine,t_Gripper_Base;
        WaitTime 1.5;
ENDPROC

PROC Lin_test_300()
        MoveJ L1P1,v150,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveL L1P1_LIN_Y,v300,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveL L1P1,v300,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveJ L1P1_LIN_X_Base,v500,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveL L1P1_LIN_X_End,v300,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveL L1P1_LIN_X_Base,v300,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveJ L1P1_LIN_Z_base,v150,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveL L1P1_LIN_Z_top,v300,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveL L1P1_LIN_Z_base,v300,fine,t_Gripper_Base;
ENDPROC

PROC Lin_test_600()
        MoveJ L1P1,v150,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveL L1P1_LIN_Y,v600,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveL L1P1,v600,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveJ L1P1_LIN_X_Base,v150,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveL L1P1_LIN_X_End,v600,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveL L1P1_LIN_X_Base,v600,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveJ L1P1_LIN_Z_base,v150,fine,t_Gripper_Base;
        WaitTime 1.5;
```

```
        MoveL L1P1_LIN_Z_top,v600,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveL L1P1_LIN_Z_base,v600,fine,t_Gripper_Base;
ENDPROC

PROC Lin_test_1000()
        MoveJ L1P1,v150,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveL L1P1_LIN_Y,v1000,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveL L1P1,v1000,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveJ L1P1_LIN_X_Base,v150,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveL L1P1_LIN_X_End,v1000,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveL L1P1_LIN_X_Base,v1000,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveJ L1P1_LIN_Z_base,v150,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveL L1P1_LIN_Z_top,v1000,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveL L1P1_LIN_Z_base,v1000,fine,t_Gripper_Base;
ENDPROC

PROC Lin_test_1500()
        MoveJ L1P1,v150,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveL L1P1_LIN_Y,v1500,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveL L1P1,v1500,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveJ L1P1_LIN_X_Base,v150,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveL L1P1_LIN_X_End,v1500,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveL L1P1_LIN_X_Base,v1500,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveJ L1P1_LIN_Z_base,v150,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveL L1P1_LIN_Z_top,v1500,fine,t_Gripper_Base;
        WaitTime 1.5;
```

```
        MoveL L1P1_LIN_Z_base,v1500,fine,t_Gripper_Base;
ENDPROC

PROC Lin_test_2000()
        MoveJ L1P1,v150,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveL L1P1_LIN_Y,v2000,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveL L1P1,v2000,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveJ L1P1_LIN_X_Base,v150,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveL L1P1_LIN_X_End,v2000,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveL L1P1_LIN_X_Base,v2000,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveJ L1P1_LIN_Z_base,v150,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveL L1P1_LIN_Z_top,v2000,fine,t_Gripper_Base;
        WaitTime 1.5;
        MoveL L1P1_LIN_Z_base,v2000,fine,t_Gripper_Base;
ENDPROC

ENDMODULE
```