



Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

Frey Dominik

KOMPLEX, SOKSZEREPLŐS, KO- OPERATÍV ÖNVEZETŐ AUTÓKBÓL ÁLLÓ RENDSZER PROGRAMOZÁSA

KONZULENS

Dr. Charaf Hassan

BUDAPEST, 2022

Összefoglalás

A közúti balesetek és a forgalmi torlódások két kritikus probléma a globális közlekedési rendszerekben. Az összekapcsolt járművek (Connected Vehicles) és az autonóm járművek (Autonomous Vehicles) a legtöbbet kutatott és legígéretesebb autóiipari technológiák közé tartoznak, a közúti balesetek csökkentésének és a mindennapi közlekedés hatékonyságának maximalizálása céljából.

A feladat egy komplex, sokszereplős, kooperatív önvezető autókból álló rendszer programozása, ahol a cél az, hogy az autonóm járművek közlekedésének hatékonyságát vizsgálhassuk egy kiber-fizikai környezetben. A járművek képesek szenzoraiuk segítségével virtuális képet alkotni a környezetükről, navigációs és manőverezési döntéseket hozni és végrehajtani. A járművek képesek egymással kommunikálni, közvetlenül és hálózaton keresztül egyaránt. A közvetlen kommunikáció csak az egymáshoz közeli járművek között tesz lehetővé információ cserét, míg a hálózatba kapcsolt kommunikáció tetszőlegesen távoli járműveket is képes valós időben összekötni. A kommunikáció célja, hogy a járművek megosszák egymással a mozgási állapotukat (pozíció, sebesség, haladási irány, stb.) és így képesek legyenek kollaboratív manőverek végrehajtására (pl. egy kereszteződésben megállás nélkül, sebességüket összehangolva az ütközés veszélye nélkül áthaladni). A hálózaton keresztüli kommunikáció lehetővé teszi a járművek központi vezérlését is, ahol a manőverek összehangolása egy központi logika segítségével történik.

A feladat

A projekt során felmerülő feladatok két részre oszthatók:

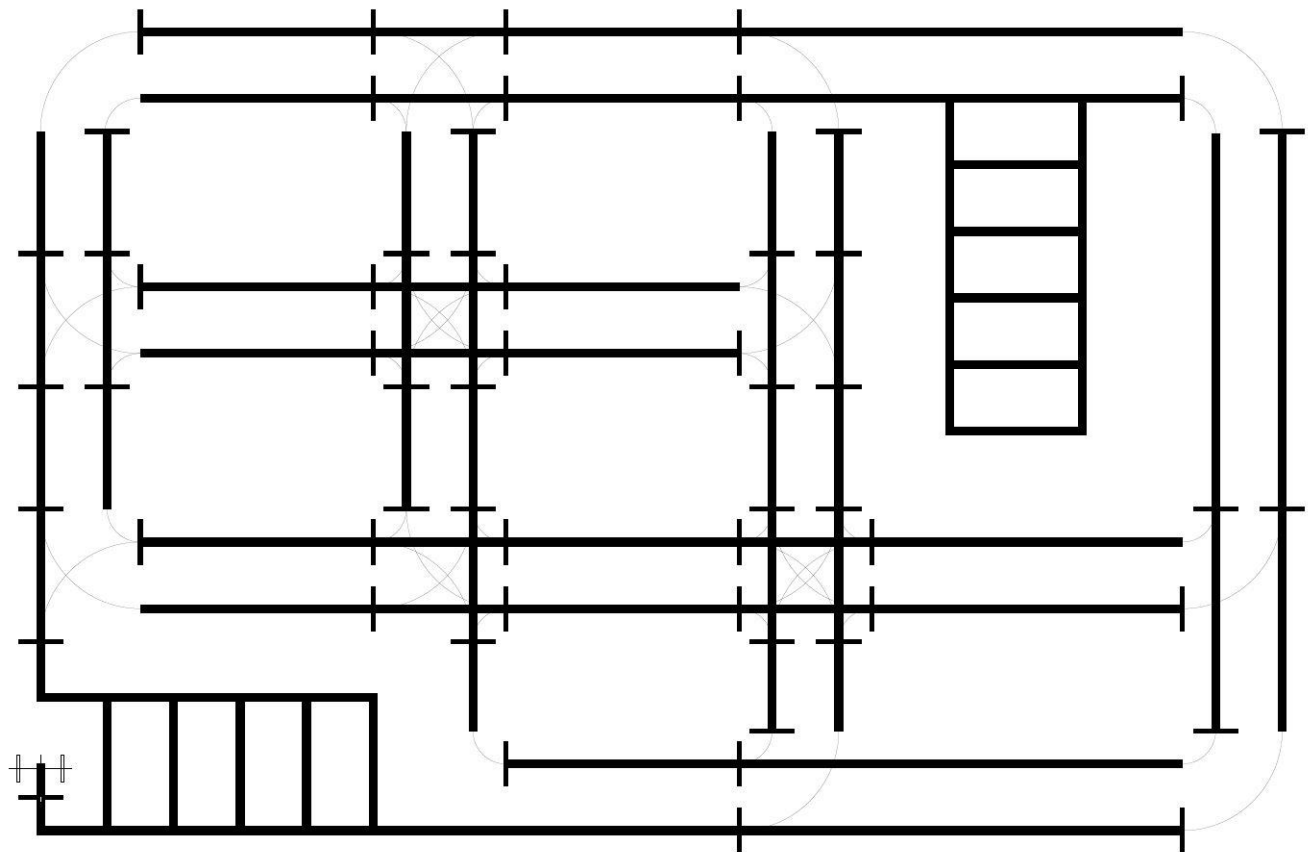
1. Autonóm két tengelyes hátsó kerék meghajtású jármű modellek mechanikus és szoftver architektúrájának megtervezése, a motorikus vezérlés kialakítása visszacsatoló szenzorok segítségével, a modell tesztelése és szoftveres manőver stabilizálása.
2. Több, autonóm jármű modellből álló flotta vezérlésének szenzorokkal támogatott és szoftveres implementációja. A cél, hogy a jármű modellek képesek legyenek egy közös területen, egymást keresztező útvonalakon közlekedni ütközések nélkül.

A rendszer áttekintése

Első lépésben, a feladatom a már meglévő, fejlesztendő rendszer áttekintése volt. Ebben a fejezetben bemutatom a pályát, amin a járművek közlekednek. Ez után a járművek hardveres és szoftveres felépítésére térek ki.

A feladat magában foglalta a jármű komponenseinek megismerését és azok működésének a munkához szükséges szinten való megértését, mind hardver, mind szoftver oldalról. Ugyan így, a komponensek közötti kapcsolatok és interfészek feltérképezése is elengedhetetlen volt. A következőkben ezeket a komponenseket és azok interfészeit részletezem.

A pálya



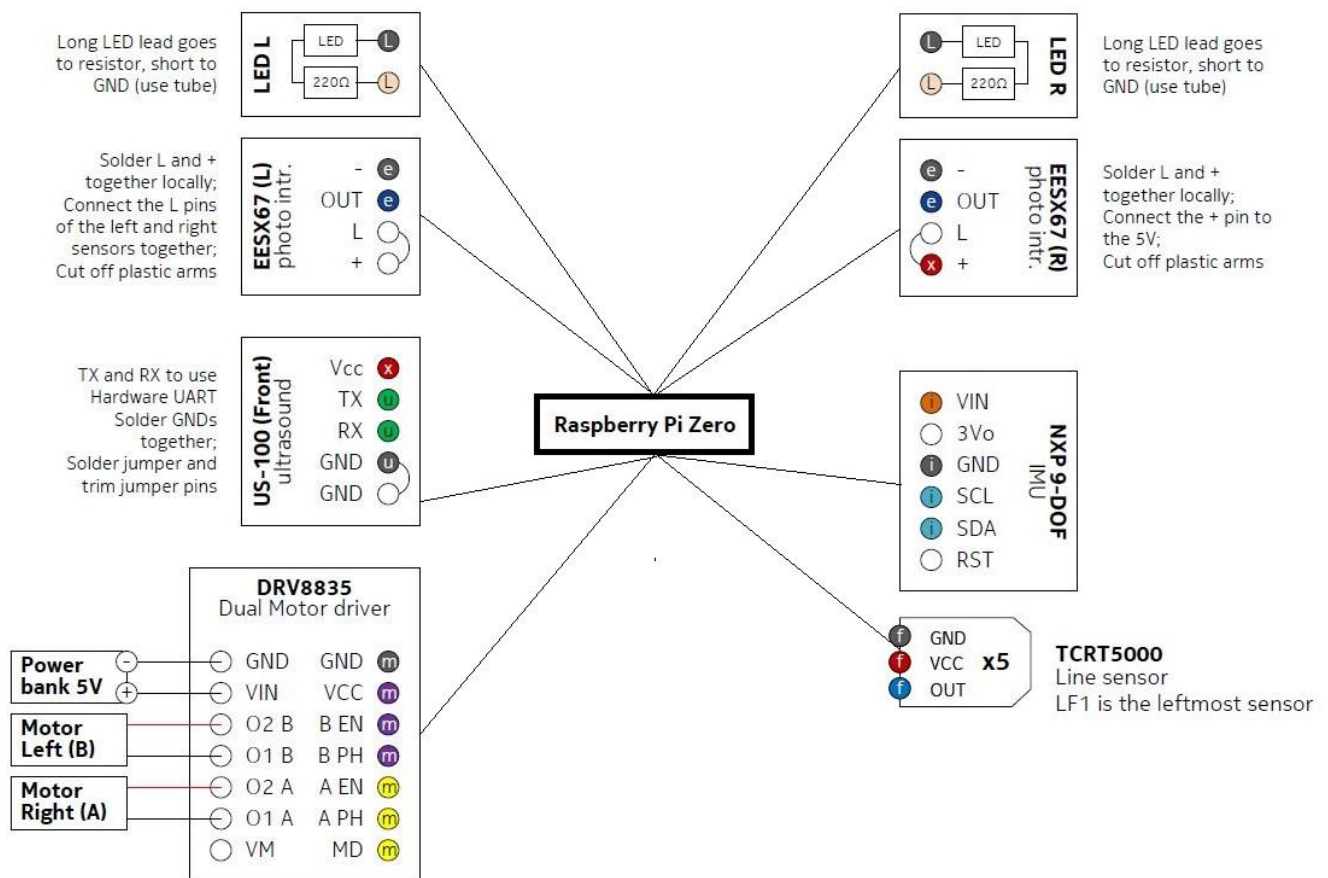
1. ábra - A járművek által bejárható útvonalak összessége

- A járművek egy fix előre megtervezett pályán közlekednek, fekete vonalakat követve, amik sávokat reprezentálnak. A **1. ábrán** látható a pálya elrendezése, ez autó memóriájába le van kódolva, tehát mindenki ismeri a teljes útvonal halmazát. A valóságban ez egy 3 x 2 méteres szőnyegre van felnyomtatva, ahol az egyes szakaszok hosszát az autók miliméterre pontosan ismerik, így tudják, hogy egy-egy szakaszon mennyi ideig és milyen sebességen kell haladni, valamint, hogy ezek függvényében éppen hol tartanak a szakaszon belül. Ezen szakaszokból áll össze a teljes pálya. A szakaszok végét merőleges vonalak jelzik (End Mark). A szakaszokra a továbbiakban szegmensként hivatkozok.

- A pályán található egy kezdeti szegmens minden autó innen kezdi a manőverezését. Ezen kívül tartalmaz egymást keresztező szegmenseket, nagy ívű, kis ívű és 90 fokos kanyarokat.
- Az End Mark pontokat az optikai szenzorok segítségével az autó fel tudja ismerni és szinkronizálni tudja a saját, nyílvántartott pozícióját az EM pozíciójával. Az éppen soron következő szegmens alapján az autó el tudja dönteni, hogy milyen manővereket kell végrehajtania. Ezek lehetnek az egyenes tartása, nagy ívű kanyar bevétele, kis ívű kanyar bevétele és 90 fokos kanyar bevétele.

Az autók hardveres áttekintése

Az egyes hardver komponensek az **2. ábrán** láthatóak.



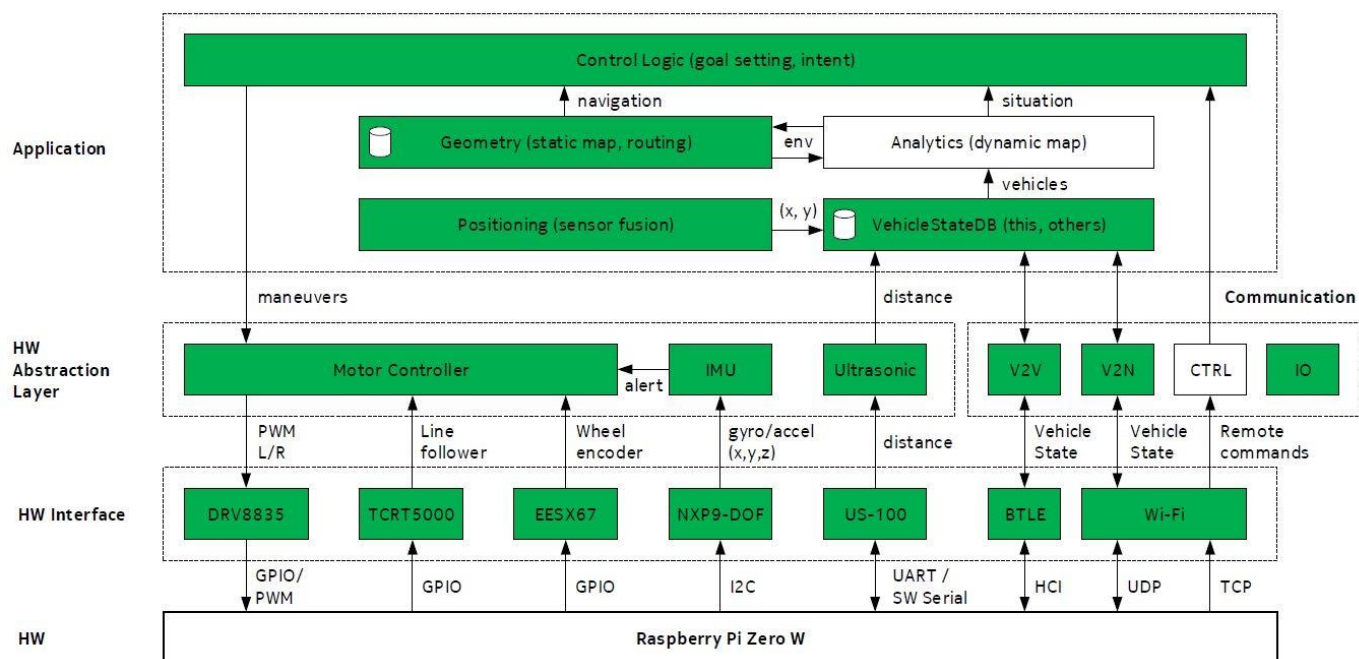
2. ábra - A jármű hardver komponenseinek diagrammja

- A központi mikrokontroller egy **Raspberry Pi Zero W** egység, ami 40 szabad GPIO pin-t biztosít, 512MB RAM-mal rendelkezik és támogat vezeték nélküli LAN és Bluetooth kapcsolatot. Tápellátását egy hozzá erősített külső akkumulátorból kapja. Kis mérete és a felsorolt tulajdonságai miatt tökéletes beágyazott IoT eszközök készítésére.

- Az egységhez két darab LED lámpa csatlakozik, egy bal és egy jobb oldali (**LED L**, **LED R**). Ezek használhatók index jelzésre, valamint vészhelyzet esetén különböző frekvencián való villogtatásra. (Például, hogy az autó felemelkedett a talajról.)
- A jármű elejére egy ultrahang szenzor van erősítve, amely hőmérséklet mérésére is képes, de számunkra a távolság szenzor funkciója az igazán érdekes. A szenzor egy **US-100 Ultrasonic** távolságmérő, amellyel a mikrokontroller UART-on keresztül kommunikál. A szenzor +15 és -15 fokos szögön belül képes érzékelni az elé kerülő tárgyakat és 2 – 450 cm intervallumon belül 0.3cm +-1% pontosságú mérési eredményt ad. Ezt a jármű elé kerülő tárgyak érzékelésére használjuk.
- A jármű egy kulcsfontosságú szenzora a 3 dimenziós giroszkóp, amely egyben egy 3 dimenziós gyorsulá mérő és magnetométer is egyben (**NXP 9-DOF IMU**). Az eszköz igen magas pontosságú, a mért értékeket fok/másodperc mértékegységben, valamint g, gravitációs állandó függvényében lehet lekérdezni I2C kommunikáción keresztül. Alapvetően a jármű stabilitásának megtartásában játszik fontos szerepet.
- Az autók alján található 5 darab **TCRT5000**-es reflektív optikai szenzor, állítható érzékenységgel. Ezeknek két állapotuk van egy aktív és egy inaktív. A szenzorok segítségével képes az autó a vonalkövetésre. Ha csak a középső inaktív, az azt jelenti, hogy a jármű a vonalon helyezkedik el. Ezen kívül a különböző kombinációkhoz különböző vonalhoz képesti elhelyezkedések vannak rendelve.
- A járművön két motor helyezkedik el, amik egy-egy kereket hajtanak meg. A két motort az egység egy **DRV8835** típusú driver-en keresztül vezérli egyszerű PWM jelekkel. A tengelyek összehangolása céljából végzett kalibráció végrehajtására egy-egy **EE-SX67**-es fotoelektromos érzékelőt használunk, amelyek villái között egy-egy a kerekekkel együtt forgó 20 réses kerék enkóder található. A szenzor be- és kikapcsolva állapotainak váltakozása miatt viszont 40 állapotváltozás rögzíthető fordulatonként, amely megduplázza a pontosságot.

Az autók szoftveres áttekintése

A járműveken futó program 3 absztrakciós szintre bontható fel, egy alkalmazás szintű rétegre, egy hardver absztrakciós rétegre és egy hardver interfészre, amely legalsó szint már közvetlenül a GPIO pin-eket állítja, amelyekre BCM notáción keresztül hivatkozik. Rétegenként az egyes szoftver komponensek és azok kapcsolatai a **3. ábrán** láthatóak. A következőkben az egyes rétegek feladatköreit ismertetem, kitérve a főbb komponenseikre.



3. ábra - A jármű szoftveres komponensei és azok kapcsolata, 3 absztrakciós szintre bontva

Alkalmazás szintű réteg

Az alkalmazás réteg felelős a járművek magas szintű viselkedésének leírására. A szenzorok által mintavételezett információ itt már rendelkezésre áll jól értelmezhető formában és a vezérlést is egyszerű interfészeken keresztül lehet megoldani. Az itt felmerülő kihívások:

- Nyilvántartani a jármű pozícióját (**Vehicle State DB**) és ez alapján megtervezni a célhoz vezető legrövidebb útvonalat (**Geometry**). Az útvonal ismeretében, annak minden szegmensére megtervezni a következő manővert vagy manőver sorozatot, majd azt végrehajtatni az alacsonyabb szinten elhelyezkedő Motor Controller-el (**Control Logic**).
- A jármű környezete és a nyilvántartott többi járműtől a kommunikáció során megszerzett információk alapján döntéseket hozni, elkerülni az ütközéseket, optimalizálni a cél eléréséhez szükséges időt.
- Állapot információt szolgáltatni a többi járműnek lokális és globális csatornákon.

A legfelső szinten a **Control Logic** helyezkedik el. Ez menedzseli a jármű egyes állapotait és hozza meg a főbb döntéseket. A saját és más járművek állapotát a **Vehicle State DB**-ben tárolja és hívja le szükség esetén.

A **Geometry** felelős a pálya szegmenseinek nyilvántartásáért hosszúsággal rendelkező vektorok formájában és az útvonalak megtervezéséért.

Hardver absztrakciós réteg

Ez a réteg felelős az egyes szenzorok és driver-ek hardver specifikus függőségeinek elfedéséért. A benne elhelyezkedő komponensek egyszerű interfészeket szolgáltatnak, hogy segítsék a vezérlést.

A **Motor Controller** felelős a felülről kapott manőverek végrehajtásáért a következő képpen:

- A manőverek típusából eldönti, hogy mekkora sebességet kell adni a motorokra.
- A sebességből PWM-et állít elő.
- Figyeli a vonal szenzorokat (**TCRT5000**) és a giroszkópot (**IMU**) és szükségyszerűen korrekciót végez.
- Jelez, ha a manőver befejeződött.

Az **IMU** felelős a gyorsulásmérő és giroszkóp (**NXP 9-DOF**) mintavételezéséért. Figyelmeztet, ha a jármű elveszti a stabilitását bármely tengely mentén. Nem triviális jövőbeli feladata a fok/másodperc értékekből abszolút orientációt előállítani, a jármű egyenes haladás közbeni driftelésének megbízható felismerésére.

Az **Ultrasonic** komponens pedig egyszerű miliméterben vett távolságot szolgáltat a magasabb szintű komponenseknek.

Hardver interfész

A hardver interfész komponensei az egység GPIO pinjeit írják és olvassák. Íráskor a magasabb szintű komponensektől kapott adatot a driverok bemenetén elvárt formátumra konvertálják, olvasáskor pedig a pinekről kiolvasott adatot egy magasabb szinten értelmezhető formátumban szolgáltatják.

Motorikus vezérlés kialakítása visszacsatoló szenzorok segítségével

Ugyan a jármű elegendő pontossággal képes az egyenes manőverezésre a tengelyek kalibrálása után, de a tesztek azt mutatták, hogy így is „kacsázik” a követendő vonal két vége között, ami megnöveli az összesen megtett utat és a cél eléréséhez szükséges időt. Ezen kívül a folyamatos irányváltás egyenes vonal követése közben instabillá teszi a haladást és előfordul, hogy a korrekciók sűrűsége és szöge egyre inkább növekszik, ezáltal egy idő után a jármű végleg elveszti a vonalat. A jelenség annak az eredménye, hogy a jármű csupán 5 darab reflektív optikai szenzorral van felszerelve, ami nem nyújt kellő pontosságot több, minimális mértékű korrekcióhoz.

Erre a problémára nyújthat megoldást az **NXP 9-DOF** giroszkóp és gyorsulásmérő. A nagy pontosságú szenzor minimális driftet is képes érzékelni így felhasználható folyamatos korrekcióhoz az optikai szenzorok mellett. Sajnálatos módon az **NXP 9-DOF** nem tud önmagától abszolút orientációt számolni, csak másodpercenkénti elfordulást és pillanatnyi gyorsulást szolgáltat. Első lépésben tehát egy orientációs

szenzor fúziós algoritmus használatára lesz szükség a hardver interfész vagy hardver absztrakciós rétegben, hogy megkapjuk a szükséges elfordulást. Egy ilyen algoritmus kombinálja a mind a giroszkóp, mind a gyorsulásmérő és magnetométer kimenetét, hogy minnél pontosabb abszolút elforgást kaphassunk.

A következőkben a feladat a megfelelő szenzor fúziós algoritmus megtalálása és rövid áttekintése, majd alkalmazása az **NXP 9-DOF** szenzor adatira. Ezután a kinyert abszolút elfordulás felhasználásával implementálni kell a tengely korrekciót egyenes haladás esetén, ötvözve az optikai szenzorok állapotai alapján történő beavatkozással.

Relatív orientáció giroszkóp segítségével

Első próbálkozásként kíváncsi voltam, hogy megúszható-e a feladat egyedül a giroszkóp kimenetének felhasználásával, ugyanis elegendőnek tűnt adott időpontoktól mért elfordulást, megfelelő időközönként mintavételezni és azokat korrigálni az ellenkező irányba. Ráadásul mindezt csak a Z, azaz a függőleges tengely körül.

A giroszkóp regisztereiből szögsebesség olvasható ki fok/másodperc mértékegységben. A stratégiám az volt, hogy 1/4 másodperces időablakokban kiátlagolom az összes mintát, amit a processzor képes volt kiolvasni majd ezt felszorozni az eltelt idővel, hogy megkapjuk az adott időn belül megtett elfordulást.

$$\theta_k = \theta_{k-1} + \frac{\sum_i^n \omega_i}{n} * \Delta t$$

Az eredmény nem volt túl rossz és a számolási költsége is minimális, talán a célra még éppen megfelelő lett volna, de igen nagy drift-tel rendelkezett hosszabb távon a zajok és a kalibráció után is fennmaradó bias miatt, ezért úgy gondoltam egy komplexebb algoritmus, több szenzor felhasználásával megbízhatóbb lenne, akár még jövőbeli funkciók is támaszkodhatnak rá.

Abszolút orientáció AHRS szűrővel

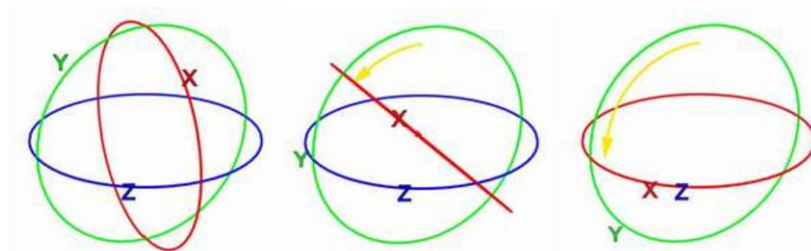
Teljes 3 dimenziós minimális driftel rendelkező abszolút orientáció kapható a gyorsulásmérő és a magnetométer szenzorok felhasználásával.

A kívánt orientációk elnevezései tengelyenként:

- **Roll:** X tengely körüli elfordulás
- **Pitch:** Y tengely körüli elfordulás
- **Yaw/Heading:** Z tengely körüli elfordulás

A cél ezen értékek valós idejű kinyerése valamely AHRS (**Attitude and heading reference system**) algoritmus segítségével.

Egy ilyen algoritmus az egyes szenzorok mérési hibáit figyelembe veszi és folyamatosan csökkenteni törekszik. A lényeg, hogy minden szenzor kompenzálja a többi által bevezetett bias-t. A két általános célú Bayes-féle becslési szűrő, ami a feladathoz megfelelő lehet a **Mahony** és a **Madgwick** szűrők [1]. Mindkét megközelítés kvaternió-reprezentációt használ, ami egy négydimenziós komplex szám és az objektum tájolását jelzi. Annak ellenére, hogy a kvaterniókat könnyebb kiszámítani és hatékonyabbak, kevésbé intuitívak és érthetőek, mint az Euler-szögek. Az Euler-szögek viszont kétértelműségnek és az úgynevezett **Gimbal lock** jelenségnek vannak kitéve, amelyek az ábrázolás két ismert problémája. A **Gimbal lock** olyan szingularitás, amely akkor jelenik meg, ha az objektum két tengelye párhuzamosan áll. Ez egy szabadságfok elvesztéséhez vezet, ezáltal mérési pontatlanságot okoz, ahogy a **4. ábrán** is látható.



4. ábra - Y tengelyen (Pitch) 90 fokon elfordulva X (Roll) párhuzamossá válik Z-vel (Heading) és elveszik a szabadsági foka. Forrás: [3]

(Szerencsére valószínűleg ezzel a jelenséggel nem kell számolnunk mivel a jármű nagyrészt csak a Z tengely körül fog rotálódni. Ezért majd a kapott kvaterniókat Euler szögekké konvertálom, a könnyebb érthetőség kedvéért.)

Ahogy különböző cikkekből megtudtam, a Madgwick a RMSE (Root-Mean-Squared Error) értéke tekintetében felülmúlja a Mahony-t zajos szenzoradatok mellett. A Mahony számítási költsége viszont kevesebb. Mindezt figyelembe véve úgy döntöttem a Madgwick algoritmust fogom felhasználni, mert a menetstabilizáció közben fontos a pontosság, mivel már nagyon minimális (0.3-0.5 fokos) elfordulások esetén is meg kell kezdeni a korrekciót, hogy elkerüljük a vonalról való letérést.

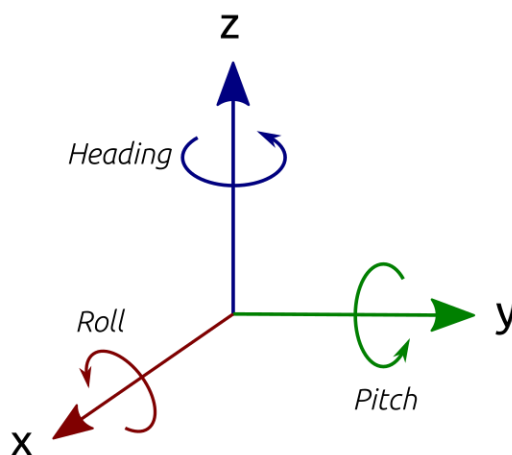
Szerencsére a Madgwick algoritmusra rengeteg implementáció és példakód született az évek során és ezek nagy része megtalálható Github-on. Tehát elegendő volt rákeresni a megfelelő C++ implementációra és beépíteni a projektbe. Azonban ebben a formában a szűrő még nem bizonyult használhatónak. Először is ki kellett elégíteni a bemenet mértékegységekre és a tengelyek irányára vonatkozó megkötéseit. Ez után

finomhangolni a tanulási rátát és a mintavételi frekvenciát, végül a kvaterniókat Euler szögekké konvertálni. Ezen kívül alkalmaztam egy algoritmust a becsült kvaterniók kiszámítására a gyorsulásmérő és a magnetométer pillanatnyi állása alapján, hogy a kezdeti állapotban kevesebb hibát kelljen javítania a Madgwick-nek.

Adatok előkészítése

Az elvárt tengelyállás:

A tengelyek elvárt állása az **5. ábrán** látható, ahol az X tengely mutat északra, a Z pedig felfelé.



5. ábra -

A szenzorokból kapott adatokat ebbe az orientációba kellett forgatni. Ez egyenlő volt a vektorok 180 fokos tükrözésével, valamint az X és Y tengelyek felcserélésével, mivel az **NXP 9-DOF** szenzor ilyen formában szolgáltatja az adatokat.

Mértékegységek:

- A giroszkóp adatait radián/másodpercre kellett konvertálni.
- A gyorsulásmérő g-ben az az 9.81 m/s^2 -ben kellett megadni.
- A magnetométer mértékegysége nem számottevő, viszont értékét eddig nem használtuk fel, ezért hozzá kellett nyúlni a szenzor regisztereikhez. Regiszter olvasás után a szembesültem vele, hogy a nyers adat lényegében használhatatlan. Implementálni kellett és elvégezni egy **Hard Iron** és egy **Soft Iron** kalibrációt, hogy pontos értékeket kapjunk. A kalibráció abból áll, hogy nagyjából 15 – 30 másodpercig forgatjuk az egységet minden lehetséges irányban. Így a mintavételezett adatok egy gömböt fognak leírni a 3D-s térben.

- A **Hard Iron** kalibráció során ennek a gömbnek a középpontjába mutató vektorral mozgatjuk el a mintavételezett adatokat és ezzel eltávolítjuk az állandó mágneses hatásokat, amit nem a föld mezője okoz.
- A **Soft Iron** kalibrációra azért van szükség mert néhány közelben lévő tárgy ugyan nem bocsát ki mágneses mezőt, de eltorzítja azt, aminek hatása van a mi adatainkra is. Ennek javítása igen költséges, egy ellipszoidra való illesztéssel oldható meg, sok mátrix művelet árán. Emiatt egy olcsóbb és egyszerűbb, de effektívnek mondható megoldást használók [2]:

- A módszer egy-egy skála bias kiszámítása a szenzor adatokra a kalibráció során, amikkel a nyers adatok felhasználás előtt lesznek felszorozva. A skálák számítását az alábbi képlettel tudom a legjobban demonstrálni:

$$\begin{bmatrix} \text{delta_x} \\ \text{delta_y} \\ \text{delta_z} \end{bmatrix} = \frac{1}{2} * \left(\begin{bmatrix} \text{max_x} \\ \text{max_y} \\ \text{max_z} \end{bmatrix} - \begin{bmatrix} \text{min_x} \\ \text{min_y} \\ \text{min_z} \end{bmatrix} \right)$$

$$\text{delta} = \frac{(\text{delta_x} + \text{delta_y} + \text{delta_z})}{3}$$

$$\begin{bmatrix} \text{scale_x} \\ \text{scale_y} \\ \text{scale_z} \end{bmatrix} = \left(\begin{bmatrix} \text{delta}/\text{delta_x} \\ \text{delta}/\text{delta_y} \\ \text{delta}/\text{delta_z} \end{bmatrix} \right)$$

Mintavételezési frekvencia és tanulási ráta:

A mintavételezési frekvencia azt jelenti, hogy milyen gyakran kérdezzük le a szenzorokat és frissítjük a kvaterniókat az algoritmussal. Ha túl kicsire választjuk nem lesz precíz a mérés, magas marad a hiba ráta. Ha túl nagyra választjuk előfordulhat, hogy a processzor nem tudja elvégezni időben a számításokat és szintén elromlik a mérés. Esetemben egy 100 Hz-es mintavételezési frekvencia bizonyult megfelelőnek.

A tanulási ráta egy konstans, ami meghatározza a korrekció mértékét. Ha túl nagy, könnyen túllőhetünk az optimumon, ha túl kicsit nem fogjuk eléggé megközelíteni azt. Jelenleg 0.4-es tanulási rátával elég jó eredményeket kapok.

Euler szögek előállítás

A Madgwick algoritmus kvaterniókat, 4 dimenziós komplex számokat állít elő. Én szerettem volna klasszikus Euler szögekben kezelni az eredményt [-180 ; 180] fok intervallumon. Szerencsére a kvaterniók könnyen konvertálhatóak a következő képlettel [4]:

$$\begin{bmatrix} Roll \\ Pitch \\ Yaw \end{bmatrix} = \begin{bmatrix} atan2(2(q_0q_1 + q_2q_3), 1 - 2(q_1^2 + q_2^2)) \\ asin(2(q_0q_1 - q_3q_2)) \\ atan2(2(q_0q_3 + q_1q_2), 1 - 2(q_2^2 + q_3^2)) \end{bmatrix}$$

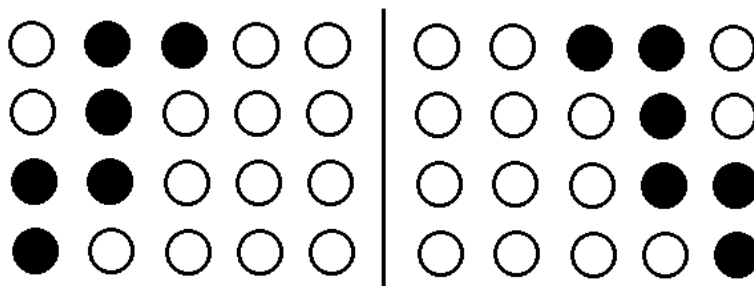
Mindezek után már kinnyerhető a jármű abszolút orientációja a 3 dimenziós térben Euler fokokban kifejezve.

A következő lépés egy szabályzó implementálása, amellyel a jármű képes lesz közel tökéletes egyenes menetirány tartására.

Menetirány stabilitás kialakítása

Most, hogy a három szenzoradat (giroszkóp, gyorsulásmérő, magnetométer) fúziójával ismerjük a jármű orientációját a 2 dimenziós síkon, de akár a 3 dimenziós térben is, minden rendelkezésre áll, hogy az egyenes manőverezést a pontatlan motorok mellett stabilizáljuk. Természetesen maguk a szenzorok pontatlansága, a drift és a pálya esetleges egyenetlensége miatt még mindig szükség lesz a vonalkövető szenzor alapján történő beavatkozásra.

A vonalkövetőnek nyolc állapota indikálja, hogy a jármű letért az útvonalról. Ezeket a **6. ábra** mutatja.



6. ábra - A vonalkövető állapotai, ha letért a vonalról a jármű

Itt az egyes körök a szenzorokat jelölik, a feketére festettek pedig, hogy egy-egy szenzor hol érzékeli a vonalat. Ebből lehet következtetni a letérés mértékére is és ez alapján kisebb vagy nagyobb korrekcióval visszaterelni a helyes állapotba, amit a **7. ábra** mutat.



7. ábra - Helyes szenzor állapot

Az alap probléma az volt, hogy a jármű túl sokszor kerül ki ebből az állapotból és helyes szabályzás hiányában elkezd „kacsázni” az állapotok között, míg végül valamelyik, a **6. ábrán** látott utolsó, leginkább nem kívánt állapotba kerül vagy teljesen elveszíti a vonalat. Az orientáció alapú szabályzás célja tehát,

hogy a járművet minnél több ideig a **7. ábrán** látott állapotban tartsa és az esetleges letérések minnél kisebb szögben történjenek. A szabályzó csak akkor fog működésbe lépni, ha a jármű a helyes állapotban van.

A vonallal párhuzamos szög megtalálása

A szabályzó alap ötlete, hogy vegyünk egy szöget, ami a vonallal párhuzamos irányba mutat. Ezt hívjuk referencia szögnek. A jármű folyamatosan méri az éppen aktuális orientációját és egy különbséget számol a referencia szöghöz képest. Ennek a különbségnek a függvényében korrigálja egyes kerekeinek a gyorsaságát, hogy a különbséget minnél kisebb értéken tarssa, így képes követni a referencia szöget, ezáltal a vonalat.

Sajnos a valóságban nem létezik olyan, hogy pontosan tudjuk, hogy milyen szög tartásával lehet a járművet tökéletesen, a vonallal párhuzamos haladásra készíteni. Első próbálkozásra, a jármű lehelyezésekor mérhetünk egy orientációt és használhatjuk mint referencia szög. Ezzel az a baj, hogy nem feltételezhetjük, hogy éppen párhuzamos az autó a vonallal, egy apró eltérés mindig lesz. A másik probléma, hogy még a szenzorfúzió után is létrejön valamennyi drift a forgások során, ezért kis időn belül ez a referencia szög a jármű szempontjából már nem a vonallal párhuzamos irányba fog mutatni. Mindezt figyelembe véve kiderült, hogy hosszabb szakaszokon elkerülhetetlen, hogy a jármű letérjen a helyes állapotáról.

Viszont ezt a letérést kihasználhatjuk, hogy egy új referencia szöget állítsunk elő, amivel még mindig elkerülhetjük a „kacsázást”, továbbá a megfelelő algoritmussal egy eredendően rossz referencia szögből egyre jobb szögeket állíthatunk elő. Ez a következőképpen történik:

- Először is feltételezzük, hogy ha van egy szögünk, akkor azt a jármű tökéletesen képes tartani. Ennek a biztosítása nem a referencia szöget javító algoritmus feladata.
- Amikor letérünk a helyes állapotról rögzítjük, hogy a szegmens hányad részét sikerült megtenni a legutóbbi referencia szöggel. Ezt az értéket használjuk, hogy egyfajta jószágot rendeljünk egyes referencia szögekhez.
- Amikor a vonalszenzorok alapján történő beavatkozás hatására visszatérünk a helyes állapotba rögzítjük, hogy mi az a szög, amit a visszatérés pillanatában mért a jármű.
- Ebből az alábbi képlet alapján számolunk egy új referencia szöget:
- $\alpha_i = \alpha_{i-1} \pm |\alpha_{i-1} - \beta| * f(\mu)$

- Ahol a $\pm$ attól függ, hogy melyik irányba tért le a jármű és
 - α_i : Az aktuális referencia szög
 - β : A visszatéréskor mért szög
 - $f(\mu)$: Egy megfelelő $[0,1]$ tartományban lévő érték μ , azaz a sikeresen megtett szegmens hossz és a teljes szegmens hossz arányának függvényében. μ szintén $[0,1]$ tartomány beli érték.

Ez azt biztosítja, hogy a következő referencia szög valahol α és β között lesz és olyan mértékben húz α felé, amennyire jól teljesített az a letérés előtt. A megfelelő $f(\mu)$ függvény megválasztása még kísérletezés alatt áll, de a lineáris $(1 - \mu)$ vagy az exponenciális $(2 - 2^\mu)$ megközelítéseket tartom ígéretesnek.

A vonallal párhuzamos szög tartása

A kalibráció során a tengelyek forgásának sebessége ugyanazon pwm érték megadása mellett kiegyenlítődik, de még így sem érhető el, hogy tökéletesen egyszerre mozogjanak és az autó egyenes szögben maradjon. Ez betudható a különböző környezeti hatásoknak például a pálya egyenetlenségének, betudható a két motor indulásának a késleltetésének vagy egyszerűen azok belső tulajdonságának. A két tengely sebességét az elméletbeli egyenestartás mellett mindenképpen folyamatosan korrigálni kell, hogy a jármű egy adott szög irányába tartson.

A korrekció úgy történik, hogy adott időközönként mintavételezem a referencia szögtől való eltérés mértékét és irányát és ezektől, valamint az aktuális sebesség értékétől függően, a megfelelő tengely gyorsaságát növelem, a másikat pedig csökkentem. A sebességet azért játszik itt szerepet, mert nagyobb sebesség esetén egy korrekció hatása gyorsabban érvényre jut, valamint az a tapasztalat, hogy nagyobb sebesség mellett a tengelyek inkonzisztenciája nem annyira számottevő mint alacsony sebesség mellett.

A korrekció mértéke az alábbi képlet alapján áll elő:

Ha az eltérés kisebb, mint 2 fok:

$$corr = |\alpha - \beta| * \frac{C}{speed}$$

Ha nem kisebb, mint 2 fok:

$$corr = \frac{2 * C + \sqrt{|\alpha - \beta| * C}}{speed}$$

Ahol:

- $|\alpha - \beta|$: A referencia szögtől való eltérés
- C : Egy megfelelően megválasztott konstans
- $speed$: Az aktuális sebesség

A korrekció mértéke 2 fokig lineárisan növekszik a szöggel arányosan majd ez után már csak gyökösen. Erre azért van szükség mert nagy eltérés mellett hirtelen túl nagy korrekciókat hajthatunk végre, ami előidézhetheti a „kacsázás” effektust. Viszont ha végig túl kicsi a korrekció, akkor meg nem tud a jármű időben a helyes szögbe fordulni és letér a vonalról. A korrekció végig lineárisan csökken a sebességgel arányosan. A C konstans határozza meg, hogy milyen agresszívan korrigáljon a jármű, ez és az egyes függések még kísérletezési stádiumban vannak, érdemes lesz még egy exponenciális megközelítést is kipróbálni.

Konklúzió

A menetirány stabilizátor rendkívül eredményesnek bizonyult nem csak az egyenes szög tartása esetében, hanem egy esetleges letérés után a helyes irány megtalálásakor is. Amikor a jármű például egy kanyar után, a motor pontatlanságából adódóan hibás szögből közelíti a vonalat kettő, maximum három referencia szög korrekció után képes viszonylag helyes irányba állni és tartani azt az irányt.

Ehhez az eredményhez jelenleg a beavatkozás másodpercenként 80-szor történik meg. Alacsonyabb frekvenciánál a jármű a szögváltozásokra lassabban reagál, ez túl alacsony frekvencia esetén még a „kacsázás” jelenséget is előidézhetheti. Túl magas frekvencia esetén elméletben persze jobb eredményt kapunk, de ezáltal a jármű mozgása töredezetté válik.

Persze létezik olyan szög egy kanyar bevétele után, amiből ezzel a menetstabilizációval nem lehet időben helyes irányba állítani a járművet és megtörténhet, hogy a rákövetkező kanyart már olyan rossz szögből veszi be, hogy teljesen letér az útvonalról. Ez egy teljesen különböző feladatot állított elé, ami a kanyarok bevételeinek stabilizálása. Szerencsére az abszolút orientáció birtokában ez már nem jelent nagy kihívást.

Manőverezés kooperatív környezetben

A következő feladat, hogy egyes járművek, szenzoraik segítségével és a pálya, valamint saját és más járművek pozíciójának ismeretében olyan döntések meghozására legyenek képesek, amik nem csak a saját, hanem az egész forgalom hatékonyságát növelik. Hatékonyság alatt itt az egyes útvonal tervek minnél gyorsabb teljesítését értem fennakadás nélkül, de akár egy esetleges fennakadás esetén is (egy jármű letért az útról vagy leállt).

Ebben a fázisban a rendelkezésre álló adatokat a következő módon szerzik be a járművek:

- **Ultrahang szenzor:** A járművek tudják, hogy egyes más járművek hol helyezkednek el a pályán, így számítanak rájuk. Ezzel szemben más objektumokra, amik nem bocsátanak ki állapot információt, a járművek nem tudnak felkészülni, amíg valamelyik a sok közül tudomást nem szerez róla valahogy. Erre való az ultrahang szenzor, amely a jármű orrára van felszerelve. Bármikor kerül az autó elé észreveszi és a távolság kiszámítása után az autó elküldi a többinek a pontos pozícióját, ezáltal a forgalom el tudja kerülni az útvonalak újratervezésével.
- **Gyorsulásmérő + vonalkövető:** Segítségével megállapítható, hogy a jármű valami miatt instabillá vált a pályán vagy felborult. Ekkor tud jelezni a többi autónak, hogy elkerülhessék a kérdéses szakaszt. A vonalkövetővel pedig azt tudja jelezni, hogy valamiért letért az autó a pályáról ezért a forgalomból való kiiktatására vár.
- **Kerék enkóder:** A kerék enkóderrel az egy szegmensben megtett távolság mérhető. Ezzel az autó meg tudja osztani a pontos pozícióját.
- **Wifi modul:** Az üzenetek továbbítása Wifin keresztül történik. Eredetileg Bluetooth-on történt volna a kommunikáció broadcast jelleggel, de az nagyon sok írás/olvasás műveletbe kerülne az amúgy is leterhelt eszközökön. Alternatívaként egy központi szerver egységet használunk, amely szűri és továbbítja az üzeneteket. Nagyon fontos, hogy az egység nem tölt be vezérlő szerepet, csakis a V2V kommunikáció emulálásáért felelős. A járművek az információ birtokában autonóm döntenek minden helyzetben, mivel a központi vezérlő egység bevezetése az ez utáni fázisban kerül megvalósításra, hogy össze lehessen hasonlítani a két módszer hatékonyságát.

Üzenettípusok

A járművek kétfajta üzenettípust közvetítenek egymás között. Az egyik a CAM (Cooperative Awareness Message), amely információkat nyújt a járművek jelenlétéről, pozíciójáról, valamint alapvető állapotáról. A másik az úgynevezett DENM (Decentralized Environmental Notification Message), amely közúti veszéllyel vagy rendellenes forgalmi körülményekkel kapcsolatos információkat tartalmaz, beleértve annak típusát és helyzetét [5].

CAM (Cooperative Awareness Message)

Ilyen típusú üzenetet minden autó akkor közvetít, ha az állapota valamilyen formában megváltozott. Egy ilyen üzenet protokollja a következő:

id	seg_curr	seg_dest	pos	speed	fdist	seq	rtt
0.-2. bájt	2.-4. bájt	4.-6. bájt	6.-7. bájt	7.-8. bájt	8.-10. bájt	10.-14. bájt	14.-16. bájt

- **id:** A küldő jármű egyedi azonosítója.

- **seg_curr**: Annak a szegmensnek az azonosítója, amin a jármű éppen tartózkodik.
- **seg_dest**: Annak a szegmensnek az azonosítója, ahová a jármű éppen tart.
- **pos**: A jármű szegmensben belüli pozíciója a 0 – 255-ös tartományra vetítve.
- **speed**: A jármű pillanatnyi sebessége.
- **fdist**: A jármű távolsága a közvetlen előtte lévő objektumtól miliméterben mérve.
- **seq**: Az üzenet sorozatszáma.
- **rtt**: Az üzenet körbefordulási ideje ms-ben mérve.

Minden autó a saját állapota mellett, tárol egy listát a többi jármű állapotáról. Ezen listaelemeket az **id** egyértelműen azonosítja. Ha egy másik autótól egy új CAM érkezik, hozzá tartozó régi bejegyzés felülíródik az új állapottal. Ha egy eddig ismeretlen **id**-t tartalmazó üzenet érkezik, a listába egy új bejegyzés keletkezik, ami az üzenet tartalmából lett konstruálva.

Emellett az autók folyamatosan nyilvántartják mely autók tartózkodnak egyes szegmenseken, így egy új szegmensre lépve könnyen kideríthető, mely más járművekre lehet számítani ott és azon belül pontosan milyen pozícióban.

DENM (Decentralized Environmental Notification Message)

Ilyen típusú üzenetet minden autó akkor közvetít, ha tudja magáról, hogy véglegesen letért a pályáról, valami miatt fennakadt vagy esetleg felborult, tehát rendellenes állapotba került manőverezés közben. Egy ilyen üzenet protokollja a következő:

id	seg	attr	pos	seq	rtt
0.-2. bájt	2.-4. bájt	4.-6. bájt	6.-7. bájt	10.-14. bájt	14.-16. bájt

- **id**: A küldő jármű azonosítója.
- **seg**: A bejelentett szegmens egyedi azonosítója.
- **attr**: A bejelentett szegmens attribútumát kódolja, azaz, hogy mit állapított meg a küldő jármű a szegmensről. Pl.: blokkolva van, várakozó autókat tartalmaz.
- **pos**: A felfedezett probléma szegmensben belüli pozíciója a 0 – 255-ös tartományra vetítve.
- **seq**: Az üzenet sorozatszáma.
- **rtt**: Az üzenet körbefordulási ideje ms-ben mérve.

Mivel az autók induláskor beolvassák a teljes pályát, nyilvántartásuk van az összes szegmensről, azok egyedi azonosítójáról és alapértelmezett állapotáról. Egy DENM érkezésekor az üzenet **seg** mezeje alapján megtalálják a kérdéses szegmenst és frissítik az állapotát az üzenet további tartalmának megfelelően.

Az eddigi munka összegzése

A tényleges fejlesztés előtt feltérképeztem a fejlesztendő rendszer komponenseit, mind hardver, mind szoftver oldalról. Megismerkedtem a felhasznált szenzorokkal és azok interfészeivel, valamint azon elemekkel, amiken beavatkozást kell végezni, például LED-ek és a motorvezérlő. A motorikus vezérlés stabilizálásához először is meg kellett ismerkedni a pálya felépítésével, a szegmensek és az azokból előállított manőverek tulajdonságaival.

A menetstabilizáció a manőverezésnek egy alacsonyabb szintjét képezi. Ahhoz, hogy egy megbízható vonalkövetést tudjak megvalósítani, gyökeresen át kellett alakítani a manőverek végrehajtásának logikáját, hogy skálázható legyen. Minden manővertípust külön fel lehet készíteni esetleges sarkallatos esetekre (például letérés a vonalról) és hogy milyen szenzorokra támaszkodjon végrehajtás során. A stabilizációhoz a kulcs a **NXP 9-DOF** szenzor felhasználása volt. Hogy a szenzorból abszolút orientációt tudjak kinyerni meg kellett valósítani egy 9 tengelyes szenzorfúziót és egy hiányzó magnetométer kalibrációt. Erre támaszkodva elkészült egy stabil vonalkövetési logika, általam megválasztott szabályzó függvények felhasználásával. Sok problémát jelentett a vonalérzékelő szenzorok pontatlansága, ami lassította fejlesztést és tesztelést valamint a **9-DOF** szenzor tengelyeinek irányítása, ami miatt transzformálni kellett őket a megfelelő irányba.

Miután a menetstabilizáció már kielégítően működött, következett a kód felkészítése az autonóm kooperatív működés kialakítására. Meghatároztam a kommunikációs csatornát és feltérképeztem az üzenet sorosítási és felhasználási logikát. Elkezdtem további járművek összeszerelését, hogy növeljem a flotta méretét a jövőbeli teszteléshez.

Az eddig elértéken alapulva a tényleges kooperatív viselkedés kialakítása még csak most kezdődik, először teljesen autonóm módon, majd a későbbiekben egy központi vezérlő egység bevezetésével.

Irodalomjegyzék

- [1] Madgwick/Mahony algoritmusok: <http://www.cs.ndsu.nodak.edu/~siludwig/Publish/papers/SPIE20181.pdf>
- [2] Soft iron kalibráció: <https://github.com/kriswiner/MPU6050/wiki/Simple-and-Effective-Magnetometer-Calibration>
- [3] Gimbal lock kép: https://www.researchgate.net/figure/Gimbal-lock-When-the-pitch-Y-rotates-90-degrees-the-roll-X-and-yaw-Z-axes-become_fig14_46720588

- [4] Euler szög/kvaternió: https://en.wikipedia.org/wiki/Conversion_between_quaternions_and_Euler_angles
- [5] ITS kommunikáció: <https://www.sciencedirect.com/science/article/abs/pii/S0968090X14001193>