Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar

Automatizált vezetett járművek irányítása ipari környezetben UWB rádiókkal mért beltéri pozíció információ alapján

Konzulens: Fehér Gábor

Készítette: Rózsahegyi Ádám

Budapest, 2018.november

Tartalomjegyzék

1 Bevezető

1.1 A Beltéri helyzet meghatározás és navigálás jelentősége

Napjainkban egyre nagyobb igény van a pontos beltéri pozíció meghatározására, mely jelentős fejlődésen megy keresztül: a gyakorlati megvalósulások eltérő technológiákat alkalmaznak, gyakran ötvözve is azokat. Beltéri helyzetmeghatározásnál számos akadály felléphet, és a megoldást nyújtó technológiák mindegyikének előnyei mellett hátrányai is vannak. Helyzetmeghatározást kínáló rendszerek telepítése előtt mindig érdemes mérlegelni azon szempontokat, amiket a rendszernek az adott környezetben teljesítenie kell és ezek után választani egy olyan megoldást, mely teljesíti ezeket a követelményeket. Ezek a szempontok elég széles skálán mozoghatnak, például mennyibe kerül a rendszer, hogyan lehet az adatokhoz hozzáférni, szükséges-e grafikus felület, illetve az adatok kezelése megfelel-e a biztonsági és privát követelményeknek. Mérnöki szempontból a pozícionáláshoz tartozó követelmények a legfontosabbak, így ezekre térek ki részletesebben.

Kérdéses, hogy mekkora területet akarunk lefedni, mekkora pontossággal kell meghatározni a kérdéses objektumok/élőlények helyét (mekkora lehet a szórás), milyen gyakran frissülnek az adatok. Fontos, hogy mennyi idő telhet el a mérés és az információnak a felhasználóig való eljuttatása között (latency) és milyen plusz információkhoz tudunk jutni a pozíción kívül (pl: orientáció, sebesség, gyorsulás). Illetve előfordulhat, hogy adott környezetben bizonyos technológiák rosszul, vagy egyáltalán nem működnek. Például nem előnyös olyan technológiával követni úszókat a vízben, ami nem képes a vízen áthatolni, vagy vaksötétben látáson alapuló rendszerrel navigálni. Ha ezeket a szempontokat sorra vesszük, akkor már biztosan le tudjuk szűkíteni a lehetséges technológiák listáját.

Felmerülhet, hogy miért nem használjuk a kültéri helyzetmeghatározó rendszereket beltérben is. Hiszen a domináns kültéri pozícionálásra használt technológia GNSS (Global Navigation Satellite Systems) kültéri körülmények között pontos, világ szintű lefedettsége van és még számos előnyét lehetne ide felsorolni, ami miatt ígéretes megoldásnak tűnne. (A GNSS körébe tartozik a hétköznapi életben ismert GPS is). Beltéri körülmények között mégis rosszul teljesít, mely a bel- és kültéri környezet közötti jelentős eltérésből fakad. Ezért érdemes kitérni általánosságban a beltéri környezet jellemzőire.

Jelentős eltérés a kül- és beltér között, hogy a nyílt szabadtéri környezethez képest egy zárt helységen/területen belülről kell információt szolgáltatni, így elkerülhetetlen a reflexiók miatt a többutas terjedés. Ez azt jelenti, hogy a jel nem csak közvetlen úton jut el az adótól a vevőig, hanem más útvonalakon keresztül is (falakról/padlóról visszaverődve). A pontos helymeghatározás érdekében ezeket a jeleket el kell tudni különíteni a közvetlenül beérkező direkt jelektől. Sokkal több akadály van a jel útjában és egyes esetekben az sem biztos, hogy a jel az adótól a vevőig közvetlen úton el tud jutni (Non-Line-of-Sight). Ilyenkor minden képpen számolni kell a többutas terjedéssel. A környezet is sokkal gyorsabban változik, például emberek haladnak át, leeresztik az ablakokon a redőnyt, kinyitnak egy ablakot/ajtót. Ennek hatására a hőmérséklet is gyakran ugrásszerűen megváltozhat.

1.2 Elterjedtebb beltéri helyzetmeghatározásra használt technológiák

1.2.1 Látáson alapuló rendszerek

Egy kamera által lefedett terület nagysága többnyire 1-10méter közé esik. Nagymértékű pontosság érhető el (µm-dm). Alapvető mérési módszere az AoA (Angle of Arrival), mellyel egy 2D-s pozíciót kapunk. Ez a módszer nem ad mélységi információt, így azt kamera mozgatásával, referencia pontokból, illetve más módszerekkel lehet megkapni. Az egyik legjelentősebb probléma a változó megvilágítás. Ezek kiküszöbölésére markereket szoktak elhelyezni referencia pontként (barcodes, színes körök). Nyilvánvaló hogy a világítás hiányában nem alkalmazható ez a technológia. Jellemző alkalmazási területe a robotnavigáció.

1.2.2 Infravörös sugárzás (IR)

Az infravörös sugárzás már nem a látható tartományba esik és nem befolyásoló tényező a megvilágítottság sem. Hatótávolsága közelítőleg 5m-ig terjed, pontossága egyes esetekben akár a tized mm-t is elérheti. Főbb alkalmazási területe az emberek detektálása (hőtérkép), nyomon követés (tracking) illetve mozgásdetektálás. Jellemző alkalmazási területei: riasztó rendszerek érzékelői és például az Xbox Kinect megoldása.

1.2.3 Tapintáson alapuló pozíció meghatározás

Hátránya, hogy kontaktus kell hozzá, illetve nagyon drága, de µm pontossággal meghatározható a vizsgált objektum pozíciója. A mérési tartomány általában pár méter.

1.2.4 'Combined polar systems'

Ide tartoznak a nagy pontosságú lézer szkennerek, lézer radarok, és még pár igen nagy pontossággal rendelkező készülék, melyek kiemelkedően magas áruk miatt navigációhoz csak nagyon ritkán, de például 3D-s modellezéshez az iparban gyakran használnak.

1.2.5 Hang alapú helymeghatározás

A hang egy mechanikus hullám, melynek terjedési sebessége jelentősen függ a hőmérséklettől (ultra hangnál 10m-en 1fok különbség 2mm-es szórást okoz). Jelentős romlást okoznak a légáramlatok is, így kültéri használat mellet nagyon pontatlan. Zavart okoz még a Doppler-effektus is, ha az adó és vevő egymáshoz képest gyorsan mozog. A hang lassú terjedése miatt az idő szinkronizáció jelentősen könnyebb, mint a fénysebességgel terjedő jelek esetében. Kedvező körülmények mellett cm pontosság érhető el.

1.2.6 WLAN (Wireless Local Area Networks)

Mint a GNSS-nél itt is felmerül, hogy ha az adott hálózat már a legtöbb helyen ki van építve, azaz közel globálisnak tekinthető a lefedettség, akkor nem kell foglalkozni a rendszer telepítésével, vagy annak kiépítése relatíve olcsó. Ezeknek a hálózatoknak a hatótávolsága 50-100méterig terjed, és nem szükséges a közvetlen rálátás sem. Alapvető mérési módszere az RSSI (Received Signal Strength Indicator), azaz a vett jelerősségből következtetünk az adó helyére. A WLAN alapú pozíció szolgáltatások igen pontatlannak bizonyultak (2-50m).

1.2.7 RFID (Radio Frequency Identification)

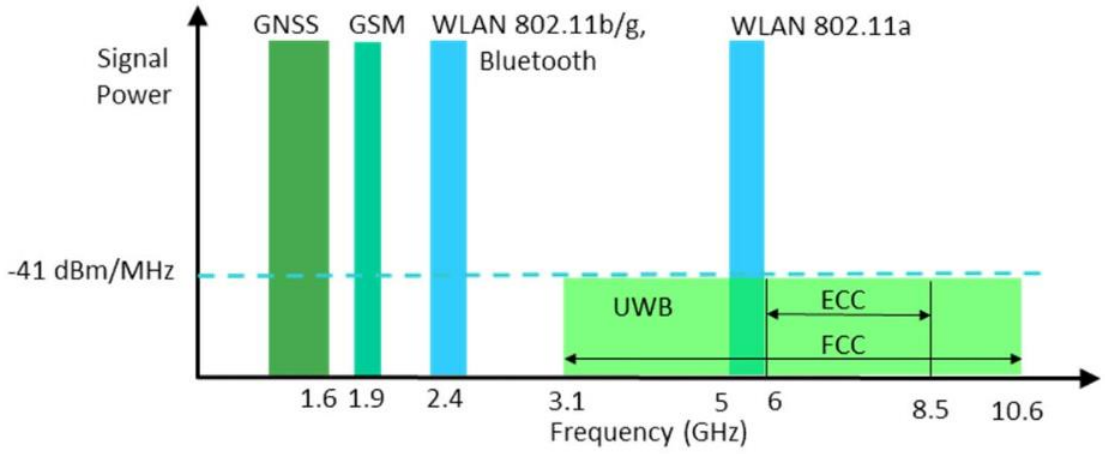
Az RFID-s rendszerek tipikus pontossága a dm-m tartományba esik. A pontossága jelentősen függ a telepített tagek (az információt tartalmazó egységek) sűrűségétől és az olvasó hatótávolságától. Alapvető mérési eljárás a Cell of Origin, mely abból áll, hogy az legközelebb lévő tagben tárolt információt megkapja az olvasó. Egyes esetekben ez még kiegészül az RSSI-vel, amely pontosabb helymeghatározást tesz lehetővé. A legelterjedtebb alkalmazási területek a közúti navigáció gyalogosok számára, illetve a termékek nyomon követése, esetleg lopás védelem.

1.2.8 Mágneses helymeghatározás

Ez a technológia egyáltalán nem igényli a közvetlen rálátást a vevő és adó között. Ezen felül invariáns a hőmérséklet illetve nyomásváltozásra. Ebből adódik, hogy ez a technológia leginkább azokon a területeken használatos, ahol anyagokon fal/talaj kell áthatolni: anyagvizsgálat, földalatti területek vizsgálata, vagy egészségügyben (MRI).

1.2.9 UWB (Ultra-Wideband)

Egy rádió hullám UWB-nak számít, ha sávszélessége meghaladja az 500MHZ-et, vagy a vivő frekvencia 20%-át. Ezzel a technológiával elég könnyen detektálható a többutas terjedés, hiszen a nagy sávszélességhez keskeny időintervallum tartozik. Ebből következik, hogy pontos távolság mérés valósítható meg ezzel a technológiával. Az alacsonyabb frekvenciás komponensek képesek behatolni a falakba, és így lehetőség van akár anyagvizsgálatra is. Hatósági szabályzások Európában (ECC) előírják, hogy a teljesítmény sűrűség nem haladhatja meg a -41.3 dBm/MHz értéket és az UWB-s kommunikációra felhasználható sávszélességnek 3.1-4.1GHz vagy 6.0 és 9GHz közé kell esnie, az amerikai korlátozást az FCC írja le (1.1 ábra). Ezek az értékek az elérhető maximális hatótávolságot is erősen korlátozzák (100m). Viszont az előírások biztosítják, hogy nem lesz interferencia más keskenysávú kommunikációval és ilyen minimális kisugárzott energia mellett az emberi szervezetre sincs káros hatása az UWB-s kommunikációnak. A legelterjedtebb mérési módszerek A ToA (Time of Arrival), TWR (Two Way Ranging) illetve TDoA (Time Difference of Arrival). A navigációs rendszereknél alkalmazott technikákkal cm-es pontosság is elérhető, de létezik olyan termék, ahol tized milliméter pontosságot ígérnek (Xethru). Az utóbb említett szenzor egymagában tartalmazza az adót és a vevőt is, mely cél alkmazása inkább légzés, jelenlét detektálásra irányul, illetve minden olyan finom/apró mozgás detektálására, ami a statikus környezettől eltér.



1.1 ábra: Az UWB spectruma és a szabályzások

Az említett technológiákon kívül is nagyon sok féle megoldás van beltéri helymeghatározásra, például High Sensitive GNSS, Pszeudolites és gyakorlatilag bármilyen rádiós jel felhasználható beltéri lokalizációra.

1.3 Az általam használt beltéri pozícionálásra használt rendszer működésének bemutatása, jellemzőinek leírása

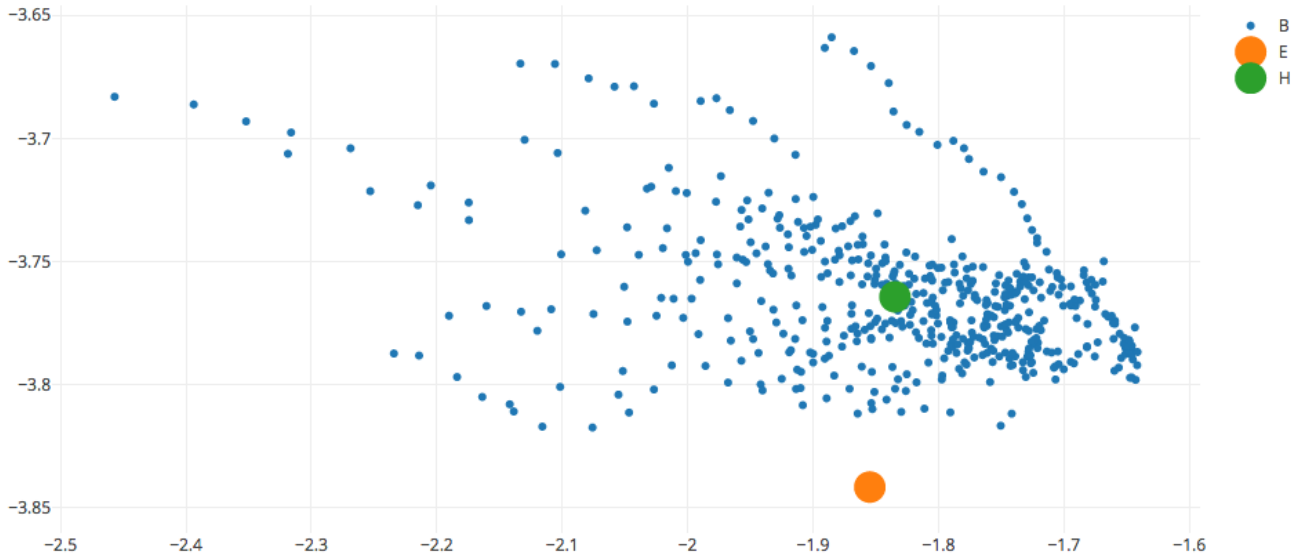
A beltéri navigáláshoz alapvetően felhasznált helymeghatározás az OMTLAB által fejlesztett SUNSTONE RTLS-en (Real-Time Locating System) alapul. Ennek felépítése 3 fő egységre oszlik: Tag-ek, Fali egységek, illetve a lefedett területi egységek számának megfelelő számú központiegység, mely az én alkalmazásomban mindig egy volt.

A tagek, kis méretű akkumulátoros tápellátású rádióadók. A bennük található gyorsulásmérő segítségével képesek érzékelni, ha megmozdították őket, amikor is periodikus rádiós jeleket kezdenek adni. Ezen rádiós jelek tartalmazzák a tag azonosítóját, a mért gyorsulás értéket és az akkuszintet. A vett jelekből az OMT RTLS rendszer pedig megállapítja a tag helyzetét.

A tagek a navigálni kívánt kocsi sarkain/távoli pontjain lettek elhelyezve, így egy multi tag rendszert alkotva, mellyel növelve a navigálni kívánt jármű pozíciójának pontosságát. Egy tag pontossága közelítőleg 40cm-en belül van, de ideális környezetben ez az érték akár a 10cm-t is elérheti.

A kommunikáció a tagek felől indul UWB-s impulzusok kibocsátásával, amit a faliegységek vesznek. A faliegységek koordinátái ismertek, így a jel beérkezési idejének különbségéből meghatározható egy 3D-s pozíció. A kibocsátott rádióhullámok közel fénysebességgel terjednek, így a beérkezési idők közötti időkülönbség nagyságrendje könnyen nanoszekundum alá is eshet. Ilyen kis időközök pontos meghatározása jelentősen befolyásolja a rendszer pontosságát ezért egy időszinkron alkalmazása is szükséges . Ezt az időszinkront a központi egység biztosítja úgy, hogy Ethernet kábellel össze van kötve az adott területi egységet lefedő faliegyégekkel. Ezen kívül a központi egység biztosítja a faliegységek tápellátását és végzi az időadatok gyűjtését. Az adatok gyűjtését a központi egységben egy Raspberry Pi3 típusú miniszámítógépen futó szoftver végzi. A beérkezett időmérési adatok egy nagy számítási kapacitású szerver gépre továbbítódnak a központi egység(ek)ből, ahol multilaterációs algoritmusok (TDoA, ToA) segítségével történik a tag-ek helyzetének megállapítása.

Az OMTLAB RTLS rendszerének rádiós egységei UWB Impulzusrádiók. Az információt spektrálisan formált rövid (néhány száz pikó szekundum) impulzusok sorozatával kódolva juttatják el egymásnak. A rádiók megfelelnek a IEEE 802.15.4a szabványban lefektetett UWB fizikai réteg specifikációjának.



1.2 ábra: 30 másodpercig végzett tesztmérés egy fix pozíciójú tag-en (Győr, Széchenyi IstvánEgyetem)

A kék pontok a nyers mérések, zöld pont a mérések átlaga, narancssárga pont a műszerrel mért referencia pont, a tengely értékek méterben értendőek.

2 A győri Széchenyi István Egyetemen végzet UWB pozíció adatok alapján megvalósított jármű irányítása akadálymentes útvonalon

2.1 A megoldandó feladat és az előforduló akadályok rövid ismertetése

A projekt a SZTAKI-val közreműködve a győri egyetem kutatói laborjában történt az Ipar 4.0 Nemzeti Technológia Platform (a továbbiakban röviden Ipar 4.0) alatt. Az én feladatom egy ún. tankhajtású kiskocsi irányítása volt az UWB-s adatok alapján. A teljesíteni kívánt feladat megnevezése: A laboratóriumban kijelölt tetszőleges pontokon (amik között feltételezzük, hogy akadálymentes közlekedés valósítható meg) a kocsinak végig kell haladnia (sorrendhelyesen). Az adott pontokat egy meghatározott hibaértéken belül kell érinteni (mely értéke konstans 0.4m). A kocsi akkor indulhat a következő pont felé, ha ez teljesült. Ha célpont megközelítése a meghatározott hibaértékén belül nem sikerült, akkor ezt detektálni kell és újabb próbát kell tenni a kijelölt pont megközelítésére.

A laborban külön nehézséget jelentet a pontos pozícionálás a környezeti tényezők, illetve a terem felépítése miatt. Környezeti tényezők alatt értem, hogy gyakran járkáltak emberek, akik esetleg több faliegységet is kitakarva zavarták a jel terjedését, a terem automatizáltsága miatt, a falmentén az ablakokra szerelt redőnyök automatikusan változtatták helyzetüket a napszaknak megfelelően, így időről időre változott a környezet. A terem alacsony belmagassága (~2.5m) miatt, illetve a terem közepén lévő oszlopok jelenéte is elősegítette a jel szóródását és a többutasterjedést hatásának növelését. Emiatt nem mindig kaptam friss pozíciót az UWB-s rendszertől, hiszen, ha a mért időadatokból nem lehetett egyértelműen következtetni a pozícióra, akkor a rendszer azt nem frissítette. Ez azért volt megengedett, mert a kocsi irányítása nagyon kis sebességgel történt (0.1m/s), így pár másodperces kimaradás a gyakorlati megvalósításban nem okozott gondot. Ritka esetekben viszont ennél hosszabb időtartamok is előfordultak. Ennek a problémának a kiküszöbölésére a kocsi mozgásának ismeretében az ún „Dead Reckoning” módszer vált be a gyakorlatban. A Dead Reckoning módszer a régebbi helyzetből, a jármű sebességéből és az eltelt idő ismeretében határozza meg a kocsi adott pillanathoz tartozó helyzetét. Ennek az eljárásnak a legjelentősebb hátránya, hogy az idő elteltével egyre nagyobb a bizonytalansága a számolt pozíciónak. Ebből adódik, hogy ez a módszer csak annak a rövid időtartamnak az áthidalására alkalmas, míg a rendszer nem frissíti a kocsi helyzetét.

A kocsi útvonalon való haladásánál, a pontok között közel egyenes vonalú mozgás megvalósítása volt a cél. A labor méretei miatt arra törekedtem, hogy a kocsi útvonala minél inkább tartson a pontokat összekötő szakaszhoz. Ennek megvalósításához egy PID szabályzót használtam, melynek paramétereit majd a későbbiekben leírt módon kísérleti hangolással határoztam meg.

A szabályzón kívül szükség volt egy irányító algoritmusra és a kocsi valós idejű nyomon követéséhez szükségesnek találtam egy grafikus felület létrehozását. A kocsi irányítását Python 3-ban a grafikus felületet QML-ben írtam.

2.2 A kocsinak kiadható parancsok és az állapotot leíró lekérhető adatok

A kommunikáció a kocsival REST API-n keresztülvalósult meg.

A kocsiról lekérhető adatok:

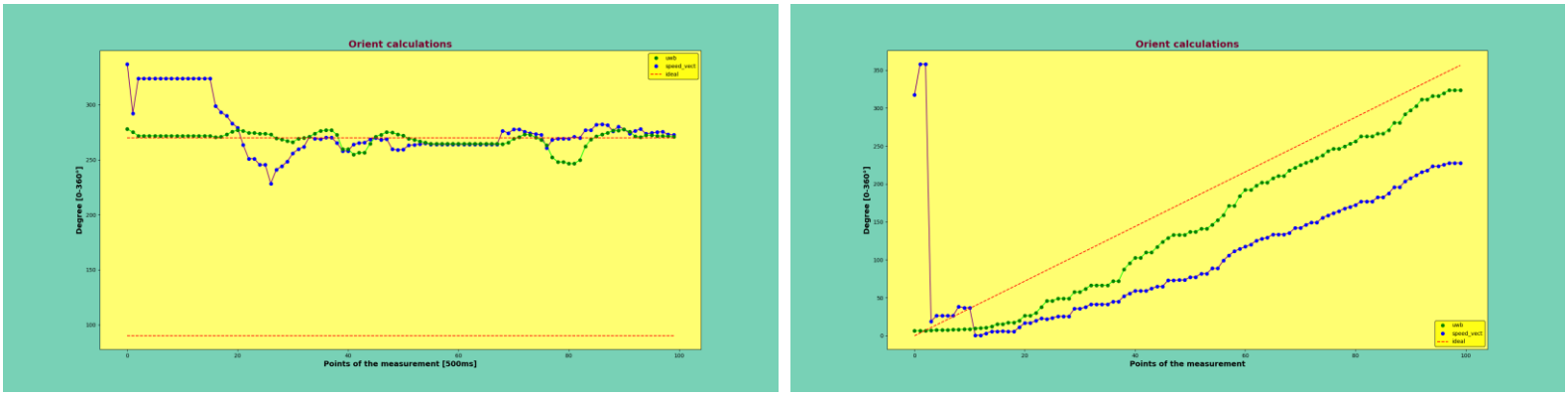
- a kocsi 2D-s pozíciója (~30cm pontosság, de helyfüggő),

- a kocsi orientációja (+/- 35fok, mozgás közben pontosabb),
- illetve egy 2D-s sebesség vektor.

A kiadható parancsok:

- a két kerék sebessége,
- a maximális gyorsulás értéke és
- a két kerék sebességének a kiadott értéken való tartásához tartozó időintervallum (timeout)

A 2.1-es ábra a sebesség vektorból számítható orientációt (kék pontok) veti össze az UWB-s orientációval (zöld pontok) egyenes vonalú egyenletes mozgás és körvonalon való mozgás során. A piros szaggatott vonal az ideális értéket szemlélteti egyenes pályán 270°, körpályán pedig egy 0°-360°-ig növekvő egyenes.



2.1 Az orientáció változása egyenes pályán ill körpályán,.való egyenletes haladáskor (pontok közötti idő: 500ms)

2.3 A grafikus felület bemutatása

A grafikus felületet QML-ben írtam. A QML egy deklaratív programozási nyelv, melynek segítségével a felhasználói felület vizuális komponensekkel és a komponensek közötti kapcsolat leírásával megadható. A főbb komponensek, amire a GUI (Graphical User Interface) feloszthető: Window, szimulációs terület, listmodel-ek, adatok bevitelére alkalmas boxok, valós idejű információk megjelenítéséért felelő boxok illetve gombok. A következőkben ezeket részletezem:

A felület egészét magába foglalja a window. Itt írható le, hogy maga a felület mekkora legyen, illetve ha átméretezzük az ablakot, akkor innen küldünk a python részére értesítést, hogy például a méter/pixel arány megváltozott, és a pythonban tárolt adatokat ennek megfelelően rajzoljuk ki újra az átméretezett ablakra. Értelemszerű, hogy a layout beépített, tisztán QML-es elemeit nem a Pythonból méretezzük át, hanem kezdetektől fogva a QML-es szülő objektumok szélességével és hosszúságával definiáljuk a méretüket: így a szülő objektum átméretezésével dinamikusan azok is változtatják a méretüket.

A szimulációs terület az ablak középső részén helyezkedik el. A szimulálni kívánt közlekedési eszköz (AGV) ezen területet nem hagyhatja el. A terület méretét Pythonban definiáljuk és a legoptimálisabban, de a valódi méretarányt tartva jelenik meg.

A szimulációs felületen beadott útvonalért illetve a faliegységek helyzetének ábrázolásáért felelősek a Listmodel-ek. A Listmodel-en belül tároljuk az adatokat és írjuk le a szerkezetüket. A komponensben tároljuk az adott elemek ábráját, a listview-ban pedig ezeket csoportosítjuk össze

(melyik adathoz melyik ábra tartozik). Én az útvonal pontjait kék keresztekkel, a faliegységeket pedig zöld kör alapon fekete kereszttel jelenítettem meg.

A jobboldalon lévő boxok-ban a kocsihoz illetve a vezérléshez kapcsolódó adatokat jelenítettem meg, míg a baloldalon lévő boxok az adatok bevitelére szolgálnak. Ezek mind alapjában véve „Rectangular" elemben elhelyezett táblázatokból épülnek fel. A bal felsővel kezdve lefelé haladva:

- Pozíció beállítása
- autó haladási sebességének állítása
- autó helyben fordulási sebessége
- jobb és balkerék sebességének beállítása/frissítése illetve
- a kocsi megállítása.

A terem koordináta rendszerének origója a szoba közepén van, illetve az X tengely a képernyő teteje felé mutat, míg az Y tengely balra. A baloldali adatbeviteli táblázatokat csak a grafikus felület teszteléséhez használtam (a beadott pontokon való megfelelő haladásra, a körmozgás tesztelésére a két kerék sebességének beállításával…), a győri teszteknél ezek érintetlenül maradnak.

A jobb oldali sáv ugyanakkor valós tesztnél is ugyanúgy hasznos információt szolgáltat (fentről lefelé haladva):

- kocsi jelenlegi pozíciójáról és a GUI-ban értelmezett orientációról (a szimuláció terület középpontjából nézve a függőleges a terület „tetejéről" kiindulva (0fok) óramutató járásával megegyezve nő egy teljes fordulat megtételéig (360fok = 0fok)) ;
- a keréksebességekről és a szabályzó bemenetére jutó hibáról és
- a szellem kocsi pozíciójáról és orientációjáról.

Valós tesztnél a szellem kocsi adatait a gyakran frissülő UWB-s adatok szolgáltatják, míg a kocsi adatok ezen adatok átlagolásával jönnek létre illetve UWB-s adat hiányában a mozgás ismeretében is frissülnek. A táblázatokhoz tartozik még a jobb alsó sarokban található ablakméretet ábrázoló téglalap is, amelyből az épp aktuális ablak méretét tudjuk leolvasni, és az e fölött található kék dobozban olvasható helyszíni tesztnél, hogy a vezérlés hányszor történt UWB-s adatok alapján és hányszor történt Dead Reckoning alapján Pythonból számított pozíciós érték alapján.

Az ablak felbontását tartalmazó táblázattal egy sorban találhatóak a GUI főbb gombjai, balról jobbra:

- „Go to start point": Multi funkciós gomb a beadott útvonal első pontjába viszi a kocsit, majd ahogy oda ért a felirat átváltozik Start going on the route-ra, erre rákattintva a kocsi végig halad a beadott útvonal pontjain. Miközben a kocsi mozog a gomb felirata RUNNING WAIT…. -re vált át. (Ez a gomb csak virtuális teszt során használható)
- A „Set Points Route": Ennek a gombnak a lenyomása után a szimulációs terület felületére kattintva útvonal pontok helyezhetők le, melyek érintési sorrendje megegyezik a letételével. Miután az összes „útvonalpontot" lehelyeztük újra kattintsunk a gombra, hogy véletlen kattintásra ne tudjunk újabb pontot lehelyezni. Útvonal pontot csak oda tudunk lehelyezni, ahol a kocsi fizikai paraméterei alapján még elfér, illetve el tud fordulni.
- A következő gombbal a legutóbbi útvonalpontot tudjuk kitörölni.
- A „Points WALL UNITS" gombbal, a faliegységeket tudjuk vizuálisan a Pythonban letárolt koordináták alapján megjeleníteni.
- A „switch agv_pos mode" gombot csak éles teszt során használjuk, ha a valós kocsit akarjuk a beadott útvonal pontjain végig vinni.

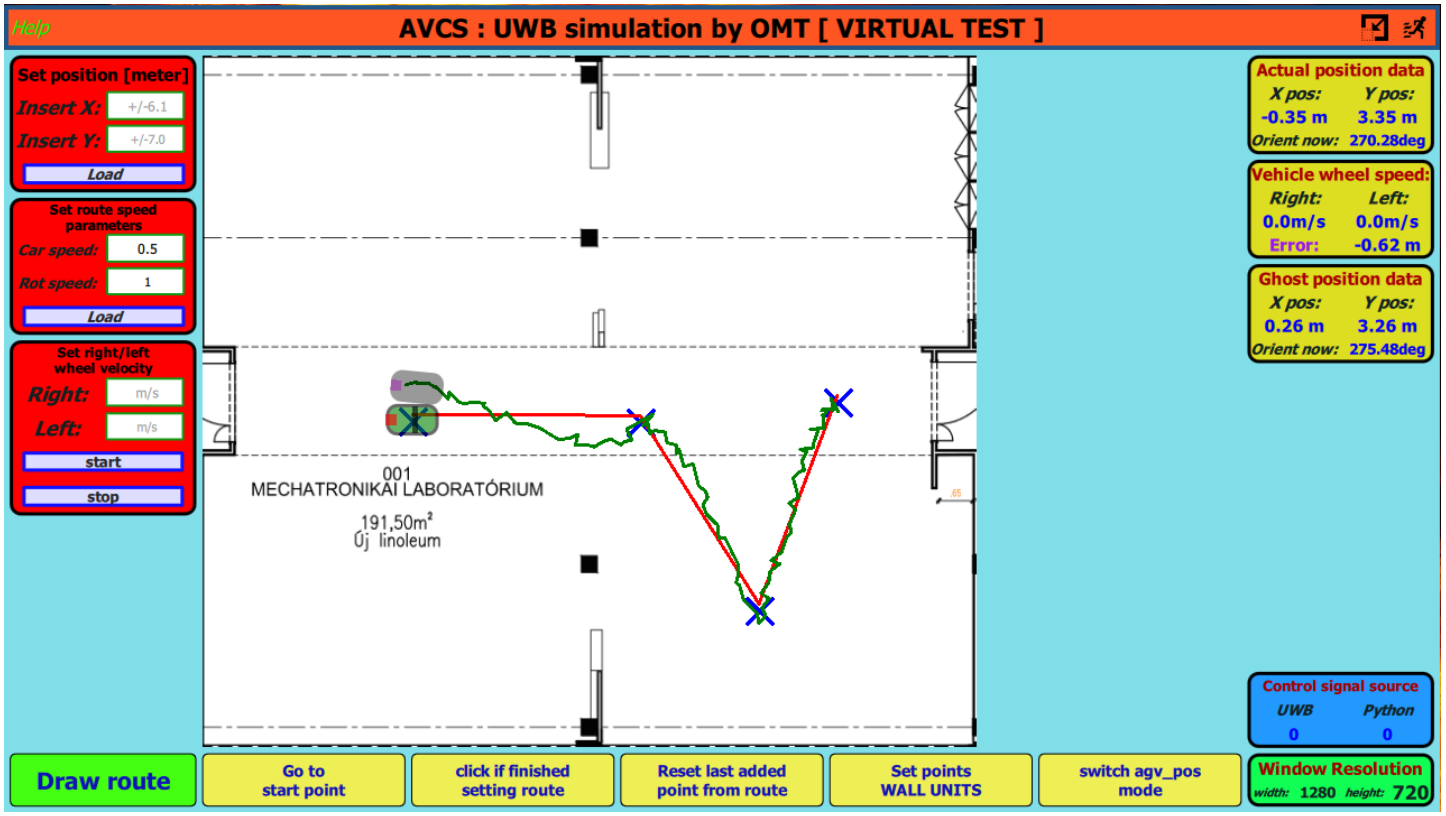
- A „Draw Route” feliratú gombbal a szellem kocsi [zöld színű csík] illetve a kocsi [piros színű csík] által a gomb megnyomásáig megtett utat lehet kirajzolni. Mivel a rajzolás nagyon sok pont között történik és elég idő igényes, így hosszabb útvonal esetén akár fél percig is eltarthat. Ezt türelmesen várjuk ki!

A Pythonból jövő PyQtSignalok kezelését, és az azok által hordozott adatok feldolgozását a QML Connection részében kell leírni.

Ahhoz, hogy az ábrák a valóságnak megfelelően jelenjenek meg, szükség volt egy függvényre, ami a valós koordinátákat a grafikus felületben felvett koordináta rendszerbe transzformálja. Ezt a transzformációt az orientáció értékekre is el kellett végezni.

A 2.2-es ábrán látható a GUI és egy virtuális teszt futtatásának eredménye. A program lehetőséget ad egy szinuszos és egy random zaj szuperponálására, amivel a környezeti zavarok emulálhatók. Az így kapott pontokon halad végig a szellem kocsi (zöld vonal), az ideális úton pedig a ténylegesen kocsi mozgását szimuláljuk ha nem lenne hibajel (piros vonal).

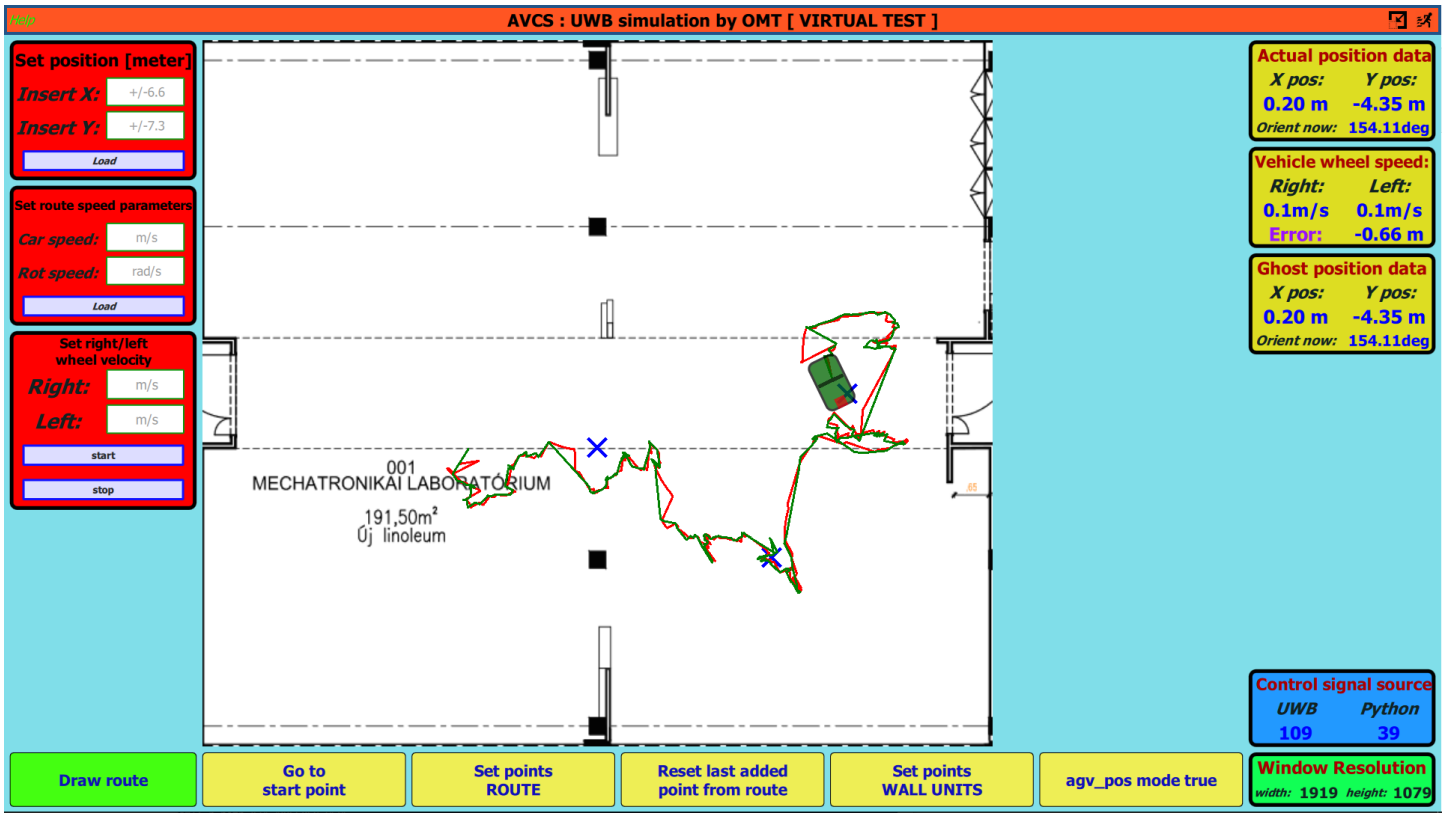
A győri tesztek során a zöld vonal az UWB-s pozíció adatok összekötésével rajzolódott ki, és ha minden esetben UWB-s adat alapján történt a jármű irányítása, akkor a piros útvonalat teljesen lefedte a zöld. Viszont, ha nem érkezett pár/több másodpercig új UWB-s pozíció akkor a Pythonból számított elmozdulást a piros útvonallal szemlélteti az útvonal kirajzolásáért felelős program. Amint újra frissült az UWB-s pozíció a piros útvonal az UWB pozíciókból meghatározott helyre „ugrik” és újra a zöld útvonal fedésébe kerül. (2.3 ábra)



2.2 Virtuális teszt futtatása

A 2.3-as ábrán az is megfigyelhető, hogy az utolsó (3.) pontot a kocsi először nem közelítette meg kellő pontossággal, így a kocsinak meg kellett fordulnia és csak ezután teljesült a kijelölt pontot 0.4m-es

hatósugárban megközelíteni. A kék boksz-ban látható, hogy a vezérlés a teljes útvonal megtétele alatt 109-szer történt UWB-s adat alapján, 39-szer pedig Python-ból Dead Reckoning alapján.



2.3 Valós teszt futtatása a győri laboratóriumban

2.4 A program felépítése

A program jelentős erőforrásokat igényel mind a grafikus felület futtatása/frissítése mind a matematikai számítások és a vezérlés részéről. Ezen okból kifolyólag célszerű volt a multi-threading használata, azaz a program párhuzamosítható részeinek több szálon történő egyidejű futtatása. Az alapból futó Python szálon kívül még két QT thread-et hoztam létre (A QT-os környezet melynek a qml is egy részhalmazát képzi, megkövetelte a QT thread-ek használatát). A három szál a következő funkciókat látja el:

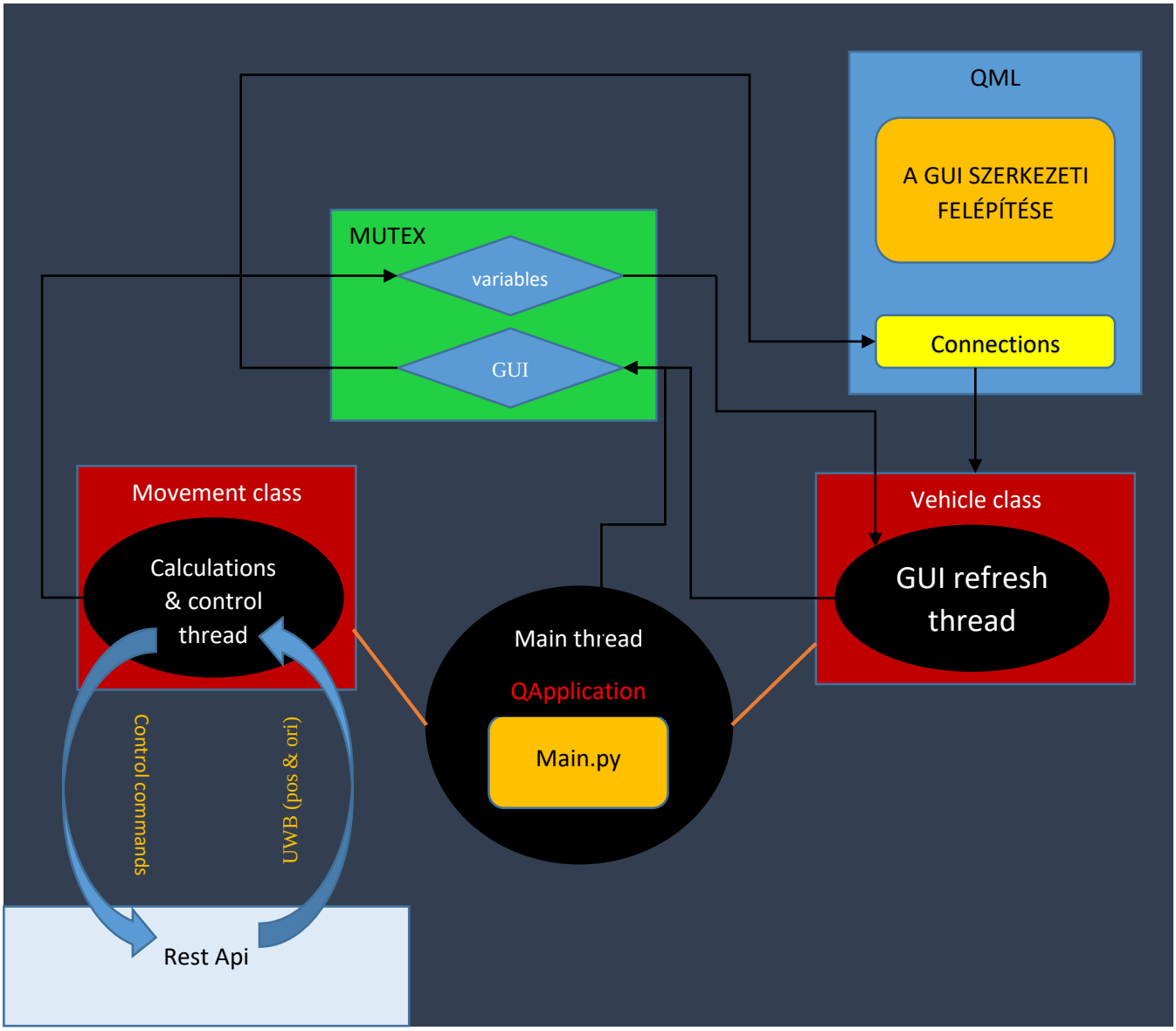
2.4.1 Az alapból futó Python thread (Main thread)

Innen futtatjuk a QGuiApplication-t amely a teljes idő alatt a háttérben fut, míg ki nem lépünk a programból. Itt példányosítjuk a másik két osztályt is (Vehicle, Movement), melyeknek a run() függvényét felülírva és a start() függvény meghívásával külön threadeket tudunk létrehozni (Csak a run() függvényen belül leírt program fog külön threaden futni). Itt adjuk meg, hogy milyen néven érjük el annak az osztálynak (Vehicle) a példányát QML-ből, aminek signal/slot-jaival a QML (GUI) rész fog kommunikálni. Innen töltjük be a futtatni kívánt QML-fájt is. Mivel gyakorlatilag ezen a szálon fut a GUI, illetve az onnan hívott Python függvények is (sorosan), ez a szál alapból eléggé leterhelt, ezen már más számításokat lehetőleg nem végzünk.

A Movement class thread-je felelős szinte minden a vezérléssel kapcsolatosabb számítás igényesebb feladatért, ami Pythonban történik. Ezen a szálon történik a szabályzás, it fut az irányítóalgoritmus, itt történik a koordináta transzformáció, körpályák számítása, az útvonal lementése, melyet a folyamat végén kirajzolunk illetve a pálya adatokkal kapcsolatos további számítás, amire még ezeken kívül szükség van.

A Vehicle class thread-je felelős azért, hogy a GUI-ban valós időben jelenjen meg a kocsi helyzete. Az itt létrehozott thread 20ms-onként küldi meg a kocsi jelenleg becsült helyzetét orientációját és a hibajelet. Innen frissítjük a GUI-ban lévő kék színű bokszot is, mely vezérlés forrásáról ad információt (Python/UWB).

A program felépítését folyamat ábrával a 2.4-es ábrán tekinthetjük meg.



2.4. ábra: A program felépítése

2.4.2 A számításokért felelős python thread részletes ismertetése

A threaden való számítások elsősorban a Rest API-tól lekért UWB-s adatok alapján történnek. Miután rendszeres időközönként lekértük a pozíciós és orientációs adatokat, első lépésként meghívjuk az ezekhez írt koordináta transzformációs függvényeket, hogy a GUI oldaláról az adatoknak megfelelő vizuális ábrázolása már könnyen menjen. A transzformáció után a kapott adatokat a mutex-el védett variable változóba töltjük, ahonnan majd a GUI frissítéséért felelős szál azokat kiolvassa és megküldi a vizuális résznek (QML). Mutex-ra a multi-threading miatt van szükség. (Például előfordulhat, hogy ugyan abban az időpillanatban írjuk az egyik thread-el a változót, mint amikor az adott változóban tárolt értékeket ki akarjuk olvasni. Ez problémákhoz vezethet, ugyanis nem definiált, hogy ilyenkor a kiolvasott értékek mik lesznek.)

Ez alapján ismert- az autó kezdeti helyzete és közelítő orientációja (Fontos lenne pontosan ismerni a kocsi kezdeti orientációját, ezért kísérletet tettem arra, hogy pontosabb kezdeti orientációt kapjak kalibrálással: a program futtatásának elején a kocsival előre és hátra tolatok, majd az így kapott pontokra egyenest illesztek és az egyenes adataiból következtetek az orientációra. A győri esetben hely szűkében és a pozíció adatok jelentős szórása miatt az így kapott orientáció sem bizonyult a valóságban pontosabbnak. Nagyobb helyiségekben viszont eredményes lehet ez az eljárás az orientáció pontosítására.) A kocsi vezérléséhez tudnunk kell az útvonalat, amit a GUI „szimulation area” nevű területén a fent leírt módon korábban már kijelöltünk, majd ezeket a koordinátákat lementettük Python-ba.

A vezérlés során szakaszonként megyünk végig az útvonalon. Az adott szakasz mindig a kocsi helyzetének és a következő pont koordinátájának összekötésével jön létre és mindig frissül, amikor elérjük a következő pontot a megfelelő hibahatáron belül (0.4m). Ha elérjük az utolsó útvonal pontot a kocsi megáll.

A vezérléshez szükséges kiszámolni a szakaszok meredekségét is (between_route_point_orient). Amikor elindulunk a kocsival a következő pontba, akkor az első lépés mindig a kocsi befordítása a pontokat összekötő távolság vektor irányának megfelelően. Ez az adat ahhoz is szükséges, hogy menetközben tudjuk, hogy a kocsi jó irányba megy-e. A későbbiekben erről még lesz szó, de a PID szabályzó hibajele nem más, mint ezen szakaszoktól mért távolsága a kocsinak. Így lényegében a szabályzó a szakaszon való előre és hátra haladása mellett ugyanúgy működik így az irányító algoritmusnak kell biztosítania azt, hogy valóban jó irányba haladjunk és ehhez nyújt segítséget szakasz irányultságának kiszámítása. Ez a szakasz orientáció fogja azt is korlátozni, hogy a kocsi milyen meredeken kanyarodhat rá az adott szakaszra és ezzel elkerülve, hogy kellően nagy hibajel hatására túlforduljon a kocsi, esetleg egy végtelen körpályára álljon (ez azért lenne lehetséges, mert a kocsi az útvonalhoz viszonyított pozíciója alapján kanyarodik vagy órajárásával megegyezően, vagy -ellentétesen az adott szakasz felé).

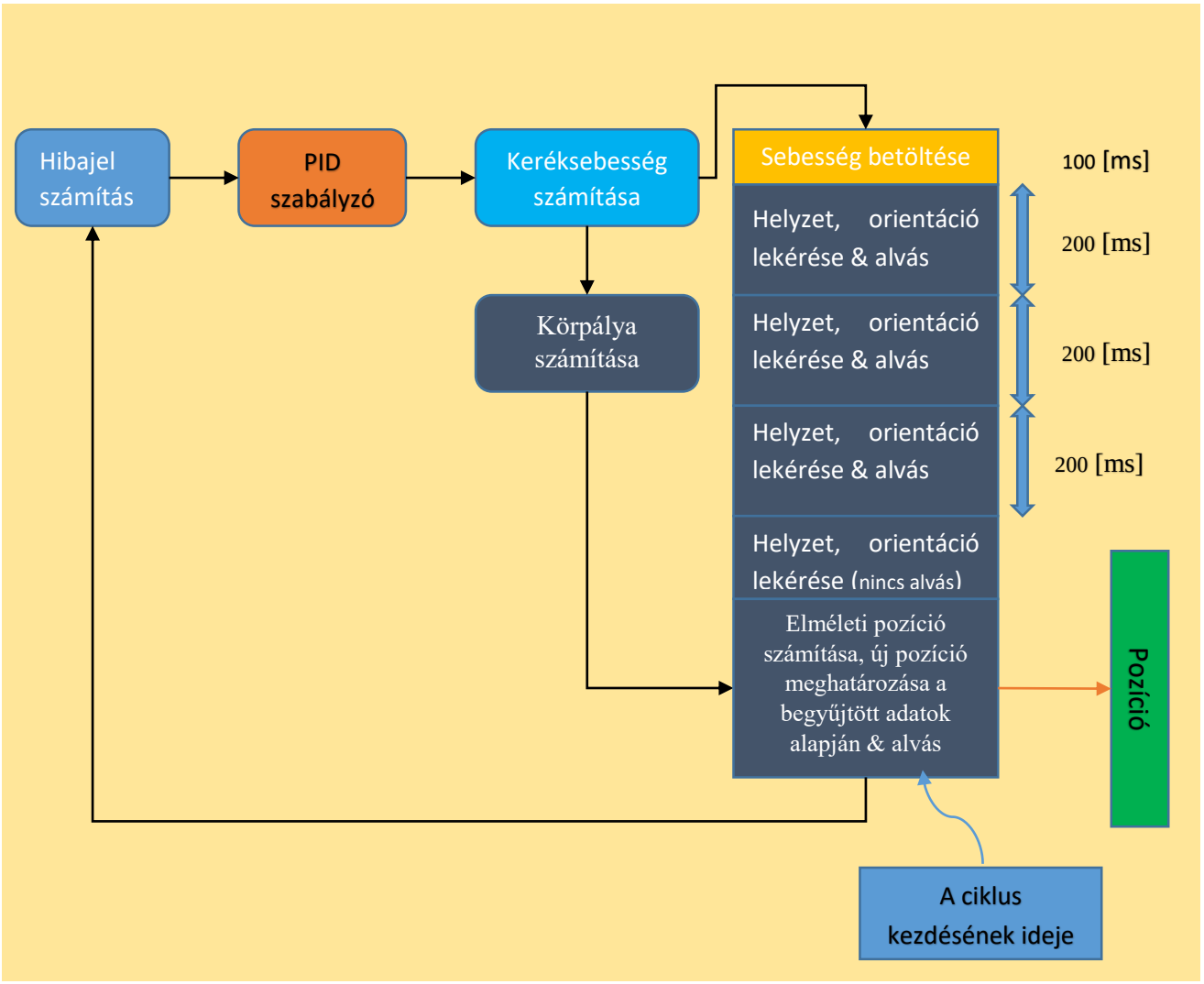
Ahhoz, hogy a megjelenítés és az irányítás időben szinkronban legyen fix időkereteket (time slots) használtam a programban. A vezérlést először 450ms-onként valósítottam meg, majd áttértem 1 másodperces keretekre, mivel a 450ms-onkénti beavatkozás a kocsi hirtelen „rángatását”, feleslegesen sok irányváltását eredményezte (az UWB-s adatok szórása miatt is érdemes a vezérlési kör „átfutását” minél nagy időre vizsgálni, hiszen hosszú távon az adatok átlagolásával szerzett információ jelentősen pontosabb). Utólag átgondolva akár 3-4 másodpercenként is vezérelhettem volna a kocsit, hiszen a két kerék átlag sebessége csak 0.1 m/s volt, ami nem igazán számít gyors mozgásnak. Ugyanakkor úgy találtam, hogy 1 másodperc alatt ezzel a sebességgel az elmozdulás 0.1m, így ezen időtartamon belül

kapott pozíció adatok egyszerű átlagolása megengedett (egy 4 másodperces időintervallumon belül kapott pozíciók átlagolását már nem tartottam volna elfogadható megoldásnak).

2.4.3 A fix időkeretes vezérlés felépítése

A vezérlés kezdetén rögzítésre kerül a kezdeti idő, amihez viszonyítva mérjük a fix 1 másodperces időkereteket. A ciklusok számát egy ciklusszámlálóval rögzítjük. A körfolyamat/ciklus először mindig a hibajel kiszámításával kezdődik (a kocsi és a szakasz távolsága, azaz egyenes-pont távolság), amit a PID szabályzó bemenetére küldünk. Ezután a PID szabályzó kimeneti jelét felhasználva beállítjuk a kocsi két kerekének a sebességét, úgy hogy a kocsi eredő sebessége mindvégig 0.1 m/s legyen. A maximális megengedett sebesség különbség a két kerék között 0.1 m/s lehet. Ha feltesszük, hogy a kerekek új sebességre való gyorsulása elhanyagolhatóan kis idő alatt megtörténik, akkor a kocsi mozgása körpályán való haladással modellezhető. Ezt a közelítő modellt használtam a szabályzó hangolására is, melyet majd a következő pontban részletezek.

Mivel a http szerverrel való kommunikációnál (a kocsi és a számítógépem közötti réteggel) időnként kisebb késleltetések léptek fel (20-30ms), ezért ezt a részt egy 100ms-os időkeretbe tettem. Ezek után 3 darab 200ms-os keretben lekértem a kocsi pozícióját és orientációját. Ilyenkor mindig eltároltam az előző lekérés értékét, és ha az akkor kapott érték megegyezett az aktuális értékkel az azt jelentette, hogy közben a pozíció nem frissült, tehát nem sikerült új adathoz jutni. Az adatok együtt frissültek, tehát akár új orientációt, akár új pozíciót kaptunk abból következett, hogy a másik adatnak is újnak kell lennie. Ha új adat jött, akkor azt az x, y pozíciót hozzáadtuk a ciklusban addig érkezett friss x illetve y pozíciókhoz összegéhez és megnöveltük a számlálót, ami az egy cikluson belüli új adatokat számát tartotta nyilván. A begyűjtött adatok alapján való új pozíció számítása úgy történt, hogy először megnéztük, hogy hányszor jött új adat. Ha ez a szám nem nulla volt, akkor az új pozíciók számával az összegzett x, y pozíciókat leosztva megkaptuk a becsült új pozíciót UWB-s adatok alapján. Ha nem kaptunk új pozíciót a ciklus alatt, akkor keréksebességek, és az eltelt idő ismeretében kiszámoltam, hogy a kocsi jelenleg hol tartózkodik. Ez a számítás a 2.5-ös ábra 6 rubrikára osztott téglalapjának utolsó pontjában történt, az ábrán a könnyebb értelmezhetőség illetve az átláthatóság kedvéért ábrázoltam párhuzamosan az UWB-s adat lekérdező folyamattal párhuzamosan. Az 5. rubrika csak annyiban tér el, hogy itt nem várjuk ki a 200ms-ot, hiszen ennek az előző 3 helyen azért volt fontos szerepe, mert az UWB-s adatok frissítési rátája 10Hz (100ms). Tehát ha ennek az időnek a kétszeresét kivárjuk, akkor már feltételezhetőleg friss adatokat kapunk a rendszertől. Viszont a 5. rubrika után, majd csak a következő ciklusban lesz pozíció kérés, ami feltételezhetően 200ms-nál hosszabb idő múlva következik be, tehát nincs szükség a program altatására. Ugyanakkor elképzelhető, hogy az egyik kérés kicsúszott az rendelkezésre álló időkeretből, így nem gond, ha inkább csak a legutolsó pontban altatjuk el a programot, arra az időre, ami a ciklusnak az egy másodpercből még vissza van. Innentől újra kezdődik a vezérlési kör a hibajel számításával.



2.5 A vezérlési kör működésének szemléltetése

2.4.4 A szabályzó hangolásának menete

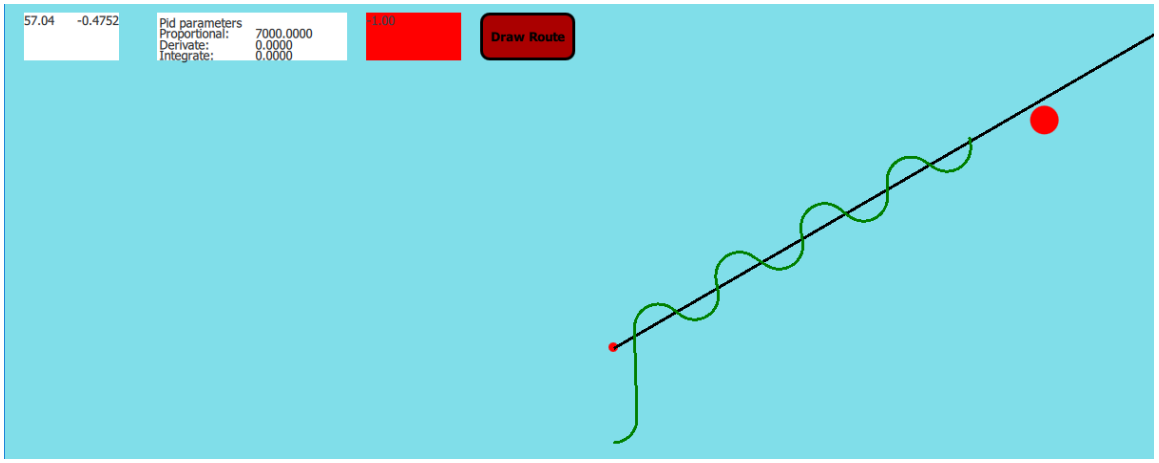
A PID szabályzó paramétereit kísérleti hangolással a Janssen-Offereins módszer segítségével állapítottam meg. Nem a valós rendszeren, hanem QML-ben általam írt grafikus felületen végeztem el, a saját kódomban is implementált vezérlő algoritmussal. Így a vezérlő algoritmus ellenőrzésére is alkalmas volt. Ezután a kapott értékeket felhasználtam a valós rendszeren, mely a várt működést eredményezte. A szabályzó hangolásánál megadható kezdeti változók:

- a kocsi helye
- orientációja, illetve
- a szakasz orientáltsága.

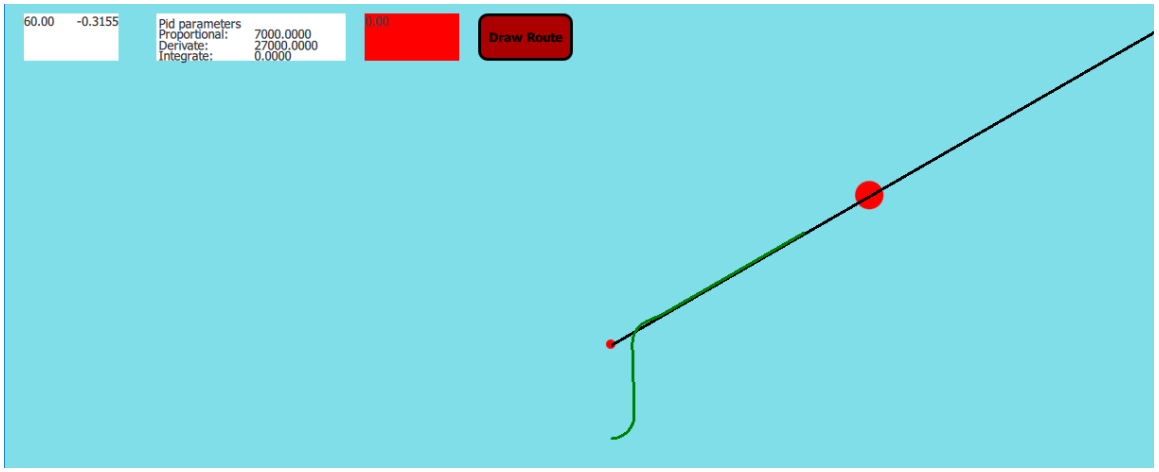
A kocsi helyzetét az ábrán látható piros kör szemlélteti (mivel a kirajzolás időigénye, mire a kirajzolt útvonal „elkészül”, a kocsi már az útvonal egy távolabbi pontján látható).

A következőkben a hangolás lépéseit ismertetem grafikusan ábrákkal szemléltetve.

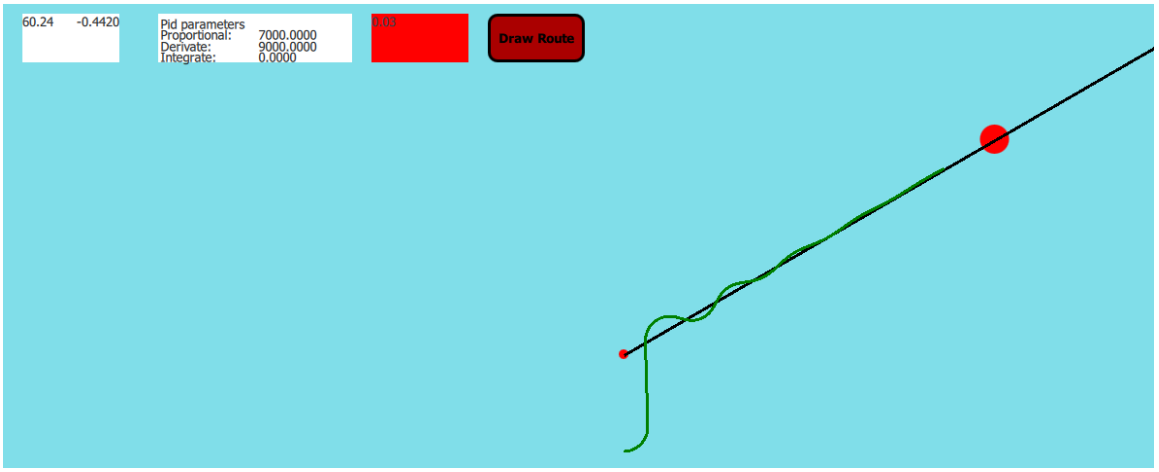
1. Az integrális és a deriválási tényezőt állítsuk nullára. Ezek után növeljük a proporcionális tényezőt (arányos tag) addig, amíg a rendszert a stabilitás határhelyzetére hozzuk és kialakul a szinuszos lengés. Így megkapjuk a max. Ap tényezőt (7000).



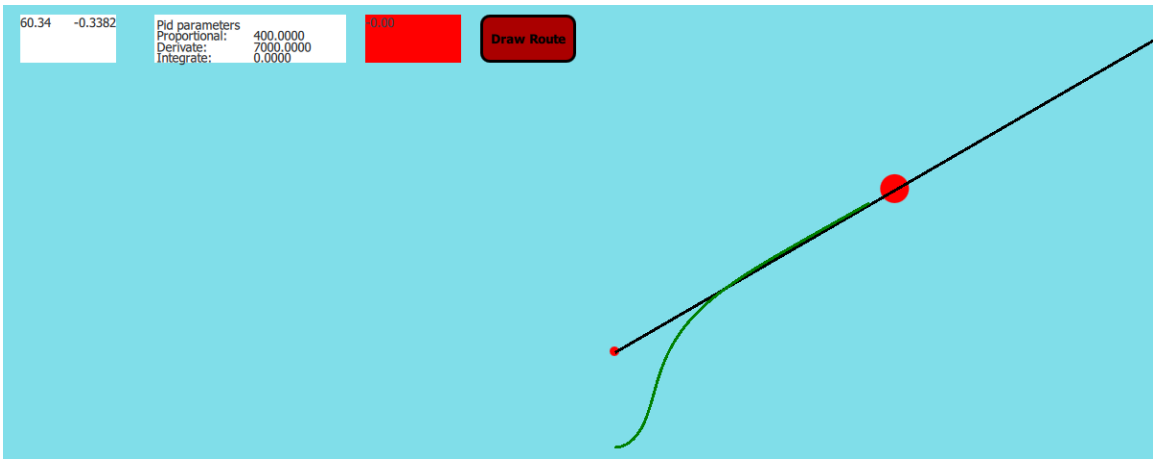
2. Fokozatosan növeljük a deriválási tényezőt, míg el nem érjük a megfelelő csillapítást. Amikor ezt elértük, akkor leolvassuk a Dmax tényezőt. (2700)



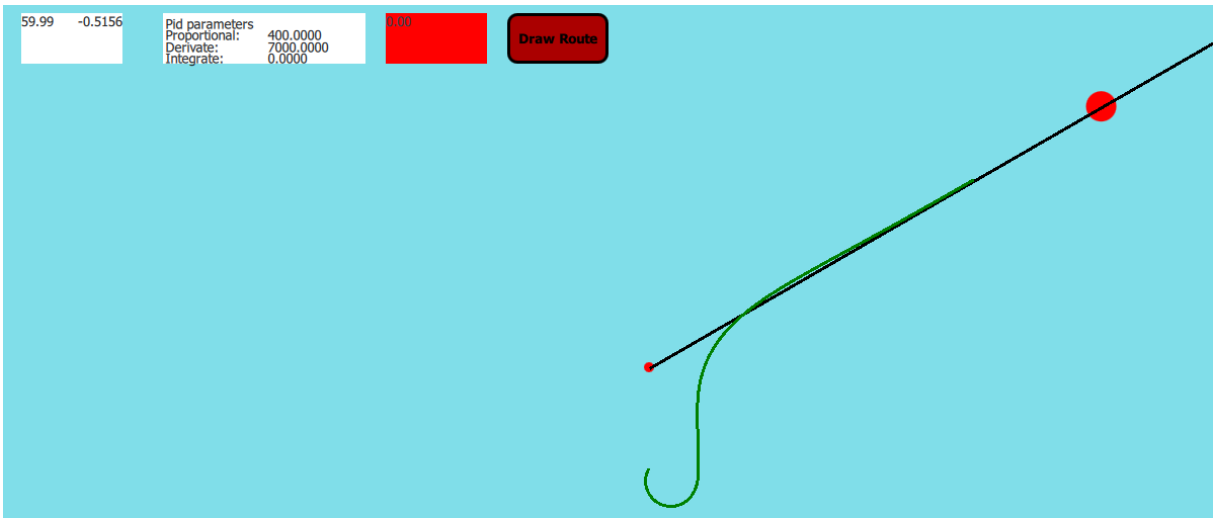
3. Ezek után állítsuk be a következő értékekre a szabályozó paramétereit:
   - [Arányos tag] Apmax
   - [Deriválási tényező] Kd = Dmax/3
   - [Integrálási tényező] Ki : 1/(Kd*4.5)



4. Végül addig csökkentsük az arányos tagot, ameddig a kellő csillapítást el nem érjük. Amint ez megtörtént csak olvassuk le a kapott szabályozó paramétereket.



A fenti hangolás menete folyamán a kocsi kezdeti orientációja 90° volt, a szakaszé pedig 60°. De azt is tesztelhetjük, hogy ha a kocsi nem a megfelelő irányba ált be eredetileg, mert a kezdeti orientációt nem sikerült jól beállítani, akkor miként viselkedik az irányító algoritmus. A következő ábrán látható, hogy 210° kezdeti orientációból hogyan fordul be a kocsi és miként halad a jó irányba:



A szabályzó paramétereinek beállítását könnyítette, hogy a tankhajtásos kocsi mozgását könnyen lelehetet modellezni körpályákkal a két kerék sebességének, illetve azok távolságainak (0.5m) ismeretében.

2.4.5 Az eredmények kiértékelés

Több tesztet is futtattam a laboratóriumban, és a legnagyobb hibát a hálózati kapcsolat minősége illetve ritka esetekben az UWB-s jelek szórásának jelentős mértékű növekedése –esetleg azok hiánya – jelentette. Ha ezek okok nem jelentkeztek, illetve a szórás 0.5m alá esett, akkor megállapítható, hogy a pontokat valóban kellő pontossággal megközelítette a kocsi és a vezérlő algoritmus egy jól működő útvonal követést valósított meg.

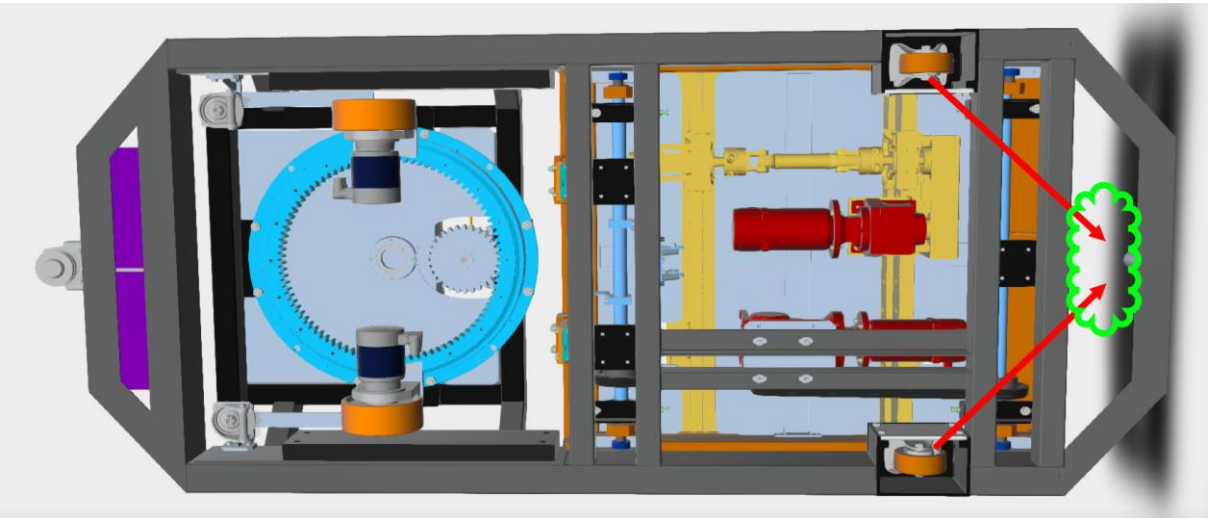
3 A martonvásári Emapot Elektrosztatikus Porfestő Üzemében működő forgózsámolyos járműhöz írt vezetett irányítás

3.1 A feladat és a környezet rövid ismertetése:

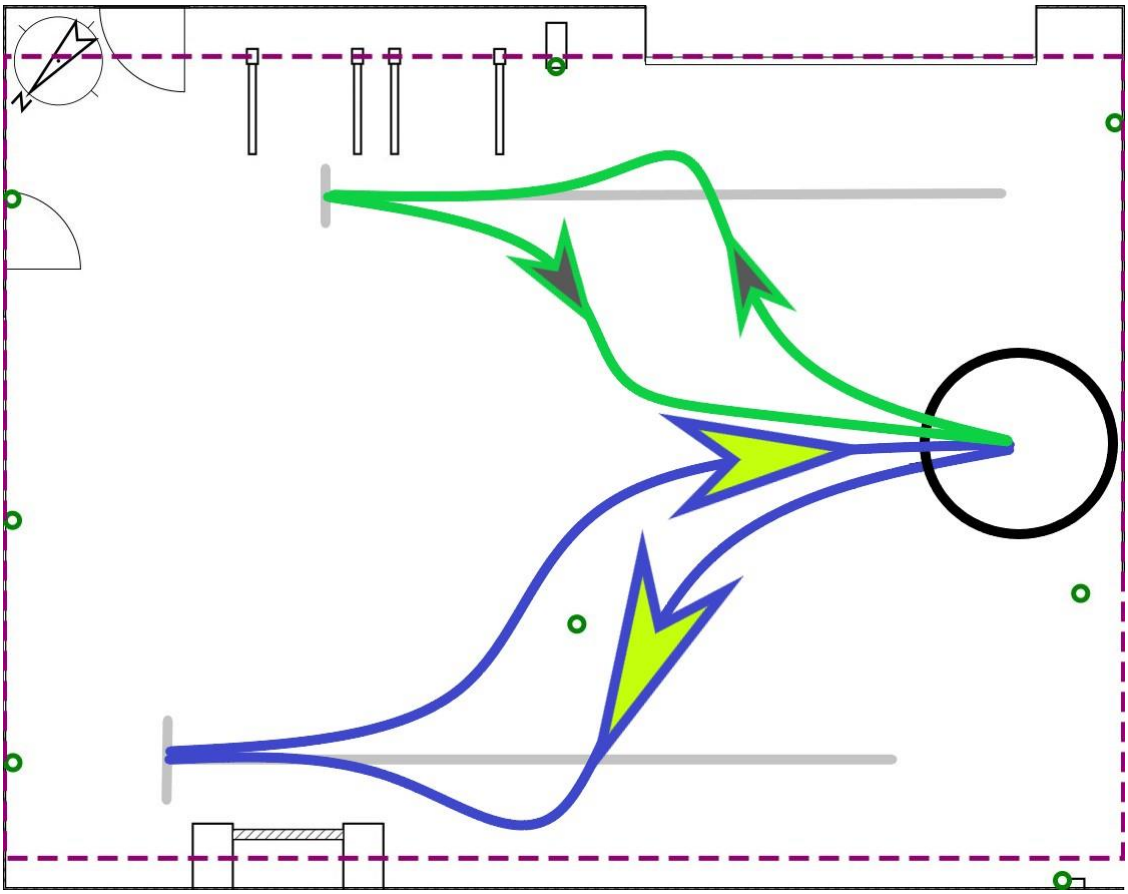
Az üzem egy nagyobb belmagasságú helyiségében (~10méter), kellett egy előre megadott útvonalon egy forgózsámolyos kocsit végig vezetni. A belmagasság miatt a faliegységek helyzete a talajtól és a plafontól is jelentősen nagyobb távolságra volt, mint a Győri egyetemen. A faliegységek mindegyike rálát az UWB-vel lefedett terület közel egészére illetve a helyiségben oszlopok sem voltak. A legjelentősebb környezeti befolyásoló tényező a hirtelen ajtónyitásokra kialakuló légáramlat, illetve az ekkor keletkező hőmérséklet ingadozás. Ezek ismeretében kijelenthető, hogy az üzemben sokkal ideálisabb környezet áll rendelkezésre az UWB-s rendszernek, mint Győrben (ezt a mérések is igazolják). Az útvonal 4 részre bontható (3.2ábra). A kiindulási pontból (fekete kör) először el kell menni egy emelőhöz, ahol a kocsira helyezi az emelő a terméket. Innen tolatva vissza kell jutni a kezdőponthoz. Ezt a két szakasz az ábrán a kék útvonal szemlélteti. Ezután a zöld színnel jelölt útvonalon kell vezetni a járművet, míg eléri az állomást, ahol a terméket leteszi a rászerelt emelővel egy állványra. Innen ismételten visszatolatunk a kiindulópontra.

Az állomásokhoz pár cm-es hibahatáron belül kell beállni és párhuzamosan az emelőre illetve az állványra (a kocsi orientációjának hibája maximum 5 fok lehet). Ezt a pontosságot az UWB-s pozíció önmagában nem garantálja, ezért az ábrán bejelölt módon (szürke vonal) a talajra mágnescsík lett rögzítve. A kocsi orrán egy mágnesszenzor található, mellyel a mágnesszalagot detektálva a kívánt pontosság elérhető. Az orientáció pontosságához a kijelölt állomások felé haladva a mágnesszalagon való túlhaladás szükséges, ugyanis a mágnesszalag metszésekor a kocsi kerekeit a mágnesszalaggal párhuzamosan fordítva a kocsi fara nem húzható be a visszamaradó 2-3 méteres szakaszon, úgy hogy a kijelölt hibahatáron belülre essen a kocsi orientációja (ez a kocsi kialakításából/méretéből következik). Később, látható lesz, hogy egyenes szakaszon haladva a kocsi orientációja hogyan függ a megtett távolságtól.

A projekt folyamán újabb ötletek vetődtek fel a kocsi irányítására, ezek megvalósítása még folyamatban, fejlesztés alatt áll. Így ezen részeket, csak elméletben fogom ismertetni, mért adatokkal alátámasztani nem.



3.1ábra: Az Emapotban található forgózsámolyos kocsi alulnézetből. (A két hátsó kerek tényleges helyzete a zöld felhővel kijelölt helyen van.)



3.2.ábra: Az útvonal ábrázolása (kék: áru felvétele; zöld: áru lerakása, fekete kör: a kiindulási pont)

3.2 Szenzorok:

A kocsin egy elfordulás szenzor (ifm electronic RM8004), egy mágnesszenzor (RoboteQ MGS1600GY) és a jövőben egy, a biztonsági előírások által szükséges lézeres távolságmérő lesz elhelyezve, mely az esetleges ütközés előtti megállást fogja biztosítani. A mágnesszenzor a kocsi orrán helyezkedik el és a horizontális tengely 16 pontján szolgáltat információt a mágneses mező nagyságáról.

3.3 Előmunkák:

3.3.1 A helyiség térképe:

Az mérések vizuális megjelenítéséhez szükségem volt egy térképre. A teremben végzett mérésekhez egy a Bosch által forgalmazott PLR50-es típusú lézeres távolság mérőt használtam. Az adatokat papíron rögzítettem, majd később a Sweet Home 3D nevű alkalmazással az adatokból rekonstruáltam a terem felépítését (3.2ábra). A teremben a szaggatott lila vonal jelöli az UWB-vel lefedett területet, a kis zöld karikák pedig a faliegységek 2D-s helyzetét.

3.3.2 A kocsi sebességéhez tartozó átváltási állandó meghatározása:

Alapból a kocsi kerekeire adott sebességet egy lineárisan növekvő változóval lehetett beállítani. Ez a sebesség érték a kerék átmérőjével arányosan változik. Ahhoz, hogy az adott jármű

kerekéhez tartozó sebességét [m/s]-ban megkapjam, a kocsival 10szer végeztem mérést különböző távolságokra változatlan sebességgel, közben az idő adatokat rögzítve. Ezek után a mért idő és út információból kapott sebességgel elosztva a kocsi sebességéhez tartozó változót, megkaptam a keresett átváltási konstanst.

3.3.3 A forgózsámolyhoz elfordulás érzékelőjéhez tartozó átváltási szám:

A szenzor 4096 lépésre bont fel egy teljes kört. A forgó zsámoly és a (3.1-es ábrán látható) kisebb fogaskerék közötti áttétel a fogak számának hányadosa: 93/30. Az utóbbi fogaskerékre van rászerelve az elfordulás érzékelő, így kiszámolható, hogy a forgózsámoly egy teljes fordulatához 4096*3.1 lépés tartozik. Ebből könnyen visszaszámolható, hogy hány lépés tartozik egy fokhoz.

3.4 A kocsi bemutatása annak legfontosabb paramétereivel:

A kocsit 4 kerék tartja: A két első, amelyek a forgózsámolyon vannak elhelyezve, illetve a két hátsó (3.1ábra). Mind a 4 kerék tömör műanyag kerék. A két hátsó kerék a kocsi síkjában van rögzítve (nem beálló kerekek). A kerekeknél feltételezzük, hogy azok a talajt érintő rész középpontján fordulnak. A forgó zsámoly kerekeinek ezen pontjai közöti távolság 89cm. A két hátsó kerék távolsága elhanyagolható, a jármű modellezésénél egy keréknek lesznek megfeleltetve, mely a kocsi hosszanti szimmetria tengelyére esik (a továbbiakban a hátsó kerék alatt ezt a helyettesítő, elméleti kereket értem). A forgózsámoly középpontjának és a hátsó keréknek a távolsága 171 cm. A kocsi teljes hossza 256cm szélessége pedig 60cm. A kocsi tömegközéppontja 115cm-re helyezkedik el a hátsó kerekektől a jármű hosszanti szimmetriatengelyén. A kocsin a forgózsámoly túlfordulását a forgózsámolyhoz tartozó fék biztosítja, mely a kocsi hosszanti szimmetriatengelyéhez képest +/-70-os szögben blokkolta a forgózsámoly további elfordulását. Erre különös képpen szükség volt, mert a forgózsámoly közepén áthaladó kábelek a forgózsámoly elfordulásával csavarodnak (forgógyűrű hiányában) és a túlzott elfordulás a kábelek szakadásához vezethetnek. A kocsi tápellátását 2darab 95Ah-s 12Voltos sorosan kapcsolt akkumulátor biztosítja. Ha a feszültség szint 22.5Volt alá esik a kocsit töltőre kell dugni.

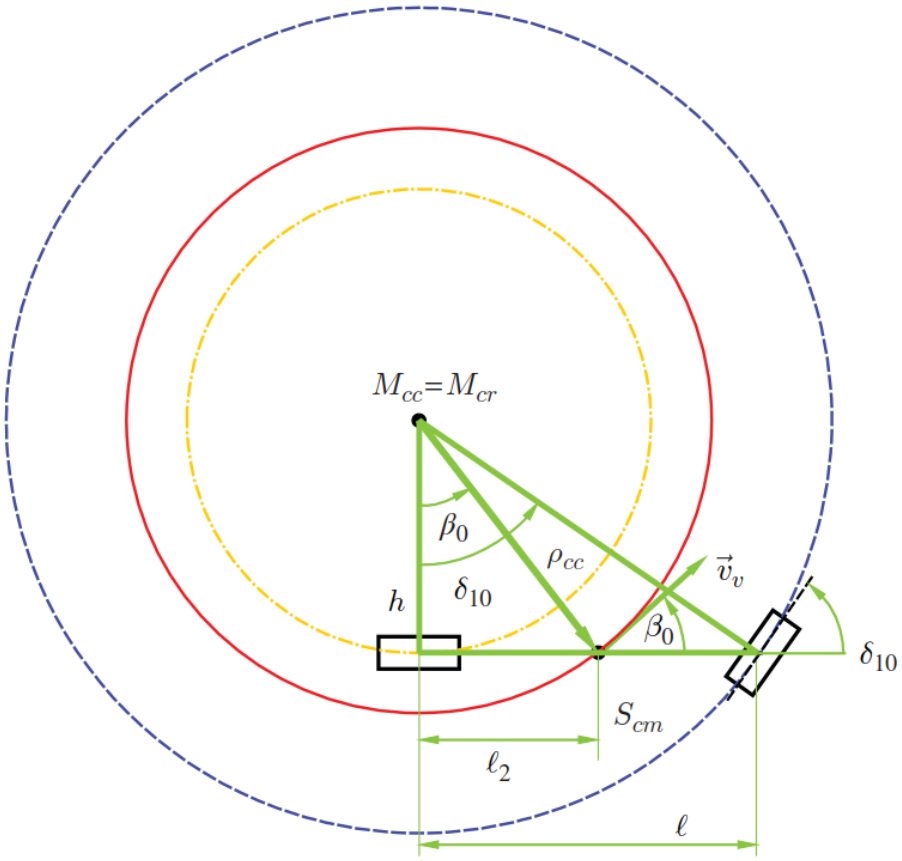
A kocsin egy háromszöget alkotva helyezkedik el az UWB-s multitag rendszer, egy a hátsó kerék felett, a másik kettő pedig a kocsi elülső részében a kocsi hosszanti oldalfalának végződésénél.

3.5 A forgózsámolyos kocsi mozgását leíró bicikli modell bemutatása:

Forgózsámolyos kocsi mozgását ú.n. bicikli modellel lehet leírni (3.3 ábra). Ezt a modellt a közúti forgalomban használt kocsikra is szokták alkalmazni egyszerűsítés képpen, de míg a kocsi első kerekei nem azonos irányba állnak, megállapítható, hogy az adott forgózsámolyos jármű leírására jobban illeszkedik, hiszen a forgózsámoly kikényszeríti a kerekek irányainak megegyezését. A helyettesítést úgy kell elképzelni, hogy a forgózsámoly középpontjába „helyezzük" a bicikli első kerekét és iránya megegyezik a forgózsámoly kerekeinek irányával. A bicikli váza pedig az első és hátsó kerék összekötésével adódik (171cm). Az első kerék helyben forgatása a forgózsámolyra adott ellentétes azonos nagyságú sebességgel történik, illetve a mozgásközben való beállítása is a két kerék sebességének állításával történik. Az első kerék iránya és a váz hátsókerekétől az első kerék felé mutató irányvektor által bezárt szöget kormányszögnek nevezzük (ackermann angle) jele δ_f (az f a front-ot jelöli, jelen esetben, mivel a hátsó kerekek rögzítettek $\delta_{rear} = 0°$). A kormányszöget pozitívnak vettem, ha a kocsi pillanatnyi körpályája az óramutató járásával ellentétes, negatívnak ha megegyezik vele. Ha a kormányszöget adott értéken megtartjuk a jármű fix sugarú pályára áll (ideális esetben). Az ezt leíró összefüggés a következő:

$$R_{front} = \frac{l}{cos(90-\delta_f)} \qquad \textbf{(k.3.1)}$$

ahol $\mathbf{R}_{front}$ az első kerékhez tartozó körpálya sugara, $\boldsymbol{l}$ pedig a bicikli „vázának” a hossza.



3.3 ábra: Geometriai szemléltetése az Ackermann szögnek (kormányszög). Látható, hogy a körpálya feltétele, hogy mind az első és a hátsó kerék iránya is merőleges legyen a kör sugarára. (S_{cm}: tömegközéppont)

A modell érvényesülését a gyakorlatban a lassú mozgás is biztosítja: forgózsámoly középpontjában helyezett virtuális kerék sebességét ($V_{central}$) mindig 0.05m/s-on tartva. Ez alapján a forgózsámoly kerekeinek sebessége a bicikli modell elülső kerekének körpályája függvényében:

$$\mathbf{V}_{slow} = \mathbf{V}_{central} * \frac{\mathbf{R}_f - r_{zs}}{\mathbf{R}_f} \quad \& \quad \mathbf{V}_{fast} = \mathbf{V}_{central} * \frac{\mathbf{R}_f + r_{zs}}{\mathbf{R}_f} \qquad \textbf{(k.3.2)}$$

ahol r_{zs} a forgózsámoly sugara (0.405m).

3.6 Az elméleti modell és a valós rendszer összehasonlítása:

A valóságban a bicikli modell szerint beállított kormányszög nem állította állandó sugarú körpályára a járművet. Ezért méréseket végeztem, hogy az elméleti modell szerint beállított forgózsámoly kerekeinek a sebessége milyen kormányszög mellett eredményez körpályán való haladást. (A forgózsámoly kerekeinek sebessége a megfelelő kormányszög mellett a valóságban is meg kell, hogy határozzák a kör sugarát, ha ez nem teljesül, akkor a jármű nem tudna körpályára állni.)

A mérések a következő képpen zajlottak:

A kocsi pályáját akkor tekinthetjük körszerűnek, ha a mozgás során a kormányszög nem változik. Például az elméleti képlet alapján a -40°-os kormányszöghöz előre fele haladásnál a kerekeket

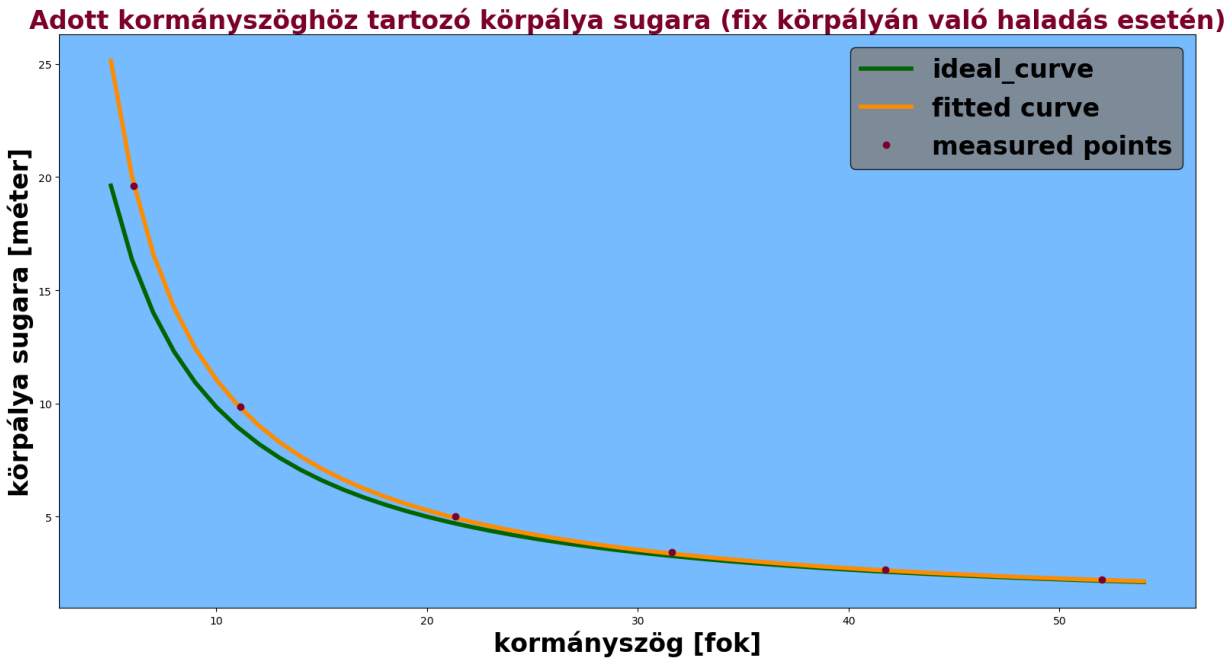
az alábbi sebességekre kell állítani: v_{left} =0.04239m/s illetve v_{right}= 0.0590726. Az bicikli modell első kerekéhez tartozó körpálya sugara ekkor 2.66 m. De a valóságban ez a körpálya nem a -40°-os kormányszöghöz, hanem -41.75°-hoz tartozik. Ennek oka leginkább a hátsó kerék csúszásából fakad, hiszen a jármű hosszanti síkjában rögzítve van, állandó körpályán való haladáskor pedig ennek a keréknek is adott körpályán kell haladni, ehhez viszont a talajon meg kell csúsznia. A mérésnél megfigyelhető volt, hogy míg előre fele haladva a körpályán a kocsi kormányszöge alig változik (konvergál egy érték felé), hátra fele tolatásnál egy kis hiba is jelentős változást vihet a körpálya sugarának változásába, a kocsi „kifarol" (ezt a részt még később részletesen kifejtem). Ezen megállapításból célszerű volt azt a kormányszöget keresni az adott keréksebességekhez, mely tolatás közben elhanyagolhatóan változik. Emellett fontosnak tartottam a tolatás folyamán azt a két szélsőértéket megkeresni, ahol a kocsi fara jelentősen elkezd kitérni jobbra illetve balra (3.5 ábra). Ezen két értéktől közel egyenlő „távolságra" választva a kormányszöget, biztosítja a legjobban a kifarolás elkerülését, de mivel a rendszer nagyon instabil, célszerűnek találtam a tolatásra szabályozót tervezni a későbbiekben leírt módon.

Az 3.4-es ábrán látható, hogy az elméleti 40°-os kormányszöghöz a valóságban körülbelül 41.75° felel meg.

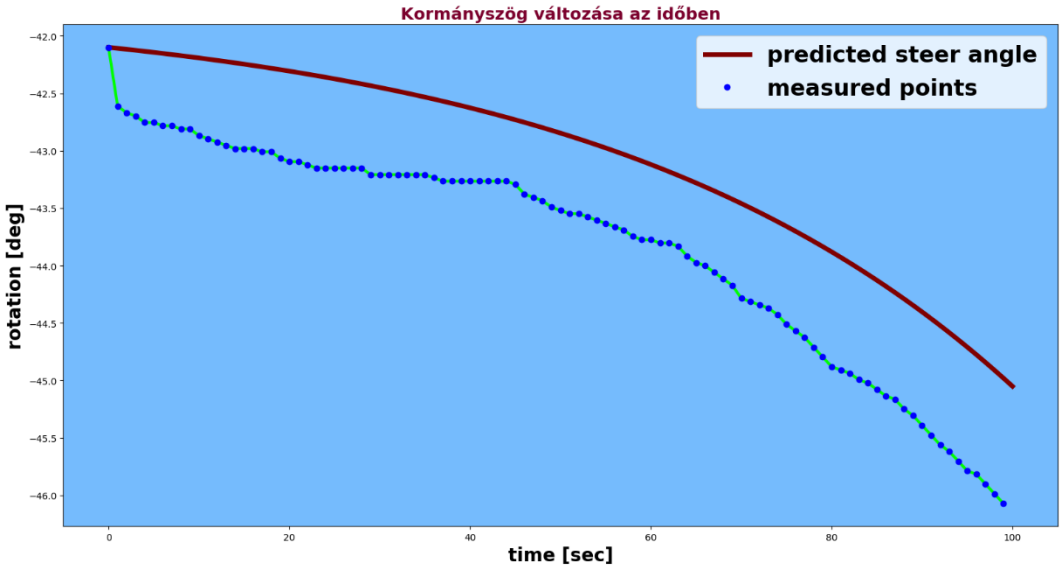
A mért értékekre illesztett görbét a biciklimodellre felírt görbe módosításával kaptam:

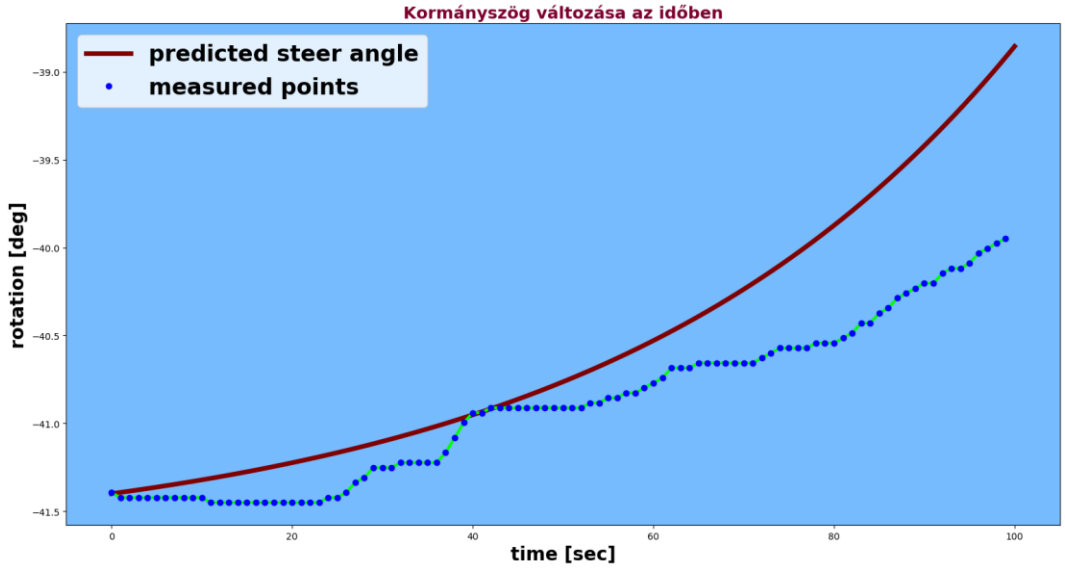
$$\mathbf{R}_{front\_real} = \frac{l}{cos(90-(\delta_f - 1.1))} \qquad \textbf{(k.3.3)}$$

A képletből látható, hogy kis kormány szögekre (<5°) nem alkalmazható, mert δ_f = 1.1 esetén a kör sugara a végtelenhez tart, ami azt jelenti, hogy a kocsi egyenes mentén közlekedik. Az egyenes mentén való haladáshoz viszont valós és ideális körülmények között is 0°-os kormányszög kell, hogy tartozzon!

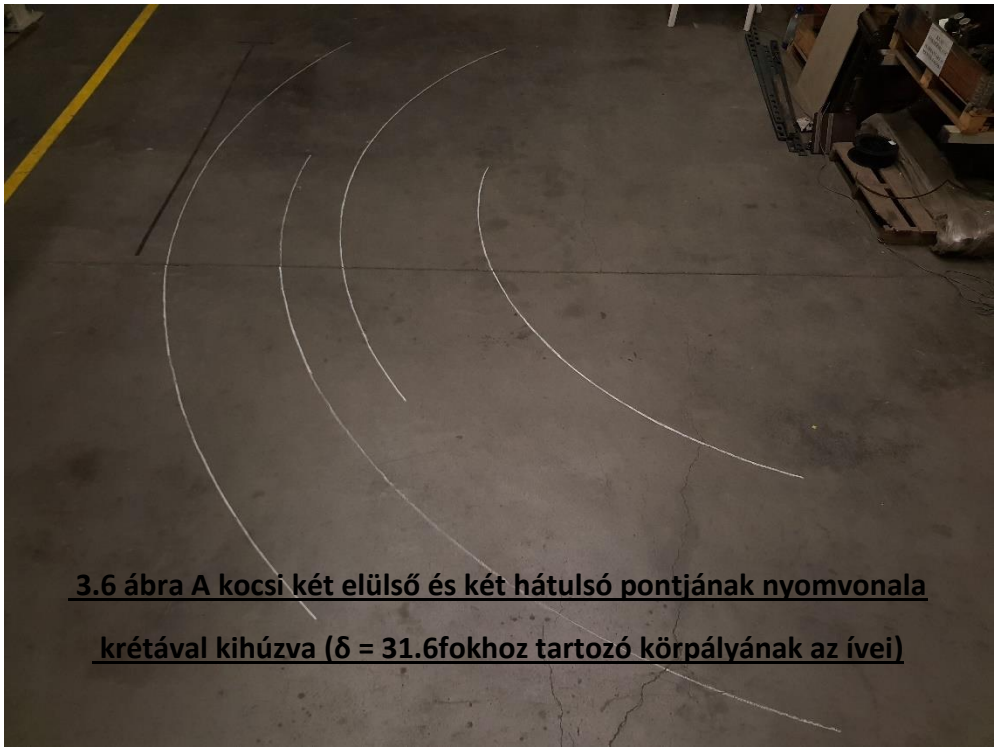


3.4 ábra: A kormányszög függvényében a bicikli modell első kerekéhez tartozó sugár változása (zöld:a bicikli modell szerint, narancs: a mért adatok alapján, piros: a mért értékek)





3.5 ábra: A mért értékek (kék) illetve a később ismertetett prediktor (piros), mely előre kiszámolja a kocsi kifarolásának következtében a kormányszög változását



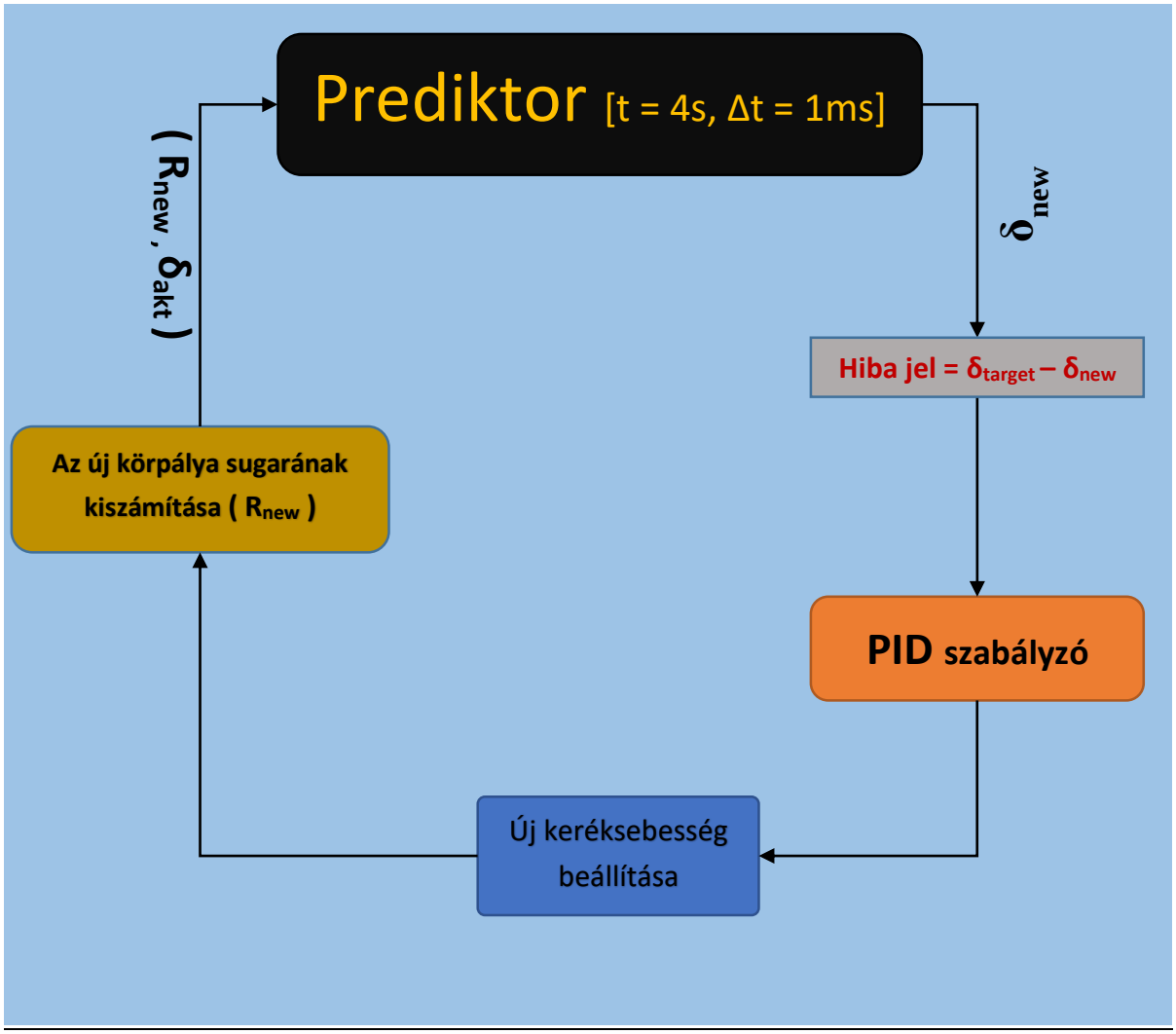
3.6 ábra A kocsi két elülső és két hátulsó pontjának nyomvonala krétával kihúzva (δ = 31.6fokhoz tartozó körpályának az ívei)

3.7 A szabályzó hangolásához vezető út

A forgózsámolyos kocsi tolatásakor jelentkező kifarolást minden képpen meg kellett akadályozni a megfelelő működés érdekében. Az egyik legkönnyebb megoldás a következő lett volna: figyeljük a kormányszögváltozását és ha az a kijelölt értéktől eltér egy előre meghatározott határértékkel, akkor megállunk, és ismét befordítjuk a forgózsámolyt a megadott kormányszögnek megfelelően. Ehhez a megoldáshoz muszáj megállni, így én e helyett inkább egy szabályzóval ellátott vezérlési kör tervezésében láttam a megoldást. A vezérlési kör felépítését a 3.7-es ábra szemlélteti. A következőkben a vezérlési kör elemein fogok végig menni, és azokat részletezni. Előbb a legkönnyeben értelmezhető dobozokkal:

A hibajel az elérni kívánt körpályához tartozó kormányszög és prediktor által megjósolt aktuális kormányszög különbségeként adódik. Ez lesz a PID szabályzó bemeneti értéke.

A PID szabályzó kimeneti jelének függvényében a forgózsámoly kerekeinek sebességét állítjuk, majd kiszámítjuk az új keréksebességekhez tartozó ideális körpályát (R_{new}).



3.7 ábra: A mért értékek (kék) illetve a később ismertetett

Ezek után csak a prediktor és a Pid szabályzó részletes bemutatása maradt, de a szabályzó hangolásához szükséges a prediktor így ennek a bemutatásával kezdem.

3.8 A prediktor

Ennek az egységnek a feladata „megjósolni”, hogy a forgózsámoly kerekeire adott sebesség által meghatározott ideális kormányszögtől való eltérés esetén hogyan változik a kormányszög értéke idő függvényében.

Ennek az egységnek a megvalósításához a kör Δs -> 0 szakaszokkal való közelítését, illetve az ezen szakaszokon való mozgást egy egyenes vonalú mozgással, majd a szakasz végén a szakaszhoz tartozó ívvel való elfordulással közelítettem. Ez a megoldás azért előnyös, mert így elég mérési eredményekből az egyenes szakaszon vizsgálni a kormányszög változását, és nem kell különböző görbékre megvizsgálni a kormányszög időbeli függését, illetve azokból egy általános összefüggést találni a kormányszög változására, mely hihetetlenül körülményes és nehéz feladat lenne.

3.8.1 Azonos keréksebességek mellett a kormányszög változásának vizsgálata:

Az első lépés annak belátása volt, hogy ha a kerekre *t* ideig *v* sebességet kapcsolok, majd ezek után azonos ideig *-v* sebességet, akkor a véghelyzetnek meg kell egyeznie a kiindulásival (mivel a vizsgált rendszer valós, így minimális eltérés előfordulhat). A következő fontos lépés az volt, hogy a jármű mozgásának elméleti vizsgálatakor elvonatkoztatunk a kocsi korlátozottságától: Ahhoz hogy jobban megértsem a vizsgálat tárgyát elképzeltem azt az esetet, amikor 70°-nál nem akad meg a forgózsámoly, hanem tovább forog. Könnyen belátható, hogy kellő idő után, mikor beáll a rendszer egyensúlyi állapota a kocsi fara lesz elől, míg az orra hátul. A folyamat megfordítható: tehát ha ekkor ellenkező sebességet adunk a forgózsámoly kerekeire az kibillen ebből az instabil egyensúlyi állapotából, és az előző folyamat idejének elteltével újra a kocsi orra lesz elől. (Elméletben a kocsi hossztengelye és a kerekek irányai által bezárt szög soha sem éri el a 0°-ot, csak az egyre inkább megközelíti azt. Valós rendszernél ezt figyelmen kívül hagyhatjuk, mert a talaj egyenetlenségei kellő idő elteltével sokkal nagyobb mértékben változtatják ezt a differenciát). A kormányszög leggyorsabban 90°-nál változik:

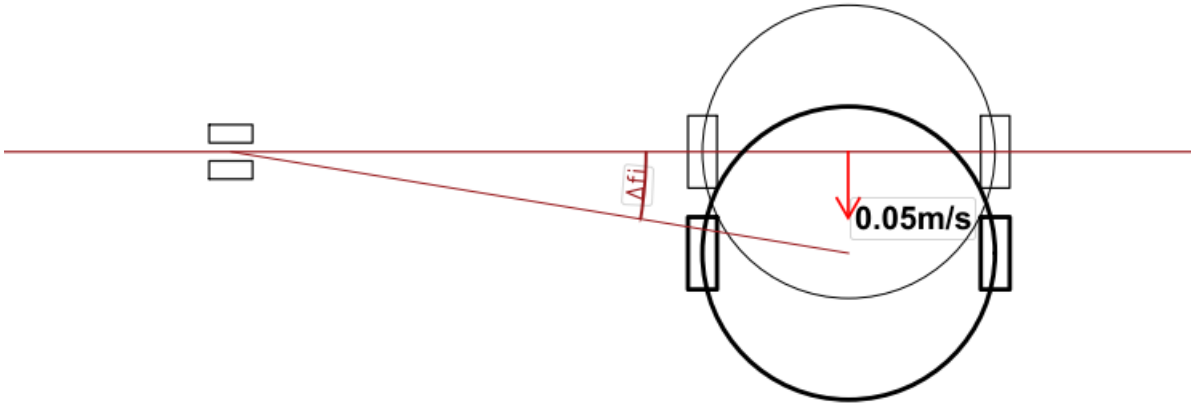
$$\frac{d\delta}{dt} = \lim_{\Delta t \to 0} \frac{\delta(t+\Delta t) - \delta(t)}{\Delta t} = \lim_{\Delta t \to 0} \frac{(90° + \tan^{-1}\left(\frac{\Delta t * v_c}{l}\right) - 90°)}{\Delta t} \quad \textbf{(k3.4)}$$

ahol v_c= *0.05m/s* és l *= 1.71 m:*

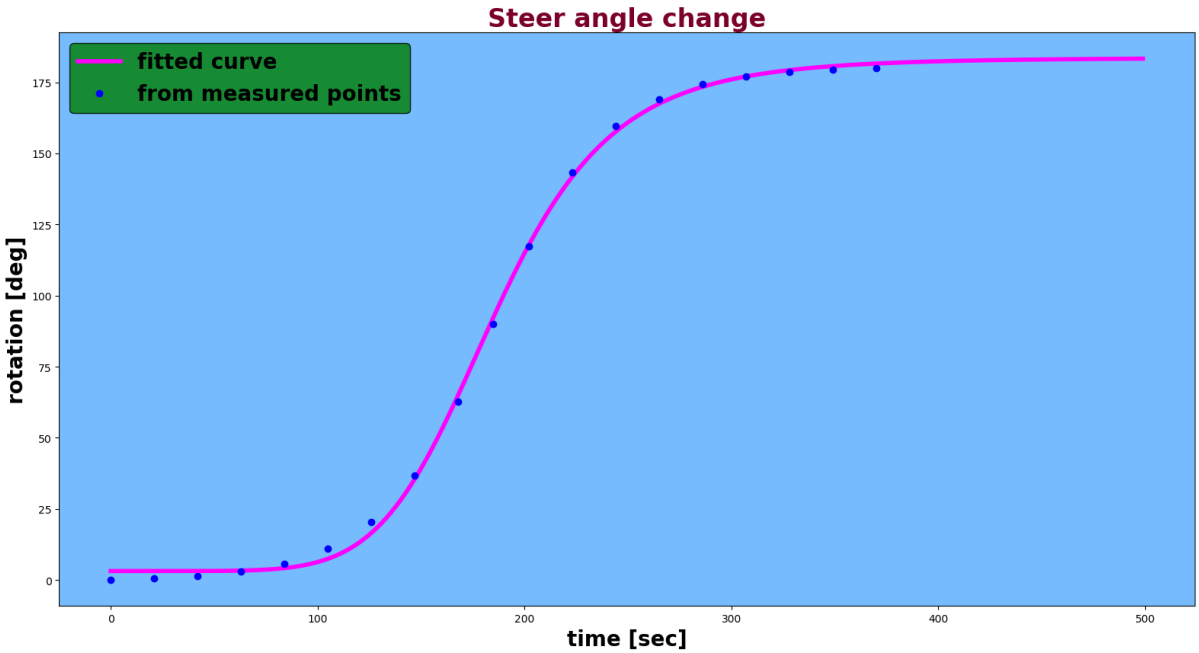
$$\lim_{\Delta t \to 0} \frac{\tan^{-1}\left(\frac{\Delta t * 0.05}{1.71}\right)}{\Delta t} = \frac{1.675°}{s} \quad \textbf{(k3.5)}$$

A mért adatok azt mutatják, hogy már 70°-nál is elég közel van a kormányszög változásának meredeksége ehhez az értékhez, így 70° és 90° között a kormányszög időbeli változásának meredekségét egy egyenessel közelítjük. Ez eddig elmondottakból következik, hogy a folyamat tükrözhető a 90° és 180° között a 90° és 0° közötti kormányszög változásának sebességére (3.9-as ábra). A pontokra illeszkedő görbe neve szimmetrikus szigmoidal (S alakú görbék). A 3.10-es ábrán a mért értékeket ábrázoltam (0 -> 70 fokig). A görbe ezen szakaszára egy 4-ed fokú polinom is nagyon illeszkedett. Mivel számomra a görbe meredeksége volt a fontos az egyes időpillanatokban, amihez a görbe egyenletét le kellett deriválni, inkább ezzel a 4-edrendű polinommal dolgoztam tovább.

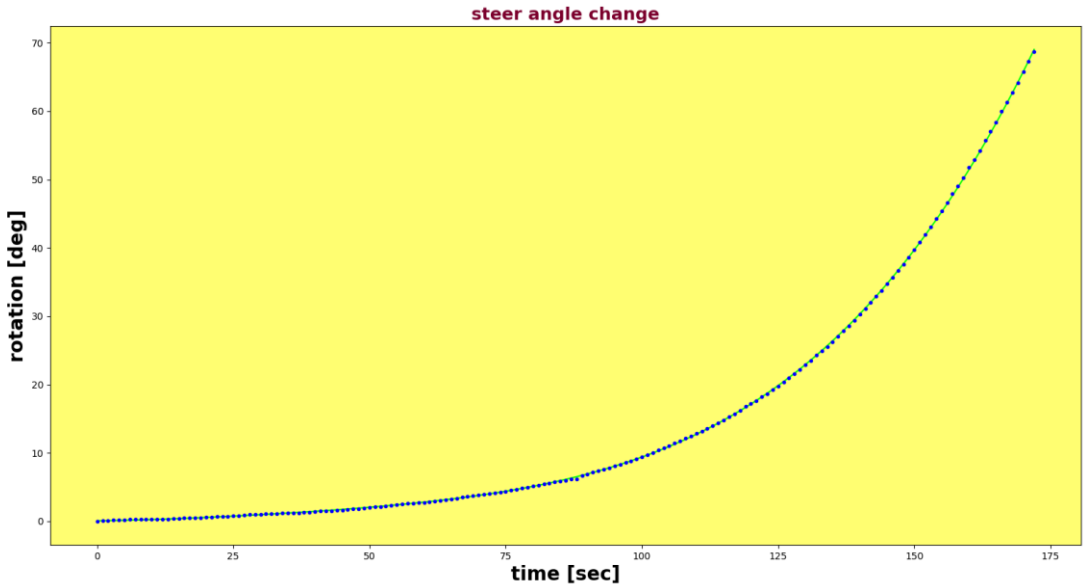
Ennek a polinomnak a deriváltja megadja, hogy adott időpillanatban a milyen gyorsan változik a kormányszög ([°/s]) 3.11. ábra



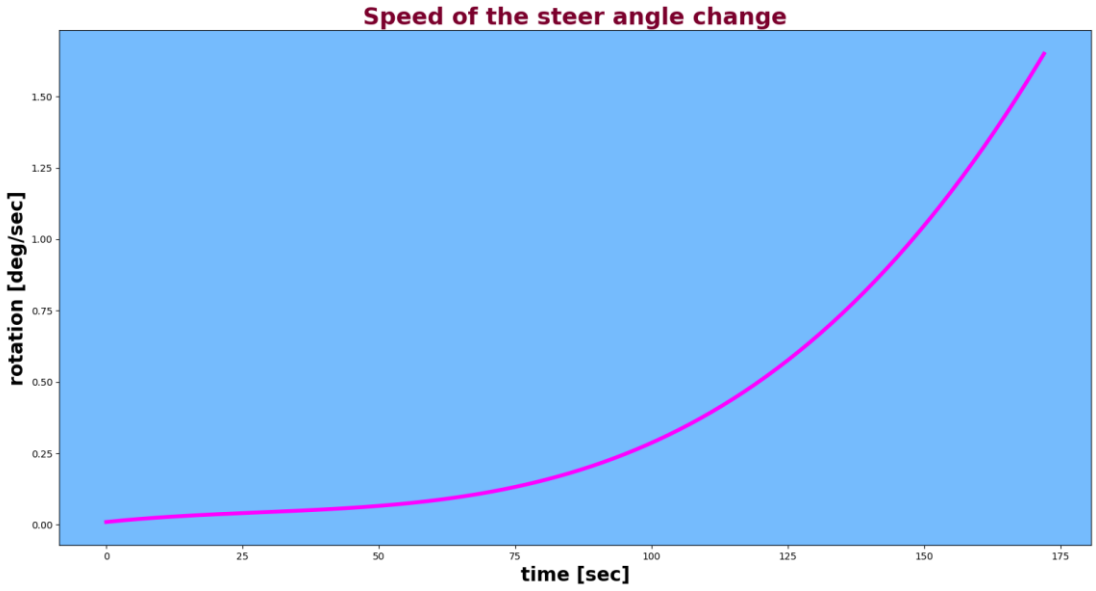
3.8 ábra: A kormányszögváltozás sebességének maximális értékéhez tartozó elrendezés



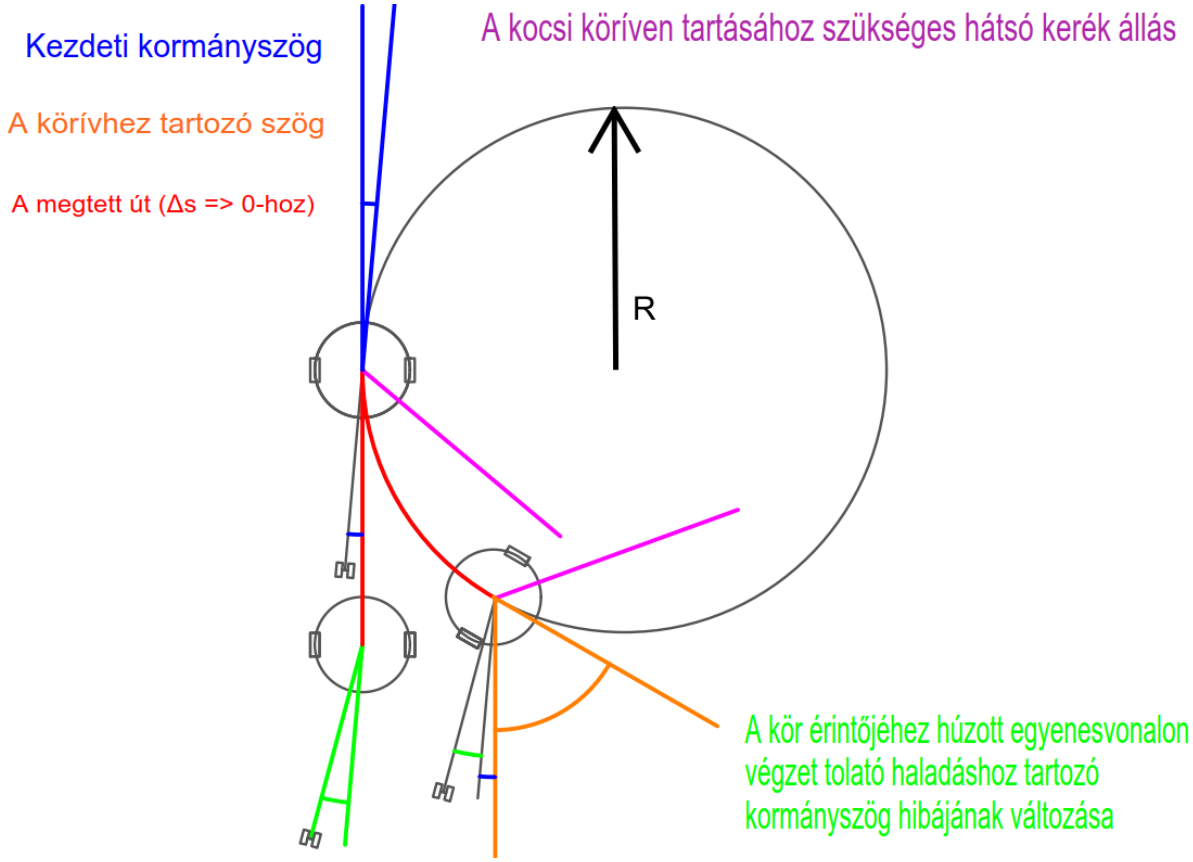
3.9 ábra: A kormányszög változása az eltelt idő függvényében



3.10 ábra: A kormányszög változása 0 és 70 fok között (kék – mérési pontok)



3.11 ábra A körpályán való mozgás két részre való bontásának szemléltetése



3.12 ábra A körpályán való mozgás két részre való bontásának szemléltetése

3.8.2 A prediktor működése:

Most már elég ismeret van a birtokunkban, hogy adott időintervallumra megjósoljuk, mennyit változik a kormányszög, ha ismerjük a kiindulási állapotot (R_{new} ill. δ_{akt}). De még ezelőtt el kell dönteni, hogy időben milyen gyakran akarjuk szabályozni a vezérlési kört, én ezt 4 másodpercnek választottam meg, illetve milyen kis szakaszokkal közelítsük a kört. A programmal különböző kis Δt időintervallumokat kipróbálva, úgy döntöttem az 1ms megfelelő érték ehhez a feladathoz. Lényeges javulást a „jósolt" értékében már nem láttam, és ennél 10-100-szor kisebb értékek választásánál a futási idő is 10 illetve 100 szorosára változott (így nem tudnánk kiszámolni az új kormányszöget 4s-on belül).
A közelítés érzékeltetése: 1ms idő alatt 0.05m/s sebességel $5*10^{-5}$méteres szakaszokkal közelítünk egy minimum 12méter kerületű kört.

Az 1ms-os időre jósolt új kormányszög számításának lépései:

I. Megnézzük hogy mekkora a kormányszög, és megkeressük az ehhez tartozó időpillanatot a 3.10-es ábra pontjaira illesztett görbét leíró polinom megoldásával az adott kormányszögre.
II. A 3.10-es ábrán látható görbe egyenletét kiintegráljuk az előző pontban kiszámolt időpillanattól: t_{start} -> t_{start}+Δt-ig, majd korrigálunk a Δt idő alatti ív elfordulással. Képlettel:

$$\boldsymbol{\delta_{new} = \int_{t_{start}}^{t_{start}+\Delta t} g(t)\,dt - \frac{\Delta t * v_c}{2\pi R_{akt}} * 360^\circ}$$

ahol: $\boldsymbol{g(t) = 0.009454423 + 0.00199091t - 0.000042261t^2 + 0.5008144 * 10^{-6}t^3}$

Könnyen belátható, hogy ha a körpályán előre fele való mozgás során kéne kiszámolni a kormányszög új értékét, azt a következő képpen tehetjük meg:

$$\boldsymbol{\delta_{new} = \int_{t_{start}}^{t_{start}-\Delta t} g(t)\,dt + \frac{\Delta t * v_c}{2\pi R_{akt}} * 360^\circ}$$

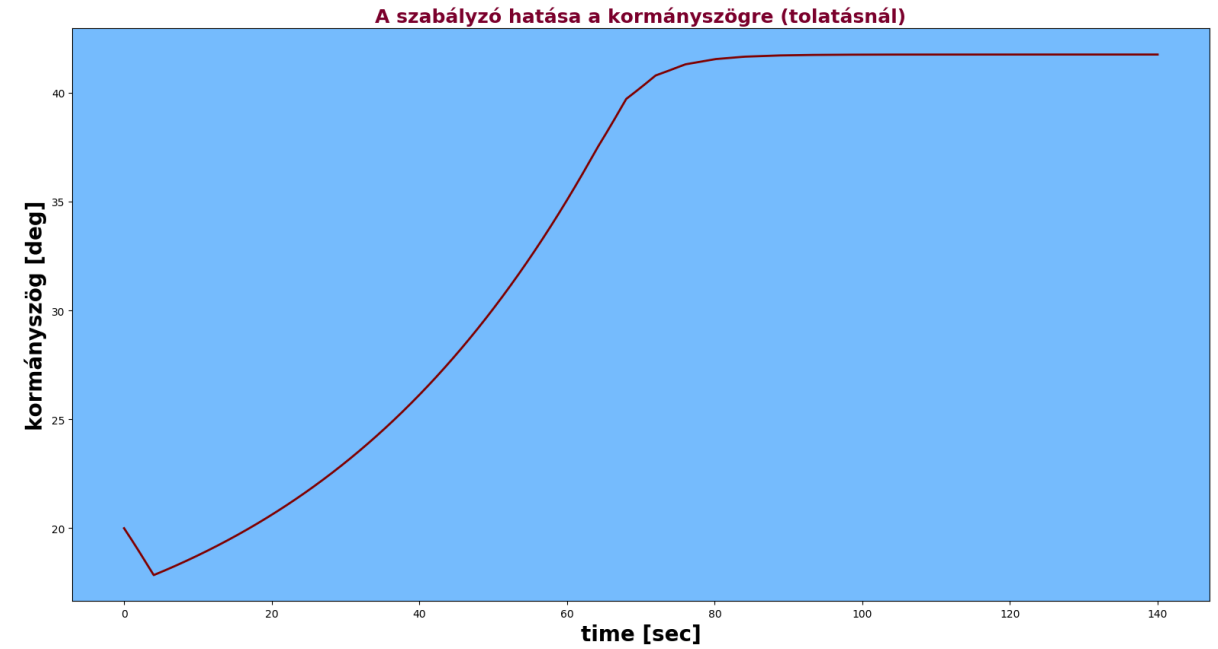
A lépéseket Pythonban valósítottam meg a numpy (numerikus python) könyvtár segítségével. Az első lépésben a 4-ed fokú polinom megoldását a roots(), az integrálást pedig a quad() függvény használatával.

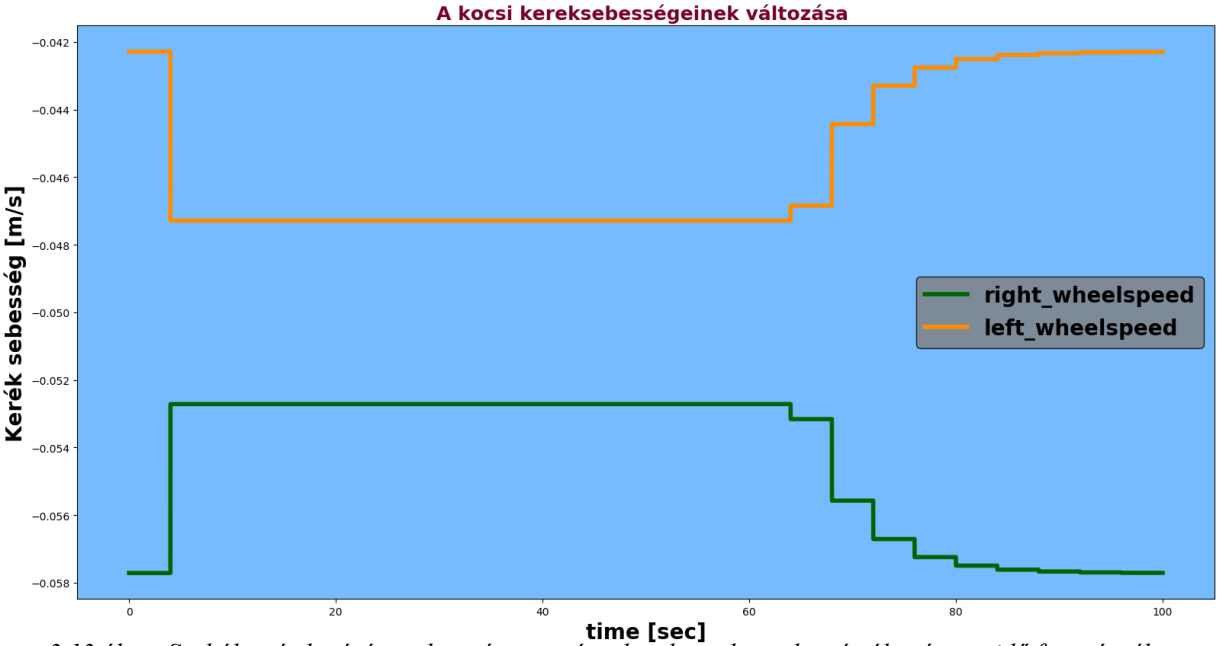
Ezután ezt 4000-szer megismételjük, így megkapjuk a 4s-os időtartam elteltével jósolt értékeket.
Működését a 3.5-ös ábra szemlélteti.

3.9 A PID szabályzó hangolása

A szabályzót a már győri részen bemutatott Janssen-Offereins módszerrel hangoltam. A szabályzónál korlátoztam a forgózsámoly kerekeire adható maximális sebességeltérést, hogy a hirtelen rángásszerű mozgások ne lépjenek fel, hanem finoman álljon be a szabályzó. A 3.13-as ábrán látható felül, amint 20°-ról 41.75°-ra beáll a kormányszög. Az első 4 másodpercben a szabályzó még nincs a körben, látható hogy a kormányszög rohamosan tér el a 41.75fok-tól, ami a körpályán való maradáshoz szükséges. Az

alatta lévő ábrán a kerekekre adott sebesség látható az idő függvényében (a szabályzóból 4s-onként változik, a szabályzó kimeneti értékének függvényében).





3.13 ábra: Szabályozás hatására a kormányszög és a kerekre adott sebessé változása az idő függvényében

A működéséhez célszerű definiálni egy +/- fél fokos tartományt, ahol nem vesszük figyelembe a szabályzó kimeneti jelét. Ezt az indokolja, hogy amikor a méréseket végeztem, az állapítottam meg, hogy egy 1 fokos tartományon belül hátra fele tolatva 5 métert sem változik a kormányszög többet, mint egy fok. Ezen kívül a talajon lévő egyenletlenségek is enyhén kitérítik a kormányszöget. Például a kerék felmegy egy bukkanóra lemegy arról, így a kormányszög eredő változása közel nulla, de a bukkanón egy kicsit kitér. Ilyen mozzanatokra nem szabad, hogy szabályozzunk.

Most, hogy a jármű mozgását modelleztük és az esetleg jelentkező kormányszögeltérésből a kijelölt pályáról való letérést kiküszöböltük, visszatérhetünk az útvonalon való haladás megvalósításához.

3.10 A kocsi irányítása Pythonból

Eleinte a saját laptopommal kapcsolódva az Emapotos hálózatra értem el a kocsin elhelyezett Raspberry Pi3-at, de a hálózati problémák miatt később az irányításért felelős kódot közvetlenül a Raspberri-n futtattam. A raspberri-re való csatlakozáshoz a PuTTY nevű programot, a fájlok módosításra, illetve azok átvitelét a saját laptopomra pedig a WinSCP nevű alkalmazással oldottam meg.

Míg a győri laborban egy httpszerveren keresztül kértem le és küldtem ki az adatokat. A parancsok kiküldése Martonvásáron UDP-s (User Datagram Protocol) csomagokkal valósul meg, melyek a parancs timeout-ja alatt többször kiküldésre kerülnek, ugyanis nem biztosított hogy a parancs célba ér. Viszont a kommunikáció sebessége sokkal jobb mint a httpszerver esetében. Az adatok lekérésének módja is változott. Az adatok kiolvasása 2 különböző websocketen érkező csomagok feldolgozásával valósult meg. Az egyik websocketen az UWB-s adatok jönnek, a másikon pedig a mágnes és a rotációs szenzor adatai. Mind kettőt a program elindulásától a befejezésig, vagy ameddig szükségünk van az adott információkra futtatnunk kell. Ezért mindkettőt számára egy külön threadet futtatok. Az adatok egyidejű írása/olvasása elkerülése szempontjából itt is mutex-el „védem” az adott változókat.

3.11 Az útvonal részleteinek ismertetése

3.11.1 Az elindulás

Az elindulás elött feltételezzük, hogy a kocsi a 3.2-es ábrán látható fekete körben helyezkedik el a mágnesszalagokkal közel párhuzamosan orral a kijelölt állomások felé. (Ezt egy Xboxokhoz gyártott kontroller segítségével tehetjük meg). A helyzetet az UWB-s adatokkal is könnyen ellenőrizhetjük, az orientációt viszont az értékek akár +/- 50 fokos szórása miatt nehéz lenne.

3.11.2 A mágnescsíkra való ráfordulás állandó sugarú körön

A pályát egy 3.42 méter sugarú körpályán való haladással kezdjük (δ=31.6°). Az UWB-s pozíció adatokat 10Hz-el mintavételezzük és eltároljuk. Ha nem jön pozíció, akkor Dead Reckoning alapján számítjuk a pozíciót a mozgás ismeretében.

A mágnescsík detektálását a mágnesesszenzorból érkező adatok vizsgálatával állapíthatjuk meg. A mágnescsík középső részén való áthaladás pillanatának megállapításához a szenzor két középső pontjában (8. és 9.) vizsgált értékek abszolút értékén maximumot keresünk egy határérték fölött, nehogy összetévesszük a kapott maximumot egy lokális maximum hellyel.

A mágnescsík metszésekor a kocsi vázának orientációját meg kell határozni, és ennek függvényében kell a mágnescsíkon túlhaladni, hogy egy ellentétes körívre való ráállással, amikor a kocsi orra ismételten a mágnesszalagra kerül, közel a fara is azon legyen.

A kocsi orientációjának meghatározása több féle képpen is történhet. Ezek közül 3-4 módszerrel egész jó becslést kaphatunk rá, ezeket megfelelő súllyal véve pedig tovább pontosítható a kocsi orientációja. A módszerek a következők:

- Az UWB-s pontokra görbét illesztek. A metszés időpontjához 1 másodperces sávon belüli pozíciós adatokat átlagolom, majd a kör origójából egyenest húzok az adott pontba és a kör azon része és az egyenes metszésének pontjába érintőt állítok. Az így kapott érintő meredekségéből a kocsi orientációja meghatározható.

- Az UWB-s pontokra görbét illesztek a kezdeti pozíciókból, majd meghatározom, hogy a körön honnan indultam és az eltelt idő függvényében kiszámolom a kör mely pontján metszettem a mágnesszalagot. Ezután erre a pontra érintőt illesztek.

- A mágnescsíkon való áthaladás idejéből meghatározom az orientációt

- A körpályán haladva RANSAC-el 1 másodpercenként (a kapott 10 helyzetre) egyenest illesztek, az UWB-s orientációkat is lementem közben és a megfelelő súllyal átlagolom őket

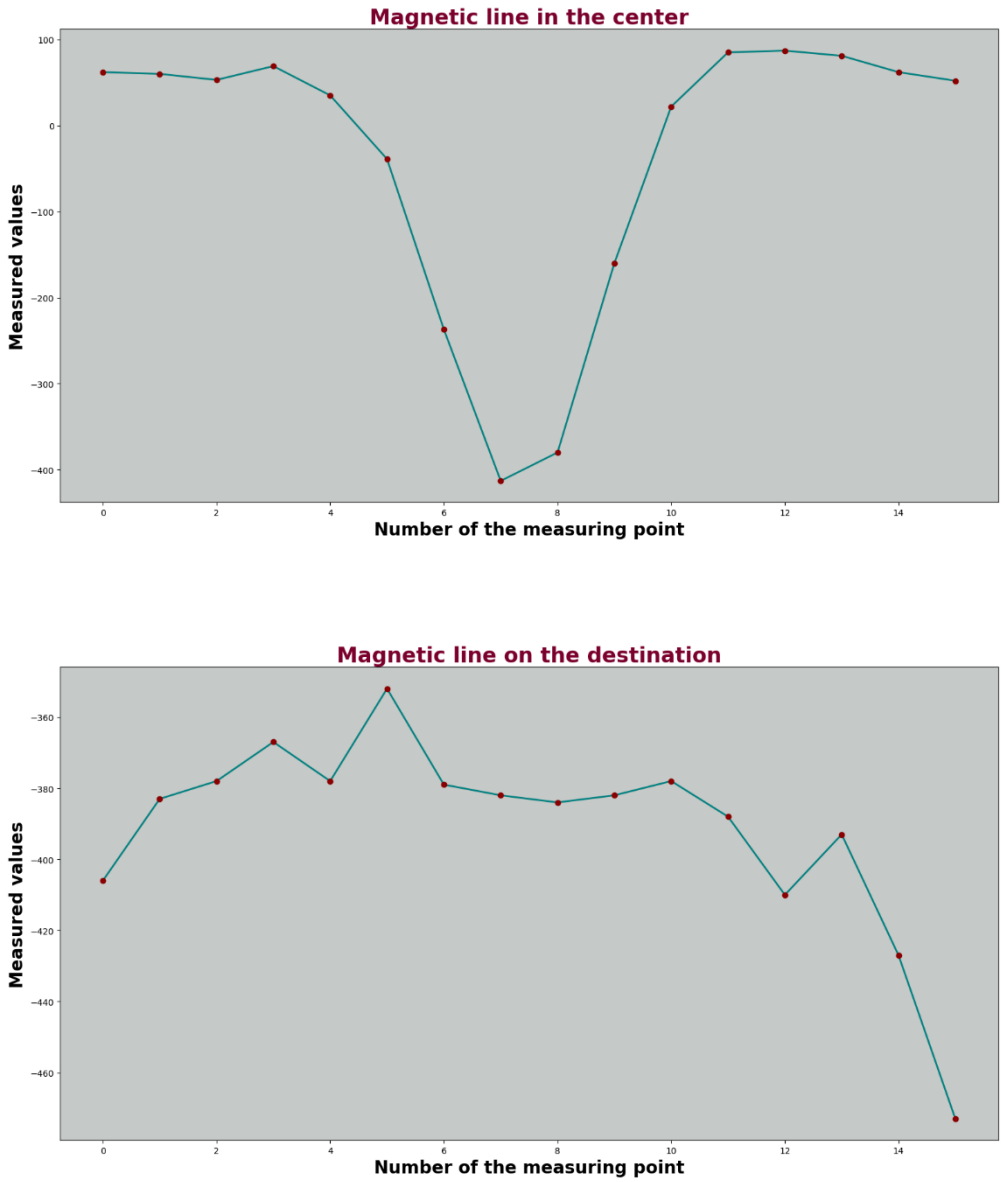
<u>A görbe illesztés a következő lépéssorozattal történik:</u>

I. A kiindulástól a mágnes csík metszéséig kapott pozíció adatokat 3 egyenlő részre osztom, a beérkezési idejük szerint. A középső részt (2.) nem használom fel a továbbiakban.
II. Kiválasztok egy pontot az első csoportból és egyet a 3-ból. Az így kapott 2 ponttól azonos távolságra lévő pontok egy egyenest határoznak meg.
III. A 2. lépést megismételve még egy egyenest kapunk. A két egyenes metszéspontja az illeszteni kívánt kör origóját adja meg, a metszés és bármely pont távolsága pedig a kör sugarát adja meg.
IV. Az 3 rész összes UWB-s pontjára megnézzük, hogy a kör középpontjától milyen messze vannak. Ha egy pont +/- 15 cm-en belül van, úgy vesszük, hogy illeszkedik a körre, ha ezen kívül esik, akkor nem. A hibahatáron belül eső pontok eltérésének abszolútértékeit összeadjuk, így egy összegzett hibát kapva.
V. Megnézzük, hogy a pontok 80%-ára teljesül-e, hogy a hibahatáron belülre esik, ha nem, akkor visszalépünk az egyes pontba, ha igen akkor megnézzük, hogy az eddigi legjobban illeszkedő körnek az összegzett hibája nagyobb-e, mint ami az épp vizsgált körhöz tartozik. Ha igen akkor lecseréljük az épp vizsgált kör adataival az eddigi legjobbat. Ezután visszalépünk az első pontba.

A módszert addig folytatjuk, míg nem találunk legalább 100 körközéppontot, amire teljesül a +/-15cm-es hibahatár illetve a 80%-os feltétel.

A mágnescsíkra való ráállást úgy valósítjuk meg, hogy a mágnescsík metszése után a kocsin a forgózsámolyt hirtelen (<1sec alatt) nullafokos kormányszögre állítjuk, majd amikor a kocsi tömegközéppontja a mágnescsíkfölé kerül a kocsi kormányszögét úgy állítjuk be, hogy az megegyezzen a mágnesszalag metszésének szögével (ezt is 1 másodpercen belül tesszük meg). Ha ez nem lehetséges, mert 70 fok feletti szögben metszettük a mágnesszalagot (bár ennek igen kicsi az esélye), akkor 65 fokos kormányszöggel, de még mielőtt a kocsi tömegközéppontja elérné a mágnesszalagot elkezdünk fordulni. (65 fokos kormányszögnél nagyobb érték beállítása nem javasolt, mivel ha elérjük a 70fokot és a motorvezérlők hatására áll meg a kocsi, akkor annak újraindítása közvetlen beavatkozást igényel). Ahogy a kocsi orrával elhelyezett szenzorral ismét detektáljuk a mágnescsíkot a kocsi orrát középre

pozícionáljuk. Majd egyenesen követjük a mágnesszalagot mindaddig, míg a keresztben elhelyezett mágnescsík darabot el nem érjük (3.14). Ennek detektálásakor a kocsit megállítjuk kiadunk egy jelet, hogy a kocsi megérkezett az állomásra és várunk egy válaszra, mely jelzi a minta darab feltételének befejezését, így haladhatunk tovább.



3.14. ábra: A mágnesszalagon haladó kocsi (felül), a megállóban a mágnesszalagra merőleges mágnesdarab következtében mérhető értékek (alul)

3.11.3 Visszatérés a kiinduló pozícióra

A kiinduló állomásra való visszajutáshoz, a kormányszöget -31.6 fokra állítva haladunk 1.5 métert, majd (hirtelen)+31.6ra váltva ugyanennyit haladunk hátra végül egyenesen 0fokra állítjuk a kormányszöget.

A hátra felé való haladást elméletileg ezen kívül, még úgy is meglehetne valósítani, hogy rögzítjük a kiinduló ponttól a megállóig a kerekekre adott vezérlést, és ezeket fordított sorrendben megismételve kiadjuk.

A termék célállomásig való eljuttatása az előző pálya mintájára történik.

3.12 Utószó:

A jármű modellezése jelentősen megnehezítette a feladatot a győri egyetemen való tankhajtásos kiskocsihoz képest. Bár az UWB-s pozícionálás kedvezőbbnek bizonyul, az állomásokra való beállás cm-es pontossággal nagyfokú precizitást igényelt, csak UWB-s adattal a feladat megoldása nem sikerült volna.

A mágnes szalagra való pontosabb (az orientációt illetően) beállás fejlesztése is folyamat alatt van, amelynek alapját a járművek pályamodellezésénél használt Clothoid görbék adják.

4 FORRÁSOK:

[1] Rainer Mautz: *Indoor positioning Technologies*

[2] Lantos Béla: *Irányítási rendszerek elmélete és tervezése*

[3] Martin Meywerk: *Vehicle Dynamics*

[4] Reza N. Jazar: *Vehicle Dynamics Theory and Application*

[5] Martin A. Fischler & Robert C. Bolles : *Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography*

[6] https://www.ifm.com/gb/en/product/RM8004

[7] https://omt.hu

[8] https://resources.altium.com/pcb-design-blog/the-advantages-and-disadvantages-of-active-and-passive-rfid-technologies

[9] https://www.superdroidrobots.com/shop/item.aspx/roboteq-mgs1600gy-magnetic-guide-sensor/2295/