

TDK-dolgozat

Palkovics Dénes

Vágner Máté

Debreceni Egyetem
Informatikai Kar
Adattudomány és Vizualizáció Tanszék

**Beltéren is használható lokalizációs helyzetmeghatározó rendszer
alkalmazása az önvezető járművek területén**

Dr. Kovács László
Adjunktus

Palkovics Dénes, MSc. II.
Vágner Máté, BSc. IV.

Bírálatra való beküldésre hozzájárulok

Debrecen
2022.06.

Tartalomjegyzék

1. Bevezetés	5
2. Irodalmi áttekintés	8
2.1. Gépi tanulás.....	8
2.2. Felügyelt tanulás	8
2.3. Neurális hálózatok.....	10
2.3.1. Neurális hálózatok felépítése.....	10
2.3.2. Neurális hálózatok működése.....	11
2.3.3. Konvolúciós neurális hálózatok	15
2.3.4. Mély tanulás	16
2.4. A Donkey® platform	17
2.5. Marvelmind szenzorrendszer	19
2.5.1. Használat célja.....	19
2.5.2. A rendszer működési elve.....	19
2.5.3. True-range multilateráció	20
2.5.4. Bázistranszformáció	23
2.5.5. Koordinátarendszer kibővítése	23
2.5.6. A rendszer felépítése	24
2.5.7. Architektúrák.....	25
2.5.8. A rendszer felállítása	26
2.6. Hibaterjedés.....	28
2.7. Pontosságígulás	29
2.8. Páros sündisznó üzemmód	31
2.9. B-spline alapú interpoláció.....	31
2.10. k-D fa.....	33
2.11. Python/C API	34

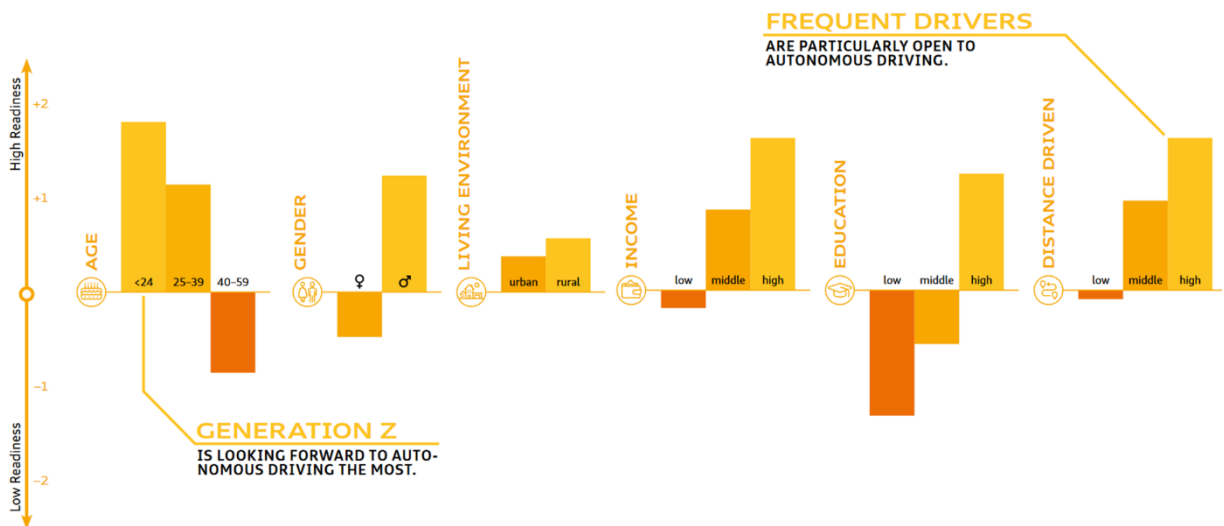
3. Készített szoftverek bemutatása	36
3.1. Marvelmind szenzoradatok rögzítése.....	36
3.2. Pályadigitalizálás.....	37
3.3. Marvelmind szenzorok integrálása a Donkey® platformba	39
3.4. Valós idejű pozíciómeghatározás.....	42
3.4.1. Sávon belüli pozíció meghatározása az autóágens számára.....	42
3.4.2. A jármű helyzetkoordinátájának megosztása más szolgáltatások számára	45
3.5. Hibaterjedés és pontosság-hígulás számítása	46
3.6. Gyűjtött adatok elemzése	48
3.7. Gyűjtött adatok jóságának meghatározása	51
4. Eredmények, értékelés	53
4.1. Adatgyűjtés.....	53
4.2. Gyűjtött adatok feldolgozása, értelmezése és elemzése.....	55
4.3. Megfigyelések, tapasztalatok	61
5. Jelenleg futó munkák.....	64
5.1. Adatok jósága a sávtartáson túl.....	64
5.2. Reinforced Learning.....	64
5.3. Kálmán-szűrő	65
6. Összefoglalás	66
7. Irodalomjegyzék.....	68
8. Saját munka leírása.....	72
9. Köszönetnyilvánítás	74

Kivonat

Az önvezető járművek kutatás-fejlesztésével kiemelten foglalkoznak, mind a szenzorok mind a járművek területén, melyet a technológiára fordított tetemes összegek is alátámasztanak. Mint szakterület fokozott multidiszciplinaritással bír, melyet a jelenleg tapasztalható jelentős társadalmi és autópári változások is segítenek. Karunkon folyó önvezető jármű kutatási és oktatási projektben a költségek racionalizálása végett kis méretű önvezető modellautók fejlesztése indult meg. Ennek célja, hogy kis méretben vizsgáljuk azon nehéz körülményeket, melyek a valós méret esetében nehézkes, illetve kockázatos. Ilyen lehet például a zaj vagy egyéb okból fellépő szenzorjeldegradáció, melyet okozhat sérülés, időjárási- és szélsőséges út-viszonyok. A kutatás jelen fázisában 1:16 méretarányú modellekkel dolgozunk, melyekhez különböző kombinációkban illesztünk szenzorokat, lefedve a lehetséges érzékelés spektrumokat. A cél, hogy a megtanult információk a valós méretben is alkalmazhatóak legyenek. Ennek részeként integráltunk a modellautóba a Marvelmind beltéri helyzetmeghatározó rendszert (IPS), melyet innovatív módon felhasználtunk az önvezető ágens szenzoraként és mérőeszközként is. Ultrahangos távmérési technológiát alkalmaz, ami újdonságnak számít az önvezető járműmodellek pozíciókövetésében. Bár léteznek hagyományos megoldások, mint a rádióhullám vagy GPS alapú helyzetmeghatározás, azonban ezek használata igencsak korlátozott. GPS szabad égbolt szükséges, míg a rádióhullám alapú technológiák esetén a pontatlanság jelentős. Az IPS megoldásunkkal valós időben $\pm 2\text{cm}$ -es pontosságot sikerült elérnünk, mely megoldásnak részét képezi az általunk kidolgozott mérési pontosságot meghatározó és zajszűrő algoritmusok is. Ezen a pozícióadatokból olyan metrikát állítottunk elő, ami alapját jelentheti önvezető ágensek megerősítő tanításához, valamint vezetésbiztonsági előírások követelhetőek meg a segítségével.

1. Bevezetés

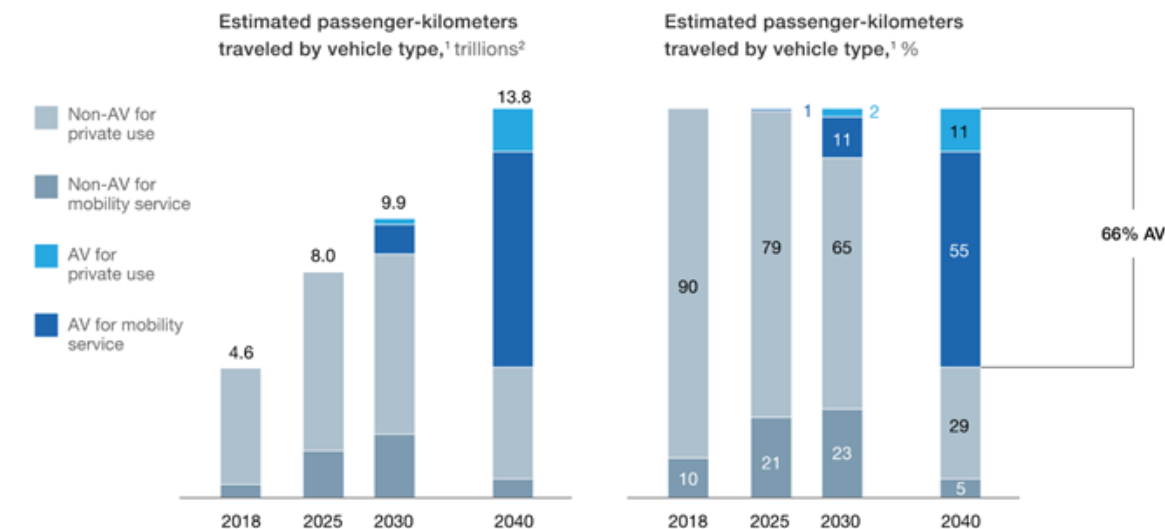
Az önvezető autók számos vélemény szerint a jövőnk mindennapi részévé válhat, és ehhez nem is szükséges a túl távoli jövőre gondolni. A Nissan szakembereinek felmérése alapján az általuk megkérdezett kisvállalatok 55%-a úgy gondolja, hogy flottáik egésze önvezetővé fog válni 20 éven belül, míg 66%-uk szerint a flottájuk teljesen elektromos járművekből fog állni [1]. 38%-uk szerint ez 10 éven belül is bekövetkezhet, míg 14% szerint ezt a váltást 5 éven belül végrehajtják. A megkérdezettek 35%-a már most használ hasonló technológiájú járműveket. A már most intelligens technológiákat használók motivációk eltérőek lehetnek. Ezen üzleteknek a három leggyakoribb motivációs tényező az üzleti hatékonyság növekedése (35%), a költségmegtakarítás (21%), és a megnövekedett fenntarthatóság (14%).



1. ábra. Az emberek készsége az önvezető járművek használatára [2].

A 1. ábra alapján az önvezető járművek használatára a legbefogadóképesebb csoportok korosztály szerint a Z generáció, valamint a vezetési szokások alapján a rendszeresen hosszú távú vezetést végző emberek [2]. Nemzetközi szinten az emberek 62%-a érdeklődéssel, míg 49%-uk optimistán fordul az önvezető technológiák felé. A globális piacon Kína lehet potenciálisan a legnagyobb önvezető jármű felvevő piac. Itt az érdeklődők és optimisták aránya már most eléri a 84%-os és 76%-os szintet, míg 2040-re előrejelzések szerint az utas-kilométerek 66%-a önvezető járművekkel történhet (2. ábra), ahol csak minden 6 utazásból 1 történne saját járművel, ha a közös utazási szokások a jelenlegi trendek szerint alakulnak [3].

Autonomous vehicles (AV) will travel about 66 percent of total passenger-kilometers in 2040.



¹Baseline scenario.

²Figures may not sum, because of rounding.

McKinsey&Company

2. ábra. 2040-es becslés a kínai utazási szokásokra [3].

A fejlesztések komolyságát jelzik az tech óriások befektetési összegei is. Itt megemlíthetjük a Waymo-t, Tesla-t, vagy az Audit is. A Waymo a Google laborjának egy önvezető autók fejlesztésével foglalkozó vállalata, akik már 5.5 milliárd \$ magántőkét gyűjtöttek 2021 június 16. bezárólag [4]. A Tesla elektromos autók gyártó vállalat 20.2 milliárd \$ tőkét gyűjtöttek össze 2020 május 9. bezárólag [5]. A cég elsődleges célja elektromos autók gyártása, de mindegyik autójukhoz érkezik vezetéssegítő rendszer, ami folytonos fejlesztés alatt áll, azzal a nem titkolt céllal, hogy egyszer teljesen önvezetővé váljanak az autók. Az Audi ezek mellett azt nyilatkozta, hogy 2023-ig közel 16 milliárd \$ fognak szálni önvezető autók és fenntartható technológiák fejlesztésére [6]. Ők 2017-ben kezdték meg az önvezető autók fejlesztését Münchenben. Több mint 200 alkalmazottal 2018-ra már 12 önvezető autójuk volt a városi utakon forgalomban. Ebből látható, hogy komolyan veszik az ilyen típusú járművek fejlesztését.

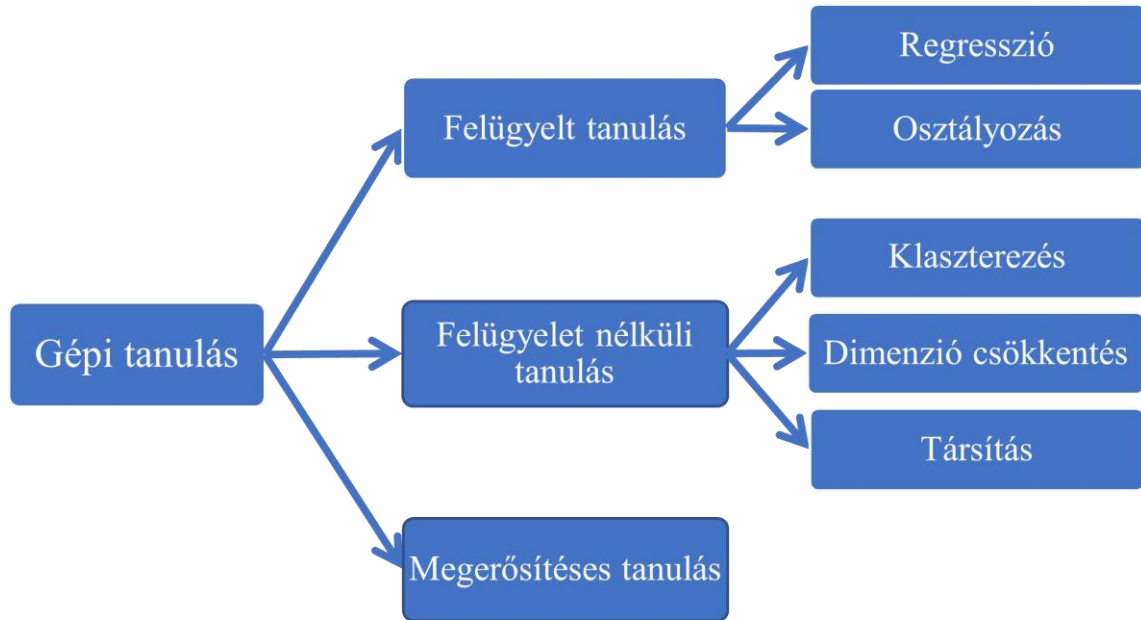
Kisebb-nagyobb startupok is jelen vannak a piacon, a már most nagy és neves cégek mellett. Ezek közül megemlíthető a Rivian nevű, elektromos teherjáratokat és SUV-okat fejlesztő cég, akik az önvezető funkció beiktatása mellett 10.7 milliárd \$ tőkét is sikeresen összeszedett, és fő befektetői között a Ford is megtalálható [7]. Továbbá érdekes a Cruise nevű cég is, akik 13 milliárd \$ tőkét gyűjtöttek össze 2021 június 15. bezárólag, és befektetőik között olyan nagy amerikai vállalatok is szerepelnek, mint a General Motors és a Walmart. Céljuk kifejezetten

olyan önvezető járművek építése, fejlesztése és szolgáltatásként történő biztosítása, amik „biztonságosan kötnek össze embereket helyekkel, dolgokkal és élményekkel” [8]. Így ők már a technológia fejlesztésén túllépve már Walmart házhozszállítási és utasszállítási szolgáltatásokat is nyújtanak saját önvezető autómobil flottájukkal [9].

A előbb említett adatok alapján látszik, hogy az önvezető járműveknek nem csak a jövőben, hanem már a jelenben is meghatározó szerepük van. A fejlesztésükhöz azonban több dolog is szükséges. A megfelelő hardvereken kívül, mint minden más mesterséges intelligencia esetén is, szükség van magas szintű, és a feladatot megfelelően megoldani képes algoritmusok implementálására, valamint rengeteg adatra és tesztelésre, amin alapján meg tudja tanulni a járművünk, hogy mi számít jó vezetésnek, legyen szó akármilyen tanulási módszerről. Ehhez nem csak egy hatalmas adatkészlet szükséges, hanem megfelelő módszerre is, ami alapján eldönthető, hogy melyik adat minősül megfelelőnek és melyik nem. Ezentúl nagy segítséget jelent a tanítási folyamat alatt, ha az adatok csoportosítva is vannak, és egy rendszer jelzi azt, hogy milyen típusú adatból van kevés. Ennek a miéértjét a következő (2.1) fejezetben tárgyaljuk.

2. Irodalmi áttekintés

2.1. Gépi tanulás



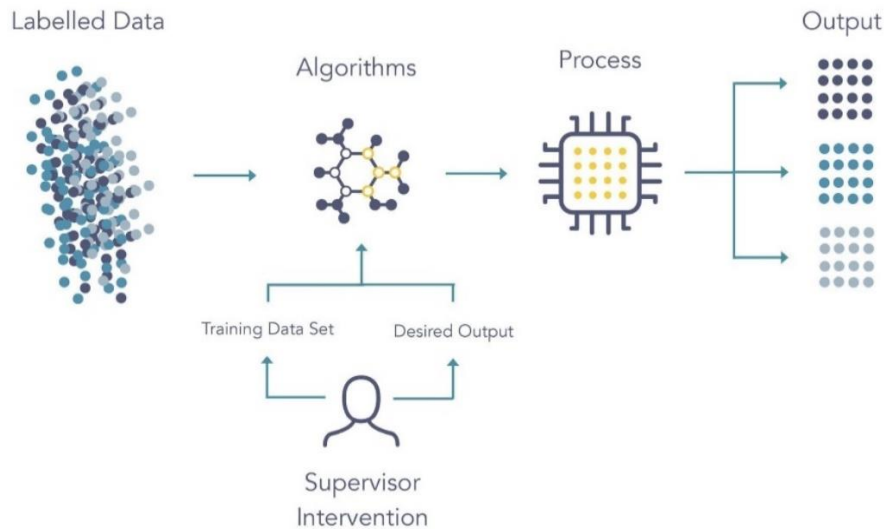
3. ábra. A gépi tanulás típusai.

A gépi tanulásnak 3 fő részhalmazát különböztetjük meg, a 3. ábra szerint bemutatott módon:

1. Felügyelt tanulás, ami egy előrejelző modellt készít a be-/kimeneti adatok és a köztük fellelhető kapcsolatok alapján.
2. Felügyelet nélküli tanulás, ami csoportosítja és értelmezi az adatokat, a bemeneti adatok és a köztük megfigyelhető kapcsolatok alapján.
3. Megerősítéssel tanulás: Egy olyan tanulási forma, ahol a tanuló folyamat egy cselekvést hajt végre, majd az erre érkező megerősítő (jutalmazás) vagy elutasító (büntetés) külső válasz alapján tanulja meg azt a viselkedésmódot, ami maximalizálja a jutalmak és minimalizálja a büntetések összegét.

2.2. Felügyelt tanulás

A felügyelt tanulás (supervised learning) a gépi tanulásnak az a részhalmaza, ahol egy algoritmus a bemenetei értékekhez egy kimeneti értéket vagy osztályt társít, előre rögzített példák kimeneti-bemeneti adatpárjai alapján [10]. A folyamat felcímkézett tanító példák halmazából álló adatbázist használ fel a tanulásához [11].

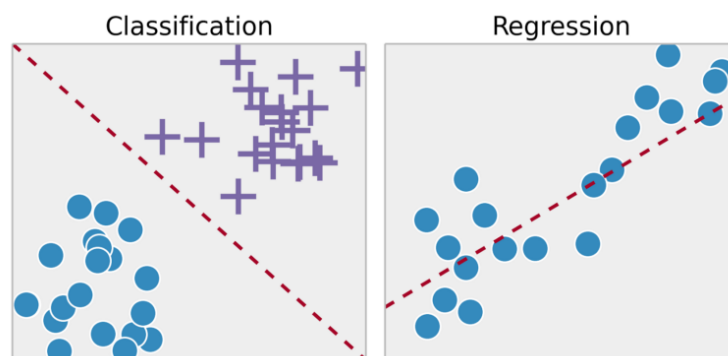


4. ábra. A felügyelt tanulás működési folyamata [12].

Általában minden ilyen adatpár egy bemeneti objektumból (általában vektor) és egy kimeneti értékből áll. Ezek alapján az algoritmus egy olyan leképezési függvényt hoz létre, amivel akár milyen új bemeneti értékeket is le tudunk képezni a kimeneti értékek valamelyikére, egy – a tanuló adatok által meghatározott – modell alapján (4. ábra). Ez optimálisan egy olyan leképző függvényt eredményez, ami minden bemeneti lehetőséghez az általunk várt kimeneti címke értéket társítja. Ez általában nem jelenthető ki 100% biztonsággal, viszont elvárt, hogy az illesztett modell előre nem látott adatokhoz is megfelelő pontossággal rendelje a megfelelő címkét. Ezen mértékmeghatározására az általánosítási hiba nevű statisztikai jellemzőt használjuk, ami a címke becslés sikerességét mondja meg ismeretlen bemeneti adatok esetén [13].

Többféle felügyelt tanítási algoritmus létezik. Ezeket sokszor használják kép és objektum felismerésre, előrejelző rendszerek működtetésére és folyamat automatizálásra is. A végrehajtott feladat célja szerint két fő feladatcsoportot különböztethetünk meg ezen algoritmusok között [14]. Az ezek közötti eltérést a 5. ábra jól szemlélteti.

Az egyik csoport az osztályozás (classification). Ennek a célja a bemeneti adatok megadott kategóriák valamelyikébe történő besorolása (pl. táblafelismerés). Itt az összes lehetséges kategória előre definiált, és az algoritmusnak a tanító adatok alapján kell egy olyan modellt létrehoznia, amely eldönti, hogy milyen értékek esetén fog egy adott csoportba tartozni egy bemeneti adatvektor. A kimenet mindig valamilyen kategóriacímke. Ebbe a csoportba tartoznak a lineáris osztályozások, tartóvektor-gépek, döntési fák, k-legközelebbi szomszéd alapú algoritmusok és random fák.

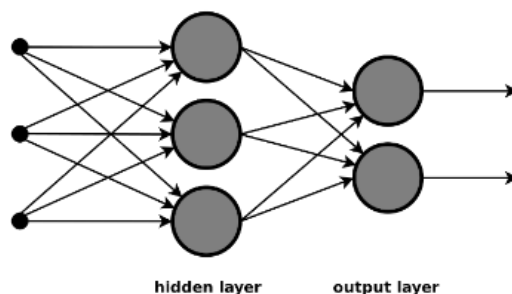


5. ábra. Az osztályozás és a regresszió közötti különbség bemutatása példával [15].

A másik ilyen csoport a regresszió (regression). Ezen esetben a bemeneti adatok függésének vagy függetlenségének a meghatározása a cél. Ekkor nem egy konkrét kategóriába kell besorolni egy bemeneti adatot, hanem a tanító adatok alapján illesztett modell segítségével, a tanító és aktuális bemeneti adatok közötti összefüggést kell megmondani. A kimenet mindig valamilyen valós érték, amit a készített modell alapján határozunk meg. Ilyen algoritmusok a lineáris, logisztikai és polinomiális regressziók.

2.3. Neurális hálózatok

2.3.1. Neurális hálózatok felépítése



6. ábra. Egyszerű neurális háló felépítése [16].

A neurális hálók a gépi tanulásnak, azon belül is a felügyelt tanulásnak egy részhalmaza. Ezek egymással valamilyen viszonyban álló, egymást befolyásoló csomópontok (node) rétegeiből (layer) épülnek fel. Mivel nevüket az emberi agy felépítéséről kapták, ezért ezeket a csomópontokat mesterséges neuronoknak is szokás nevezni. A csomópontokhoz és a közöttük levő kapcsolatokhoz is tartoznak bizonyos súlyok és függvények, amik a modell működését befolyásolják. Így egy adott csomópont a bemeneti értékeit bizonyos módon súlyozza, és a saját értékét

is egy bizonyos függvény által határozza meg. Ezt az értéket továbbítja a többi, hozzá kapcsolódó csomópontnak, akik a kapcsolathoz társított súllyal arányosan fogják ezt figyelembe venni a saját kiértékelésük folyamán. Ezeket az értékeket a hálózatok egy tanító adatkészlet alapján elvégzett optimalizáció során határozzák meg. Céljuk az egyre pontosabb kimenet meghatározás a bemeneti értékek alapján [17]. Egy neurális hálózat legalább egy bemeneti (input layer), egy rejtett (hidden layer) és egy kimeneti (output layer) réteget tartalmaz. Az így kapott hálózat és a részeihez tartozó határértékek és súlyok együttesét modellnek nevezzük (6. ábra).

2.3.2. Neurális hálózatok működése

A fent említett kiértékelés elvégzéséhez minden egyes csomópont (neuron) elvégzi a saját kis regresszióját. Különböző regressziós algoritmusokat is beállíthatunk a csomópont értékének meghatározásához, aminek a legegyszerűbb esete a lineáris regresszió. Ez esetben a neuron bemeneti egyenletének a meghatározásához tegyük fel, hogy:

- az m a neuron bemeneti kapcsolatainak a száma,
- az i az aktuális elem indexe,
- az x_i az i . bemenetről érkező érték, és
- a w_i az i . bemenetről érkező értékhez társított súly.

Így az alábbi egyenletet kapjuk:

$$w_1x_1 + w_2x_2 + \dots + w_mx_m = \sum_{i=1}^m w_ix_i$$

Az így meghatározott érték alapján meghatározhatjuk a kimeneti értéket is, ami az aktiváló függvénytől függ.

- A legalapvetőbb esetben ez a határérték függvény. Ennek használata esetén:

$$kimenet = \begin{cases} 1, & \text{ha } \sum_{i=1}^m w_ix_i \geq 0 \\ 0, & \text{ha } \sum_{i=1}^m w_ix_i < 0 \end{cases}$$

- Sigmoid aktiváló függvény esetén:

$$kimenet = \frac{1}{1 + e^{-\sum_{i=1}^m w_ix_i}}$$

- Kijavító (rectifier) aktiváló függvény esetén:

$$kimenet = \max\left(\sum_{i=1}^m w_i x_i, 0\right)$$

- Hiperbolikus tangens aktiváló függvény esetén:

$$kimenet = \frac{1 - e^{-2 \times \sum_{i=1}^m w_i x_i}}{1 + e^{-2 \times \sum_{i=1}^m w_i x_i}}$$

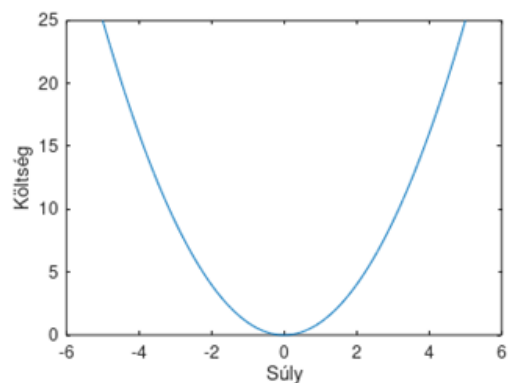
Amennyiben az algoritmusunkat felügyelt tanítással, megfelelően felcímkezett adatokkal (például képekkel) szeretnénk tanítani, akkor a tanítás alatt a modell pontosságát is ellenőrizni kell. Ezt egy költségfüggvénnyel (vagy más néven veszteségfüggvénnyel) tehetjük meg. Ehhez a legtöbb esetben (például az előző példában is használt lineáris regresszió esetén is) az átlagos négyzetes hibát használjuk (mean squared error, MSE), amit az alábbi módon írhatunk fel:

$$\frac{1}{2m} \sum_{i=1}^m (\hat{y}_i - y_i)^2$$

Ahol:

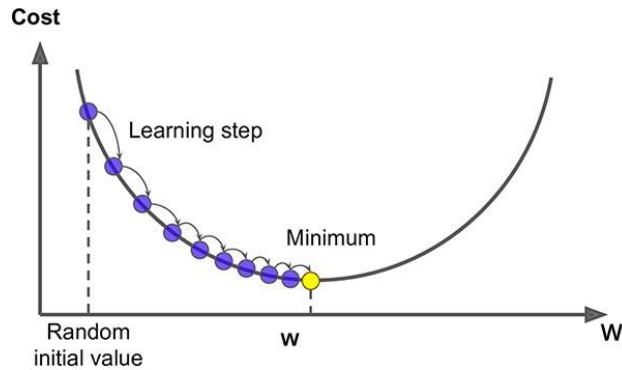
- Az m a minták darabszáma.
- Az i az aktuális elem indexe.
- Az $\hat{y}$ a becsült eredmény (a fenti példa alapján a legutolsó, kimeneti réteg $f(x)$ értéke).
- Az y pedig a valós eredmény (az aktuális tanító adatpár kimeneti értéke).

Ennek a minimalizálása lenne a célunk a tanítási folyamat alatt, azzal a céllal, hogy megfelelő módon rendeljünk egy osztályt vagy egy várható értéket bármilyen bemeneti adathoz. Ez egy neuron esetén egy egyszerű másodfokú függvény minimumkeresése (7. ábra), de komplexebb hálózatoknál az optimalizálandó függvény is komplexebb lehet.



7. ábra. Egy lineáris neuron hibafülete egy tanítási érték esetén.

A tanulás alatt egy visszaterjesztési algoritmus (backpropagation) próbálja a lehető legjobb eredményt adó súlyokat megtalálni a hálózatunkhoz. Ezt a súlyok módosításával, és a validációs adatok ezen változtatott modellen történő ellenőrzésével éri el, a tanítási folyamat iterációi alatt.



8. ábra. A gradiens leszámazási módszer [18].

A súlyok megfelelő optimalizálását egy úgynevezett gradiens módszerrel (gradient descent, 8. ábra) éri el. Ez minden ciklus alatt veszi a költségfüggvény deriváltját, és az így kapott meredekség segít meghatározni, hogy a súlyok módosítása milyen irányba és nagyságban történjen. Ezt addig szeretnénk futtatni, ameddig elérjük a minimum értéket, viszont ez a módszer csak egy lokális minimum megtalálását biztosítja. Az iteráció az alábbi, ha a rendszer nem elfogult (vagyis nincs benne rendszerszintű hiba) [19]:

1. Inicializáljuk a w paramétert (random).
2. Határozzuk meg a tanulási rátát.
3. Ismételjük konvergálásig a következő műveletet:

$$w_{t+1} = w_t - \alpha \left(\frac{\partial C}{\partial w_t} \right)$$

Ahol:

- A w_{t+1} és w_t az új súly és az eddigi súly, vagyis t az iteráció indexe.
- Az α a tanulási ráta.
- A C az költségfüggvény aktuális súlyokkal kiszámolt értéke az m elemű tanító adathalmazon.

Ha szeretnénk, hogy globális minimum értéket találjunk a folyamatunk, akkor érdemesebb az eljárás egy sztochasztikus változatát használni. Ugyanis a sztochasztikus gradiens algoritmus a

visszaterjesztési lépést nem csak az egész tanító adathalmazon történő végig tanulás után, hanem minden egyes értékéhez tartozó kimenet megbecslése után is végrehajtja. Ha ugyanezt a tanító adat minden eleme helyett csupán kisebb részegységei után szeretnénk végrehajtani, akkor ugyanezt a folyamat mini-batch módban történő elvégzésével tehetjük meg.

Ezek hatására így módosul az elvégzendő művelet:

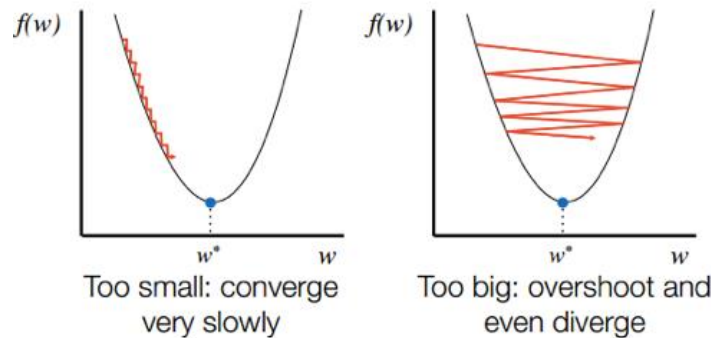
- Sztochasztikus esetben:

$$w_{t+1} = w_t - \alpha \left(\frac{\partial C_i}{\partial w_t} \right), \text{ ahol } C_i = \frac{1}{2} (\hat{y}_i - y_i)^2$$

- Mini-batch esetben:

$$w_{t+1} = w_t - \alpha \left(\frac{\partial C_b}{\partial w_t} \right), \text{ ahol } C_b = \frac{1}{2n} \sum_{i=j+1}^{j+n} (\hat{y}_i - y_i)^2, \text{ ahol}$$

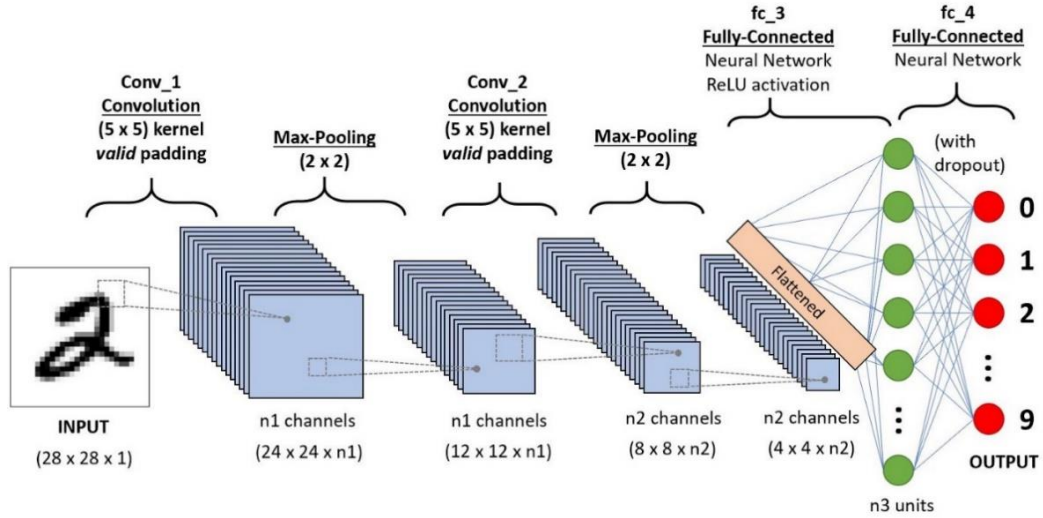
- n a mini-batch csomagmérete és
- $j = 0, n, 2n, \dots, m - n$.



9. ábra. A tanulási ráta nagyságának hatása [20].

A tanulási ráta, mint a súlyok iterációk közötti módosításának mértékét meghatározó tényező, megfelelő meghatározása fontos (9. ábra). Ugyanis túl nagy értéket felvételekor a folyamat túl-
 lőheti az új veszteségi értéket a minimumértéken, és elég nagy érték esetén akár divergálhat is a folyamat a minimum értéktől. Túl kicsi érték megadása esetén pedig túlságosan lassan fog konvergálni a lokális minimum értékhez a folyamat, ami feleslegesen növeli a tanításhoz szükséges idő és erőforrások szükségességét [20].

2.3.3. Konvolúciós neurális hálózatok

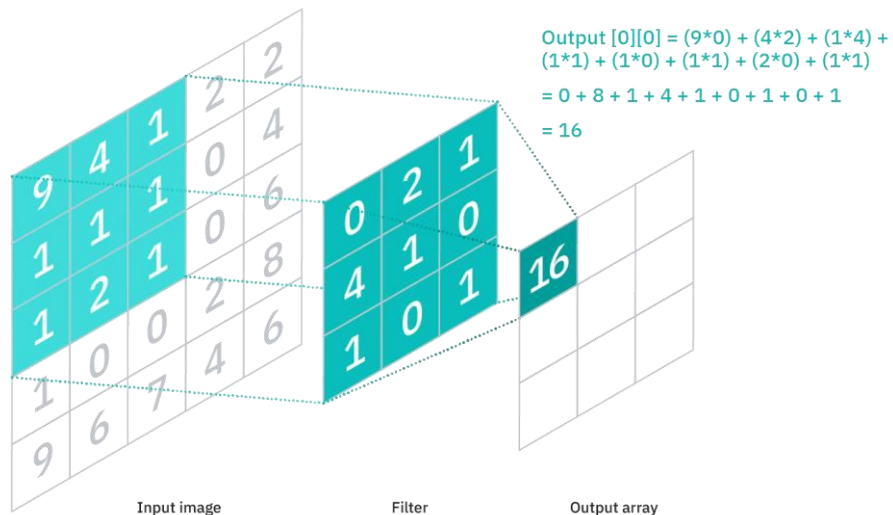


10. ábra. Egy kézzel írt számokat felismerő konvolúciós neurális háló [21].

A konvolúciós neurális hálózatokban háromféle rejtett réteg is előfordul [22]:

1. Konvolúciós réteg (convolutional layer, 11. ábra):

Konvolúciót hajt végre, vagyis a bemeneti mátrixon végigmegy egy tulajdonságfelismerő szűrővel. A szűrő egy tetszőleges méretű, de általában 3x3-as súlymátrix. A réteg a bemeneti mátrixnak veszi egy, a szűrővel megegyező méretű részmátrixát, és ezen konvolúciót hajt végre.



11. ábra. A konvolúciós réteg működése [22].

Ezután tovább lép egy vagy több oszloppal, és megismétli a műveletet. A sor végére érve tovább lép egy vagy több sorral, és megismétli az előbb végzet folyamatot. Az így elvégzett konvolúciók eredményeképpen kapott kimeneti mátrixot adja ki a kimenetén, amit tulajdonságtérképnek nevezünk. Ebben a rétegben történik a számítások nagyrésze.

2. Összevonó réteg (pooling layer):

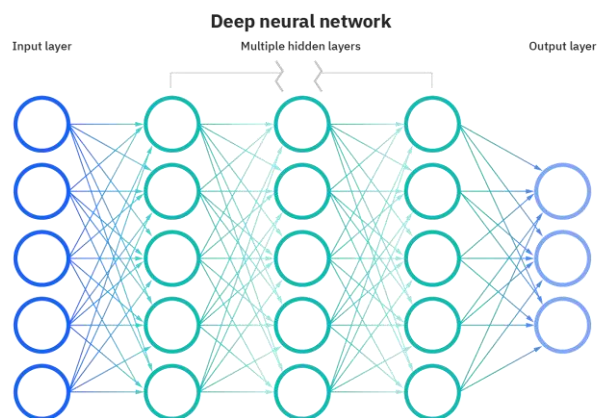
Célja a paraméterek, és így a komplexitás csökkentése. Ez is hasonlóan végigmegy egy szűrővel bemeneti mátrixán, de nem hajt végre konvolúciót, vagy más teljesítményigényes feladatot, hanem csak egy összesítést végez vele. A szűrőn belüli értékek közötti maximum vagy átlag értékekből összeállt mátrixot adja kimenetként.

3. Teljesen összekapcsolt réteg (fully connected layer):

Neve önmagarázó. Az előtte lévő bemeneti mátrixok kilapításra kerülnek, és ez a réteg az összes konvolúció és összevonás után maradt legvégső kimeneti mátrix összes elemével összekapcsolja az összes csomópontját, vagyis egy teljesen összekapcsolt réteget kapunk. Ezek alapján az ilyen típusú rétegek végzik el az előszűrt adatok elemzését.

Ez a gyakorlatban elősegíti, hogy nagyobb méretű bemeneti adatokat is elemezzünk úgy, hogy azok méretcsökkentése alatt megőrizzük az adatok lényegi jellemzőit. Ezentúl nagyon jól használható kinézeti, működési vagy egyéb más hasonlóságok és összefüggések megtalálására az azonos kategóriájú elemek között. (például meg tudja tanulni a háló, hogy milyen színű a piros, sárga és zöld lámpa a jelzőlámpán és fel is tudja később azokat ismerni.)

2.3.4. Mély tanulás



12. ábra. Mély neurális háló felépítése [17].

A mesterséges intelligencia területén egy gyakran használt fogalom. A mély tanulás esetében nem is egy konkrét algoritmusra kell gondolni, hanem egy olyan gyűjtőfogalomra, amely magában tartalmaz minden olyan tanulási folyamatot, de legfőképpen neurális hálózatokat, amelyeknek a mélysége nagyobb mint 1 (12. ábra). Ez a neurális hálózatok esetén több rejtett réteg használatát jelenti. A fent megemlített konvolúciós neurális hálók ezek alapján mély tanuló hálózatoknak számítanak, ugyanis legalább egy konvolúciós, egy összevonó és egy feldolgozó rejtett réteget is tartalmaz a hálózatában.

Ezek alapján láthatjuk, hogy a tanításhoz használt adatok és az azokhoz társított kimenetek lényegében határozzák meg a neurális hálók betanulását és a modellek működését is, ezért szükség van ezek megfelelő felcímkézése, sokszínűsége és nagy mennyisége mellett azok előszűrésére és előkészítésére is. Ami egyre fontosabb lesz, amint egyre komplexebb gépi tanulási módszereket kezdünk használni.

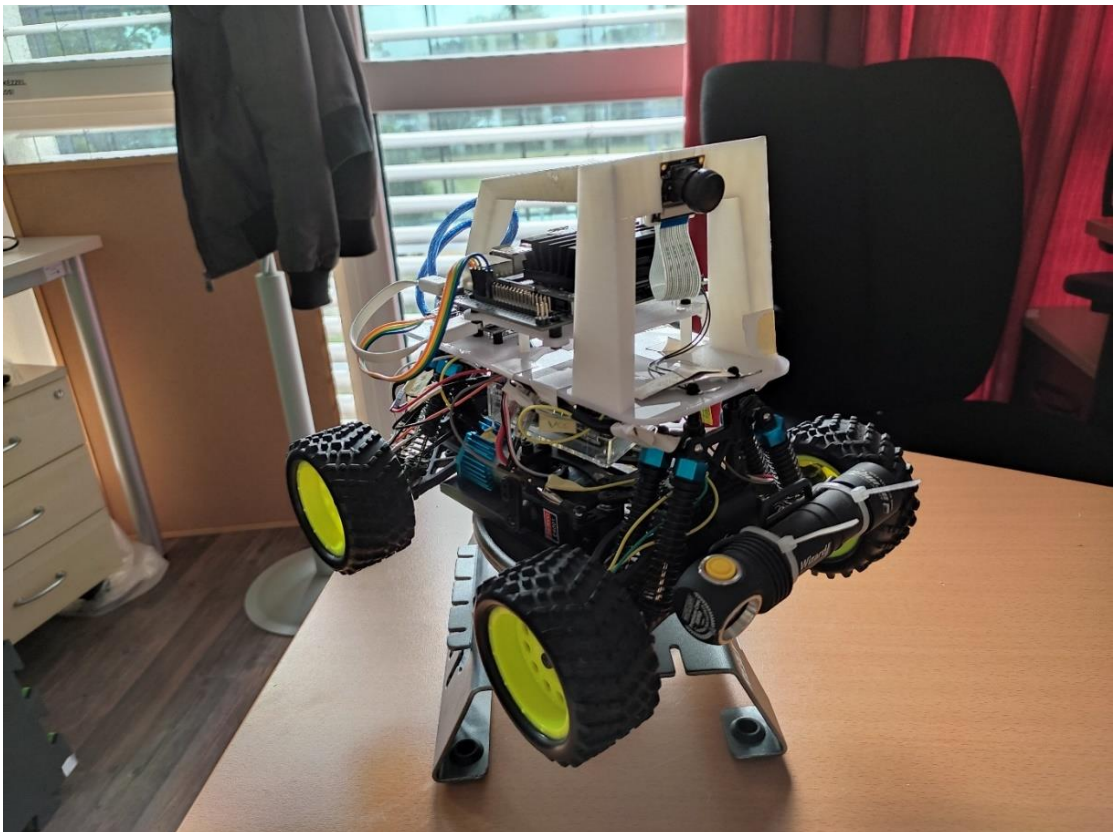
2.4. A Donkey® platform



13. ábra. Egy Donkey® leírás alapján készített kisautó [23].

Az önvezető rendszerek fejlesztéséhez mind biztonságos fejlesztési környezetre, mind pedig különböző szenzorok beintegrálása szükség van az épp fejlesztés alatt álló járműv rendszerébe. Az önvezető járművek fejlesztési előmunkáinak megkönnyítését, egyszerűsítését és valamint az ilyen jellegű projektek fejlesztésének elérhetőbbé tételét segíti elő a Donkey® DIY (Do It Yourself – csináld magad) nyílt forráskódú önvezető platform, amit önvezető autók építésével és fejlesztésével kapcsolatban érdeklődő önkéntesek fejlesztenek. A projekt kisméretű önvezető modellautók megépítéséhez biztosít háromdimenziós modelleket (13. ábra), és egy már előre felállított vezérlési, tanítási és önvezetési keretrendszert [23].

Ez a biztonságos környezet megteremtése mellett lehetőséget biztosít a kamera és vezérlési adatok mellett más szenzorok integrálására is a környezetbe. Központi feldolgozó egységként alapértelmezetten egy Raspberry Pi eszközt használ, de ehelyett egy Nvidia Jetson Nano (14. ábra) is használható, ha számításigényesebb feladatokat is akarunk elvégezni a kisautón. Az irányítást egy szervó és egy villanymotor végzi. A felület továbbá egy egységes keretrendszert is biztosít tetszőleges szenzorok csatlakoztatásához hardveres és szoftveres szinten is.



14. ábra. A projekt alatt használt egyik kisautó.

2.5. Marvelmind szenzorrendszer

2.5.1. Használat célja

Munkánk során célunk volt a kisautó helymeghatározása a tanító és tesztpályán, mivel megfelelő pontosságú pozicionálás esetén lehetőség nyílik az adatok szelektálásának és kiértékelésének automatizálására, és a fizikai eszközünk elhelyezésére egy kiterjesztett valóságú környezetben, ami tanulókörnyezet virtualizációját is lehetővé teszi. Ebben a kamerakép feldolgozása is segítségre lehet, de ez esetben egy nagy számítási igényű folyamattá válik a sávon belüli pozíciómeghatározás, emiatt célszerűbb egy külső szenzor használata a pozíciós koordináták meghatározására.



15. ábra. Egy kezdőkészletnyi Marvelmind beltéri helyzetmeghatározó rendszeregység [24].

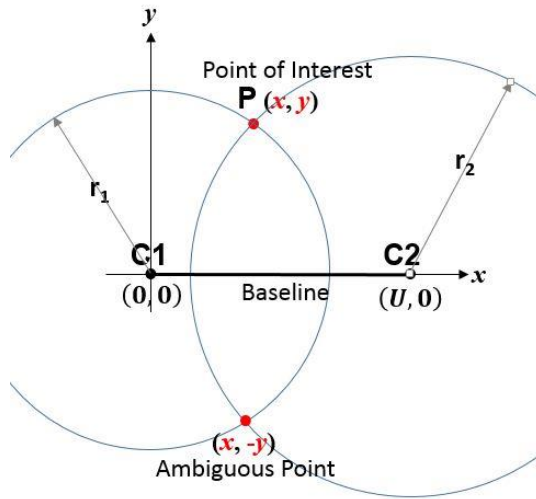
2.5.2. A rendszer működési elve

Ehhez választanunk kellett egy lokalizációs szenzorrendszert, amit integráljunk a Donkey® ke-retrendszerbe. A feladat betöltésére a Marvelmind Robotics beltéri helymeghatározó szenzor-rendszerért választottuk [24]. Ez ± 2 centiméteres pontossággal tud távolságot mérni a rendszer-pontok között ultrahangos távolságmeghatározással, amely távolságértékekből egy vagy több mozgó szenzor, a rendszer saját koordináta-rendszerén belüli, pozíciójának meghatározásra képes, multilateráció segítségével. A rendszer előnye, hogy nagy pontosságot biztosít a WiFi, Bluetooth és Ultra-Szélessávú technológiákat használó megoldásokhoz képest, és hogy tetsző-leges mértékben skálázható.

2.5.3. True-range multilateráció

Mind ahogy már az előbb említettük, a Marvelmind szenzorrendszer lehetőséget biztosít előre lehelyezett, referenciapontként szolgáló szenzorok segítségével, egy vagy több mozgó szenzor pozíciójának meghatározására. Ezt true-range multilaterációs algoritmussal végzi el. Az eredeti dokumentáció [25] két true-range multilaterációs [26] módszerre hivatkozik, amivel két- és háromdimenziós Descartes-féle koordináta-rendszerben tudja meghatározni a szenzorok helyzetét.

A kétdimenziós meghatározás alapvető esetét trilaterációnak is nevezik (16. ábra). Ez egy kifinomult háromszögelési folyamat, ami képes egy síkban megadni egy pont helyzetét, két vagy több más állomásponthoz viszonyított távolsága alapján.



16. ábra. Kétdimenziós eset.

A kétdimenziós eset számítási módszere a következő:

- Legyen $B_1(0, 0)$ és $B_2(U, 0)$ két fixen elhelyezett szenzor középponti koordinátája, ahol az U a B_1 és B_2 közötti távolság.
- Legyen $P(x, y)$ a meghatározandó szenzor középponti koordinátája.
- Legyen r_1 és r_2 a $B_1 - P$ és $B_2 - P$ közötti távolság, ahol Pitagorasz-tétele alapján:

$$r_1^2 = x^2 + y^2$$

$$r_2^2 = (U - x)^2 + y^2$$

Ebből pedig meghatározhatjuk, hogy

$$x = \frac{r_1^2 - r_2^2 + U^2}{2U}$$

$$y = \pm \sqrt{r_1^2 - x^2}$$

Mivel a rendszer az x tengelyt a két referenciapont között határozza meg, így mindig az alapvető eset áll fent a rendszer saját koordináta-rendszerében. A P pont így ezen koordináta-rendszer x tengelye által meghatározott két félsík valamelyikében van. Mivel a munkaterület meghatározását elvárja tőlünk a rendszer, így a két pont közül a megfelelő kiválasztását automatikusan elvégzi a rendszer.

A Marvelmind szenzorrendszer 3 fix szenzorpontot ajánl, mind a stabil meghatározás, mind a háromdimenziós működés támogatása végett. Ez esetben a kétdimenziós meghatározás elvégezhető az alábbi egyenletrendszer megoldásával, tetszőleges referenciapontokkal a koordináta-rendszerünkben:

$$(x - x_1)^2 + (y - y_1)^2 = r_1^2$$

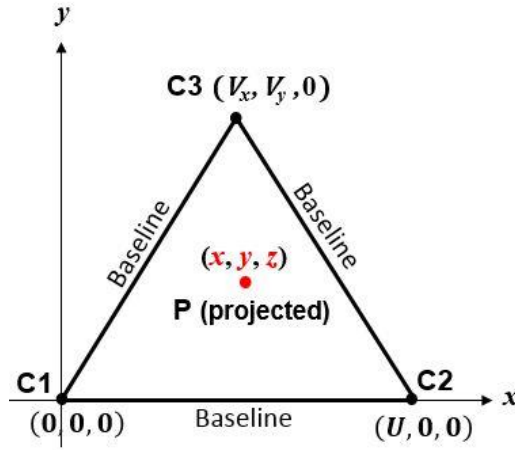
$$(x - x_2)^2 + (y - y_2)^2 = r_2^2$$

$$(x - x_3)^2 + (y - y_3)^2 = r_3^2$$

Ahol:

- $B_1(x_1, y_1)$, $B_2(x_2, y_2)$ és $B_3(x_3, y_3)$ három fixen elhelyezett szenzor középponti koordinátája és $P(x, y)$ a meghatározandó szenzor középponti koordinátája.
- r_1 , r_2 és r_3 a $B_1 - P$, $B_2 - P$ és $B_3 - P$ közötti távolság.

Amennyiben az adott pont háromdimenziós helyzetét akarjuk meghatározni, akkor ahhoz már minimum 3 referenciapont szükséges, mivel egy 3. ismeretlen is bekerül az egyenletbe. Ha a 3 referenciaponthoz 1 z koordináta tartozik, akkor így leírható a pontmeghatározás egyenlete (17. ábra):



17. ábra. Háromdimenziós eset.

- Legyen $B_1(0, 0, 0)$, $B_2(U, 0, 0)$ és $B_3(V_x, V_y, 0)$ három fixen elhelyezett szenzor középponti koordinátája, ahol az U a B_1 és B_2 közötti távolság, V_x és V_y adott koordináta értékek.
- Legyen $P(x, y, z)$ a meghatározandó szenzor középponti koordinátája.
- Legyen r_1 , r_2 és r_3 a $B_1 - P$, $B_2 - P$ és $B_3 - P$ közötti távolság, ahol Pitagorasz-tétele alapján:

$$\begin{aligned} r_1^2 &= x^2 + y^2 + z^2 \\ r_2^2 &= (x - U)^2 + y^2 + z^2 \\ r_3^2 &= (x - V_x)^2 + (y - V_y)^2 + z^2 \end{aligned}$$

Ha feltesszük, hogy $V^2 = V_x^2 + V_y^2$, akkor P koordinátái a következők:

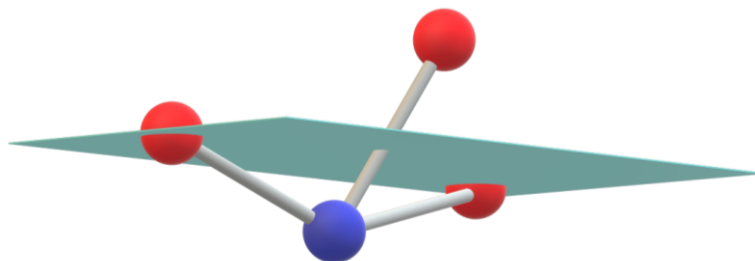
$$\begin{aligned} x &= \frac{r_1^2 - r_2^2 + U^2}{2U} \\ y &= \frac{r_1^2 - r_3^2 + V^2 - 2V_x x}{2V_y} \\ z &= \pm \sqrt{r_1^2 - x^2 - y^2} \end{aligned}$$

Mivel multilaterációról van szó, ezért a helymeghatározáson pontosíthatunk azzal, ha nem csak a minimális szenzorszámú eseteket használhatjuk. Lehetőségünk van ugyanis mind két- és mind háromdimenziós esetben is négy referenciapont használatára altérképenként, mivel egy extra pont esetén lehetőség van, hogy mindig azt a 3 szenzorértéket vegyük figyelembe a számításnál, amelyek a legnagyobb valószínűséggel fogják a legkisebb hibát tartalmazó eredményt adni. Ez

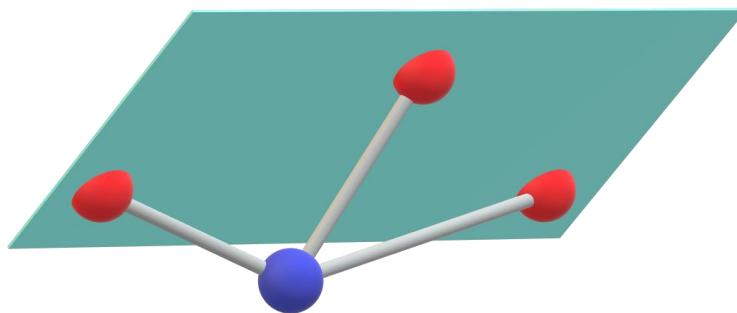
megoldás lehet azon esetekben is, amikor az egyik szenzor nem tudna jelet kapni, mert vagy egy fizikai tárgy, vagy más környezeti zaj zavarja vagy blokkolja a felé érkező jelet.

2.5.4. Bázistranszformáció

Ha nem egy z síkban lennének a referenciapontok, akkor bázistranszformációval meghatározható egy olyan új koordináta-rendszer, amiben ezek mind 1 „ z ” koordinátát kapnak. Az x tengely meghatározása még továbbra is az első két definiált referenciapont alapján történik. A transzformálás előtti állapotot a 18. ábra, míg a transzformálás utánit a 19. ábra mutatja meg. Piros gömbbel vannak jelölve a rendszer fix viszonyítási koordináták, kékkel a meghatározandó pont, szürke hengerrel a távolságok és türkiz lappal az aktuális térkoordináta-rendszer bázissíkja.



18. ábra. A bázistranszformálás előtt a referenciapontoknak különbözik a z értékük az Euklideszi koordináta-rendszerben.



19. ábra. A bázistranszformálás után az új koordináta-rendszerben a referenciapontok közül háromnak a 3. dimenzióban és kettőnek a 2. dimenzióban is megegyezik az értéke.

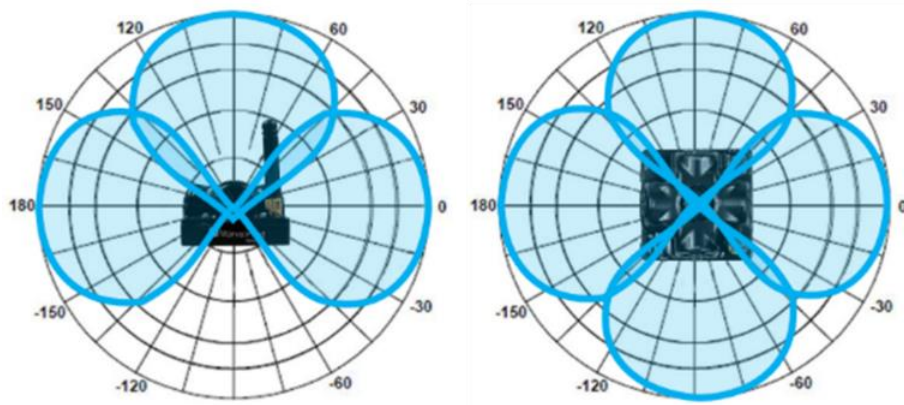
2.5.5. Koordináta-rendszer kibővítése

A meghatározott pont koordinátái ezután természetesen bármilyen másik bázisú koordináta-rendszerbe is átvethető, a megfelelő transzformációk elvégzése után. A rendszer magában biztosít lehetőséget az NMEA protokoll szerinti koordináta-rendszerbe történő automatikus átranszformálására, az origó hosszúsági és szélességi fokainak megadása után.

2.5.6. A rendszer felépítése

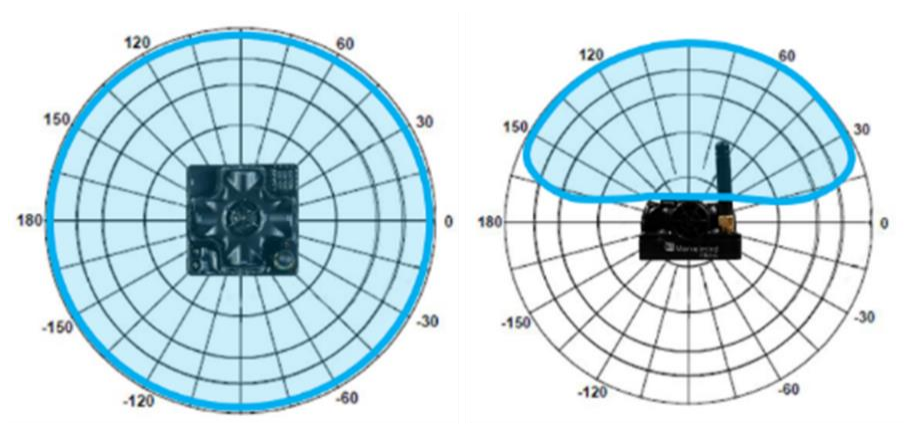
A rendszeren belül háromféle csomópontot különböztethetünk meg. Ezek a következők:

1. Beacon: Egy előre beállított frekvenciájú ultrahang jel vételére vagy adására képes, aminek segítségével távolságmérést végez. A jeladást és jelvételt 5 ultrahangos jel vevő/adón keresztül végzi (20. ábra). Ezentúl egy beépített inerciális mérő egységet (Inertial Measurement Unit – IMU) is tartalmazhat, ami giroszkópot, gyorsulásmérőt és magnetométert foglal magába. Az adatokat rádióhullámú adatként továbbítódnak a modemnek. A kapott adatok alapján a soros kimeneten továbbítja a saját vagy más szenzorok helyzetét is, az IMU szenzor által mért adatok mellett, az eszköz felszereltségétől és beállításaitól függően.



20. ábra. A Beacon öt ultrahang adó-vevőjének térbeli hallási szöge [24].

2. Super-beacon: Egy digitális többfrekvenciás mikrofonnal (21. ábra) is felszerelt beacon, aminek a segítségével egyszerre több, különböző frekvenciájú jelet is képes venni a szenzor. IMU-t mindig tartalmaz.



21. ábra. A Super-beacon digitális mikrofonjának hallási szöge [24].

3. Modem (22. ábra): Az összes szenzor által mért adatot fogadja és értelmezi. Ez szervezi össze a rendszert, hogy lehető lehessen a pozíciómeghatározás, és adja át az adatokat a kimeneti adatbusszal rendelkező készülékeknek, hogy meg tudják határozni a mozgópontok koordinátáit a rendszer koordinátarendszerén belül.



22. ábra. Egy Marvelmind 4.9-es verziószámú modem [24].

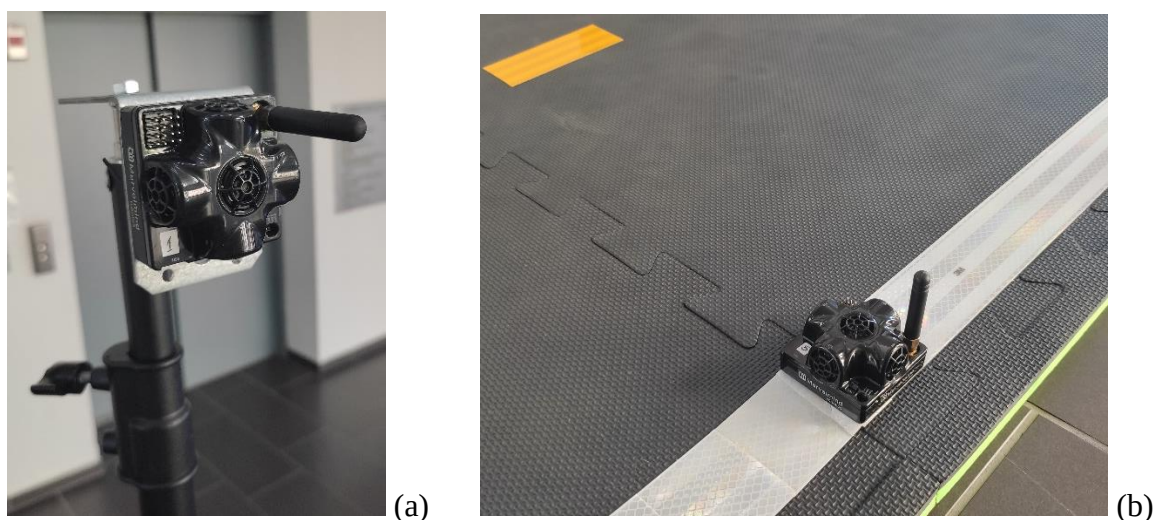
Míg a modemnek egyértelmű a működési feladata, addig a beacon-öknek kétféle működési módja is van, amit az úgynevezett „sündisznó mód” (hedgehog mode) ki vagy be kapcsolott állapotával jelzünk. A mód kikapcsolt állapota esetén a beacon maga a rendszer egyik referenciapontja. Ekkor a saját koordinátája a rendszeren belül fix, és feladata a sündisznó módban futó beacon-ök helyzetmeghatározásának segítése. A sündisznó mód bekapcsolt állapota esetében pedig a rendszer egyik mozgó beacon-jének számít a szenzor. Ez esetben a pozícióját a többi, referenciapont szerepét betöltő eszköz segítségével határozza meg a rendszer valamilyen sebességgel. A rendszer képes 16 Hz-es frissítési időre, de bizonyos feltételek betöltése esetén ez sűrűbb is lehet. A meghatározott pozíciók között is képes azonban köztes pontok meghatározására a rendszer a sündisznó módban lévő beacon-ök IMU és Ultrahang szenzorjainak adatfúziójával.

2.5.7. Architektúrák

A rendszer kétféle architektúrával is használható, aminek a működése alapjaiban különbözik. Az egyik a nem invertálható, a másik pedig az invertálható architektúra.

A nem invertálható architektúrában (NIA) a sündisznó módban lévő rendszeregységek adják az ultrahang jeleket, míg a többiek veszik ezeket a jeleket. Ezen jelek hossza szoftveresen állítható. A jeleket a sündisznók (23. ábra) egy meghatározott sorrendben küldik, hogy ne lépjen fel jelütkezés. Ezen sorrend és esetlegesen a más, alapjelhez csatolt egyéb információs jelek alapján az

ezeket vevő, referenciaponti szerepet betöltő rendszeregységek (23. ábra) meg tudják határozni, hogy melyik sündisznó milyen messze van tőlük. Ezen távolságadatokat alapján végzi el a rendszer a helyzetmeghatározásokat a multilaterációs számítások elvégzésével. Az architektúrának a nem invertálhatósága a jelek frekvenciáira vonatkozik, ugyanis az összes küldött jelnek és a vevő szenzorok szűrőbeállításainak is megegyező frekvenciájúnak kell lenniük, hogy a rendszer megfelelően működjön. Ehhez a rendszer dokumentációja [25] a 31 KHz-es frekvenciatartományt ajánlja alapértelmezetten.



23. ábra (a, b). Egy NIA-t használó rendszer egyik referenciapontként funkcionáló (a) és egyik sündisznó módban futó (b) beaconje, a pálya vonalainak rögzítése közben (3.2).

Az invertálható architektúrában (IA) a sündisznó módban lévő rendszeregységek veszik a fix pontként viselkedő rendszeregység által adott jeleket. Az invertálhatóság itt is a frekvenciákra tartozik, ugyanis a jeladók különböző frekvenciájú jeleket adnak, így tudnak egyszerre jelütkezés nélkül is jelet adni. Ha több lenne az adókból, mint ahány támogatott frekvenciatartomány, akkor a szenzoregységek TDMA módszer szerinti késleltetéssel vagy más, azonosító tartalommal bíró kód jittereléssel azonosíthatóak. Ezen jelek utazási idejéből szintén meghatározhatóak a szenzorok közötti távolságok, amikből a pozíciók is. A 19 KHz-es, 25 KHz-es, 31 KHz-es, 37 KHz-es és 45 KHz-es frekvenciák körüli tartományok támogatottak a jelküldésre.

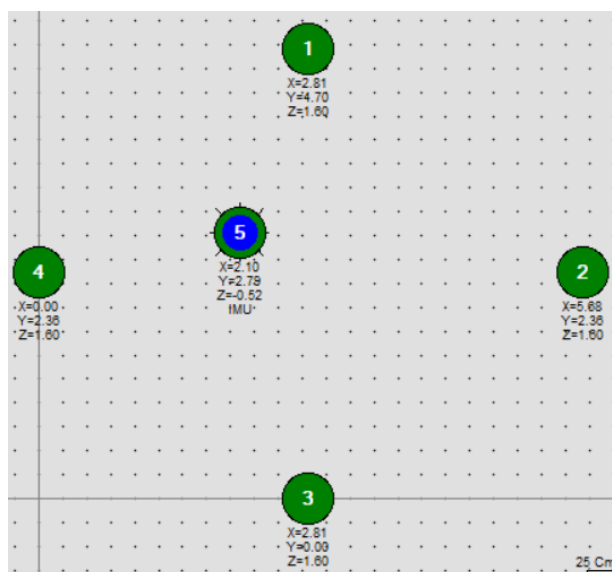
2.5.8. A rendszer felállítása

A rendszer egyszerűen felállítható. Először a hardverekre fel kell telepítenünk a megfelelő és általunk használni kívánt architektúrához tartozó firmwaret. Ezután a nem sündisznó módban működő rendszerelemeket rögzíteni kell a megfigyelni kívánt tér körül (24. ábra).



24. ábra. A menetpálya és a Marvelmind szenzorrendszer egységei felállítva a pálya körül.

Az ezek közötti távolságot rögzíteni kell a rendszer programja segítségével a modem adatbázisában (25. ábra). Ezen értékek meghatározásához a mérőszalaggal történő lemérésen kívül maga a rendszer kezelőfelülete is lehetőséget biztosít. A szenzorokon lévő öt jeladó és vevő ki- és bekapcsolható a beállítási szakaszban, így lehetőség van a rendszer távolságmérő funkciójának a használatára is, két szenzor egymás melletti, vagy egymással szembeni távolságának le-
mérésére.



25. ábra. A 24. ábra látható pályát körülvevő szenzorrendszer koordinátarendszere.

A felállított rendszerben működő modem és sündisznók kimeneti soros portjáról lekérdezhetőek a rendszer adatai. A lekérdezés megkönnyítését a fejlesztők által szolgáltatott API és mintakódok is segítik. Ezek segítséget nyújtanak saját API, vagy a biztosított API-t és a rendszer adatait felhasználó, feldolgozó szoftver fejlesztésében is [27].

2.6. Hibaterjedés

A Marvelmind rendszert első alkalommal úgy teszteltük, hogy egy alapszintű próba konfigurációt telepítettünk ki a gyártó által kiadott felhasználói kézikönyv alapján. Első benyomásunk az volt, hogy ez egy nagyon gyorsan kitelepíthető és felkonfigurálható rendszer, azonban egyre gyakrabban kaptunk hibás pozíciókat. Mivel eleddig nem találkoztunk hasonló munkával – ahol egy 1:16 –os méretű modell jármű pontos helyzetét határozták meg ultrahang alapú távolságméréssel –, ezért különböző ad-hoc hibakeresési módszereket alkalmaztunk.

A korábban feltett kérdést fogalmazzuk meg másképpen: Mekkora lesz a pozíció – x és y koordináták – hibája a lokalizációs folyamatban, ha a távolságmérés hibája $\pm 2\text{cm}$, valamint hogyan befolyásolja a koordináták hibáját a referenciapontok között megmért távolságok abszolút hibája? Nem lineáris rendszerek hibaterjedését a Gauss-hibaterjedési formulával írhatjuk le, ha mért mennyiségek normális eloszlást mutatnak:

$$s_f \approx \sqrt{\sum_i \left(\frac{\partial f}{\partial x_i}\right)^2 \cdot s_{x_i}^2}$$

ahol f a rendszer működését leíró függvény, x_i az f függvény i . paramétere, s_{x_i} az x_i paramétert terhelő hiba, végül s_f a rendszer hibája.

Mint az korábban kifejtettük a rendszer a helyzetmeghatározást multilaterációval végzi. Adott U, r_1 és r_2 , referencia pontok között – automatikusan vagy manuálisan mért –, valamint a keregett pont kiszámításakor megmért távolságok ismeretében:

$$x = \frac{r_1^2 - r_2^2 + U^2}{2U}$$
$$y = \pm \sqrt{r_1^2 - x^2}$$

Ezen formulák parciális deriváltjai:

$$\frac{\partial}{\partial r_1} \frac{r_1^2 - r_2^2 + U^2}{2U} = \frac{r_1}{U}$$
$$\frac{\partial}{\partial r_2} \frac{r_1^2 - r_2^2 + U^2}{2U} = \frac{-r_2}{U}$$
$$\frac{\partial}{\partial U} \frac{r_1^2 - r_2^2 + U^2}{2U} = \frac{U^2 - r_1^2 + r_2^2}{2U^2}$$

$$\frac{\partial}{\partial r_1} \sqrt{r_1^2 - x^2} = \frac{r_1}{\sqrt{r_1^2 - x^2}}$$

$$\frac{\partial}{\partial x} \sqrt{r_1^2 - x^2} = \frac{-x}{\sqrt{r_1^2 - x^2}}$$

behelyettesítve a gauss-hibaterjedési formulába a következő egyenleteket kapjuk:

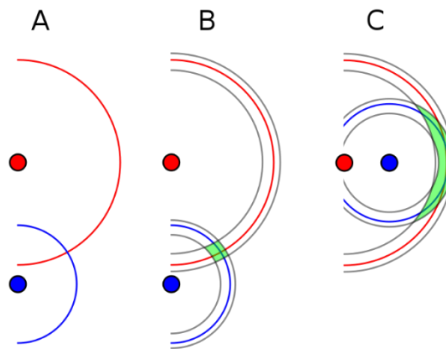
$$\mu_x = \sqrt{\left(\frac{r_1}{U^2}\right)^2 \cdot \mu_r^2 + \left(\frac{-r_2}{U}\right)^2 \cdot \mu_r^2 + \left(\frac{U^2 - r_1^2 + r_2^2}{2U^2}\right)^2 \cdot \mu_U^2}$$

$$\mu_y = \sqrt{\frac{x^2}{r_1^2 - x^2} \cdot \mu_x^2 + \frac{r_1^2}{r_1^2 - x^2} \cdot \mu_r^2}$$

ahol μ_x, μ_y az x és y koordináták, valamint μ_r az r_1 és r_2 távolságok abszolút hibája. Mi a μ_U referencia pontok között mért távolság hibájának μ_x, μ_y változókra gyakorolt hatását vizsgáltuk a fenti képletek segítségével.

2.7. Pontosság-hígulás

Geodéziában, főleg a globális helyzetmeghatározásban ismert fogalom és jelenség a pontosság-hígulás, vagy DOP (dilution of precision). A hibaterjedés megnyilvánulása térbeli pontok mérésekor. A térbeli vagy síkbeli pont meghatározásánál vett referenciapontok geometriai elrendezése és viszonya a mérendő ponttól nagyban befolyásolja a meghatározott pont koordinátáinak pontosságát.



26. ábra. A pontosság-hígulás oka. [28]

A pontosság-hígulás okát a 26. ábra szemlélteti. Az ábrán a piros és kék körívek sugara egyenlő a mért távolságokkal. Az őket közbezáró szürke ívek jelentik a távolság mérés abszolút hibahatárát. A 2.5.3 fejezetben leírt multilateráció a geometriai ívmetszésen alapul. Belátható, hogy

kedvezőtlen elrendezés esetén az ívek nagyon lapos szögben metszik egymást [29], emiatt kis hiba a távolságmérésben is nagy hatást gyakorol a mérésre.

A DOP értékek fejezik ki a helyzethiba és a műhold-vevő távolsághibájának (URE, azaz User Range Error) arányát [29]:

$$helyzethiba = DOP \cdot URE$$

Tehát alacsony DOP érték jelent kedvező mérési viszonyokat.

Különböző DOP értéke előre meghatározható matematikai úton. Az eljáráshoz szükséges meghatározni a mérendő pontból a referencia pontok irányába mutató egységvektorokat:

$$\left(\frac{x_i - x}{||x_i - x||}, \frac{y_i - y}{||y_i - y||}, \frac{z_i - z}{||z_i - z||} \right)$$

ahol x_i, y_i, z_i az i -edik referenciapont x, y, z pedig a mérési pont koordinátái. Ezekből képezzük a pontmeghatározás alakmátrixát:

$$A = \begin{bmatrix} \frac{x_1 - x}{||x_1 - x||} & \frac{y_1 - y}{||y_1 - y||} & \frac{z_1 - z}{||z_1 - z||} & -1 \\ \frac{x_2 - x}{||x_2 - x||} & \frac{y_2 - y}{||y_2 - y||} & \frac{z_2 - z}{||z_2 - z||} & -1 \\ \vdots & \vdots & \vdots & \vdots \\ \frac{x_n - x}{||x_n - x||} & \frac{y_n - y}{||y_n - y||} & \frac{z_n - z}{||z_n - z||} & -1 \end{bmatrix}$$

Ebből azután meghatározzuk a $Q = (A^T A)^{-1}$ kovariancia mátrixot.

$$Q = \begin{bmatrix} \sigma_x^2 & \sigma_{xy} & \sigma_{xz} & \sigma_{xt} \\ \sigma_{yx} & \sigma_y^2 & \sigma_{yz} & \sigma_{yt} \\ \sigma_{zx} & \sigma_{zy} & \sigma_z^2 & \sigma_{zt} \\ \sigma_{tx} & \sigma_{ty} & \sigma_{tz} & \sigma_t^2 \end{bmatrix}$$

Amiből a következőképpen kapjuk meg a különböző DOP értékeket:

$$\begin{aligned} GDOP &= \sqrt{\sigma_x^2 + \sigma_y^2 + \sigma_z^2 + \sigma_t^2} \\ PDOP &= \sqrt{\sigma_x^2 + \sigma_y^2 + \sigma_z^2} \\ TDOP &= \sqrt{\sigma_t^2} \end{aligned}$$

ahol PDOP a térbeli helyzetre, TDOP pedig az időmeghatározásra vonatkozó pontosságígulási érték. A GDOP együttesen fejezi ki mind a két DOP értéket.

2.8. Páros sündisznó üzemmód

A Marvelmind rendszerben implementálták a mobil jeladóknak egy olyan üzemmódot, amikor is két jeladó szerelhető egyazon járműre. Ebben az üzemmódban a jeladók között UART1 összeköttetést kell kialakítani. A járművön a jeladókat haladási iránnyal merőlegesen kell elhelyezni anélkül, hogy tengelyeik mentén elforgatnánk őket. A két jeladó koordinátájából meghatározhatjuk az őket összekötő szakasz egységnyi normálvektorát. A normálvektor iránya ekkor megfeleltethető a jármű haladási irányával.

Legyen A és B két pont, a jeladók pozíciói. Ekkor:

$$\Delta v = \begin{pmatrix} B_x - A_x \\ B_y - A_y \end{pmatrix}$$

$$\hat{n} = \begin{pmatrix} \frac{-\Delta v_y}{\|\Delta v\|} \\ \frac{\Delta v_x}{\|\Delta v\|} \end{pmatrix}$$

ahol $\hat{n}$ az AB szakasz egységnyi normálvektora. Az irány ebből meghatározni magától értendő, csak a koordináták hányadosának arkusztangensét:

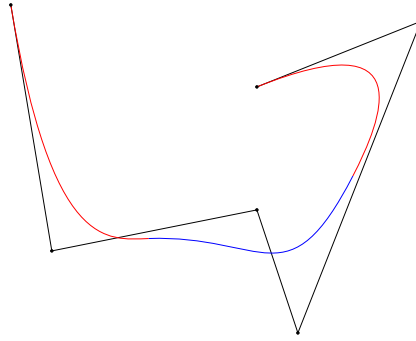
$$\theta = \tan^{-1} \left(\frac{\hat{n}_y}{\hat{n}_x} \right)$$

Ezt a számítást a Marvelmind rendszer elvégzi, és a pozíció mellett kiolvasható az elfordulási szög is belőle.

2.9. B-spline alapú interpoláció

Az n -ed rendű b-spline, vagyis basis spline, egy olyan függvény, amely $n - 1$ -ed fokú szakaszpolinomokból áll össze. Ezen polinomok érintkezéseit csomóknak (knot) nevezzük. Fontos tulajdonságuk, hogy deriváltak folytonos lehet a csomók számától függően. A csomók helyét megadó pontokat csomópontoknak nevezzük.

¹ Universal Asynchronous Receiver-Transmitter: aszinkron soros kommunikációt megvalósító hardver.



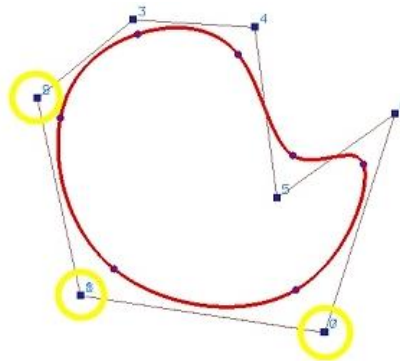
27. ábra. Egyszerű b-spline görbe [30].

Az n -ed rendű b-splineok bázisfüggvényei, azonos rendű és azonos csomópontok alapján meghatározható spline függvényeknek, így minden ilyen függvény előállítható b-splineok lineáris kombinációjaként. Ezek formaoptimalizációs és interpolációs eljárásoknál jól használhatóak, mivel az így előállítható függvények segítségével görbékét írhatunk le (27. ábra). Ezek megadhatóak a szakasz-polinomokat befolyásoló, úgynevezett kontroll pontok lineáris kombinációjának és b-spline bázisfüggvények együttesével. Ha ezt a görbét $C(u)$ -nak nevezzük el, akkor a

$$C(u) = \sum_{i=0}^n N_{i,p}(u) * P_i$$

függvény írja le a görbét, ahol adott:

- $n + 1$ a kontroll pontok, $m + 1$ a csomópontok, p a használt b-spline függvények foka,
- $n + 1$ kontroll pont $P_0, P_1, \dots, P_n$,
- $m + 1$ csomópontot tartalmazó vektor $U = \{u_0, u_1, \dots, u_m\}$,
- Az U vektoron értelmezett p -ed fokú B-spline bázisfüggvények $N_{i,p}(u)$,
- és $m = n + p + 1$ teljesül [31].

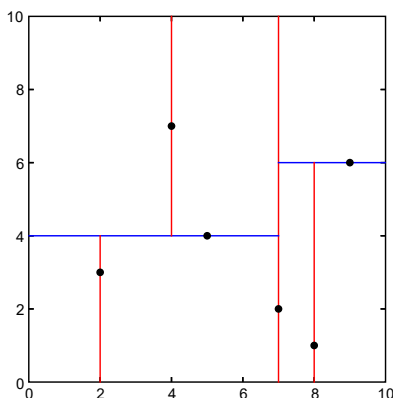


28. ábra. Zárt b-spline görbe. [32]

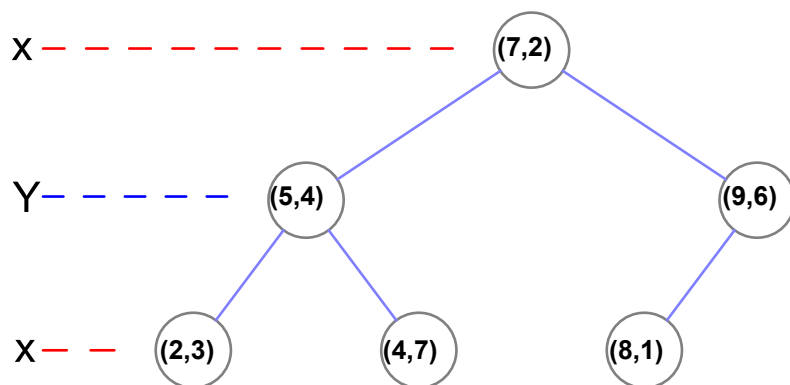
Mi ilyen függvények segítségével (3.2) zárt görbeillesztést hajtottunk végre. Ennek a legegyszerűbb módja, hogy ha a legutolsó p (ami a felépítéshez használt b-spline függvények fokszáma) kontrolpontot egyenlővé tesszük a legelső p kontrolponttal, ez ugyanis egy zárt b-spline görbét eredményez (28. ábra) [32].

2.10.k-D fa

A k-D fa egy olyan bináris fa, aminek minden csúcsa egy k-dimenziós pont. Minden nem levél csúcsot lehet egy olyan hipersíkként értelmezni és ábrázolni, ami két zárt féltérre bontja az adott teret, ahol a csúcs bal gyermekei a baloldali, míg a jobb gyermekei a jobboldali féltérben helyezkednek el. Minden egyes csúcs által elvégzett térszeleteléshez mindig egy dimenziót rendelünk hozzá a k-ból, amelyik dimenzióra merőleges. Így, ha a fa csúcsait ábrázoljuk, akkor a közöttük lévő leszármazási kapcsolatokat mindig, az egyik koordinátarendszere merőleges egyenesként ábrázolhatjuk, ahol minden, ezen egyenes által felvett értéknél kisebb k. (pl. x) dimenziós értékkel rendelkező pont, az egyenes egyik, és minden ennél nagyobb értéket felvevő pont az egyenes másik oldalán helyezkedik el, míg maga a csúcs az egyenes része [33].



29. ábra. 2-D fa ábrázolva a síkban [34].

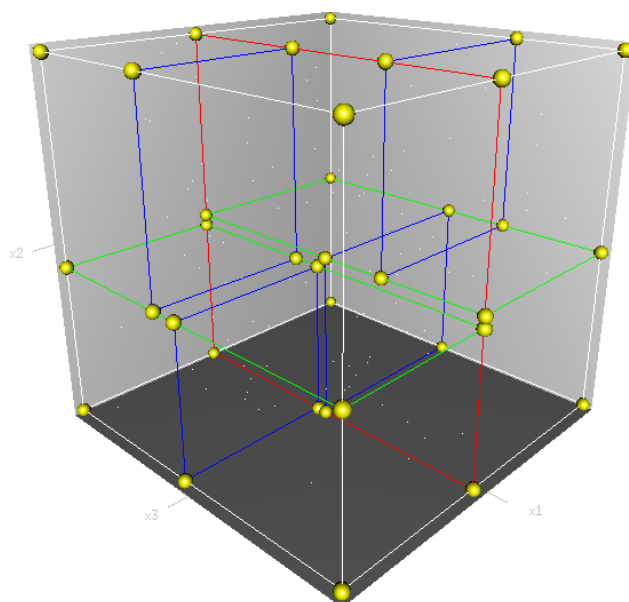


30. ábra. 2-D fa bináris faként ábrázolva [34].

A 29. ábra és 30. ábra bemutatják, hogy hogyan lehet ábrázolni és eltárolni egy $k = 2$, azaz 2-D fát, ha azt a (2,3), (5,4), (9,6), (4,7), (8,1), (7,2) pontok alapján építjük fel.

Egy k -D fa létrehozásának időkomplexitása $O(n)$. A beszúrás, törlés és keresés átlagos időkomplexitása pedig $O(\log n)$, ami legrosszabb esetben $O(n)$ futásidőig romolhat. Ez akkor valószínű, ha fán belül felvett pontok nem közel azonosan osztják fel a teret. Mivel a pontok közötti távolság nem előírt, így szélsőséges esetben előfordulhat, hogy a k -dimenziós tér egyik, kis alterében van a pontok jelentős része, és így a fában történő reprezentációjuk mélysége is az n -hez közeli értéket fog felvenni. Ez természetesen növeli a műveletek elvégzéséhez szükséges időt, de a felvett pontok megfelelő eloszlása esetén ez elkerülhető.

Az 31. ábra pedig egy megfelelően szétszórt értékekkel felépített 3-D fát ábrázol.



31. ábra. 3-D fa [34].

A k -D fákon belül gyorsan kereshető a legközelebbi szomszéd, legyen a viszonyított pont akár a fa része akár nem, mivel a fa maga egy k dimenziós térben értelmezett, így az adott térben bármilyen felvehető két adat közötti távolság értelmezhető. A fejlesztett programjainkban (3.4.1, 3.7) a „`scipy.spatial.KDTree`” k -D fa implementációt használjuk [35].

2.11. Python/C API

A Python/C API [36] lehetővé teszi, hogy C-s kódokban hozzunk létre és használjunk Python típusokat és függvényeket. Ennek használatára egy `Python.h` header meghívásával a c-s kódkunkból az alábbi módon:

```
#include <Python.h>
```

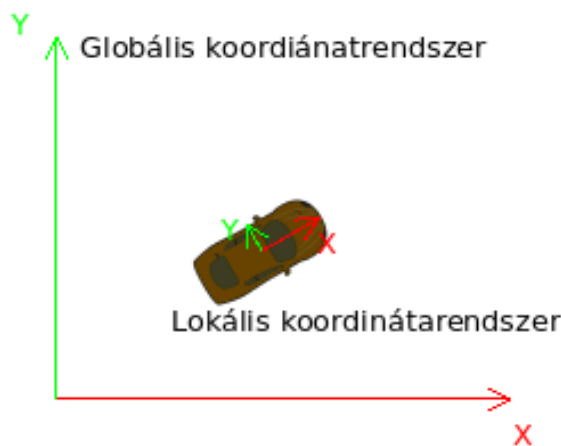
Ez olyan c-s könyvtárak létrehozását teszi lehetővé, ami Pythonba importálható és közvetlen hívható, így teljes értékű Python könyvtárat hozhatunk létre C-s kódokból és futtatható forrásállományokból. Ezekhez teljeskörű csomagtelepítési és törlési támogatást kapunk, aminek kihasználásához megfelelően definiált C-ben megírt függvényekhez és egy megfelelően megírt setup.py fájl megírására van szükség. A könyvtár függvényei a pip-pel történő telepítés után bárholnnan hivatkozhatóvá válnak a Python interpreter számára. A telepítés alatt a forrásállományok teljes értékű gépi kóddá fordulnak le, így a C-ben megírt kódok ténylegesen megőrzik a gyorsaságukat. Ez több, és kicsit bonyolultabb, lassabb C-s kódokat eredményez, mintha csak a leghatékonyabb, C kódolási paradigmák alapján írnánk meg a kódot, mivel a visszatérési értékeinket a C-s oldalon kell felépítenünk Python Objektum típusú adatként. Viszont biztosítja, hogy a visszaadott értékeink a Python számára konvertálás nélkül felhasználhatóak legyenek, ami egy sokkal tisztább Python kódot tesz lehetővé, mintha valamilyen memóriakapcsolási eljárás, például CType lenne használva ugyanerre a feladatra.

3. Készített szoftverek bemutatása

3.1. Marvelmind szenzoradatok rögzítése

A szenzor által vett adatok rögzítése egy, a munkánk során többször is használt, alapvető feladat. Ennek a legalapvetőbb megoldását egy Pythonban megírt szoftverben implementáltuk, ami a Marvelmind eredetileg is biztosított Python API-ján keresztül kommunikál [27].

Ez egy „MarvelmindHedge” nevű osztály létrehozását teszi lehetővé, ami folytonosan eltárolja a sündisznó módban levő szenzoregységgel, vagy a szenzorrendszer modemével, összekapcsolt soros porton beérkező adatokat egy kétvégű sorban. Ezek a rendszer beállításától függően tartalmazhatják a szenzoregységek helyzetét, a szenzoregységek között mért távolságot, a rendszer telemetriai és pontossági értékeit. Ez sündisznó módban lévő szenzoregységek esetén kiegészülhet azok nyers IMU adataival, vagy a pozíciós és IMU adatok értékegyesítésével megkapott, a rendszer globális koordináta-rendszerben értelmezett, elmozdulási vektorokat is (ezek különbözhetnek az IMU szenzortól alapból megkapott, a szenzor saját lokális koordináta-rendszerében értelmezett elmozdulási vektoroktól, 32. ábra).



32. ábra. A szenzoregység lokális és globális koordináta-rendszere közötti különbség.

Ezeket a beérkező bináris adatokat a szoftver lefordítja és eltárolja, ezzel egyszerűen lekérdezhetővé téve azt az általunk megírt többi folyamata számára. Ezen adatok közül a használt szoftvernek csak az előre beállított azonosítójú sündisznó x, y és z koordinátájára van szüksége, így azokat kérdezi le egy, leállítási parancsra leálló (pl. q billentyű vagy megszakítási parancs kiadása esetén), végtelen ciklusban. Minden egyes iterációban a lekérdezett számhármast összehasonlítja az előző számhármassal, és ha teljesen különböző adatokról van szó, akkor kiírja és

eltárolja ezeket az értékeket egy „.csv” kiterjesztésű fájlba, majd frissíti a legutóbbi adatot tartalmazó változót. Ha az értékek megegyeznek, akkor nem csinál semmit se az adott ciklusiterációban. A programnak parancssori argumentumként megadható a sündisznó azonosítója, egy „--addr” kapcsolóval, és a kimeneti adat azonosítására használt név is a „--map” és „--line” kapcsolókkal.

Ezt a lekérdezési és mentési folyamatot egy gomb lenyomása esetén hajtja végre a program, és ismétli egészen a gomb felengedéséig. Azért ezt a rögzítési módot választottuk, hogy a rögzítést végző személynek legyen tetszőleges ideje bemenni a pályára, arrább tenni a szenzort, kimenni a pályáról és rögzíteni a szenzor helyzetét minden egyes adatrögzítés között. Erre azért van szükség, mert a szenzorrendszer pontosságának romlásához vezethet, ha egy magas tárgy vagy személy leárnyékolja a sündisznó módban lévő egységet. Ha pedig korlátozzuk a mozgathoz rendelkezésre álló időt, az kapkodáshoz, és így gyengébb minőségű adatok rögzítéséhez vezethet. Magát a programot csak a fizikai pályánk vonalainak rögzítésére használjuk, mivel a többi adat lekérdezési feladat már másik program alapján működik (3.3.), így az „--map” és „--line” kapcsolók használhatóak az épp rögzített pálya és annak vonalának (INNER, MIDDLE, OUTER, azaz belső, középső és külső sávelválasztó vonal) azonosítására.

3.2. Pályadigitalizálás

A kocsni által bejárando pályát 1x1 m-es szivacsokból szereljük össze. Ennek 3 vonalán pontokat veszünk fel az előző program (3.1.) segítségével. A pályadigitalizáló program ezen elmentett három vonal koordinátáinak x és y helyzetét olvassa be három „.csv” kiterjesztésű fájlból, amiket három kétdimenziós tömbként tölt be a programunk, ahol minden sor egy pont x és y koordinátáját tartalmazza. A program a pontok közül kiszűri az összes ismétlődő elemet, így minden pozíció csak egyszer fog szerepelni az újonnan előállított tömbünkben. Majd ezeket, a csak egyedi elemeket tartalmazó tömböket, sorba rendezi bejárasi irány szerint. A folyamat ehhez egy legközelebbi szomszédkeresésen alapuló sorba rendezést használ. A pontok legközelebbi szomszédját euklidészi távolság alapján határozzuk meg (1. algoritmus).

Algorithm 1: EuclideanDistance (P_a, P_b)**Input:** pont $P_a(a_x, a_y)$, pont $P_b(b_x, b_y)$ **Output:** P_a és P_b pont közötti euklidészi távolság P_T

1 $Táv_x \leftarrow (a_x - b_x)^2$

2 $Táv_y \leftarrow (a_y - b_y)^2$

3 $P_T \leftarrow \sqrt{Táv_x + Táv_y}$

4 **return** P_T

1. algoritmus. Két pont közötti euklidészi távolság.

Ez amiatt is jó, mert a sorba rendezésen túl kiszűri azokat az adatokat is, amik egyetlen másik pontnak sem a legközelebbi szomszédjai, így a később elvégzett interpoláció is figyelmen kívül hagyja ezeket a pontokat, amik nagy valószínűséggel mérési hibákat tartalmaznak.

A szoftvert az előkészített adathalmazokon egy-egy B-Spline interpolációt hajt végre, amik három zárt görbét generálnak. Ezen három görbéből milliméterenként mintavételezünk pontokat, amiket numpy tömbökként mentünk le (ezen lementett adatokra a későbbiekben többször is „térképként” hivatkozunk). A B-Spline algoritmus simasági együtthatóját az első parancssori argumentumban adhatjuk meg a programunknak. Ez befolyásolja, hogy melyik adatpontokat érinti ténylegesen az interpolált görbe (amit nem minden pont esetén akarjuk, a rendszerben szereplő zaj miatt), és a görbe fokszáma is hatással van. Minél kisebb ez az érték, annál kevésbé lesz sima a görbénk, azaz annál jobban akar az adathalmaz összes pontjához illeszkedni az interpolált görbe. Így egy kellően alacsony, de a görbét nem túlillesztő együtthatót kell keresnünk, minden különböző pályaadat interpolálása esetén. Ezt az értéket az interpolálást elvégzettetni akaró felhasználó befolyásolhatja a programunkban, mivel a simasági együttható optimális értéke $[m - \sqrt{2 * m}, m + \sqrt{2 * m}]$ között van, ahol m az interpoláció során használt pontok száma. A B-spline interpoláció elvégzéséhez a *scipy.interpolate* könyvtár implementációját használjuk, azon belül is az *splprep* [37] és *splev* [38] függvényeket.

Az interpolálás elvégzésével a felvett adathalmaz pontjai, és az interpolált zárt görbe is megjelenítésre kerül egy, a szenzorrendszer koordináta-rendszerében értelmezett ábrán, így grafikusan is ellenőrizhető az interpolált görbe és a pont-halmaz kapcsolata.

A három görbét még egy „.svg” kiterjesztésű fájlba is elmentjük. Ezt azért tesszük meg, mert így könnyen integrálhatóvá válik maga a pálya más, két és háromdimenziós szoftverekbe is.

Ezt használták például a 3.4.2. fejezetben említett valós idejű háromdimenziós virtuális megjelenítési környezetben, a pálya háromdimenziós modelljének elkészítéséhez.

3.3. Marvelmind szenzorok integrálása a Donkey® platformba

Mint ahogy azt már a 2.5.8. és 3.1. fejezetekben említettük, a Marvelmind Robotics beltéri helymeghatározó szenzorrendszeréhez a gyártó biztosít API-t C, Python, Java és Arduino C nyelven is. Ezekhez biztosít példa kliens alkalmazásokat is, melyek bemutatják, hogyan lehet a rendszert vezérelni, és abból adatokat kiolvasni. Mivel a Donkey platform maga Python nyelvet használ, ezért egyértelmű, hogy nekünk a Python nyelvű környezetben kell elérnünk a szenzor adatait, amit a Pythonos API-t [27] használva megtehetünk, de megeshet, hogy nem az elsőre kézenfekvő megoldás a jobb megoldás. Mi ennek a megoldására egy C és Python hibrid API integráció megírását választottuk. Ennek okait és az implementációnk működését a következő bekezdésekben részletezzük.

A Marvelmind Python API-ját alaposabban elemezve előjön több gyengesége is. Először is, az adatokat nem folytonosan, bájtanként olvassa be és értelmezi, mint a C nyelvű API, hanem már a fentebb (3.1.) tárgyalt módon egy kétvégű sorba tárolja el, amit állandóan végig ellenőriz, és mintaillesztés alapján próbálja meg beazonosítani a megkapott adatokat. Ez sebességben lassabb az beérkező adatok folytonos feldolgozásnál, hiszen bármilyen fajta feldolgozás megkezdése előtt az egész adatsomag beérkezését meg kell várnia a szoftvernek, és állandóan ellenőriznie kell az egész puffert, mintákat keresve abban.

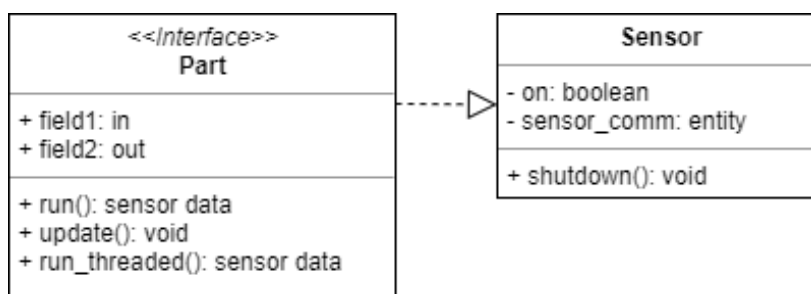
Továbbá, a Python API megvalósítása egy, multithreadinget nem támogató, Python thread típusú osztályban lett megvalósítva, ami elsőre még jól is hangzik, de a CPythonban működő Python Interpreter hiába kezeli ezt külön folyamatszálként, mivel benne működik egy úgynevezett „Global Interpreter Lock” (GIL), ami megakadályozza a konkurens szál futtatást, és mivel ő maga csak egy szálon fut, nem tudja kiszervezni a különböző logikai folyamatokat tényleges fizikai szálakra.

Ezeket a dolgokat úgy változtattunk, hogy megírtunk egy olyan C-s könyvtárat, ami a Marvelmind C API-t használva kommunikál a szenzoregységek soros portjaival, amivel egy gyorsabb adatelérést kapunk már az implementáció mögötti logika miatt is. Ezen persze tovább gyorsíthat az, hogy a C fordító általában gyorsabban végrehajtható gépi kódokat eredményez, mint a Python Interpreter által értelmezett kódok, és hogy megfelelő odafigyeléssel C nyelvben telje-

sítményhatékonyabb szoftverek állíthatók elő. A megírt könyvtár Python kódokból hívhatósá-
gát a Python/C API biztosítja (2.11). A megfelelő működés érdekében a C-ben deklarált függ-
vényekhez kell egy kiegészítő „PyMethodDef” típusú struktúratömböt és egy „PyMODI-
NIT_FUNC” típusú függvényt is deklarálni, de ezekért cserébe teljes körű Python támogatást
és hívhatóságot kapunk.

A Marvelmind C API esetében az adatlekérés egy külön pthread-en történik, így az ténylegesen
egy másik fizikai szála kerül kiszervezésre. A működés elkezdéséhez inicializálnunk kell egy
„MarvelmindHedge” struktúrát, ami adatpufferként és az elindított pthread szál hivatkozási
pontjaként is funkcionál. Az adatlekérő függvények ezen a struktúrán keresztül kéri le az ál-
taluk visszaadandó adatokat az eszköz memóriájából, és ezentúl biztosítják, hogy észlelhetően
hibás adatok esetén Python kivételeket dobjanak vissza az őket hívó programoknak.

Donkey platform alatt az egyes szoftveres és hardveres elemek – és itt most elsősorban autóra
épített szenzorokra gondolunk – integrálását úgynevezett „part” osztályok implementálásával
lehet megoldani. Egy part objektum tárolhatja az adott szenzor állapotát és biztosít interfészt új
adatok beolvasására, lekérdezésére, valamint az eszköz leállítására. Az autó rutinjának főciklu-
sában – ahol a szenzoradatok és vezérlő utasítások beolvasása, és a vezérlés kiadása a hatjómű-
veknek történik felváltva – az egyes part objektumok lekérdező metódusainak hívása külön
logikai szálakban történik. Mivel a valós lekérdezést az előbb említett módon kiszerveztük egy
fizikailag is külön szála, így a vezérlő szoftver valós időben képes a beolvasott adatokat feldol-
gozni. Ezzel képessé vált a főfolyamat reagálni a szenzor és vezérlő jelekre is egyszerre, tehát
nem veszítjük el az irányítást az autó felett, miközben egy időigényesebb adatfeldolgozás vagy
lekérés zajlik.



33. ábra. Egy szenzor part objektum osztálya. Megvalósítja a Part interfészt. Valójában nincs
ilyen interfész definiálva a Donkey platformban, ez az ábra csak szemléltetési céllal készült.

Algorithm 2: SensorConstructor ()

Input: comm_interface a szenzor és a főrutin közötti kommunikációs interfész objektuma

```
1 sensor_comm ← comm_interface
2 on ← true
3 sensor_comm.start()
```

2. algoritmus. A `Sensor` osztály konstruktora.

Algorithm 3: Run ()

Output: A szenzorból kiolvasott adatok

```
1 data ← Szenzor adat(ok) kiolvasása
2 data ← Adatok átkonvertálása
3 data ← Adatok szűrése
4 return data
```

3. algoritmus. A `Run` metódus kiolvassa, majd feldolgozza a szenzortól kapott adatokat.

Algorithm 4: Update ()

```
1 while on do
2     out ← Run()
3 end
```

4. algoritmus. Az `Update` metódus külön szálon fut és mindig aktualizálja az `out` attribútumot.

Algorithm 5: RunThreaded ()

Output: Az objektum `out` attribútuma

```
1 return out
```

5. algoritmus. A `RunThreaded` metódus akkor hívódik meg, ha az `update` külön szálon fut.

Algorithm 6: Shutdown ()

```
1 on ← false
2 sensor_comm.stop()
```

6. algoritmus. A `Shutdown` metódus egyetlen feladata, hogy az `on` attribútumot hamis értékre állítsa és a kommunikációt a szenzorral megszüntesse.

A 33. ábra és a 2. algoritmus, 3. algoritmus, 4. algoritmus, 5. algoritmus és 6. algoritmusok egy Part interfész konkretizálására adnak példát. Megmutatják a szenzor integrálásának szoftver oldali megvalósítását. Az `Update()` metódus egy külön számban fut, míg az autó főciklusa ismételten hívja a `RunThreaded()` függvényt.

A szoftveres integrációt a `setuptools` eszközzel egy állományba csomagoltuk, ezáltal a `pip` segítségével telepíthető az autón. Ennek előnye, hogy a további változtatások és javítások aktualizálása egy újratelepítési folyamattal elvégezhető. Ezáltal ennek az integrációs résznek a karbantartása egyszerűbbé válik. A becsomagolás lényeges része a `setup.py` állomány, mely tartalmazza a C kódbázisra vonatkozó fordítási információkat, az esetleges függőségeket – esetünkben a Donkey platform verziójától való függés a releváns –, és az adott csomag verzióját és azonosítóját. Az integrálási folyamatba beletartozik az autó főprogramjának módosítása is. Mivel ezt más fejlesztői csapatok is módosítják integrációs céllal, nem jöhetett szóba annak lecserélése amikor a mi kódjainkat telepítjük. Ezért a főprogramhoz készítettünk egy patch állományt, amit felhasználva a `patch_mm.sh` Bash szkript felfrissíti a már meglévőt. A `patch_mm.sh` arra készült, hogy a Donkey platform kódbázisát befoltozza az szenzorintegráció megvalósításához. Manuálisan kell lefuttatni a telepítés után.

Sajnos a patch alapú verziófrissítés nem biztonságos, a változások automatikus visszavonása nem garantált, viszont a patch állományok szöveg alapúak és viszonylag kevés idő alatt manuálisan helyre lehet állítani az módosított kódbázist. Kutatjuk a lehetséges alternatívákat, amik kiküszöbölik ezt a hiányosságot. Egyelőre Donkey platform forkolása tűnik jó megoldásnak, amihez szükség lesz az autó fejlesztésével foglalkozó csapatok munkavégzését átszervezni.

3.4. Valós idejű pozíciómeghatározás

Ezeknél a szoftvereknél a hangsúly mindig az autó működése közben történő adatszolgáltatáson és számítások elvégzésén van, a többi szoftverrel ellentétben, ahol mindig előre rögzített adatok feldolgozása a cél. Ezeknél így figyelni kell a végrehajtásukhoz szükséges erőforrásokra, különösen, ha a kisautón akarjuk futtatni őket.

3.4.1. Sávon belüli pozíció meghatározása az autóágens számára

Ez a program a kocs számára határozza meg a saját pozíciója alapján, hogy a térkép haladási sávjának középpontjához képest hol helyezkedik el, azzal a céllal, hogy ezen adat segítségével meg tudja mondani, hogy esetlegesen a sávon belül, kívül, vagy esetleg épp a szélső vonalán

halad-e. Működési logikáját az 1. algoritmus, 7. algoritmus, 8. algoritmus és 9. algoritmusok írják le.

Algorithm 7: NearestNeighboursInCurve (C_{map} , P_{act})

Input: A térkép három görbéjének háromdimenziós listája $C_{map}(C_{out}, C_{mid}, C_{in})$,

pont(ok) listája $PL(P_0 [\dots, [P_i]])$, ($i = 0, 1, \dots, n - 1$), pontok száma $n \in \mathbb{N}$

Output: A PL lista pontjaihoz legközelebb eső pontok listája a C_{map} görbéin

$PL_{NN} (P_{NN_{out}}(P_0 [\dots, [P_i]]), P_{NN_{mid}}(P_0 [\dots, [P_i]]), P_{NN_{in}}(P_0 [\dots, [P_i]]))$,

($i = 0, 1, \dots, n - 1$)

```

1   $PL_{NN}$  legyen üres lista
2  for  $i = 0$  to  $n - 1$  do
3       $curveTree \leftarrow KDTree(C_{map}[i])$ 
4       $dists, indexes \leftarrow curveTree.query(PL)$ 
5       $P_{NN}$  legyen üres lista
6      for  $j = 0$  to  $n - 1$  do
7           $C_{map}[i, indexes[j]]$  hozzáfűz  $P_{NN}$ -hez
8      end
9       $P_{NN}$  hozzáfűz  $PL_{NN}$ -hez
10 end
11 return  $PL_{NN}$ 

```

7. algoritmus. Bizonyos pozíció(k) legközelebbi szomszédja(i) a térkép görbéin.

Algorithm 8: Midpoint (P_a , P_b)

Input: pont $P_a(a_x, a_y)$, pont $P_b(b_x, b_y)$

Output: P_a és P_b pont közötti felezőpont P_F

```

1   $Mid_x \leftarrow (a_x + b_x)/2$ 
2   $Mid_y \leftarrow (a_y + b_y)/2$ 
3  return  $P_F(Mid_x, Mid_y)$ 

```

8. algoritmus. Két pont közötti egyenes felezőpontja.

Algorithm 9: NormalisedLanePosition (P_a, P_b)

Input: A térkép három görbéjének háromdimenziós listája $C_{map}(C_{out}, C_{mid}, C_{in})$, autópozíciós pont(ok) listája $PL(P_0 [., \dots, [P_i]])$, ($i = 0, 1, \dots, n - 1$), autópozíciós pontok száma $n \in \mathbb{N}$

Output: A PL lista pontjainak normalizált távolsága a sáv a menetirány szerinti forgalmi sáv közepétől $NormDists_{pos}(P_0 [., \dots, [P_i]])$, ($i = 0, 1, \dots, n - 1$)

```
1  $P_r \leftarrow NearestNeighboursInCurve(C_{map}, PL)$ 
2  $NormDists_{pos}$  legyen üres lista
3 for  $i = 0$  to  $n - 1$  do
4   if  $EuclideanDistance(P_r[0, i], PL[i]) < EuclideanDistance(P_r[2, i], PL[i])$  then
5      $OutSide \leftarrow P_r[0, i]$ 
6   else
7      $OutSide \leftarrow P_r[2, i]$ 
8   end
9    $MidSide \leftarrow P_r[1, i]$ 
10   $P_{mid} \leftarrow Midpoint(OutSide, MidSide)$ 
11   $D_{norm} \leftarrow EuclideanDistance(P_{mid}, PL[i]) / EuclideanDistance(P_{mid}, MidSide)$ 
12   $D_{norm}$  hozzáfűz  $NormDists_{pos}$ -hoz
13 end
14 return  $NormDists_{pos}$ 
```

9. algoritmus. A pozíció(k) és sávközéppont(ok) közötti normalizált távolság(ok).

A folyamat egy „Track_Position” nevű Python osztályként lett implementálva, ami egy, a sávra merőleges vonalon értelmezett relatív helyzetet ad vissza, ha beadunk neki egy, a szenzorrendszer által definiált globális koordinátarendszerben értelmezett pozíciós értéket, ami legtöbb esetben a kocs beacon-je által szolgáltatott aktuális pozíció. A számítás elvégzéséhez meg kell határozni a térkép 3 vonalának a ponthoz legközelebb eső pontjait. Ezen három pont és az adott pozíció közötti távolságok alapján meghatározható a jelenlegi haladási sáv (abban haladok, amelynek szélső vonalához közelebb vagyok), annak a három legközelebbi pont között meghúzott egyenesen lévő középpontja, és a KRESZ által előírt haladási sáv középpontjának távolsága is a jelenlegi pozíciótól. Ezt a távolságot normalizálom úgy, hogy akkor vegye fel értéként az 1.0, ha a kocs kereke épp a sáv széléhez ér. Az így felvett értéknek a 0.0-hoz való

közelsége a jármű és a sáv középpontjának közelségét, valamint 1.0 feletti érték felvétele a sáv elhagyását jelenti. Ha a sáv középpontját a jármű ideális haladási helyének tekintem, míg, ha ezt más nem írja felül, a sáv elhagyását rossz viselkedésmódnak ítélem meg (sávtartás esetén). Így a kapott értékem az optimálistól való eltérés mértéke is egyben.

A legközelebbi pont megkereséséhez a térképet egy k-D fában mentem el, amiben nagyon hatékonyan, $\sim O(\log n)$ -es időkomplexitással lehet keresni a fában eltárolt ponthalmaz azon pontját, amely a legközelebbi egy általunk megadott másik ponthoz. A fent már említett (2.10) SciPy könyvtár *spatial.KDTree* osztálya, amelyiket kifejezetten a legközelebbi szomszéd keresési feladatok megoldására hoztak létre, egyszerre több megadott ponthoz tartozó legközelebbi pont keresését is lehetővé teszi, így sok feldolgozandó adat esetén még hatékonyabb tud lenni az algoritmus. Ezt a 3.5. fejezetben leírt programba ki is használtuk, így itt is ezen implementáció használatát választottuk. A térkép adatainak fává történő átalakítását viszont kiszerveztük az osztály inicializációs részébe, így azt csak egyszer kell elvégeznünk, így az elkészült fákat az osztály mezőiben eltárolva tudjuk felhasználni minden egyes érték lekérdező függvényhívás esetén a visszatérési érték meghatározásához.

Mivel ezt a számítást csak minden új GPS koordináta-hoz kell számolni, így a számítási igénye jobban kielégíthetőbb, mintha azt képenként kéne számolni. De amennyiben a GPS pozíciónk folytonosan becsülhetővé válik egy Kálmán szűrő alkalmazásával, akkor mindenképpen egy NVIDIA Jetson Nano lapka ajánlott, mind a Kálmán-szűrő és mind ennek a szoftvernek a működtetésére.

3.4.2. A jármű helyzetkoordinátájának megosztása más szolgáltatások számára

Ez a program egy http post üzenetben küldi el egy szerver számára a kisautó jelenlegi helyzetét, a sündisznó módban futó szenzoregység helyadatának beolvasása, és megfelelő formátumúvá történő konvertálása után. A szoftvernek két típusai is van. Az egyik megjeleníti az autó helyzetét egy kétdimenziós térképen az azt futtató számítógépen, a másik nem. Az előbbi a gazdagépen, míg az utóbbi a kisautón történő futtatásra lett tervezve.

Ez használható az autó reinforced learning alapú tanítására és vezetés alatti viselkedéselemzésre is, amit a 5.2. fejezetben tárgyalunk bővebben. A szolgáltatás használatára jó példa egy, a Debreceni Egyetemen más hallgatói által fejlesztett projekt, ami a kisautó háromdimenziós virtuális környezetbe helyezéséről szól. Ebben az eddig tárgyalt program küldi el egy szerverre a jármű

aktuális koordinátáját, ami alapján az leképezi a kisautó frissített helyzetét a virtuális környezetben is.

3.5. Hibaterjedés és pontosságígulás számítása

A hibaterjedés és a pontosságígulás számításai Jupyter jegyzetekben lettek lefuttatva. A jegyzet fájlok tartalmazzák a programkódokat, a legutolsó futás eredményét továbbá magyarázatot formázott szöveg, valamint képek formájában.

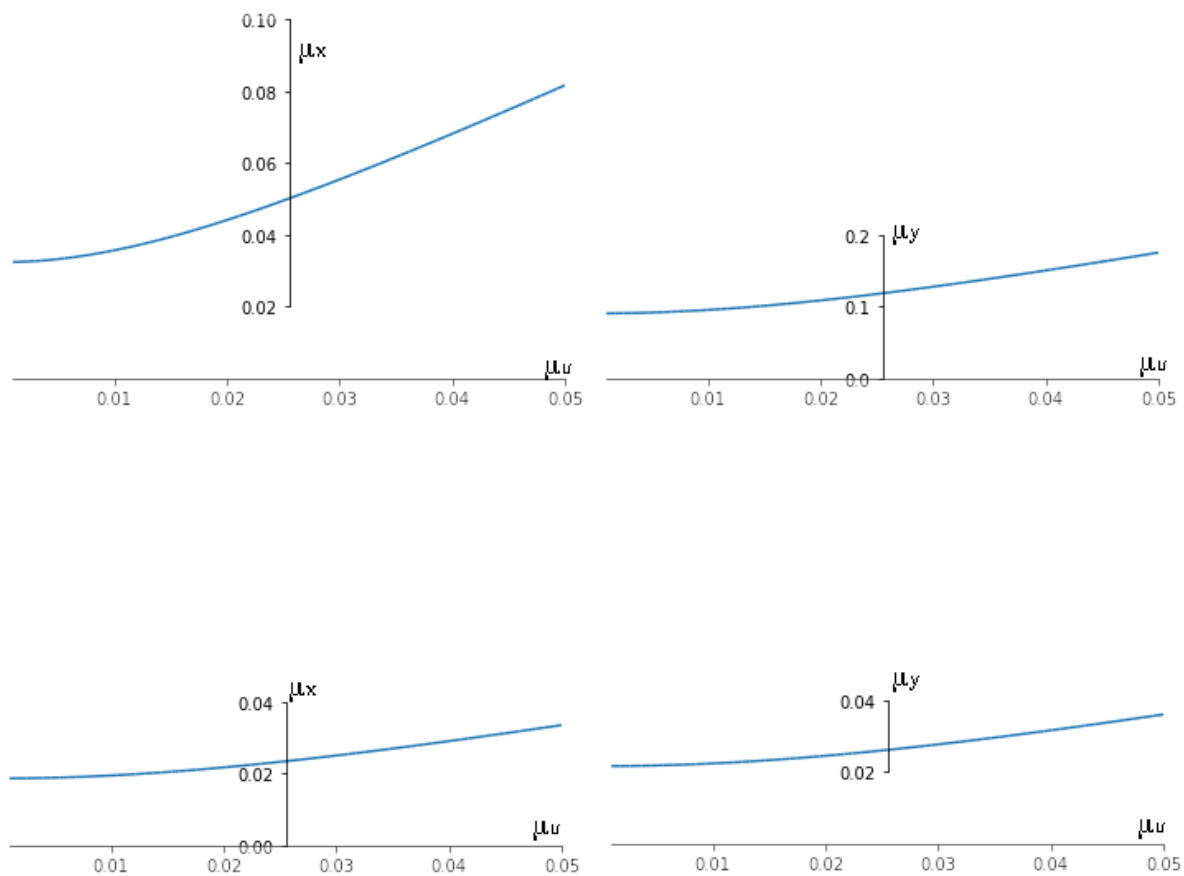
A hibaterjedés számításakor alkalmazott képletek SymPy nevű Python könyvtárát használtuk, ami szimbolikus matematikai számításokhoz lett kifejlesztve. A kiindulási egyenletekből és formulákból hatékonyan lehetett meghatározni a parciális deriváltakat, elvégezni behelyettesítéseket és az eredményül kapott egyenleteket grafikonját megjeleníteni. Külön jegyzet fájlban lett kidolgozva a hibaterjedés számítás 2, valamint 3 dimenziós multilaterációra. Az áttekinthetőbb programkódfejlesztés és futtatás érdekében az y koordináta számításának formulájában az x változó behelyettesítésre került, hogy a két koordinátát egymástól függetlenül tudjuk kiszámítani:

$$y = \pm \sqrt{r_1^2 - x^2} = \pm \sqrt{r_1^2 - \left(\frac{r_1^2 - r_2^2 + U^2}{2U} \right)^2}$$

Ennek a parciális deriváltjait és az azokból képzett hibaterjedés egyenletét az y koordinátára a SymPy segítségével határoztuk meg.

További előnye a SymPy-nak, hogy a tárolt formulákat a Jupyter képes nyomdai formátumban megjeleníteni, valamint Latex formátumként másolhatjuk ki azokat.

Két geometriai elrendezést vizsgáltunk meg. Az első esetében a rögzített változók következők voltak: $\mu_r = 0.02m$ $r_1 = 2.236m$ $r_2 = 6.083m$ $U = 4.0m$. A helyzethiba nagyságának változását μ_U függvényében a 34. ábra felső két grafikonja mutatja. Innen leolvasható, hogy még $U = 4.0m \pm 1mm$ mellett is az y koordinátát terhelő hiba közel 10cm.

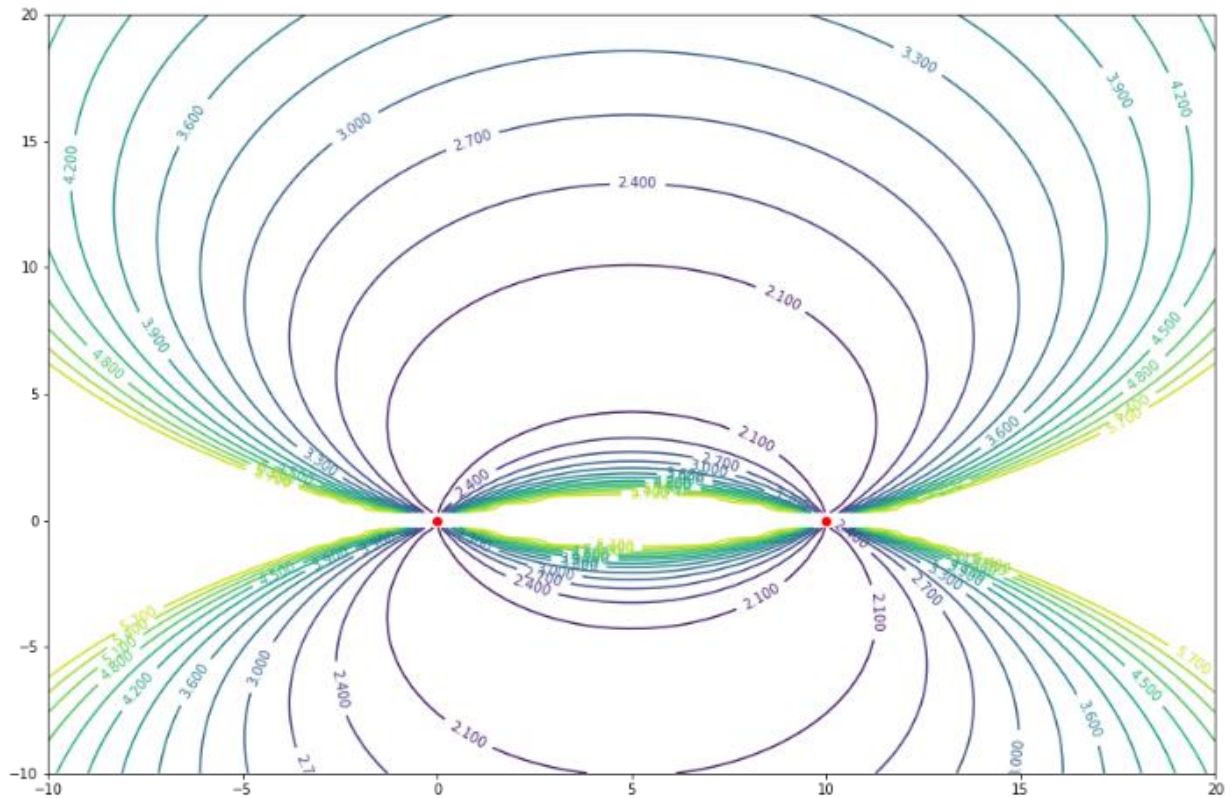


34. ábra U abszolút hibájának hatása az x és y koordináták hibájára. Felül egy rossz, míg alul egy jobb geometriai elrendezés mellett mutatja a helyzethibát az U hibájának függvényében.

A második esetben $\mu_r = 0.02m$ $r_1 = 2.463m$ $r_2 = 2.810m$ $U = 4.0m$ volt a paraméterek értéke. Ekkor a helyzethiba közel azonos volt a távolságmérés hibájával. Az U paraméter hibája csak kis mértékben befolyásolta ezt. Innen következtettünk arra, hogy más módszert kell keresnünk a helyzethiba meghatározására.

A pontossághígulás számítása legnagyobb mértékben mátrix műveleteken alapul. Ehhez a numpy könyvtár elegendő volt. Az ábrák létrehozására a matplotlib könyvtárat használtuk. Kiszámítottuk és kontúrdiagramon megjelenítettük, hogy a referencia pontokhoz viszonyított helyzetekben mekkora a várható PDOP érték. A 35. ábra piros pontokként lettek ábrázolva a referencia pontok. A diagramon látszik, hogy a PDOP – és így helyzethiba is – változik, ahogy az autó mozog. Azt is kimutattuk ezzel, hogy a PDOP érték 2 és 3 referenciapont esetén se jobb 1,93-nál, ami azt jelenti, hogy a helyzethiba nagyjából $\pm 4cm$ és ennél csak rosszabb lehet $\pm 2cm$ -es

távolságmérési pontosság mellett. Bár a Marvelmind rendszerben pontosságot javító szűrő eljárások vannak implementálva – melyek működését nem ismerjük a hiányos dokumentáció és a zárt firmware miatt – ilyen kis méretarányú modellek esetén mindenképp szükséges további pontosságot javító eljárást implementálni a rendszerben.



35. ábra PDOP mező.

3.6. Gyűjtött adatok elemzése

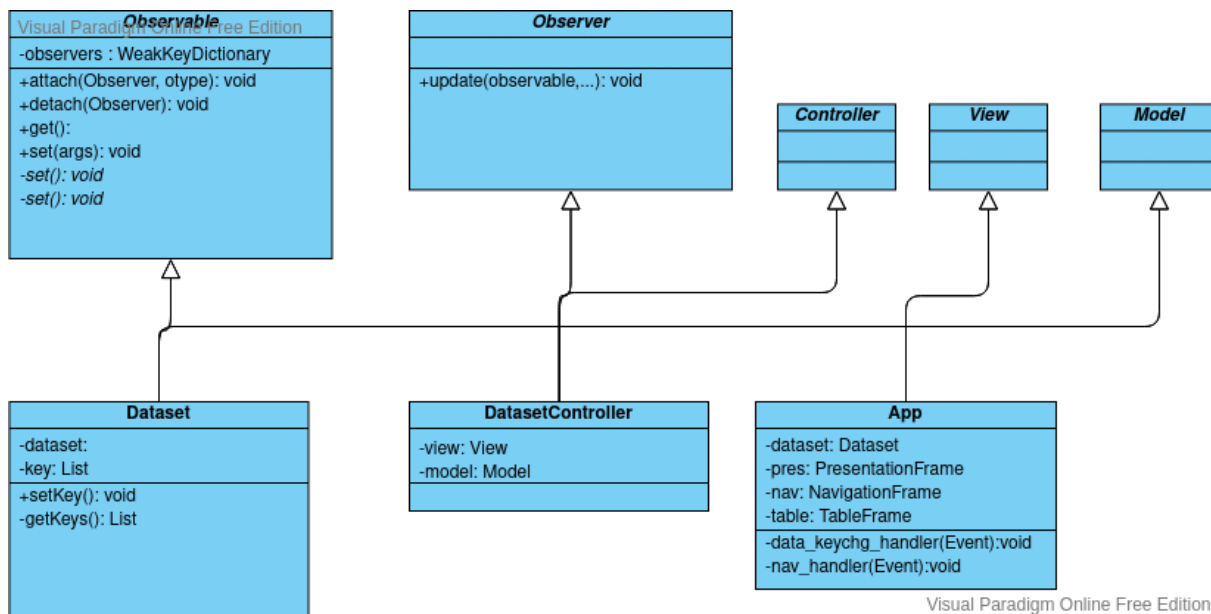
Ahhoz, hogy ellenőrizni tudjuk az IPS² konfigurációjának helyességét, próbaméréseket végeztünk a rendszerrel. Egyaránt gyűjtöttünk adatokat a pozícióval, gyorsulással és elfordulással kapcsolatban a Marvelmind jeladóba beépített IMU szenzor segítségével. A gyűjtött adatokon vizuális elemzést végeztünk. Ehhez saját interaktív alkalmazást fejlesztettünk.

A vizuális adatmegjelenítő alkalmazásunkat Python és Tkinter alapon fejlesztettük. A programhoz grafikus felület fejlesztettünk ki MVC³ mintára. A Tkinter egy Python csomag, aminek motorja a Tcl/Tk környezet. Előszeretettel használják a Tkintert Python alapú gyors prototípus

² IPS: Indoor Positioning System, azaz beltéri helyzetmeghatározó rendszer.

³ MVC: Model-View-Controller programtervezési minta. Az adatmodell, annak nézetei, valamint az őket összekötő irányító eljárások külön programegységeket képeznek és egymás között kommunikálnak.

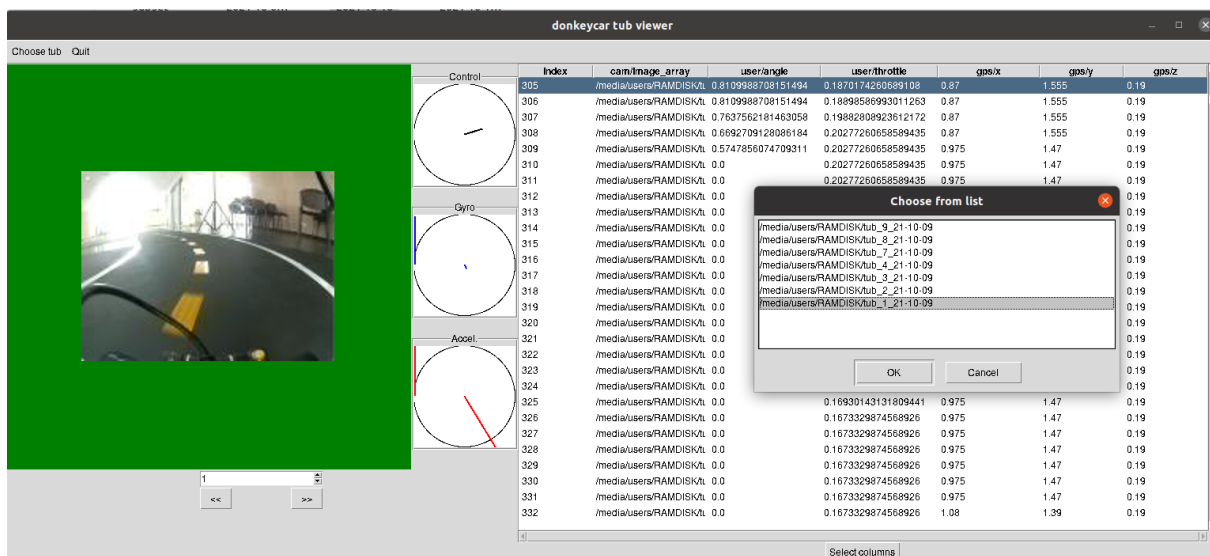
fejlesztésre olyan esetben, amikor a cél alkalmazás grafikus felülettel kell rendelkezzen. Sajnos elég korlátozott eseményvezérlést támogat, ezért szükséges volt kiterjeszteni az MVC és megfigyelő (Observer) programtervezésimintát követő osztályokkal. A megfigyelő minta az eseménykezelés kiterjesztését szolgálja. Az alkalmazás vázát a 36. ábra mutatja.



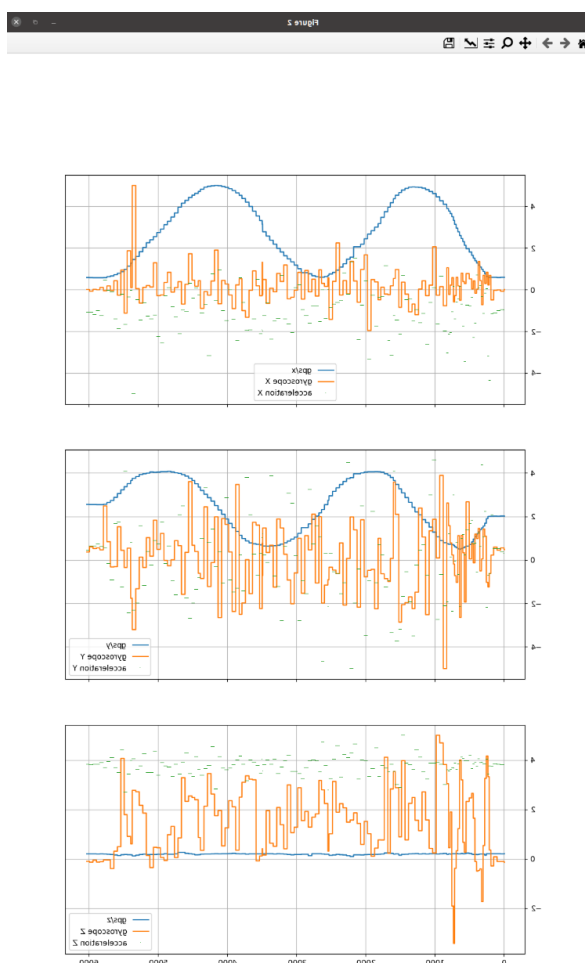
36. ábra Az alkalmazás osztályhierarchiája.

Távlati elképzelésünk ezzel az összetett programtervezéssel az volt, hogy a jövőben egy jól használható Python alapú adatvizualizációs keretrendszert fejlesszünk ki belőle.

Az alkalmazás a Donkey által generált „tub” könyvtárakat dolgozza fel. Kétféle üzemmódban lehet elindítani. Az egyik üzemmódban egy interaktív felületen navigálhatunk az egy könyvtárakban tárolt rekordok között. Képesek vagyunk benne váltani egyik könyvtárból a másikba (a választható könyvtárakat kilistázza a program). A rekordok tartalmát megjeleníti táblázat formájában, illetve a rekordhoz tartalmazó kameraképet, valamint IMU adatokat grafikusan is. A program segítségével ebben az üzemmódban ellenőrizni tudtuk például azt, hogy gyorsulásvektor tényleg a gyorsulás irányába mutat-e, valamint a giroszkóp az elfordulás mértékét jó tengelyekhez párosította-e.



37. ábra Interaktív adatvizualizációs felület. A kameraképtől jobbra van megjelenítve a gyorsulásmérő és giroszkóp adatai.



38. ábra Nem interaktív üzemmód ablaka.

A másik üzemmód – egyelőre – szerényebb felhasználói felülettel rendelkezik. Meghatározott „tub” könyvtárakat kiválasztva megjeleníti a bennük rögzített számszerű vagy számszerűsíthető szenzoradatokat több grafikonon egyszerre. A vektor típusú adatokat pontfelhődiagramon jeleníti meg. A pozíciókat pontokkal, a gyorsulásvektorokat – ill., ha van sebességvektorokat – az adott pozícióban nyíllal ábrázolja a program.

3.7. Gyűjtött adatok jóságának meghatározása

Az kis önvezető autónk a viselkedésmódjának betanuláshoz neurális hálózatot használ. Mint ahogy a róluk szóló részben (2.3.) már említettük, a neurális hálózatoknak sok és megfelelően felcímkezett adatra van szükség a sikeres betanulásukhoz. Ennek okából tettük fel az alábbi kérdést: Hogy mégis hogyan dönthetnénk el a saját tanítási környezetünkben, hogy egy adott adat mennyire számít jónak?

A Donkey minden egyes adatrögzítési során elmentett adatokat úgynevezett „tub”-ba menti (továbbiakban kádak). A program elindulásakor lehetőségünk van a feldolgozandó adatbázisban szereplő kádak közül az épp általunk feldolgozni kívántak kiválasztására. A program futása alatt ezeken a kádakban szereplő adatokat dolgozza fel a program egymás után.

Minden egyes adatpont jóságának a meghatározásához az adott készített képhez tartozó autó haladási sávjához viszonyított pozícióját használjuk, a 3.4.1 fejezetben megírt módon. Az ott már említett k-D fa legközelebbi szomszéd keresést és számításokat használjuk itt is, de mivel itt már több darab pozícióról beszélünk, ezért a kereső algoritmusnak is több pontot adunk át, amikhez keressük a térkép görbéinek legközelebbi pontjait (7. algoritmus).

Eltérés a sima sáv középpontjához viszonyított, normalizált távolságot meghatározáshoz használt programhoz képest, hogy ez a szoftver az 9. algoritmus visszatérési értékekének négyzetét használja a pont jóságának meghatározására, és ezen értékek átlaga, mediánja, minimuma és maximuma segítségével végzi minden egyes „tub” összehasonlítását. Ezek segítségével eldönthető, hogy tanításhoz teljesen, részlegesen, vagy egyáltalán nem használhatóak fel az egyes adatok.

Ezen elemzés elvégzése végett a szoftver képes minden egyes kád feldolgozása után, a szoftver változóitól függően, megjeleníteni két különböző ábrát is, amik célja a minősített adatok könnyen értelmezhetővé tétele és az adatszelektálás megkönnyítése.

Az egyik ábrán a rögzített adatpontok vannak a térképen megjelenítve, a pozíciók jósága szerint szín kódolva (zölddel a sávon belüli, sárgával a sávonalat érintő és pirossal a sávot elhagyó pozíciók). Ezzel láthatjuk, hogy a pálya melyik részéről vannak és milyen adataink.

A másik ábra az y tengelyén a rögzített adatok jóságát, míg az x tengelyén az ezekhez tartozó haladási irányát jeleníti meg. Balra kanyarodás esetén negatív, egyenesen haladás esetén 0 körüli, míg jobbra kanyarodás esetén pozitív értékek tartoznak az ábrázolt adatokhoz az x tengelyen, mivel az giroszkóp által mért z tengely körüli, két mérés ideje alatt bekövetkezett szögelfordulást veszik fel az ábrázolt pontok értéként.

A giroszkóp szenzor kimeneti adatait viszont ehhez a megfelelő adattípusúra kell konvertálni. Az adatot a pozíciókhoz társítva kapom meg a rögzített adatok listájában, 0.0175 fok-per-másodperc / legkisebb helyiértékű bit (dps/LSB) adattípusban ábrázolva. Ez azt jelenti, hogy minden egyes értéknövekedés 0.0175 fok/másodperces szögsebességet jelent a rögzített adatban. Ezt az alábbi módon (10. algoritmus) konvertáljuk át, a két adatmérés közötti idő segítségével, szögelfordulássá.

Algorithm 10: ConvertGyroDatas (GL_z , TL_{ms})

Input: Rögzített szögsebességi mérések listája 0.0175 dps/LSB-ben megadva GL_z ,
mérések időpontjának listája milliszekundumban megadva TL_{ms} , mérésszám n

Output: Fokba konvertált giroszkóp mérési adatok listája GL_{deg}

```

1   $GL_{deg}$  legyen üres lista
2  for  $i = 1$  to  $n$  do
3     $AngVel \leftarrow 0.0175 * GL_z$ 
4     $\Delta T_s \leftarrow (TL_{ms}[i] - TL_{ms}[i - 1])/1000$ 
5     $G_{deg} \leftarrow AngVel * \Delta T_s$ 
6     $G_{deg}$  hozzáfűz  $GL_{deg}$ -hez
7  end
8  return  $GL_{deg}$ 

```

10. algoritmus. A giroszkóp adatainak konvertálása.

A kiszámolt értékeket az adatkád és a térkép nevével együtt egy „json” kiterjesztésű fájlba mentjük le. Ezekből egy másik program segítségével több kádnyi adat, de akár az egész adatbázis adataira is elkészíthető az előbb leírt két ábra.

4. Eredmények, értékelés

4.1. Adatgyűjtés

Mint ahogy azt már korábban említettük (2.3) a felügyelt tanulás alapú neurális hálók betanításához sok megfelelően felcímkezett adat szükséges. Ezért mi se restelkedtünk a tanítóadatok gyűjtésében. A projekt alatt, még a Marvelmind helymeghatározó szenzorrendszer Donkey rendszerbe integrálása előtt 176,3 MB, pozíciós információ nélküli adatot rögzítettünk. A beintegrálás elvégzése után 913,3 MB, a kocsí menet közbeni koordinátáit, és legtöbb esetben IMU adatokat (amennyiben az volt a kocsira felszerelt beaconben) is, tartalmazó adatot rögzítettünk még. A rögzítések előkészítése alatt elvégzett rendszerfelállítási folyamatok alatt 8,97 MB – a referenciapontok térbeli helyzetét és más, a pozíciómeghatározást lehetővé tevő adatokat tartalmazó – szenzorrendszer inicializációs adat került lementésre. Ezek három különböző felépítésű pálya köré kiépített szenzor és referenciapont rendszert definiálnak, amiket az adatrögzítés során használtunk.



39. ábra. Az első, padlóra ragasztott pálya.

A legelső pálya (39. ábra) az egyetem Informatikai Karának első emeleti folyosójára lett ráragasztva fehér és sárga ragasztószalagokkal. Ez egy nagyobb méretű pálya, viszont a jármű ezen a pályán történő önvezetésének megoldása nehézséget is rejt magában.

Ezek közül a mi munkánkat érintő nehézség a következő volt. A pálya környezetének kialakítása miatt nem lehet egy olyan helymeghatározó rendszert felállítani, ami egy altérképben teljes pontossággal lefedné az egész pályát. Így a pálya egyik részén (egész pontosan az üvegkorlát által határolt részen) a pozíciómeghatározás pontossága romlik.



40. ábra. A pálya egyik tatami eleme.

Mivel egy padlóra ragasztott pályát mindig csak akkor lehet használni, ha az adott területen nem tartózkodnak sokan, így a következő pálya készítésekor már a pálya mozgathatósága is figyelembe lett véve. Ezt több, hasonló projekten dolgozó, egyetemi hallgatóval együtt 1x1 méteres tatamikból készítettük el. Ezek fekete oldalára, azok összefűzése után, fehér és sárga fényvisszaverő ragasztószalagokkal lettek felragasztva el rajtuk a sávok, amik aztán a szőnyegdarabok széle mentén lettek szétvágva (40. ábra).



41. ábra. A második pálya.

A második és harmadik pálya már ilyen tatamikból lett összetéve, annyi eltéréssel, hogy a harmadik pálya (42. ábra) kisebb, mint a második (41. ábra), és a második pálya darabjait felhasználva lehet megépíteni. Erre sajnos amiatt volt szükség, mert az addigi laborunk beázott, így a pályát át kellett költöztetni egy kisebb helyre. De a pálya hordozhatóságának köszönhetően könnyen és gyorsan tudtuk reagálni erre a váratlan helyzetre, és tudtuk folytatni a munkánkat.



42. ábra. A harmadik, kisebb pálya.

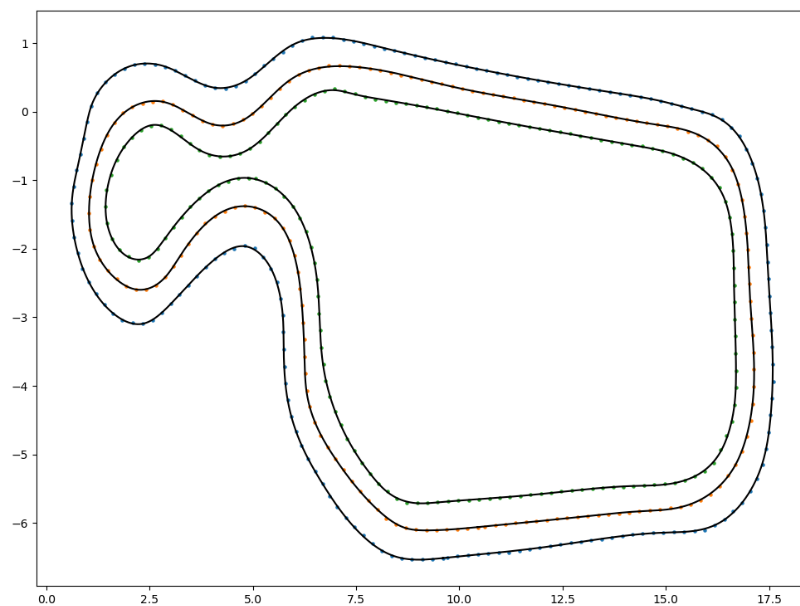
Ezen pályák sávválasztóinak és útszéleinek elhelyezkedését is rögzíteni kellett a kitelepített helymeghatározó rendszer koordináta-rendszerében, a rögzített menet- és vezetési adatok későbbi feldolgozhatóságának lehetővé tételéhez. Összesen 1,3 MB pályaadatot rögzítettünk, amiből 38,43 MB digitalizált térképinformációt generáltak le szoftverjeink a digitalizálási folyamat (3.2) alatt.

Ezen túl még megemlítenéd, hogy a projekt alatt elvégzett adatok rögzítését és elemzését 3980 sornyi Python és 1760 sornyi C nyelvben megírt kód segítette elő. Ezentúl a kódok rendszerbe integrálását még 37 sornyi makefile és 68 sornyi patch kód is megkönnyíti. A szükséges számítások (Hibaterjedés, Pontosság-hígulás, Kálmán-szűrő) elvégzését pedig 387 sornyi Jupyter notebook kód és leíró szöveg segítette elő.

4.2. Gyűjtött adatok feldolgozása, értelmezése és elemzése

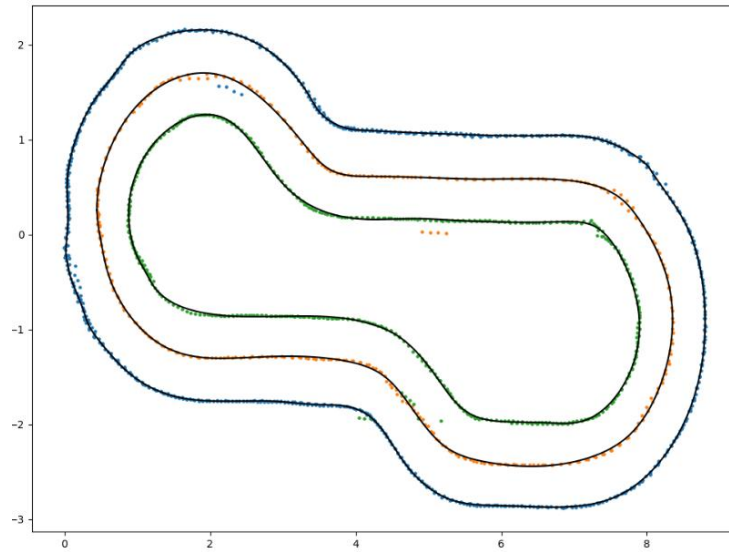
Az első pályát hibrid módszerrel digitalizáltuk. Először is végig húztunk egy sündisznó módon futó beacont a pályát meghatározó három vonalon. Viszont az előző fejezetben (4.1) leírt pozíciómeghatározási problémák miatt a rögzített értékeknek nagy volt a szórása. Ezért a felvett

többszöri megismétlése után az összes adatot beimportáltuk a QGIS nevű nyílt forráskódú tér-informatikai rendszerbe, ami segítségével kézzel rögzítettük a pálya görbéit, a rögzített pontok közül kiválogatott pontokkal. Az ebből az adathalmazból exportált pontfelhőn futtattuk végig a pályadigitalizáló szoftvert (3.2), ami a 43. ábra szereplő digitális térképet eredményezte, aminek az x tengely 0 és 7.5 méter közötti szakaszain jól, a többi részén viszont bizonytalanul tudta meghatározni a jármű pozícióját a rendszer (a környezet kialakítása miatt). A leírt digitalizálási folyamat eredménye az általunk elkészített térképek között a legpontosabb, viszont a legtöbb munkaórát igénylő pályatérkép lett.



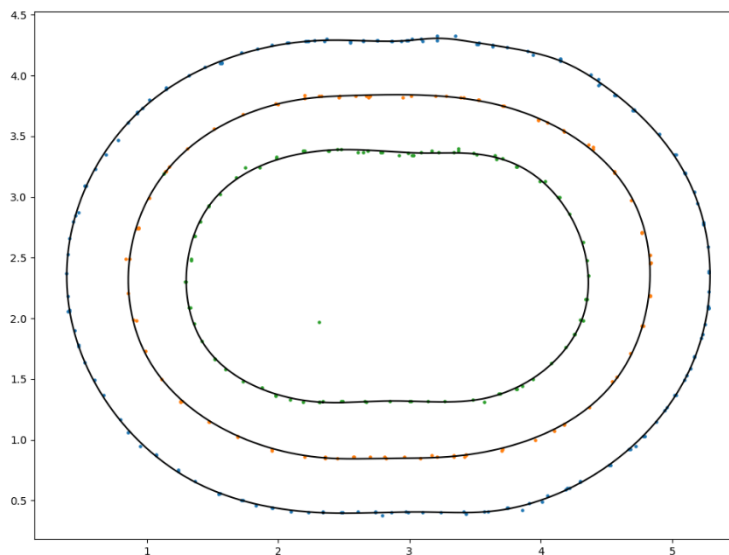
43. ábra. Az első pálya digitalizált térképe.

A második pályának (44. ábra) rögzítése is, a pálya sávjainak határoló vonalainak mentén többször végigfuttatott, sündisznó módban futó beacon segítségével történt. Viszont itt a pályadigitalizáló szoftverre (3.2) bízunk a teljes adatszűrést, sorba rendezést és görbeillesztést is. Látszik, hogy ezen módszer okozhatja, a szenzort végig gurító ember testének takarása miatt, a rögzített koordináták időnkénti eltolódását, amik miatt a szűrés minden esetben szükségessé válik.



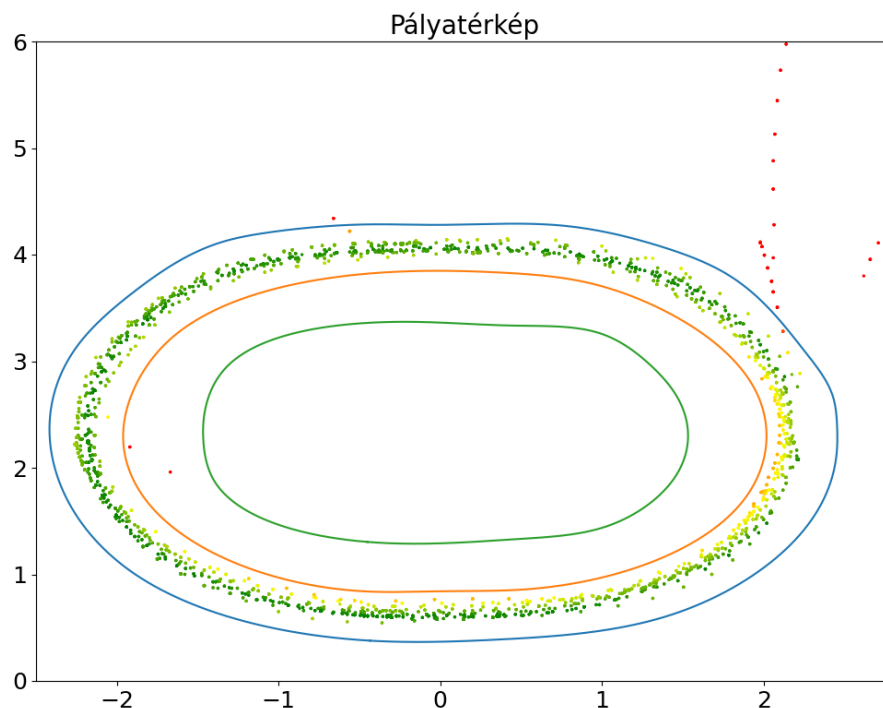
44. ábra. A második pálya digitalizált térképe.

A harmadik pálya (45. ábra) esetén kihasználtuk az interpoláció által biztosított lehetőségeket, és a pályasávoknak csak bizonyos pontjait rögzítettük. Ezen módszer alkalmazásával a rendszerben szereplő zaj is minimalizálhatóvá vált, mivel a rögzítést végző beacont mozgató embernek nem kellett a rögzítés pillanatában a megfigyelt területen belül tartózkodnia. Így a pályadigitalizáló szoftver (3.2) egy tisztább pontfelhőből tudott pályatérképet készíteni.

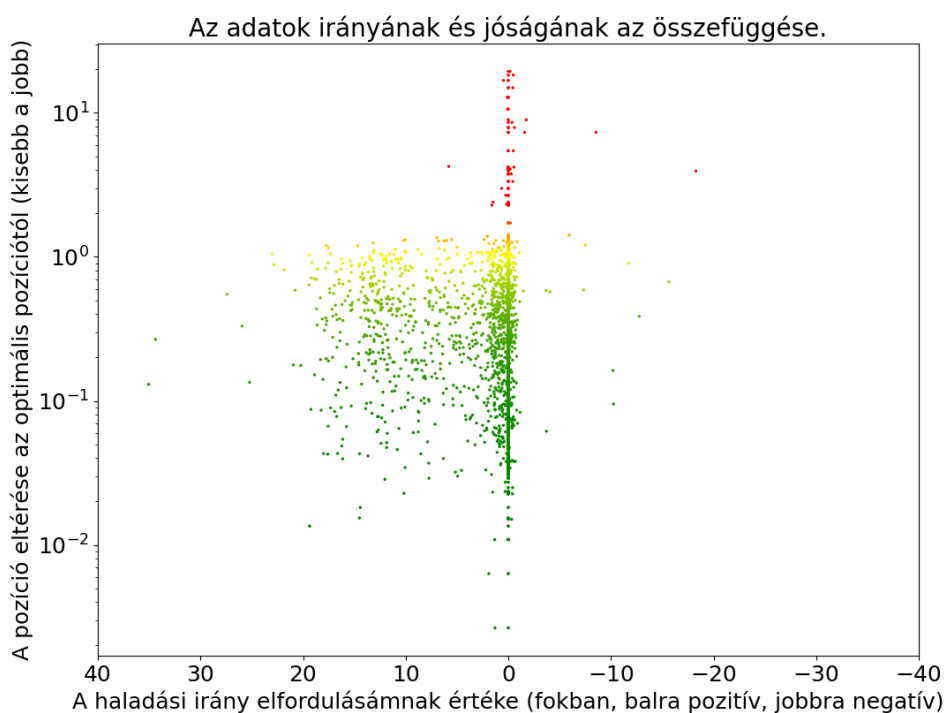


45. ábra. A harmadik pálya digitalizált térképe.

Az adatok feldolgozása és ábrázolása a 3.7. fejezetben leírt módon történt. Ezek közül az alábbi bekezdésekben és ábrákban mutatunk be pár, a harmadik pályán, rögzített adatot, és elemezzük őket röviden a szoftver által elkészített ábrák alapján.



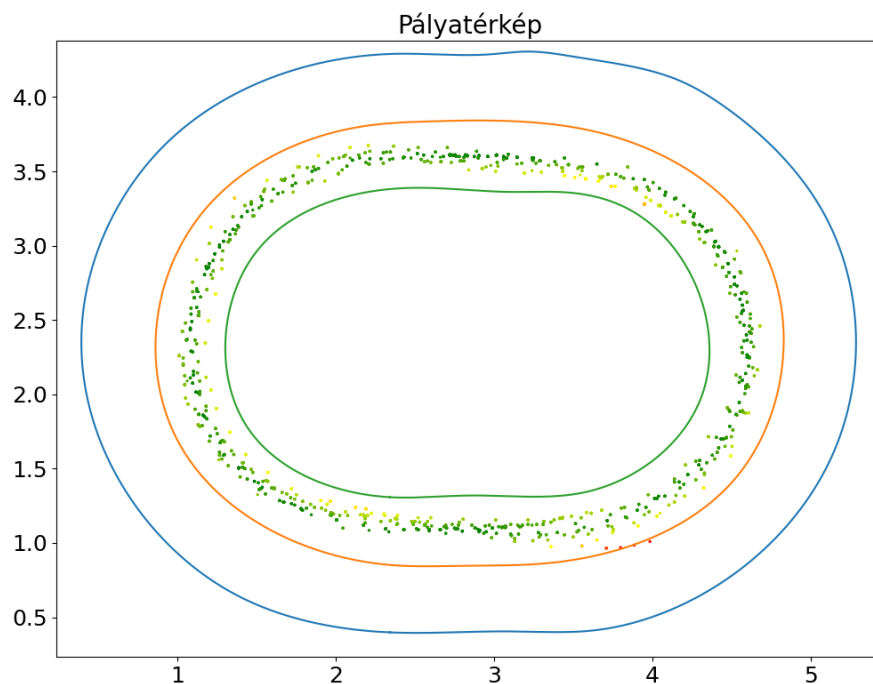
46. ábra. Egy sofőr által rögzített, külső sávú vezetési adat ábrázolva egy térképen.



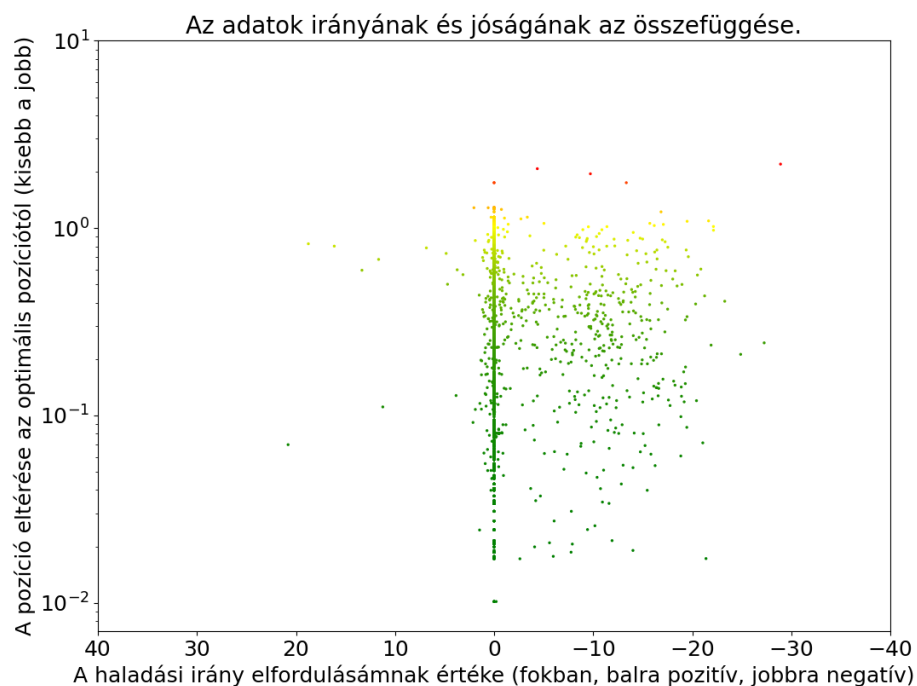
47. ábra. A 46. ábra adathalmazát ábrázolva a haladási irányok szerint.

A 46. ábra és 47. ábra által ábrázolt adatokon jól látszik a balra forduló adatok túlnyomó többsége, a pálya kialakítása miatt. Ha folytonosan tartjuk a pálya ívét, akkor ugyanis csak így kell kanyarodnunk, jobbra egyedül korrekciós mozgásokat végzünk. A pozíció eltérés alapján

felismerhető a pálya elhagyása, valamint az is, hogy az egyik kanyar esetén sokszor érintettük a sáv belső szélét.



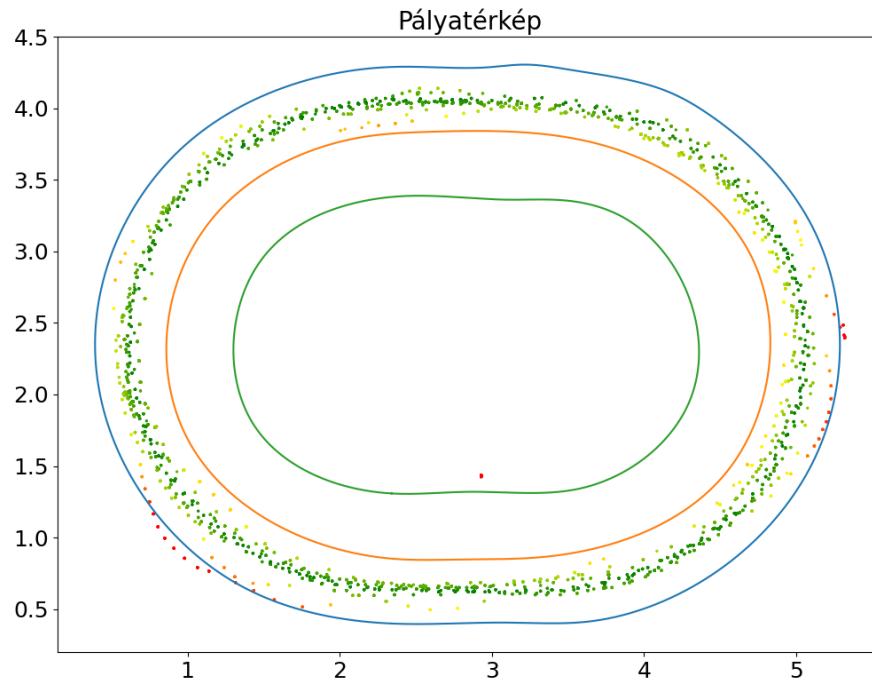
48. ábra. Egy másik sofőr által rögzített, belső sávú vezetési adat ábrázolva egy térképen.



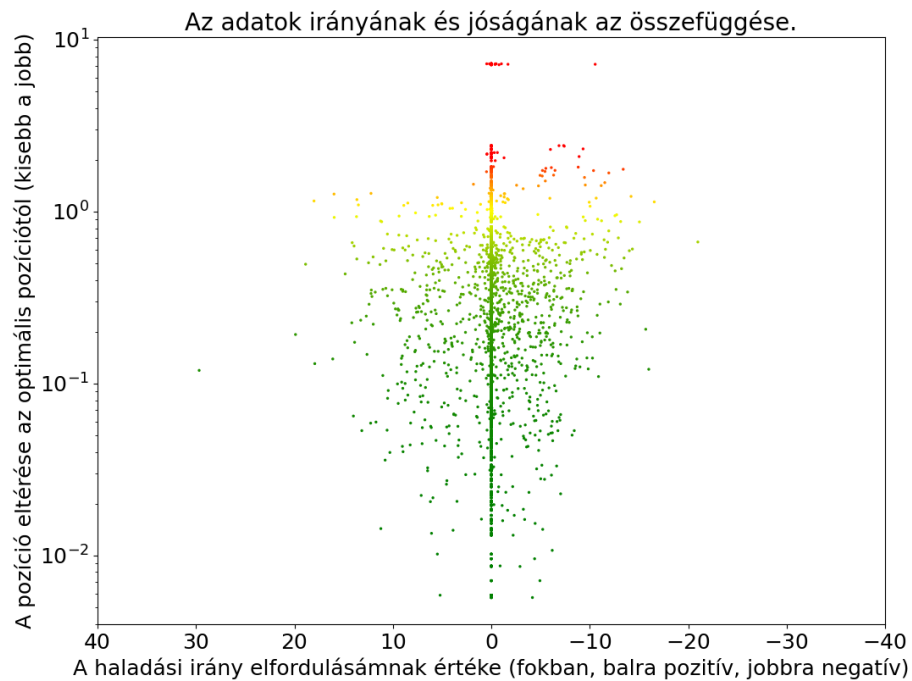
49. ábra. A 48. ábra adathalmazát ábrázolva a haladási irányok szerint.

A 48. ábra és 49. ábra ábrázolt adatokon egy másik sofőr, belső sávban történő közlekedését láthatjuk. Itt a jobbra forduló adatok többsége fedezhető fel a pálya formája miatt, ugyanis

a folytonos pályáiv követéshez csakis ilyen irányú elfordulásra van szükségünk. Balra időnként korrekciós mozgásokat végzünk. A sofőrváltás a jármű és a sáv belső szélének általános érintkezési pontjainak változásában is megfigyelhető.



50. ábra. Az előző sofőr által rögzített, külső sávú vezetési adat ábrázolva egy térképen.



51. ábra. adathalmaz ábrázolva a haladási irányok szerint.

A 50. ábra és 51. ábra ábrán megjelenített adatokon, az előző sofőr által rögzített, külső sávban haladást láthatunk. Itt a balra és jobbra forduló adatok között még mindig a balra fordulók dominálnak, viszont a viszonylagosan nagy számú jobbra kanyarodó adat arra utal, hogy már sokkal többször kellett korrigálni a jármű helyzetét vezetés közben. Ezen túl a sofőr azon szokása, hogy a kanyarok második felében érinti a sáv bal szélét továbbra is megfigyelhető.

Ezen feldolgozott adatokkal és rövid elemzésünkkel a feldolgozó programjaink működését, és az általuk elvégzett munka értékét szerettük volna bemutatni. Ezen elkészített ábrák alapján ugyanis elemezhetőek és csoportosíthatóak az adathalmazok, mind tanításra való felhasználhatóság, mind a sofőrök vezetési stílusa alapján is. Ezzel ki tudjuk választani a tanításra legalkalmasabb sofőröket és adathalmazokat is, miközben magukon az adathalmazokon belül már most is elemezni és osztályozni tudjuk az adatokat részegységenként is.

4.3. Megfigyelések, tapasztalatok

Ebben az alfejezetben a projekt alatt tett megfigyeléseinket és tapasztalatainkat gyűjtjük össze. Ezeknek témája bekezdésenként változik.

Ajánlott egy közel azonos arányú egyenesen, balra és jobbra haladást megkívánó pályát készíteni, mivel az optimális betanításhoz ezek azonos arányban szükségesek, és ekkor nem szükséges külön adatok rögzítése valamilyen haladási irányhoz, ami ajánlott, amennyiben ez a rögzítési feltétel nem teljesül.

Az adatokat felvelő kisautót Raspberry Pi 4 vagy NVIDIA Jetson Nano hajthatja meg, mint ahogy azt már korábban említettük (2.4). Ezek között lényegi különbség, a Nano kártya erősebb számítási teljesítményéből fakadóan, hogy ugyanazt az adatfeldolgozási folyamatot lassabban végezheti el a Raspberry kártya. Ebből fakadóan, mind a rögzített képek, mind az önvezetés alatt a betanított hálónak átadott képek minősége kisebb lesz a Raspberry kártya esetében. A mi esetünkben egészen pontosan 120x160-as képekről van szó. A Nano kártya esetében lehet szó 240x240 vagy még nagyobb képek rögzítéséről és felhasználásáról is. Ezentúl a Nano alatt simább rendszerfutást tapasztalhatunk, valamint több külső szenzor egyidejű feldolgozását tudja elvégezni az autó működése alatt is, így mind a valós idejű sávhoz viszonyított helyzetmeghatározás használatát, mind a helymeghatározó rendszer más szenzorrendszerek történő együttes használatát (például LIDAR) csakis Nano kártya használata esetén ajánljuk.

A helymeghatározó rendszer felállításánál próbálkoztunk a beaconök falra történő felszerelésével is, amit csak akkor ajánlunk, ha ténylegesen lehet fixen rögzíteni legalább az őket megtartó csavarokat a falba, mert csak akkor garantálható azok biztos helyzete. Amúgy jobb egy hordozható, de fix állványra felszerelni, mert az garantálja szükség esetén a mozgathatóságot, de biztosítja, hogy ne mozduljon el a rendszer használata közben a referenciapont.

A referenciapontként működő beaconök alatt a koordinátatér egy 15-20 cm sávban kicsit torzul, görbül. Ezt jó figyelembe venni, és úgy elhelyezni ezeket a szenzorokat, hogy a megfigyelni kívánt terület szélétől legalább 20 cm odébb legyen. Lehetséges a rendszeren „kívüli” helymeghatározásra is, de ekkor az adatfolytonosság és helyzetmeghatározási pontosság csökken.

Több altérkép egy térképbe rendezése esetén fontos, hogy mekkora az úgynevezett „átadási terület” (handover zone). Ez a terület a két altérkép metszete, és ezen belül a szoftver átállítja a rendszert, hogy már ne a kiindulási, hanem az érkezési tartomány határozza meg a sündisznó helyzetét. Fontos, hogy ez ne legyen se túl kicsi, hogy legye ideje a műveletet elvégezni, se túl nagy, hogy ne legyen nagy a bizalmatlan terület, ahol nem tudja eldönteni, hogy a térkép melyik alterébe fog beleérkezni a sündisznó.

Ezentúl lehetséges két teljesen átfedő altérképet létrehozni (invertálható architektúra használata esetén), de ez a szoftver v6.240 verziójában csak nagyon speciális helyzetekben ajánlott, az előbb leírt működési módszer miatt, például, ha egy sűrű oszlopsor eltakarja a megfigyelendő terület nagy részét. Ennek a működése esetén ugyanis a jelenleg használt szoftverek verziójában kétféle működési mód lehetséges. Az egyik esetben a két altérkép által mért pozíció átlaga lesz a meghatározott pozíció, míg a másik esetben a szerinte közelebbi szenzorok által mért értéket veszi meghatározott értéként. Ekkor két forgatókönyv lehetséges, ami miatt mindkét módban hibás értéket kaphatunk meg. Elsőnek lehetséges, hogy egy teljesen rosszul mért érték elhúzza az átlagolt pozíciót egy hibás helyre. Másodjára lehetséges, hogy a hibás értéket veszi a rendszer jó értéknek, mivel a hibás mérési érték miatt szerinte ahhoz az altérképhez van jobban közelebb. Mivel ezek valószínű események egy hibásan működő altérkép esetén, ezért a bekezdés elején említett példához hasonló eseteken kívül nem ajánlott ennek a használata, mert nem fog átlagos esetekben egy pontosabb helymeghatározáshoz vezetni, sőt a rendszer hibája miatt csak még több hibát visz a számításba. Azt ajánljuk tehát, hogy minden megfelelően belátható terület egy altérképpel figyeljünk meg, és a pontosságot az egy altérkép lehetséges referenciapontjainak növelésével érjük el (ez 4 darab minden altérkép esetén).

A referenciapontként és sündisznóként működő beaconök közötti kis távolságok esetén a környezetből jövő ultrahangos zajok nagyon elronthatják a pontosságot. Ez az autómata zajsztűrő algoritmusnak köszönhető, ugyanis a zaj és a rendes jel nagyon közeli erőssége / beérkezési ideje esetén előfordulhat, hogy a zajjal együtt a rendes jelet is kiszűri a rendszer. Ekkor a jel visszhangját fogja valós értéknek venni a rendszer, ami alapján a távolságmeghatározás hibája ± 2 centiméterről akár ± 20 méterre is romolhat (a környezettől függően).

A legutóbbi, 2021. szeptemberében kiadott Marvelmind szoftver 7.0 verziója [39] már a normál beaconöket csak visszamenőlegesen támogatja, az új funkciók működését csak a szuper beaconök esetében garantálják, így azok használata ajánlott a jövőben, hogy a helymeghatározó rendszer legfrissebb verzióját használhassuk.



52. ábra. A beacon IMU tengelyeinek iránya [24].

Fontos megjegyezni, hogy a beacon burkolata alapján lehet beazonosítani annak IMU tengelyeinek irányát [24] (52. ábra). Ez alapján levonható a következtetés, hogy a rögzített adathalmazok közötti eltérés elkerülése végett mindig ugyanúgy ajánlott felszerelni a járműre a beacont.

5. Jelenleg futó munkák

5.1. Adatok jósága a sávtartáson túl

A jelenlegi projektünkben az adatokat csakis sávtartási szempontból tudjuk osztályozni és a jóságukat meghatározni. De folyamatban van különböző mozgási műveletek felismerésének és összehasonlításának automatizálása.

Az adatok osztályozását például úgy tudjuk javítani, ha a pálya ívéhez viszonyítjuk az autó elmozdulását. Ugyanis a jelenlegi adatfeldolgozó szoftverek (3.7) alapján rossz egyenes adat egy kanyarban történő pályaelhagyás esetén valójában egy rosszul bevett kanyar lesz, de jelenleg ez csak egy rossz egyenes adatként kerül definiálásra.

Továbbá fejleszteni lehet még az adatok szelektálását az alapján, hogy nem csak a jelenlegi haladási irányt, és annak az optimálishoz viszonyított eltérését nézzük, hanem ha a jármű tartási irányát is figyelembe vesszük az adatelemzési folyamat alatt. Ez alatt azt értjük, hogy különbség van egy, az optimálistól éppen eltérő irányba haladó adatrész, és egy, az optimális felé közelítő, másik adatrész között. Mivel ez esetben az előbbit nem akarjuk a tanító adatok között felhasználni, mivel csak hibát vinne a rendszerbe, viszont az utóbbi tanító adatba építése segíthet a hálózat számára, hiszen megoldást mutat arra, hogy hogyan tud visszatérni a számára kijelölt útra és sávba, ha eltérne attól. Ezentúl az optimálistól eltérés és közelítés mértéke alapján is összehasonlíthatóak ezek az adatok. Ezen osztályozási feladat megoldására segítség lehet a jármű jelenlegi mozgásának deriváltja, és annak az optimális ív által bezárt szöge.

5.2. Reinforced Learning

A szoftverjeink által szolgáltatott pozíciós adatok (3.4) felhasználhatók megerősítéses tanulás alapú neurális hálók fejlesztésére is. A rendszer koordinátarendszerében meghatározott pozíció segíthet lehet az algoritmus jutalmazásában, ugyanis ezzel meg tudjuk mondani a rendszer számára, hogy a mikor tett meg egy kört, mikor tette meg a kör megtételéhez szükséges út egy részét (például virtuális ellenőrzőpontok felállításával), és hogy milyen jól követi az általunk optimálisan megítélt utat. Ezek mind olyan mutató értékek, amik szükségesek a megerősítés alapú tanításhoz, és ezek elérhetővé tételével megnyílik a lehetőség a használatára is.

Ezen tanítási módszer használatával az is lehetővé válik, hogy egy előre betanított hálózatot tanítunk tovább élő adatokkal és visszajelzéssel. Ez esetben az előre betanított háló garantálni

tudja azt, hogy ne csináljon semmilyen nagyobb balesetet és hibát a járművünk, viszont azt is lehetővé teszi, hogy az általa hozott hibás döntéseket észlelje és megpróbálja korrigálni. Mivel folytonosan visszajelzést kap a kisautó ezen esetben a hozott döntéseinek következményéről, így még a rossz javítás kísérleteket is hamar észreveszi, és tanulva az előző hibájából, már egy jobb megoldást tud megpróbálni működése korrigálására, egészen addig, míg meg nem tanulja, hogy egyes események vagy hibák esetén milyen reakció a megfelelő.

5.3. Kálmán-szűrő

Miután rájöttünk, hogy pusztán a referencia pontokként szolgáló jellevők precíz elhelyezésével nem lehetséges elérni a kívánt pontosságot, olyan megoldást kerestünk a szenzor adatok szűrésére, amit jelenleg is használnak valós idejű helyzetmeghatározásra. A Kálmán-szűrő sokszor alkalmazzák navigációs rendszerek pontosságának javítására, valamint önjáró robotok lokalizációjára [40] [41]. Alkalmazhatóságát az önvezető járművek területén az is jelzi, hogy az Udacity online oktatási platformon indított „Self Driving Car Engineer” kurzus oktatási anyagában is szerepel [42].

A Kálmán-szűrő alkalmazásának előnye, hogy a helyzetmeghatározás szűréséhez több különböző típusú szenzor adatot is képes felhasználni, így a mért múltbeli pozíciókon túl az IMU adatait figyelembe veszi az időben következő pozíció becslésére, pontosítására. Ideális esetben származtatott mennyiségeket is becsülhetünk a segítségével, például a jármű sebességét.

Jelenleg a Python környezetre implementált FilterPy program könyvtárat használjuk, melyben különféle Bayes-féle rekurzív becslésen alapuló eljárás – köztük a Kálmán szűrő – hívható meg. Jelenleg folyik egy csak múltbeli koordinátákon alapuló szűrő definiálása a Marvelmind rendszerhez. Ezután kibővítjük a szűrőt a pozíciókon túl az múltbeli IMU adatok feldolgozására.

6. Összefoglalás

Mint ahogy a bevezetésünkben is bemutatásra került, az önvezető járművek már most is jelentős szerephez jutnak a nemzetközi piacon, a jövőben pedig csak növekedni fog a jelenlétük, mind a mindennapi életünkben, mint a szolgáltatási szektorban is. Szakmai és kutatási körökben pedig egyértelműen húzóiránynak számítanak. Az ilyen eszközök fejlesztéséhez viszont hatékony gépi tanuló algoritmusokra és megfelelően szelektált, felcímkézett és nagyméretű adatbázisokra van szükség. Bár a kapcsolódó területekhez kötődően jelentős eredményeket értek el, számos kihívást jelentő feladat vár még magára, mind elméleti, mind technológiai oldalról.

Ezen területek közé tartozik a költségek racionalizálását célzó megoldások, a nehéz körülmények és szituációk vizsgálata, illetve ezek megoldását jelentő elméleti és technológiai eredmények is. Különös tekintettel a vezetéstámogató asszisztensek és önvezető rendszerek megbízhatóságnak és kutatásfejlesztésének a vizsgálataira zaj vagy egyéb okból fellépő szenzorjel-degradáció esetén, melyet okozhat sérülés, időjárási- (köd), nehéz út-viszonyok (kátyú, csúszós vagy tükröződő útfelület). Ezen területhez kapcsolódva kis méretű önvezető modellautók segítségével kívánjuk vizsgálni azon nehéz körülményeket és szituációkat, melyek a valós méretű autók esetében nehézkes, illetve kockázatos, jelentős anyagi terhet vonva magával. Ennek egy lépéseként foglalkoztunk kis méretben alkalmazható lokalizációs technikákkal, mivel ennek alkalmazása jelen ismereteink szerint elengedhetetlennek látszik az önvezető járművek területén. Míg az életnagyságú járművek esetén találunk számtalan késznek mondható megoldást és technológiát a helymeghatározási feladatok elvégzésére, addig az olyan jellegű munkák, ahol kis méretekkkel kell számolni, illetve beltéri alkalmazással kapcsolatosak, ott ilyen megoldást nehezen találunk. Bár régóta léteznek rádióhullám alapú beltéri helyzetmeghatározó rendszerek, ezek pontossága nem kielégítő olyan alkalmazások esetén, ahol pár centiméteres hibatűréssel kell méréseket végezni. Végül az önvezető járművek területén újszerű megoldást jelentő ultrahangos távolságmérésen alapuló megoldás kidolgozása mellett döntöttünk.

Dolgozatunkban bemutattuk, milyen módon sikerült integrálnunk egy GPS független, beltéren is alkalmazható helyzetmeghatározórendszert önvezető járműmodellek méretben. A hagyományos GPS rendszerekre támaszkodó megoldásokkal ellentétben a lokalizációs precizitás, és a beltéri alkalmazhatóság pozitívumai mellett előre nem várt komplex konfigurációs kihívások is a rendszer velejárói. Ennek elsődleges oka, hogy itt a komplett rendszer felállítása szükséges szemben a már széles körben alkalmazott GPS esetén, ahol elegendő egy kis komponens az

úgynevezett GPS lokátor minimális konfigurációt igénylő alkalmazása. A megoldásunkban sikeresen feltérképeztük a rendszer környezetének, mind a szenzorok helyzetének megfelelő definiálását és rögzítésének lehetőségeit, melyek révén sikerrel stabilizáltuk és minimalizáltuk a rendszer lokalizációs gyengeségeit és a számítási hibát. Ennek eredményeképpen nem csak a konkrét jármű pozíciót határozhatjuk meg, hanem különböző abszolút, illetve relatív pozíciókhoz viszonyítva, a jármű valós idejű digitalizálása, viselkedésének értékelése és diverzifikált tanítása valósítható meg.

Nevezetesen a kijött eredmények segítségével, a fejlett diverzifikált adatgyűjtés, teljes pályadigitalizáció és vizualizáció alkalmazásával elkezdünk az automatikus kiegyensúlyozott adatgyűjtés kidolgozását, melynek folytatása az önmege erősítő tanulás alkalmazása. Bízunk benne, hogy a kijött eredmények amellet, hogy hozzájárulnak a szakterület eredményeihez, sikerrel alkalmazhatjuk további új elméleti eredményekhez is.

7. Irodalomjegyzék

- [1] K. Zachary, "Businesses need smarter tech in their fleets to survive e-commerce boom," Nissan News, 26. 02. 2019. [Online]. Available: <https://global.nissannews.com/en/releases/release-41181d20da4d17b77ed88d700817c540-businesses-need-smarter-tech-in-their-fleets-to-survive-e-commerce-boom?source=nng>. [Accessed 27. 09. 2021].
- [2] Audi AG, "The pulse of autonomus driving: An international user typology and an Imotional landscape of autonomus driving," 25. 09. 2019. [Online]. Available: <https://www.audi.com/content/dam/gbp2/company/research/audi-beyond/2019/pulse-of-autonomous-driving/and-audi-study-auto>. [Accessed 27. 09. 2021].
- [3] L. Pizzuto, C. Thomas, A. Wang and T. Wu, "How China will help fuel the revolution in autonomous vehicles," 25. 01. 2019. [Online]. Available: <https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/how-china-will-help-fuel-the-revolution-in-autonomous-vehicles>. [Accessed 27. 09. 2021].
- [4] "Waymo: Company Financials," Cruchbase, [Online]. Available: https://www.crunchbase.com/organization/waymo/company_financials. [Accessed 27. 19. 2021].
- [5] "Tesla: Company Financials," Cruchbase, [Online]. Available: https://www.crunchbase.com/organization/waymo/company_financials. [Accessed 27. 09. 2021].
- [6] CBInsights, "40+ Corporations Working On Autonomous Vehicles," 16. 12. 2020. [Online]. Available: <https://www.cbinsights.com/research/autonomous-driverless-vehicles-corporations-list/>. [Accessed 27. 09. 2021].
- [7] "Rivian: Company Financials," Cruchbase, [Online]. Available: https://www.crunchbase.com/organization/waymo/company_financials. [Accessed 27. 09. 2021].
- [8] "Cruise: Company Financials," Cruchbase, [Online]. Available: https://www.crunchbase.com/organization/waymo/company_financials. [Accessed 27. 09. 2021].
- [9] Cruise, [Online]. Available: <https://www.getcruise.com/>. [Accessed 27. 09. 2021].
- [10] S. J. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, Third ed., Prentice Hall.
- [11] M. Mohri, A. Rostamizadeh and A. Talwalka, "Foundations of Machine Learning," *The MIT Press*, 2012.

- [12] G. Grieve, "Explained SIEMply: Machine Learning," Logpoint, 17. 07. 2020. [Online]. Available: <https://www.logpoint.com/sv/blogg-sv/explained-siemply-machine-learning/>. [Accessed 28. 09. 2021].
- [13] V. A. Cabannes, F. R. Bach and A. Rudi, "Fast rates in Structured Prediction," *COLT*, 2021.
- [14] IBM Cloud Learn Hub, "Supervised Learning," 19. 08. 2020. [Online]. Available: <https://www.ibm.com/cloud/learn/supervised-learning>. [Accessed 28. 09. 2021].
- [15] D. Soni, "Supervised vs. Unsupervised Learning," towards data science, [Online]. Available: <https://towardsdatascience.com/supervised-vs-unsupervised-learning-14f68e32ea8d>. [Accessed 28. 09. 2021].
- [16] Wikipedia Commons, "Multi-Layer Neural Network-Vector.svg," [Online]. Available: https://commons.wikimedia.org/wiki/File:Multi-Layer_Neural_Network-Vector.svg. [Accessed 28. 09. 2021].
- [17] IBM Cloud Learn Hub, "Neural Networks," 17. 08. 2020. [Online]. Available: <https://www.ibm.com/cloud/learn/neural-networks>. [Accessed 28. 09. 2021].
- [18] S. Varma and S. Das, "Deep Learning, Chapter 7: Training Neural Networks Part 1, Figure 7.2," [Online]. Available: <https://srdas.github.io/DLBook/GradientDescentTechniques.html>. [Accessed 28. 09. 2021].
- [19] S. Kansal, "Quick Guide to Gradient Descent and Its Variants," towards data science, 18. 08. 2020. [Online]. Available: <https://towardsdatascience.com/quick-guide-to-gradient-descent-and-its-variants-97a7afb33add>. [Accessed 28. 09. 2021].
- [20] S. Bhattarai, "What is Gradient Descent in machine learning?," A tech blog, [Online]. Available: <https://saugatbhattarai.com.np/what-is-gradient-descent-in-machine-learning/>. [Accessed 28. 09. 2021].
- [21] S. Saha, "A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way," towards data science, 15. 12. 2018. [Online]. Available: <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>. [Accessed 28. 09. 2021].
- [22] IBM Cloud Learn Hub, "Convolutional Neural Networks," 20. 10. 2020. [Online]. Available: <https://www.ibm.com/cloud/learn/convolutional-neural-networks>. [Accessed 28. 09. 2021].
- [23] Donkey Car, [Online]. Available: <https://www.donkeycar.com/>. [Accessed 01. 10. 2021].

- [24] Marvelmind Robotics, [Online]. Available: <https://marvelmind.com/>. [Accessed 01. 10. 2021].
- [25] Marvelmind Robotics, "Marvelmind Indoor Navigation System Operation Manual," 04. 07. 2021. [Online]. Available: https://marvelmind.com/pics/marvelmind_navigation_system_manual.pdf. [Accessed 29. 09. 2021.].
- [26] "True-range multilateration," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/True-range_multilateration. [Accessed 01. 10. 2021].
- [27] Marvelmind Robotics, "Marvelmind support software package," [Online]. Available: https://marvelmind.com/pics/marvelmind_SW.zip. [Accessed 07. 10. 2021].
- [28] "Dilution of precision (navigation)," Wikipedia, the free encyclopedia, 25 május 2021. [Online]. Available: [https://en.wikipedia.org/w/index.php?title=Dilution_of_precision_\(navigation\)&oldid=1025023225](https://en.wikipedia.org/w/index.php?title=Dilution_of_precision_(navigation)&oldid=1025023225).
- [29] J. Ádám, L. Bányai, T. Borza, G. Busics, A. Kenyeres, A. Krauter and B. Takács, *Műholdas helymeghatározás*, Budapest: Műegyetem Kiadó, 2004.
- [30] Wikipedia, "B-spline," [Online]. Available: <https://en.wikipedia.org/wiki/B-spline>. [Accessed 10. 10. 2021].
- [31] D. C.-K. Shene, "B-spline Curves: Definition," Michigan Technological University, Department of Computer Science, [Online]. Available: <https://pages.mtu.edu/~shene/COURSES/cs3621/NOTES/spline/B-spline/bspline-curve.html>. [Accessed 10. 10. 2021].
- [32] D. C.-K. Shene, "B-spline Curves: Closed Curves," Michigan Technological University, Department of Computer Science, [Online]. Available: <https://pages.mtu.edu/~shene/COURSES/cs3621/NOTES/spline/B-spline/bspline-curve-closed.html>. [Accessed 10. 10. 2021].
- [33] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, p. 509–517, 1975.
- [34] Wikipedia, "k-d tree," [Online]. Available: https://en.wikipedia.org/wiki/K-d_tree. [Accessed 10. 10. 2021].
- [35] The SciPy Community, "SciPy documentation: spatial.KDTree," [Online]. Available: <https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.KDTree.html>. [Accessed 10. 10. 2021].
- [36] Python Software Foundation, "Python/C API Reference Manual," [Online]. Available: <https://docs.python.org/3.9/c-api/index.html>. [Accessed 10. 10. 2021].

- [37] The SciPy community, "SciPy documentation: interpolate.splprep," [Online]. Available: <https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.splprep.html>. [Accessed 10. 10. 2021].
- [38] The SciPy community, "SciPy documentation: interpolate.splev," [Online]. Available: <https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.splev.html>. [Accessed 10. 10. 2021].
- [39] Marvelmind Robotics, "Marvelmind SW pack v7.0/v7.1 release notes," [Online]. Available: https://marvelmind.com/pics/marvelmind_SW_v7.000_v7.100_release_notes.pdf. [Accessed 10. 10. 2021].
- [40] N. Z. Z. M. A. R. S. & b. A. M. Y. Mohd Nasir, "Autonomous mobile robot localization using," *MATEC Web of Conferences*, vol. 90, 2017.
- [41] J.-H. P. T. & E. B. Won, "Characteristics of Kalman Filters for GNSS Signal Tracking Loop," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 48, pp. 3671 - 3681, 2012.
- [42] Udacity, "Become a Self-Driving Car Engineer," [Online]. Available: <https://www.udacity.com/course/self-driving-car-engineer-nanodegree--nd0013>. [Accessed 14 10 2021].

8. Saját munka leírása

A munkánk során figyeltünk annak hatékony elvégzésére. Emiatt a legtöbb munkát feladatonként felosztva végeztük el, miközben egymás munkájára építettünk. Ez lehetővé tette a legtöbb feladat egy személy általi megoldását, miközben felhasználtuk egymás, már elkészített megoldásait is a saját munkánk megkönnyítésére és elvégzésére. Ezek mellett arra is lehetőségünk volt, hogy komplexebb feladatokat kisebb részfeladatokra bontva oldjunk meg közösen. Ezen elvek alapján a munkánk két részre bontható.

Palkovics Dénes által készített munkák felsorolása:

- Kutatás a hibaterjedéssel és pontosságáthígulással kapcsolatban. A hibaterjedés számítását bemutató és elvégző, valamint a pontosságáthígulást számító Jupyter jegyzetfüzetek írása.
- Kutatás a Kálmán-szűrő implementálásával és felhasználásával kapcsolatban.
- Adatvizualizációs alkalmazás fejlesztése. Az első működő verziót én készítettem, de a későbbi verziók fejlesztésében Máté is segített, főleg a Donkey 4.2.1-es verzióval való kompatibilitási részben.
- A felhasznált adatok rögzítése a modellautókkal. A 42. ábra fényképének készítése.
- Marvelmind rendszer szoftverintegrációjának (3.3) Python nyelven írt részei:
 - part modul implementálása
 - telepítő csomag létrehozása
- Dolgozat szerkesztése:
 - 2.6. Hibaterjedés
 - 2.7. Pontosságáthígulás
 - 2.8. Páros sündisznó üzemmód
 - 3.3. Marvelmind szenzorok integrálása a Donkey® platformba
 - 3.5. Hibaterjedés és pontosságáthígulás számítása
 - 3.6. Gyűjtött adatok elemzése
 - 5.3. Kálmán-szűrő
 - 6. Összefoglalás

Vágner Máté által készített munkák felsorolása:

- Kutatás a gépi tanulással, legközelebbi szomszéd kereséssel, b-spline interpolációval és a Python/C API-val kapcsolatban.

- A 3.1., 3.2., 3.4., 3.7. fejezetek által bemutatott pályarögzítő és digitalizáló, pozíció-meghatározó, és adatfeldolgozó és vizualizáló kódok fejlesztése.
- Az adatok rögzítésére használt pályák térképének digitalizálása és a rögzített adatok elemzésének elvégzése a készített programok segítségével.
- A 18. ábra és 19. ábra 3 dimenziós modelljének elkészítése.
- A 14. ábra, 23. ábra, 24. ábra, 39. ábra, 40. ábra és 41. ábra fényképeinek készítése. A 4.1. és 4.2. fejezetekben szereplő képek és ábrák készítése.
- Marvelmind rendszer szoftverintegrációjának () C nyelven írt részei:
 - Adat lekérdező és szolgáltató folyamat.
 - A Python/C API integrációja a C kódba.
- Dolgozat szerkesztése:
 - 1. Bevezetés
 - 2.1. Gépi tanulás
 - 2.2. Felügyelt tanulás
 - 2.3. Neurális hálózatok
 - 2.4. A Donkey® platform
 - 2.5. Marvelmind szenzorrendszer
 - 2.9. B-spline alapú interpoláció
 - 2.10. k-D fa
 - 2.11. Python/C API
 - 3.1. Marvelmind szenzoradatok rögzítése
 - 3.2. Pályadigitalizálás
 - 3.4. Valós idejű pozíciómeghatározás
 - 3.7. Gyűjtött adatok jóságának meghatározása
 - 4. Eredmények, értékelés
 - 5.1. Adatok jósága a sávtartáson túl
 - 5.2. Reinforced Learning

9. Köszönetnyilvánítás

A kutatást és a publikáció elkészítését a Debreceni Egyetem Informatikai Kar Önvezető járművek kutatási és fejlesztési laboratóriuma támogatta. Köszönettel tartozunk **Dr. Kovács László** témavezetőnknek a szakmai támogatásért és iránymutatásért. A közös munka során innovációban, kihívásokban gazdag, nehéz, de ugyanakkor nagyon előremutató, céltudatos, eredményes munkában vehettünk részt, melyhez témavezetőként jelentősen hozzájárult. Ez úton is köszönjük értünk és az Önvezető járművek kutatási és fejlesztési laboratóriumért tett munkáját, mely lehetővé tette a részvételünket is.

Szeretnénk továbbá megköszönni a laborban dolgozó **Mészáros László** és **Girászi Tamás** hallgatóknak azt a sok segítséget, amivel támogatták munkánkat. Tamásnak külön köszönjük a kávé.