TDK-dolgozat

Lázár Tamás

Széchenyi István Egyetem
Gépészmérnöki, Informatikai és Villamosmérnöki Kar

Digital Twins az esportban

Lázár Tamás

(I. éves angol nyelvű programtervező informatikus MSc. hallgató)

Konzulens: Dr. Kovács Ákos
(egyetemi adjunktus)

Széchenyi István Egyetem
2022. November
Győr

Tartalomjegyzék

1 Bevezetés 1
2 Digital Twins 3
2.1 Projekt célja 4
3 Szimulátor 6
3.1 Carla 7
4 Adatok 8
4.1 Current_pose 8
4.2 RollPitchYaw: 8
5 Gyakorlati munka 11
5.1 Autók megjelenítése 11
5.2 Interpoláció 11
5.3 Magasság 11
5.4 Szimulátor irányítása 12
6 Irányítást akadályozó tényezők 14
6.1 Pozíció 14
6.2 Orientáció 16
6.3 Adatfolyam 16
7 Versenyzés 18
7.1 Autók elkészítése 18
7.2 Sebesség 21
7.3 Időmérés 21
8 Konklúzió 22
9 További fejlesztési lehetőségek 23
10 Ábrajegyzék 24
11 Egyenletjegyzék 25
12 Irodalomjegyzék 26

1 Bevezetés

A mai modern felgyorsult világunkban a felhasználóknak vagy fogyasztóknak már nem felelnek meg a régi médiumok, mint például a TV nézés, egyre kevesebbet fogyasztják azt, más platformokat, más eszközöket használnak. A világjárvány számos változást hozott, többek között az emberek ingerküszöbét is megváltoztatta, új fajta ingerekre van szükségük, sokszor már nem tudnak csak leülni és tartalmat fogyasztani. Ezen megállapítás igaz, a sporteseményekre is, például az autóversenyzésre. Motivációm bemutatni egy olyan új, innovatív megoldást, amellyel az emberek az autóversenyzéshez közelebb érezhetik magukat, átélhetik a vezetés közbeni izgalmakat, mégpedig azzal, hogy digitálisan ők is részt vesznek egy valós versenyben.

A dolgozatom célja egy olyan technológia és gyakorlati alkalmazás bemutatása, mely innovatív elemeivel megreformálhatja az autóversenyzést. Hipotézisem, hogy létre lehet hozni egy olyan szoftvercsomagot összekötve egy szimulátorral, amely képes autókat valós időben megjeleníteni egy versenypályán, amelyekkel versenyezni tudunk.

Dolgozatomban bemutatom a Digital Twins (digitális iker) témakörét, elméleti hátterét a kezdetektől, hogy mikortól alakult ki a fogalom, mit tekintünk digitális ikernek, milyen felhasználási területei vannak a technológiának.

Részletezem a Proof of concept verziót, hogy milyen követelményeknek kell megfelelnie a szoftvernek, és mi a víziója az elvégzett kutatásomnak és munkámnak.

Ezek után bemutatom a munkámhoz használt szimulátor követelményeit, és bemutatom az ezen követelmények által kiválasztott szoftvert.

Röviden bemutatom a ROS-t, és az adatátvitel módját, illetve, hogy milyen adatokkal dolgoztam, az ehhez szükséges numerikus konverziókkal együtt, többek között a referenciapontok fontosságát hangsúlyozva.

A dolgozat további részében a gyakorlati munkámat részletezem. Lesz szó az autók megjelenítéséről, az interpoláció fontosságáról, a szimulátor beállításáról és annak irányításáról, illetve, hogy miért nem használtam a magasság szenzor értékeit.

A gyakorlati munkám jelentős része volt az irányítást akadályozó tényezők megértése, az ezzel kapcsolatos problémák elemzése, és a megoldás elsőkörben elméleti, majd gyakorlati implementálása. Problémát észleltem többek között a pozíció és orientáció adatokkal

kapcsolatban, illetve az adatküldés módjában. Ezen problémák detektálása és megértése után számos hibát sikerült kijavítanom.

A munkám során elkészült egy, a szimulátorhoz tartozó kliens szoftver, amelynek elindítása után egy digitális autóval tudunk versenyezni. A digitális autó szintén egyedileg készült el a projekthez, festéssel és az autó mechanikus beállításaival együtt.

A dolgozatom legvégén az összegzett eredmények és a további fejlesztési lehetőségek találhatóak.

2 Digital Twins

A Digital Twins témakör korunk egyik legújabb technológiájának gyűjtőneve, az egyik fő iránya az ipar digitalizációjának, olyan innovatív fogalmak mellé sorolhatjuk, mint Smart City, Big Data, Deep Neural Network, Artificial Intelligence.

A Digital Twins a NASA Apollo programjáig vezethető vissza, ekkor készítettek el az első ikreket, hogy az űrhajósokat tudják segíteni a kritikus manőverekben. Ekkor még nem DT-ként hivatkoztak rájuk. A következő lépcsőfok 2002-ben jött el, ahol Michael Grieves tartott egy előadást a Michigani egyetemen, ahol ugyan még mindig nem fogalmazódott meg a digitális iker fogalma, de az akkori model elemeit a mai ikrek már tartalmazzák.

2010 ben a NASA kiadta az ütemtervét, ahol szerepel, mint fogalom. Ekkortól DT-ként hivatkoztak rájuk, létrejött a digitális iker fogalma.

2014-től a legnagyobb technológiai vállalatok már használják a marketing szövegekben.

Manapság konstans fejlesztés és kutatás folyik ezen téma berkein belül.

A digitális iker definíciójára nincsen egyértelmű válasz, mivel egy olyan technológiai innovációról van szó, amely jelenleg is kutatás és fejlesztés alatt van. Számos különböző megfogalmazást találhatunk, de egységesített, mindenki által elfogadottat nem. A digitális iker minden bizonnyal egy való életbeli objektum minél pontosabb digitális mása. Számos esetben vizuális megjelenítést is alkalmaznak, a könnyebb érthetőség érdekében. Azonban nagyon fontos tényezője minden megfogalmazásnak, hogy milyen „use-case” -re készítik.

Szenzorokkal a modern világban számos dolgot mérhetünk, ezeket le kell szűrni azokra, amelyekre az adott „use-case” -nek szüksége van.

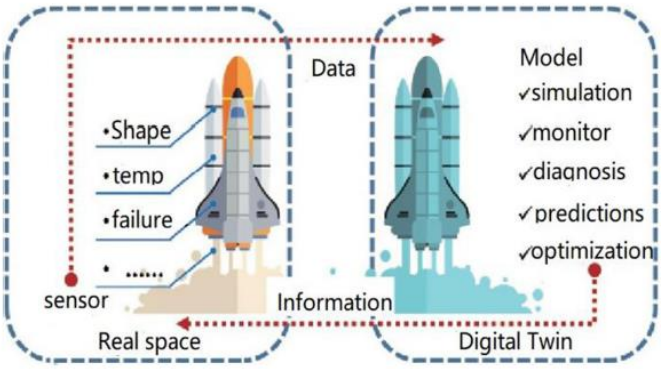
A szenzor adatokat fel kell tudnia dolgozni a rendszernek, és azáltal kell működnie, hogy milyen szenzor adatokat kap. Az adatok érkezhetnek tiszta szenzor adatokként is a rendszerbe, ekkor feldolgozása történhet analitikai modellekkel, de már érkezhet előre feldolgozott adat is a rendszerbe.

A digitális ikrek a következőekre lehetnek képesek

- szimulálják a való életbeli történéseket
- monitorozzák az adatokat és diagnosztizálják a hibákat.
- jósolnak, hogy milyen esemény fog bekövetkezni

- Optimalizálhatják a folyamatokat

Az alábbi képen láthatjuk, hogyan kell elképzelni a digitális iker felépítését adatok tekintetében:



1. ábra A digitális iker adatútja

A bal oldalon található a való világ, a jobb oldalt a digitális ikre. Az adat a szenzorokról beáramlik a szimulációba, ahol a feljebb már említett dolgokat lehet képes végrehajtani a szimuláció. Majd ezek után az információt tudja visszaadni a való világba, ahol ez alapján akár döntéseket vagy cselekvéseket is képesek lehetünk végrehajtani.

A digitális ikreket számos helyen fel lehet használni többek között az iparban, egészségügyi területen, ahol akár egy emberi test működését, és a gyógyszerekre való hatását is lehet szimulálni. Az okos városokkal összekötött fejlesztésekben, ahol egy digitális iker nagy segítséget nyújt például a közlekedés optimalizálása, a forgalom előrejelzése, vagy akár a gyors vészhelyzeti reagálások, tűz esetén.

A bemutatott összefüggések függvényében kijelenthetjük, hogy egy digitális iker az élet számos területére létrehozható, és a jövőben még több iparágban fognak megjelenni ezzel kapcsolatos fejlesztések. [1] [2] [3]

2.1 Projekt célja

A projekt célja elhozni az autóversenyzés egy új formáját, ahol a versenyző vagy valós időben vagy visszajátszva képes versenyezni egy olyan térben vagy szimulációban, ahol a többi ellenfél valós versenyző a pályán. Ebben az esetben képesek vagyunk becsatlakozni egy már futó versenybe, vagy visszajátszani és újra élni régebbi versenyeket. Becsatlakozhatunk akár a későbbiekben a Forma versenysorozat királykategóriájába, a Forma 1-be vagy a WTCR, a World Touring Car Cup versenyébe, és megmérettethetjük magunkat a legjobbakkal. Az ehhez szükséges proof of concept verzió készült el a projekt keretén belül.

A proof of concept verzió a következőképpen épül fel:

- Egy versenypálya digitális másolata, amelyen a versenyzők versenyeznek.
- A való életbeli autókat tudjuk megjeleníteni ezen a versenypályán
- Az autókat meghatározott pozícióra és orientációra tudjuk megjeleníteni
- Legyen egy kliens, amellyel a szerverre csatlakozva ezekkel az autókkal valós időben, vagy visszajátszva tudjunk versenyezni.

Ehhez elsősorban a dolgozatom szempontjából szükséges a:

- versenypálya minél pontosabb digitális reprezentációjának elkészítése
- pályán levő versenyzők digitális, vizuális leképezése
- versenyzők aktuális időpillanatbeli pozíciója és orientációja

A versenyzők megjelenítéséhez szükséges lesz az autók digitális megjelenítése. A digitális megjelenítéshez pedig az autó, esetünkben autók 3D modelljei, az autók mechanikai paraméterei, és a festés.

A versenyzők pozícióját és orientációját egy GPS és egy IMU szenzor szolgáltatja.

A technikai megvalósításhoz szükséges volt még az autók tetejére helyezni egy antennát, amelyre a GPS-nek volt szüksége, az autókba be kellett építeni egy fedélzeti egységet, amely az adatokat szolgáltatta a szervernek, és a kapcsolatért felelt az autók és a szerver között, illetve egy RTK bázis egységet is be kellett konfigurálni, hogy a GPS centiméter pontosan tudja küldeni a koordinátáit.

A fedélzeti egység, az antenna és a bázis összeépítését, konfigurálását a projekten belül nem én végeztem, így ennek a részletes kifejtése nem valósul meg a dolgozatomban.

Az 1. ábrán található 5 tulajdonságból a dolgozatomban részletezett digitális iker képes szimulálni a valós autók pozícióját és megjeleníteni ezen autókat, ezen felül képes a bejövő adatokat monitorozni, és ezen adatok alapján minél valósághűbb megjelenítést eszközölni.

3 Szimulátor

A Proof of Concept verzióhoz, hogy valós időben megtudjam jeleníteni az autókat, szükségem lesz egy szimulátorra. Ehhez a piacon fellelhető szimulátorok közül, azon belül is az open source verziók közül kellett kiválasztanom egyet, amely alkalmas a feladatra. A szimulátornak a következő kritériumoknak kell megfelelni:

1. Szerver kliens architektúrával épüljön fel a szoftver, lehessen egy szerverre akár több, akár távoli klienssel csatlakozni.
2. Rendelkezzen API-val, amelyen keresztül lehet manipulálni az objektumokat, jelen esetben a megjelenítendő autókat.
3. Lehessen koordináták alapján állítani az objektumok pozícióját, orientációját, valós időben
4. Lehessen autót irányítani benne kormánnyal és pedállal. Amennyiben lehet kormánnyal és pedállal irányítani, akkor lehessen egyedi UI-t tervezni és implementálni hozzá.
5. Lehessen egyedi pályát, autót tervezni és beilleszteni a játékba.
6. Az egyedi autó paramétereit lehessen módosítani, ezáltal lehessen több autót is létrehozni különböző paraméterezéssel, amelyeket különböző erősségű pilótáknak oda lehet adni.
7. A szimulátor rendelkezzen egy olyan függvénnyel, amellyel megtudjuk határozni egy adott 3D koordinátán egy megfelelő irányban kiadott láthatatlan lézersugárral, hogy milyen objektumokon halad át, ezzel lehetőséget adva, hogy az autót a pályára „ragasszuk”.
8. Az objektumokhoz rendelhessük hozzá, hogy egy másik objektum fennakadjon-e bennük, vagy ne. Erre azért volt szükség, mivel a versenyzés közben a szimulátorban levő autó, amelyet a pilóta vezet, nem tud a való életben interaktálni az autókkal, ezért szükség van arra, hogy áttudjon haladni rajtuk.

A 7.-es számú kritérium az első pályafelmérés alkalmával fogalmazódott meg, amikor a használt GPS magasságbeállításai nem voltak megfelelőek, és az elkészült ROS bagek bizonyos topicjain, amelyek arra lettek kijelölve, hogy használjam őket, nem jelentek meg magasságértékek.

Ezen kritériumokkal megvizsgáltam első körben több játékszoftvert, amelyek érthető módon nem feleltek meg az elvárásoknak, mivel egy játékszoftver ritkán tartalmaz olyan API-t,

amellyel tetszőlegesen kiválasztott programnyelvből lehet interaktálni. Ezután fordultam a ténylegesen open-source szoftverek felé, amelyek közül egy emelkedett ki a mezőnyből. Az open-source szoftverek előnye, hogy a közösség által lettek fejlesztve, így általában olyan keretrendszerekre épülnek, olyan mérnöki megoldásokat használnak, amelyek ipari standardnek számítanak, így könnyedén lehet akár igény szerint módosítani is őket a saját igényeinknek megfelelően.

3.1 Carla

A Carla [4] egy open-source szimulátor, amely Unreal Engine 4-re [5] épül, annak a fizikáját, megjelenítési eszközeit használja. Elsősorban autonóm vezetésre tervezték, ennek megfelelően számtalan lehetőséget biztosít a Carla Ecosystem, olyanokat, mint például forgalom generálása, RGB kamera megjelenítése, Lidar nézet megjelenítése, ROS-Bridge, amellyel össze lehet kötni a szimulátort egy ROS-al, és kétirányú kommunikációt lehet létrehozni.

A szimulátor rendelkezik egy Python nyelven írt API-val, egy széleskörű dokumentációval, és számtalan példa programmal, amelyek együttes használatával könnyedén lehet olyan python scripteket írni, amelyekre szükségem van.

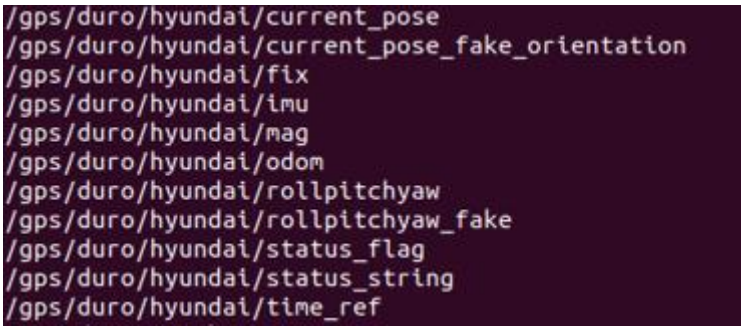
A szimulátort lehet bináris verzióként is használni, illetve lehet Windows és Linux rendszerre source-ból felbuildelt verziót is létrehozni. Ebben az esetben, mivel a pálya, a környezet, és a különböző beállítások előreláthatólag módosításra fognak kerülni, én a Linux rendszeren sourceból buildelt verziót használtam. A Linux rendszer alapvetően megkötés volt, mivel a ROS csak ezen operációs rendszer alatt működik megfelelően.

Az Unreal Engine széleskörű beállításainak és támogatottságának köszönhetően egyszerűen lehet egyedi pályát és egyedi objektumokat hozzáadni, az autók paramétereit könnyedén lehet változtatni. Az autók transzformálására ad lehetőséget az API, gyakorlatilag egy teleportálást hajt végre egyik frame-ről a másikra. Van olyan példa program, amellyel manuálisan tudunk vezérelni egy járművet.

Alapvetően szerver-kliens architektúrában működik a szoftver, akár azonos hálózaton belüli másik gépről is könnyedén el lehet érni IP-cím alapján. Van olyan függvény, mellyel megtudjuk határozni egy koordinátából kiindulva bizonyos irányban és távolságra, hogy milyen objektumok találhatók. Legvégül pedig betudjuk állítani, hogy a manuális irányítás közben milyen pálya elemeken vagy a többi autón fennakadjon-e a mi autónk.

4 Adatok

A ROS (Robot Operating System) egy nyílt forráskódú keretrendszer, amelynek segítségével robotikai alkalmazásokat készíthetnek a kutatók vagy fejlesztők. Az információ topikokon és node-okon keresztül áramlik, ahol vannak publisherek, akik bizonyos topikokra küldenek adatokat, és listenerek vagy subscriberek akik bizonyos topikokon fogadják ezeket. A dolgozatomban ezt a rendszert az adatok fogadására használtam, ezen keresztül érkeztek a GPS adatok az autókról, 10Hz gyakorisággal.



2. ábra ROS topicok

Az autókban elhelyezett fedélzeti egység által biztosított adatok ezeken a topicokon keresztül érkeztek meg a szerverre. A számomra releváns adatok a current_pose és a rollpitchyaw topicban találhatóak.

4.1 Current_pose

UTM koordinátarendszerben X, Y, Z értékeke érkeznek a topicra, a Z érték a magasságot jelöli, melyet egyedül a pályafelmérés alkalmával és a pálya kialakításánál alkalmaztunk a projektben. Az X és Y koordináták pedig megfelelnek a Hungaroring pálya UTM koordinátáinak.

A topic a geometriai üzeneteken belül a Point.msg által meghatározott szintaxist követi:

```
# This contains the position of a point in free space
float64 x
float64 y
float64 z
```

3. ábra A Point.msg szintaxisa

4.2 RollPitchYaw:

Az adatok radiánban érkeztek, azonban a szimulátornak fokokban kellett megadnom az autó ezen értékeit. Ezt a következő képlettel számoltam át:

$$\deg(x) = rad(x) * (\frac{180}{\pi})$$

1. egyenlet Fok számítása radiánból

A $rad(x)$ a fok érték radiánban, a $\deg(x)$ pedig a számításból adódó érték fokban. Az így eredményként kapott fok értékeket már könnyedén betudtam állítani az autókat.

A topic a geometriai üzeneteken belül a Vector3.msg által meghatározott szintaxist követi, amely megegyezik a current_pose topic szintaxisával. Az X, Y, Z értékek rendre a Roll, Pitch és Yaw értékek radiánban.

A pozíció adatoknál nem, de az orientáció adatoknál egy meghatározott konstans értéket hozzá kellett adni ezekhez a fok értékekhez. A yaw értékek esetén a konstans -90, a pitch értékek esetén pedig az egyik autónál 3, míg a másiknál nem kellett hozzáadni. Erre azért volt szükség, mivel a fedélzeti egységet egy meghatározott helyzetben tudtuk rögzíteni, és ezek a konstans értékek szükségesek voltak a szimulátorban való megfelelő megjelenítéshez.

A feldolgozást a rospy modullal végeztem, amelynél egyszerűen a kiválasztott topikokra fel kellett iratkozni, és amelynél egy callback függvény hívódott meg minden alkalommal, ahányszor frissítés érkezett arra a topicra. A callback függvények értékeit több globális változóban tároltam, amely periodikusan beállította az autók helyzetét, a pozícióját és az orientációját.

A GPS modul biztosított UTM koordinátarendszerű értékeket, és lat lon koordinátarendszerű értékeket. Annak érdekében, hogy a sok tizedesjegy során ne vesszen el adat, dolgozatomban az UTM koordinátarendszerben érkező adatokat használtam, amelyet egy lineáris transzformációval alakítottam át, hogy a szimulátorban levő koordinátarendszerhez alkalmazkodjon. Ehhez szükségem volt egy valós életbeli koordináta digitális másolatára, amelyeket egymás, különböző koordinátarendszerekben található referenciapontjaiként értelmeztem. Mind az UTM, mind a játékbeli koordinátarendszer méter alapú, így a valós életbeli referenciaponttól való eltérés pontosan ugyan annyi, mint a szimulátorban meghatározott referenciaponttól. Ezen szimulátorbeli referenciapont meghatározása azonban nem triviális, a minél pontosabb elhelyezéshez segítségül vettük azon, a projekt során elkészült, rosbagek adatait, ahol a GPS-el felszerelt autó a pálya két oldalán (külső és belső ív) végig ment.

Az így összeálló diszkrét pontok centiméteres pontossággal adják meg a való életben az aszfaltcsík oldalait, melyből elkészülhetett a digitális másolata. A pálya digitális másolatának

létrehozásánál figyelembe vettük az autó orientációjának értékeit is, hogy minél pontosabb másolatot készítsünk. Ezek után iteratívan került meghatározásra a digitális referenciapont, az aszfaltcsík minél pontosabb meghatározásával.

5 Gyakorlati munka

5.1 Autók megjelenítése

A ROS rendszeren keresztül jövő adatok 10Hz gyakorisággal érkeznek meg. Ez azt jelenti, hogy másodpercenként 10-szer kell minimum frissíteni a szerver állapotát. Azonban, ha belegondolunk abba, hogy egy versenyautó akár 180km/h-val is halad a versenypályán, ami 50 m/s sebességet jelent, akkor 5 méterenként egyszer fogjuk frissíteni a pozícióját. Ennek elkerülésére interpolációt alkalmaztam.

5.2 Interpoláció

Az interpoláció jó megközelítést ad, hogy az autó milyen koordinátákon halad két GPS általi frissítési pont között, és növeli azon pontok számát, amelyeken meg kell jeleníteni az autót, így sokkal jobb vizuális élményt nyújtva a felhasználónak.

Az interpolációt a következő módon számoltam ki:

- Megvártam míg érkezik 2 frissítés a ROS topicra.
- Ezen 2 frissítés között X számú pontot határoztam meg, ahol X egy olyan pozitív egész szám, amelyet gyakorlati úton állapítottam meg, mivel több tényező is befolyásolta, amelyek közül a legnagyobb hatással bíró a monitorok száma volt.
- A pontok meghatározásánál kiszámoltam x, y koordinátarendszerben az irányvektort, majd ennek a vektornak a hosszúságát X résszel elosztottam, amely eredményezett X+1 olyan pontot, amelynél pozíció és orientáció értékekre egyaránt meg kellett jeleníteni az autót.

A kritérium az autók megjelenítésénél, hogy nem „csúszhatnak”, azaz nem lehet a számolás és megjelenítés, belevéve a szimulátor frissítési idejét is, 100 ms feletti idő. Ezen kritérium segítségével került meghatározásra az interpolált pontok száma.

Elsősorban az interpoláció segítségével oldottam meg azt a problémát, hogy az autók „ugrálnak”. Az „ugrálás” a vizuális megjelenítésnél fordult elő, mivel az autók túl nagy távolságot tettek meg két frissítés között.

5.3 Magasság

Ahogy azt dolgozatomban már említettem a ROS rendszeren keresztül érkező magasság értékeket csak a pálya készítéséhez, a pálya felvételezéséhez használtuk. Ennek oka az volt,

hogy a magasság értékek, amelyeket a GPS biztosított, nagyban függött attól, hogy milyen magas járműre szerelték rá az eszközt. A jármű magassága ebben az esetben egy konstans értéket jelentett, amelyet minden esetben hozzá kellett adni, vagy éppen ki kellett vonni a GPS-ről jövő adatok tükrében. Több fajta beállítása volt a GPS-nek, ahogy a magasságot számolta, az egyik ilyen például a tengerszint feletti magasság. A kezdeti tesztek során nagyon gyorsan kibukott, hogy ez a megoldás nem megbízhat, több gondot is okozott, és a konstans értéket folyamatosan változtatni kellett. A megoldás ebben az esetben a szimulátor kritériumaiban megfogalmazott cast ray, amellyel megtudjuk határozni egy pozícióban az objektum magasságát. Amennyiben ezt a függvényt egy bizonyos X, Y koordinátára lefuttatjuk, megkapjuk az aszfaltcsík magasságát, amelyhez betudjuk állítani az autót, vagyis letapasztottuk a földre. Külön érdekesség, esetünkben fellépő hiba volt, hogy az autó ugyan kifejezetten nagy sebességgel, de még szemmel láthatóan oszcillált 2 érték között, mivel a lézer a szimuláció során áthaladt az autón is. A megoldás erre a problémára, hogy a kapott objektumok magasságát összevetve mindig a legalacsonyabbra állítottam be az autó magasságát.

5.4 Szimulátor irányítása

A kiválasztott szimulátor szerver-kliens architektúrával épül fel, így alapvetően aszinkron módban futnak. Az aszinkron mód azt jelenti, hogy külön szálakon az erőforrások operációs rendszer általi eloszlásából adódóan maximális sebességen futnak. Ezt az aszinkron módot állítottam át szinkron módú futásra.

A szinkron módú futás előnye, hogy az szimulátorhoz tartozó API segítségével lehet irányítani, hogy mikor számolja ki a következő updatet, tehát irányítani lehet, hogy a szerver mikor álljon le, és mikor induljon el. Ilyen „tick”-ek meghívására van szükség minden autó beállításakor, ezzel irányítva a szimulátort.

Azt tapasztaltam, hogy amennyiben aszinkron módban futnak a kliensek és a szerver, akkor az autók megjelenítése akadozik. Abban az esetben is fellép ez a fajta akadozás, ha nem minden update-nál változik az autó pozíciója, hanem minden másodiknál, így külön figyelmet kellett annak szentelni, hogy szinkron módban mind a szimulációs környezet, mind pedig a szimulációt irányító scriptek ügyeljenek ennek a kritériumnak a minél pontosabb betartására.

A szimulációt irányító/manipuláló program beállításánál arra is ügyelni kellett, hogy minél közelebb legyen az autók beállításához szükséges periódusidő a 100 ms-hez, mivel, ha ez az érték 90 ms, akkor körülbelül másodpercenként fellép az az eset, amikor az autónak be

szeretnénk már állítani azt a pozíció és orientáció értéket, amely még nem érkezett meg a ROS-on keresztül. Ez pedig vizuálisan azt eredményezi, mintha az autó másodpercenként ugrana egyet, holott ebben az esetben csak az autó pozíciójának frissítése állt le 1 periódusidőre.

A „tick” -ek meg száma egy másodperc alatt meghatározza az FPS számát is a szimulátornak. Ezen érték nagyjából 60 FPS, ha aszinkron módban futnak, szinkron módban ennél kevesebbet sikerült csak elérni.

Az autóknál a rájuk ható fizikát ki kell kapcsolni, mert különben a szimulátor fizikája az autók teleportálgatásánál egy akadályozó tényező, mivel befolyásolja az autók pitch, roll és yaw értékeit, amelynél vizuális megjelenítésben az autó véletlenszerű fordulását tapasztaljuk mindhárom koordináta tengely irányába.

6 Irányítást akadályozó tényezők

A szimulátor irányításánál és az adatok beérkezésénél vagy feldolgozásánál számos akadályozó tényezővel találkoztam. Ezen tényezőket valamilyen módon le kellett kezelni, mivel a szimuláció működésénél komoly problémát okoztak.

A szimuláció működésénél, a szimulátor irányításánál kiemelten fontosak a szenzorokról érkező adatok minősége. Hibás vagy nem megfelelő adatokkal a szimuláció működése megbízhatatlanná, instabillá válhat. A nem megfelelő minőségű adatokkal való probléma mind a pozíció mind az orientáció esetén számos alkalommal fellépett, olykor akár 1 körön belül is.

6.1 Pozíció

A Hungaroring pálya nem sík, a legmagasabb és legalacsonyabb pontja között több mint 30 méter található, illetve számtalan helyen valamilyen objektum alatt vagy magas épület közvetlen közelében haladunk el. Ennek az egyik következménye, hogy a GPS elveszti a Fixed RTK jelét, és a megfelelő centiméter pontos adatok helyett, egy, az elvesztés helyétől számított fals pozíció helyet küld a szimulátornak, amelyet az eddig sebességből, és irányból jósol. Ez a fajta hiba a legszembetűnőbb, mivel ugyan tudjuk, hogy való életben az autó nem ment át a pályát határoló gumifalon, a szimulátorban mégis ez történik. Ez a fajta hiba, ahol az autó lemegy a pályáról, a pálya csak egy bizonyos szakaszán fordult elő, és az autó visszanyerte eredeti pozícióját, amint a GPS visszaállt Fixed RTK státuszba. A hibás GPS adatok utáni visszaállás során azonban egy másik fellépő hibával szembesültem, mégpedig azzal, hogy az autó, a szimulátorban való teleportálásnak köszönhetően gyakorlatilag visszaugrik a helyére. A visszaugrás 2 pozíció üzenet között történik meg, amely olykor pár száz méter különbséget jelent a 2 üzenet koordinátái között.

A GPS státusz változására, és a hibás, vagy nem megfelelő minőségű pozíció adatokra a szimulátor oldaláról az adatok tekintetében detektálásra nincs lehetőség. Ellenben, amikor visszanyeri eredeti pozícióját, az detektálható. Ennek köszönhetően a feljebb említett hibák közül az egyiket sikerült megoldani, az alábbi módon:

Folyamatosan monitorozom az autó UTM koordinátarendszerbeli értékeit x és y koordinátára nézve. Ezen monitorozásból fakadóan minden üzenetre megvan az előző üzenetre vetítve a delta érték. Ezen delta érték, amennyiben legalább kétszer nagyobb, mint az előző üzenet delta értéke, akkor elméletben egy ilyen ugrást detektáltunk. Ha belegondolunk, hogy az autó

pozíciójáról 10 Hz gyorsasággal kapunk frissítést, akkor láthatjuk, hogy ez megfelelően gyors ahhoz, hogy a delta értékek ilyen gyorsan ne tudjanak egyik üzenetről a másikra.

A kivétel abban az esetben lép fel, ha az autó pontosan az egyik koordináta mentén mozog, ekkor a delta értékek lecsökkenhetnek egy olyan nullához közeli számra, ahol biztosan fogunk ugrást detektálni, holott az nem történt meg. A fals detektálások, holott logikailag hibásak, a program működését nem befolyásolták. Ennek oka, az algoritmus, amely az ilyen ugrások simításáért felel.

Amennyiben ilyen ugrást detektáltunk, akkor az autót nem állítjuk be a pozícióra, hanem kiszámoljuk a jelenlegi és a jövőbeni pozíciók közötti delta értékeket x,y koordinátára, és az ötvened részét mindig hozzáadjuk, majd csökkentjük ezeket a delta x és y értékeket a hozzáadott rész értékével.

Az ötvened részt meghatározása iteratívan történt, a szimuláció újraindításával és az autók pozíciójának, ugrásokra adott válaszok vizsgálatával. Ezen érték alatt számos esetben az autó lecsúszott a pályáról, olyan esetekben is, amelyeknél eddig nem.

Az alábbi kép szemlélteti az előbbiekben részletezett megoldást:

4. ábra Az autó ugrásának interpolálása

Az egyenes vonal jelzi, hogy az autó milyen útvonalon haladt az ugrás előtt, a szaggatott vonal, hogy az ugrás után milyen útvonalon mozog. A kis körök a két vonal között jelzik a delta értékek hozzáadását minden iterációnál. Megoldásomnak köszönhetően az szenzor adatokról érkező pozíció értékekre nem csak beállítja az autót, hanem leinterpolálja a különbséget, és pár másodperc alatt az autó elfoglalja a valós helyét.

6.2 Orientáció

Amennyiben az orientáció szenzor hibás adatokkal szolgál, az hasonló módon feltűnő, mint a pozíció szenzor esetében. Ebben az esetben az autónál azt tapasztaljuk vizuálisan, hogy nem megfelelő irányba fordul, számos esetben akár 90 vagy 180 fokot is tévedhet.

A hibás adatokatszenzor oldalról több, különböző változó is befolyásolhatja az egyik ilyen például a hőmérséklet. A szenzor túlmelegedése esetén gyakran fellépett nem megfelelő minőségű adat.

Azonban ideális körülmények között is számtalan alkalommal tapasztalható, hogy ugyan csak pár üzenet erejéig, de hibás értékeket kap a szimulátor, és az autó rendellenes állapotba fordul, akár 100 km/h feletti sebességnél. Ezen értékek könnyedén detektálhatóak, a megoldásomban vizsgáltam, hogy hány fok volt a legutolsó yaw értéke az autónak, és amennyiben a delta érték a mostani és a legutolsó érték között nagyobb volt mint 30 fok, akkor az autó az utolsó olyan orientáció értéket kapta beállításul, amely megfelelt a követelményeknek. Ezzel a megoldással kivédhetőek az autó yaw értékeinek ugrálásai. A roll és pitch tengelyen ilyen ugrálásokat nem tapasztaltam.

6.3 Adatfolyam

A ROS rendszeren keresztül jövő adatoknál előfordul, hogy a koordináták és az orientációk nem egyszerre vagy időben kis távolságra érkeznek meg egymástól, hanem olykor egy teljes másodpercnyi csúszás is van a 2 ugyanabban az időpontban felvet adat között, amely két dolgot okozhat:

- Az autó előbb kanyarodik, mint ahogy az autó elérné a kanyart.
- Az autó később kanyarodik, mint ahogy az autó elérné a kanyart.

Ha előbb kanyarodik, akkor az autó „driftelését” tapasztaljuk, amely ugyan vizuálisan szemet gyönyörködtető, de nem felel meg az elvárásainknak a rendszerrel kapcsolatban. UDP adatküldés esetén az is előfordulhat, hogy egyes csomagok nem érkeznek meg, illetve az is lehet, hogy a fedélzeti egység nem szolgáltat adatot egy nagyon rövid ideig, amely a példának okáért 1 másodperc. Mivel az adatkapcsolaton keresztül folyamatosan áramlik a szimulációba az adat, és a 2 üzenet között eltelt idő 99%-ban megegyezik az autók pozícionálásának az idejével, ezért bizonyos üzenetektől meg kell szabadulni annak érdekében, hogy a feltorlódott üzenetek problémáját megoldjam.

Ezt a fajta problémát néhány esetben sikerült orvosolni, azonban ez sem védte ki száz százalékban az előforduló hibát. A probléma orvosolására a megoldásom az volt, hogy folyamatosan monitoroztam, hogy mely topicon mikor érkezett legutoljára adat. Amennyiben az egyik autónál azt tapasztaltam, hogy a current_pose topicra érkezik adat, de a rollpitchyaw topicra nem, akkor current_pose topicra jövő adatot egyszerűen eldobtam, nem jelenítettem meg az autót, nem töltöttem fel a listát ezen értékekkel. A határértéket, mivel 100 ms az elméleti adatküldés gyakorisága, 200 msben határoztam meg. ezen megoldás gyakorta kivédte a felmerülő hibát, de szükség volt egy további megoldásra is.

A feljebb említett ugrások detektálásánál végeztem el az alábbiakat:

Megnéztem, hogy milyen hosszú üzenetlista van még hátra, rendre a pozícióra és orientációra. Amennyiben az egyik hosszúsága eltért a másiktól, tehát vagy az orientáció nem felel meg a pozíciónak, vagy a pozíció az orientációnak, akkor a hosszabb üzenetlista elemszámát annyival csökkentettem, a lista elejéről, hogy a frissebb üzenetek maradjanak meg, amennyi a két lista közötti elemszám. Eredményként előállt 2 ugyanakkora hosszúságú tömb, ahol az üzenetek közel ugyanazon időpillanatból származnak, ennek megfelelően, amennyiben a szenzor adatok megfelelnek a valóságnak, a vizuális megjelenítés is megjavult.

7 Versenyzés

A POC verzió egyik része az autók megjelenítése, és a szimulátor, vagyis a szimuláció olyan módon való vezérlése, hogy minél pontosabb képet kapjunk egy digitális környezetben a valós autók pozíciójáról, orientációjáról. A másik része egy digitális autó irányítása pilóta vagy versenyző által, amely képes versenyezni a való világban haladó autókkal a szimulátorban.

7.1 Autók elkészítése

A szimulátorhoz hozzá kellett adni 2 olyan egyedi autót, amely kinézetre pontosan megfelel a valós autóknak.

Az autó digitális váza rendelkezésre állt, így nem kellett a nulláról elkezdeni egy járműmodell fejlesztését, a grafikai elemek megegyeztek a valósággal. A feladatom a minél pontosabb leképezése volt az autó festésének, a díszítésének, a matricáknak. Az Unreal Engine-ben ezt egy robbantott ábra segítségével lehet megoldani.

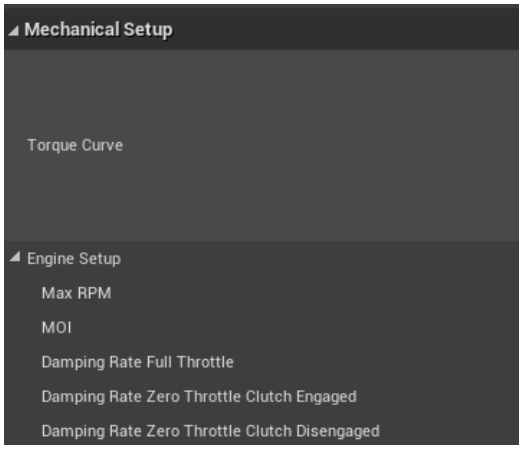
5. ábra Bal oldalon a Hyundai, jobb oldalon a Suzuki ábrája látható.

A következőekben két feladatot párhuzamosan végeztem. Az egyik feladat egy olyan kliens program írása volt, amely a Carla-hoz tartozó példaprogram segítségével tud csatlakozni a szerverre, megadott helyre beilleszt egy autót, amelyet kormánnyal és pedállal tudunk vezérelni. Az ezzel párhuzamos feladat, hogy az autó digitális paramétereit minél jobban illesszük hozzá a valós autó paramétereihez, úgy, mint végsebesség, a gyorsulás, a tapadás, a fékek erőssége, a hajtott tengelyek száma.

A szimulációt biztosító grafikus motor számos lehetőséget biztosított ezen paraméterek beállítására, amelyet iteratívan állítottam és a tesztelések során felvett rosbageken teszteltem. Eredményképpen előállt egy olyan digitális autó, amely nagymértékben hasonlít a valós autó paramétereihez, ezzel egy valós vezetési élményt nyújtva a pilótának, hiszen amennyiben a

szimulátorban leszeretné győzni az autókat, akkor nem, vagy csak nagyon kicsi hibákat véthet a futam, vagy akár csak körök során.

Az alábbi képen láthatóak az autó bizonyos paraméterei, melyeket az Unreal Engine-n belül lehetett állítani:



6. ábra Az autó mechanikai paraméterei

A Torque a forgatónyomatéka az autónak, amelyet a fordulatszám függvényében egy grafikonnal lehet beállítani. Ezen felül a csillapítási tényezőt, és a maximum fordulatszámot lehetett beállítani az Engine Setupon belül. Az autónak a fentebb említett paramétereken felül számos beállítási lehetősége van, azonban ezen beállítások megértéséhez egy jóval mélyebb, és összetettebb tudás szükséges. Az autót automatikus váltóval használtuk, a jobb játék élmény érdekében.

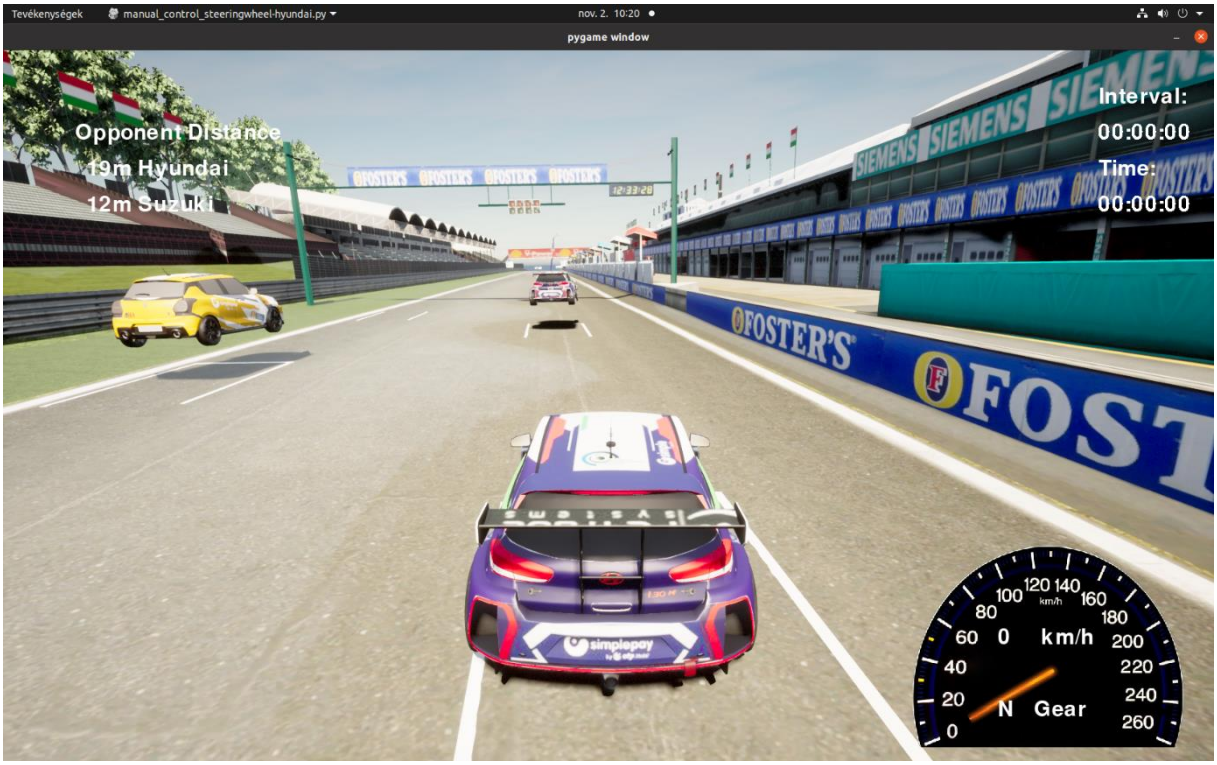
A legnagyobb nehézséget az okozta, hogy a szimulációnak van egy úgynevezett frissítési ideje, melyet az autó paramétereinek első körös beállítása után találtam meg. Ezt a frissítési időt a szimuláció periódus idejének a gyorsaságaként lehet értelmezni a legjobban, tehát hogy mennyi idő telt el 2 szimulációs számítás között. Ez a paraméter kritikus a szimuláció szempontjából, hiszen a teljes működésre hatással van. Két fajta módú értéket vehet fel:

- Variable time-step
- Fixed time-step

A Fixed time-step esetén 2 szimulációs számítás között eltelt idő fix. Az értéket tetszőlegesen bármekkorára változtathatjuk, így egy olyan autóval, amelynek a paraméterei jóval gyengébbek, ugyanazt, vagy hasonló eredményt lehet elérni. Azonban ennek a paraméternek a változtatásával olyan rendhagyó viselkedést lehet előidézni a szimulációnál, amely indokolta azt, hogy a dolgozatomban a variable féle verzióját használtam. Az első indok, hogy az autó

paramétereit a variable féle szimulációhoz igazítottuk hozzá, így a fixed time-step valószínűleg csak ront a helyzeten. A második, hogy a kliens script paraméterei is a variable time-step féle verziónál helyesek. Ez elsősorban az autó sebességénél látványos, túl alacsony fixed time-step esetén az autó sebessége, és a sebességértéke, amelyet a kliensnél láthatunk, túl alacsony a valós sebességéhez képest.

A kliens, amely az autót irányítja a következőképpen néz ki:



7. ábra A kliens képe

Ezen kliens irányításában kismértékben, a grafikus elemeken nagymértékben változtattam. Hozzáadtam egy sebességmérőt, időmérőt, és az autók távolságának mértékét méterben.

A sebességmérő kiválasztásánál figyelembe vettem, hogy fokok szerint lehessen beállítani a sebességet, tehát a 0 és a 90 km/h közötti szög 90 fok legyen.

Az többi versenyző távolságát, legyen az való életbeli, vagy digitális pilóta, megjelenítem minden kliens esetében. Ez a távolság légvonalbeli távolság, amely a kanyarok esetén ugyan mutathat nem olyan pontos értéket, de az esetek nagy részében megfelelő a kiíratás.

A kép baloldalán látható az autóktól mért távolság méterben megadva. A jobb oldalán az időmérés megjelenítése, az Interval a legutolsó mért kör, a Time az aktuális kör. Mivel a versenyző még nem haladt át a rajtvonalon, ezért még nulla értékek jelennek meg. A jobb alsó sarokban látható az egyedileg készített sebességmérő.

7.2 Sebesség

A sebesség értéket a szimulátoron belül az egyes koordináta irányába történő sebességekkel lehetséges kiszámolni.

A képlet a következő:

$$V = 3.6 * \sqrt{X^2 + Y^2 + Z^2}$$

2. egyenlet A sebesség számítása

Ahol az egyes koordináták irányába való sebesség négyzetösszegének a gyöke szorozva 3,6-tal megadja az adott autó sebességét. A szimulátoros autó sebessége variable time-step esetén jó közelítést ad a valós autó sebességéhez. A projekt során számtalanszor végeztünk pályafelmérést különböző okokból, mint az 5G-s kapcsolat gyorsasága, a hálózat lefedettsége, vagy a pálya minél pontosabb meghatározása GPS segítségével. Ilyen pályafelmérések alkalmával meghatározott sebességgel haladt az autó, mely a felmérést végezte, és ezzel tökéletesen lehetett igazolni a számolás pontosságát.

7.3 Időmérés

Minden fajta versenyzéshez szükséges az időmérés a szimulátoron belül. Az időmérés esetünkben a rajtvonalon való áthaladáskor kell, hogy induljon vagy hogy újra induljon, megjelenítve az előző kör mért idejét.

Megoldásomban meghatározott számú objektum felett kell, hogy áthaladjon az autó, ekkor indul el/indul újra az időmérés. Az objektumok száma 10, melyet egyedileg helyeztem el a rajtvonal alatt. A pályán sehol máshol nem található 10 objektum közvetlen egymás alatt, kivéve ezen a meghatározott vonalon. Az objektumok felett való áthaladást az unreal engine cast ray funkciójával ellenőriztem, mely már a szimulátor kiválasztásánál is egy kritérium volt.

8 Konklúzió

Létrejött egy digitális iker szoftver, amely képes bármilyen rendelkezésre álló versenypályán megjeleníteni objektumokat, legyenek azok a dolgozatomban tárgyalt autók, vagy más valós objektumok.

A bemutatáshoz szükséges autók digitális reprezentációja elkészült, a paraméterek minél pontosabb beállításával a dolgozatban tárgyalt megoldással.

A szoftver képes megjeleníteni objektumokat pozíció és orientáció adatok alapján, a versenyzéshez szükséges pontossággal megfelelő GPS kapcsolat esetén.

A szoftver a nyers GPS adatokat képes átalakítani valós és digitális referenciapontok segítségével digitális koordinátákká.

Az autók megjelenítéséhez szükséges volt az adatok interpolációja, melyet szintén a dolgozatomban tárgyalt megoldással oldottam meg.

A szoftver, a szimuláció irányítását akadályozó vagy befolyásoló tényezőket sikerült megfelelő szint alá hozni, kivéve egyetlen esetben, amikor a GPS szenzor adatai olyan mértékben nem feleltek meg a valóságnak, hogy azt szoftveresen a szerverről nem lehetett megoldani, erre megoldási javaslat készült egy jobb pontosságú GPS kipróbálására, amelynek a frissítési ideje is nagyobb a mostani 10Hz-nél.

Elkészült egy kliens szoftver, amellyel csatlakozni lehet az elindított szerverre, versenyezni lehet a szimulátorban, attól függetlenül, hogy a való életből érkezik-e szenzor adat vagy nem.

Elkészült a pályához az időmérés, így még teljesebb vezetési élményt adva.

A hipotézisem teljesült, a GPS Fixed RTK jelének elvesztéséből fakadó hibán kívül az autók vizuális megjelenítése megfelelő, és egy klienssel csatlakozva mi is részt tudunk venni a versenyben.

9 További fejlesztési lehetőségek

A digitális iker továbbfejlesztésével kapcsolatban több irányba is el lehet indulni.

Az első javaslatom új, jobb pontosságú GPS modul kipróbálása, hogy megoldódjon a fennálló hiba a GPS-el kapcsolatban.

Javítanám a vezetési élményt, a szimulátorban levő autó paramétereinek további állításával, illetve adnék visszajelzést a felhasználónak, force-feedback formájában. Úgy gondolom ez javíthatná a vezetési élményt.

A digitális iker bővítésénél megpróbálnék egy másik olyan szimulátort, ahol az autó több paraméterét lehet beállítani, mint például a rugók megnyúlása. Ezen szenzor adatokat CAN üzenetek formájában az autó CAN bus-áról lehet megszerezni, és amelyek most is rendelkezésünkre állnak, azonban a Carla nem adott lehetőséget ezzel kapcsolatban állításra.

Megpróbálnék egy valós autót vezérelni, úgy, hogy csak a szimulátorban levő autó pozícióját nézem, távoli vezérléssel 5G kapcsolaton keresztül, ez azonban még jóval több kutatást és fejlesztést igényel.

10 Ábrajegyzék

1. ábra A digitális iker adatútja 4
2. ábra ROS topicok 8
3. ábra A Point.msg szintaxisa 8
4. ábra Az autó ugrásának interpolálása 15
5. ábra Bal oldalon a hyundai, jobb oldalon a suzuki ábrája látható. 18
6. ábra Az autó mechanikai paraméterei 19
7. ábra A kliens képe 20

11 Egyenletjegyzék

1. egyenlet Fok számítása radiánból 9
2. egyenlet A sebesség számítása 21

12 Irodalomjegyzék

[1] J. a. L. Z. Guo, „Application of Digital Twins in multiple fields,” *Multimedia tools and applications,* pp. 1-27, 2022.

[2] P. van Schalkwyk, S. Dr Malakuti és S. Lin, „A short introduction to digital twins.,” *Ind. Internet Consortium J. Innov,* 2019.

[3] H. van der Valk, H. Haße, F. Möller, M. Arbter, J.-L. Henning és B. Otto, „A Taxonomy of Digital Twins.,” in *AMCIS,* 2020.

[4] A. Dosovitskiy , G. Ros , F. Codevilla , A. Lopez és V. Koltun, „{CARLA}: {An} Open Urban Driving Simulator,” Mountain View, California, USA, 2017.

[5] E. Games, „Unreal Engine,” 25 április 2019. [Online]. Available: https://www.unrealengine.com.