Kevert valóság fejlesztési környezet autonóm járművek számára

A cikkben egy olyan kevert valóságot alkalmazó keretrendszert mutatnak be, amelyben egy valós tesztjármű és több valós objektum is egyszerre kezelhető, miközben mindezek köré a virtuális valóságban virtuális forgalmat generálnak 3D megjelenítésben.

DOI: ***https://doi.org/10.24228/KTSZ.2021.3.2***

Szalai Mátyás – Dr. Tettamanti Tamás

egyetemi tanársegéd egyetemi docens

Budapesti Műszaki és Gazdaságtudományi Egyetem, Közlekedés- és Járműirányítási Tanszék

e-mail: szalai.matyas@mail.bme.hu, tettamanti@mail.bme.hu

1. BEVEZETÉS

Napjainkban a járműgyártással kapcsolatos fejlesztések egyre nagyobb arányban fókuszálnak az önvezető képességek megvalósítására [1]. A fejlesztések fő célja a teljes önvezetés. Ennek elérése azonban egy hosszú folyamat. Még számtalan szimuláció, teszt végrehajtása és validációs eljárás kidolgozása szükséges ahhoz, hogy a felhasználók számára is teljes körben elérhető legyen az önvezető jármű.

Egy önvezető jármű - felépítése szempontjából - több architekturális rétegre osztható. A főbb rétegek: az érzékelési, a döntési, a navigációs, valamint a beavatkozó réteg [2]. Ezek a rétegek az általuk megvalósított különböző funkciók miatt eltérő tesztelési és validációs eljárásokat igényelnek. Ahhoz, hogy ezeket mind szimulációs úton tesztelni tudjuk, számos szimulációs szoftver áll rendelkezésre. Léteznek szimulátorok jól definiált szenzormodellekkel, amelyek kimenete bemenetként szolgálhat az érzékelési réteg számára (pl. Carla, rFpro, Vires VTD, AirSim, PreScan). Járműdinamikai szimulátorokat használhatunk a trajektóriatervező algoritmusok tesztelésére (pl. CarMaker, CarSim). A forgalomszimulációs szoftverek pedig a közlekedési rendszerekben való viselkedés fejlesztésében lehetnek segítségünkre (pl. SUMO, Vissim). Ahhoz, hogy minél több réteget és ezáltal funkciót egyidejűleg is tesztelni lehessen, több szoftver együttes alkalmazása szükséges (ez az ún. „co-simulation").

Napjainkban már több olyan szimulációs szoftver is létezik, amely valamely funkciókat együttesen tesz elérhetővé [3], [4], [5], [6]. Az elérhető szoftvereken kívül pedig számos kutatás eredményei takarnak megoldásokat több szoftver együttes alkalmazására szimulációs célból. [7] valós tesztjármű és Vires VTD segítségével a virtuális térben elhelyezett objektumokat sötétben megvilágító szimulációs rendszert mutat be. [8]-ban egy olyan vezető központú szimuláció kerül megvalósításra, amelyben virtuális forgalom került generálásra SUMO mikroszkopikus forgalomszimulátor [9] és Unity 3D játékmotor segítségével. [10] egy olyan rendszert mutat be, amelyben a DYNA4 dinamikai szimulátort kapcsolják össze a SUMO forgalomszimulátorral. [11] egy járműirányító algoritmusok tesztelésére fejlesztett rendszert mutat be, amelyben az IPG CarMaker járműdinamikai szoftver kerül összekapcso-

lásra a SUMO forgalomszimulátorral. [12] egy megoldást mutat be arról, hogyan modellezhetünk Vehicular-Ad-Hoc-Network (VANET) rendszereket 3D környezetben. A Carla [3] egy olyan virtuális szenzor készletet nyújt, amellyel már megfelelően lehet érzékelni egy virtuális környezetet ahhoz, hogy az információkkal további teszteket végezhessünk. Ezenkívül lehetőséget biztosít forgalom generálására, tanuló algoritmusok tesztelésére, és rendelkezik ROS (Robot Operating System) interfésszel is. A PreScan szoftver szintén jó lehetőségeket biztosít az ADAS rendszerek teszteléséhez a Model-in-the-Loop (MiL), Software-in-the-Loop (SiL) és Hardware-in-the-Loop (HiL) szimulációs környezeteivel [4].

Az általunk bemutatott megoldás a korábbi kutatásainkon alapszik [13], és a SUMO mikroszkopikus forgalomszimulátorra, valamint a Unity 3D játékmotorra épül. A kommunikációs interfész a tesztjármű és a tesztkörnyezet között Pythonban lett implementálva. A SUMO segítségével tetszőleges közlekedési szituáció modellezhető, előállítható a valós forgalomra jellemző paramétereket tartalmazó virtuális járműforgalom, ami segíti a tesztjármű forgalomban való tesztelését. A Unity segítségével szabadon alkothatjuk meg a virtuális tesztelési környezetet úgy, hogy az teljes mértékben a mi igényeinknek feleljen meg. Mindezek mellett a rendszer lehetőséget biztosít egy valós tesztjármű valós időben és valós tesztkörnyezetben történő szimulációba csatlakoztatására, amelyet így Vehicle-in-the-Loop (ViL) szimulációnak hívunk. A virtuális környezetben ezen valós tesztjármű klónja, azaz digitális ikerpárja kerül definiálásra. A megalkotott rendszer lehetőséget biztosít Scenario-in-the-Loop (SciL) tesztek elvégzésére is [15].

2. A SZIMULÁCIÓS RENDSZER FELÉPÍTÉSE

A szimulációs rendszer fejlesztésekor a fő cél az volt, hogy olyan rendszert alkossunk, melylyel egy valós tesztjármű virtuális környezetben tesztelhető. Mindez a gyakorlatban azt jelenti, hogy a valós tesztjárművünk minden hardverével, szenzorával és a számítógépein futó algoritmusokkal együtt egy valós tesztpályán mozog, önvezető üzemmódban. Ez a járműmozgás kerül digitializálásra, majd a virtuális térben is elhelyezett jármű köré saját igényeink szerint generálhatunk virtuális környezetet. A feladat komplexitását az adja, hogy a tesztjármű nem csak a valóságos tesztkörnyezetet érzékeli, hanem a virtuálisan definiált akadályokat is. Ezen az elven haladva kerül megvalósításra a kevert valóságon alapuló ViL szimulációs környezet.

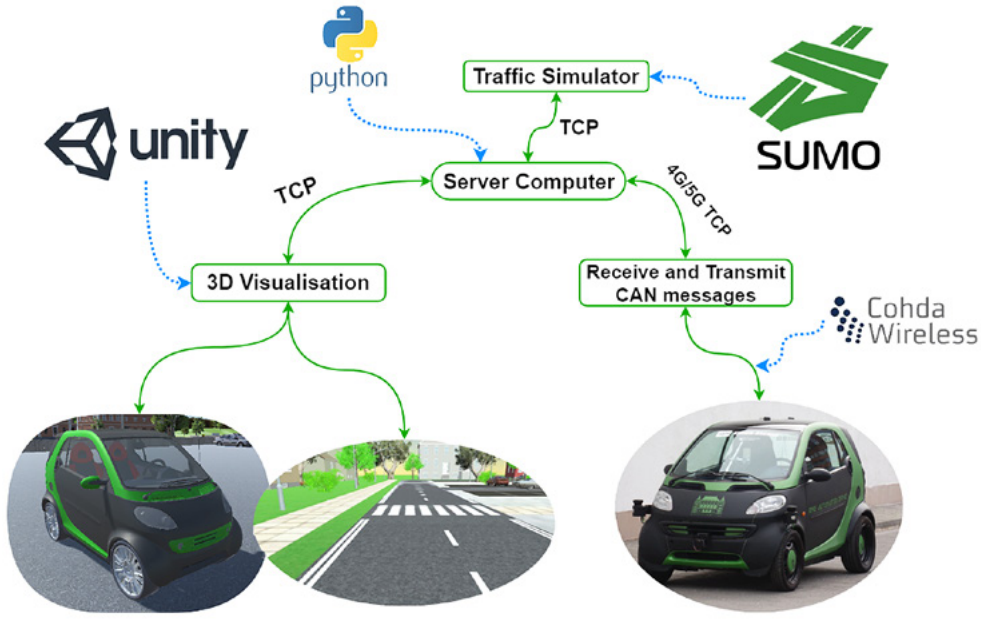
Egy ilyen szimulációs környezet a járművel való valós idejű kommunikációt igényli, amelyre a tesztjármű CAN[1] hálózatán keresztül adódott lehetőség. A járműtől kapott információk feldolgozását követően definiálásra került a valós tesztjármű digitális ikerpárja. Ez a digitális ikerpár a virtuális környezetben mozog, mégpedig a valódi járműtől megörökölt mozgás szerint. A virtuális környezet tartalmazza az összes virtuális szimulációs elemet: az úthálózatot, a forgalmat, az akadályokat, a gyalogosokat, a közlekedési táblákat és forgalomirányító berendezéseket. A virtuális ikerpár környezetét így ismerjük, erre alapozva a mozgásviszony alapján szenzorinformációk csatolhatók vissza a valós jármű felé. A CAN üzenetek dekódolása és a küldéshez megfelelő formátumúvá kódolása Python környezetben történik. A Pythonban megalkotott központi rendszeregység TCP/IP kapcsolaton keresztül kommunikál a forgalomszimulációs egységgel (SUMO), valamint a vizuális megjelenítő egységgel (Unity). A valós tesztjármű és a megalkotott szimulációs rendszer között egy Choda V2X eszközzel került megvalósításra a kommunikáció (CAN üzenetek továbbítása és fogadása rádiófrekvencián, 4G kapcsolaton keresztül). Az így megvalósított keretrendszer az 1. ábrán látható.

2.1. Python szerver

A rendszer középpontjában egy Python környezetben fejlesztett szerver áll. Ez a

1 CAN (Controller Area Network): elterjedt autóipari szabvány járművön belüli kommunikációra, amely lehetővé teszi a jármű különböző logikai egységeinek kommunikációját egymás között, ill. a központi számítógép irányába.

1. ábra: A megalkotott rendszer strukturális felépítése



rendszeregység felelős a járművel való kommunikációért, a vizualizáció és a forgalom generálásának megfelelő működéséért. A szerver felelős továbbá a rendszer egyes elemeinek megfelelő szinkronizációjáért is. A szerver megnevezés itt nem csak mint a megalkotott TCP/IP kapcsolat szerver oldalára utal, hanem mint vezérlő, központi elemre is. Ezen egység feladata a szcenáriók vezérlése, beleértve a különböző virtuális akadályokat is. A virtuális forgalom generálása nem a szerver feladata, de itt szabhatók meg annak paraméterei akár időben való változással is. Az objektumok vezérlésén felül az EGO objektum, azaz a tesztjármű digitális ikerpárja és a további objektumok közötti távolság kiszámítása is itt megvalósított feladat. A kiszámított távolságok alapján adódik lehetőség a jármű valós szenzorait ismerve virtuális szenzorinformációk generálására, amelyeket a jármű CAN hálózatára való küldéssel juttathatunk annak vezérlő számítógépéhez. A megalkotott rendszer alkalmas több valós objektum kezelésére is, így akár több valós jármű, dummy[2] gyalogos, vagy valós forgalomirányító berendezések is csatlakoztathatók hozzá. Ezen objektumok kezelése megegyezik a tesztjármű digitális ikerpárjának kezelésével. A megjelenítés és a távolság értékek számítása során minden valós objektum, amely kapcsolatban áll a szerver számítógéppel a valóságnak megfelelően mozog, és a virtuális világban vett távolsága a digitális ikerpárral a valós távolságnak megfelelő. Ezzel a kiterjesztéssel élve az eddigiekben leírt ViL szimulációs környezet SciL szimulációvá bővül.

2 dummy: Az autóipari tesztek során valós tesztszemélyek helyett ún. dummy tesztbábukat alkalmaznak. A dummy gyalogosnak EuroNCAP (European New Car Assessment Programme az európai autók biztonságával foglalkozó szervezet) által definiált specifikációja van, ugyanis meghatározott radar reflexióval és vizuális megjelenéssel kell rendelkeznie a valósághű tesztelés érdekében.

2.2. Unity 3D – digitális iker, virtuális környezet, objektumok kezelése

A Unity 3D játékmotor használata a vizuális megjelenítés miatt vált indokolttá. A virtuális szimuláció megjelenítése nagyban segíti a szimulációk kiértékelését, és jó lehetőséget biztosít az egyes önvezető funkciók biztonságos környezetben való demonstrálására.

A vizualizációs modulon belül az elsődleges feladat a tesztjármű digitális ikerpárjának virtuális világban való megalkotása volt. A digitális ikerpár megalkotásakor figyelembe kell venni, hogy annak mozgása a virtuális világban a valós jármű mozgásával teljesen azonos kell legyen. Ehhez információkra van szükség a jármű mozgásállapotával kapcsolatban, amelyeket a jármű GPS szenzoraiból, valamint a kormány elfordulását mérő szenzorból kaphatunk meg. Az így kapott pozíció adatok 1-2 cm pontosságúak, amelyek így biztosítják a pontos megjelenítés lehetőségét. Abban az esetben, ha nem állna rendelkezésre ilyen pontosságú információ, becslési eljárások alkalmazásával vagy szenzorfúzióval lehetne elérni a megjelenítéshez szükséges pontosságú adatokat. A kapott adatok azonban a globális földi koordinátarendszer egy pontjára mutatnak, amit valamilyen transzformáció nélkül nem lehet egy korlátos kiterjedésű virtuális térbe illeszteni. Ezért szükséges ezen laterális és longitudinális koordináták átváltása a megjelenítő saját x-y koordinátarendszerébe. A szimuláció során minden alkotóelem ezen az x-y koordinátarendszeren alapszik. A kapott koordináták, állásszög, valamint a jármű kormánykerekének állásából származtatott elkormányzási szög ismeretében már elhelyezhető a jármű a virtuális térben.

A szimulációkhoz szükséges virtuális környezet létrehozása több módon is lehetséges. Ez mindig attól függ, hogy a megalkotni kívánt környezetet egy valós környezet pontos, esetleg kiegészített másaként vagy teljesen szabadon egy új, csak virtuális környezetként szeretnénk megalkotni. A valós környezeten alapuló rendszerek esetében a GPS koordináták adják a környezet alapját. A valós helyszínt leíró longitudinális és laterális koordinátapárokat a SUMO forgalomszimulátorának NETEDIT úthálózat szerkesztőjével konvertálhatjuk át a korábban említett x-y koordinátarendszerbe. Így a forgalomszimulációs modul számára előáll a valós környezet digitális másolata, amely egyaránt rendelkezik a pontok földi, globális és szimulációs lokális koordinátáival is. Ezek alapján már minden pont koordinátája ismert a lokális rendszerben, azok alapján modellezhető a teljes virtuális környezet. Az így kapott környezetet saját igényeink szerint alakíthatjuk, mind a forgalomszimulációs, mind a vizualizációs modulban élve a virtuális valóság adta szabadság lehetőségével. Amennyiben a környezetet teljes mértékben magunk szeretnénk megalkotni, úgy is szükséges a forgalomszimulátor NETEDIT moduljának alkalmazása. Ebben az esetben a valós tesztkörnyezet határoló pontjainak meghatározásához szükséges koordinátatranszformáció alkalmazása. A határoló pontok megállapítását követőn már teljes szabadsággal alkotható meg a kívánt virtuális környezet.

A virtuális tesztkörnyezet és a tesztjármű digitális ikerpárján („digital twin") kívül további, a szimuláció során mozgást végző objektum megalkotására is szükség van a Unity környezetben. Ezek az objektumok a forgalomszimulátor által generált járművek, a virtuálisan generált dummy gyalogos vagy a valós dummy objektumok digitális párjai lehetnek. Ezek megalkotása a szimuláció futása során kell, hogy megtörténjen a vizualizációs modulban, mivel nem tudhatjuk előre, hogy melyik időpillanatban melyik alkotóelemből hány darab alkotja majd a szimulációt. A be- és kilépő járművek és egyéb objektumok kezelését így előre definiált variánsok klónozásával, és az azokra való hivatkozással tudjuk kezelni.

2.3. Úthálózat és forgalom generálása SUMO forgalomszimulációs szoftverrel

A SUMO mikroszkopikus forgalomszimulátor alkalmazása az úthálózat generáláshoz, és a virtuális forgalom megalkotásához szükséges. A SUMO a TraCi (Traffic Control Interface)

[14] interfésze segítségével könnyen kezelhető a választott Python környezetből is. A már korábban említett NETEDIT kiegészítője segítségével modellezhetünk úthálózatokat OpenStreetMap (OSM) térképek alapján vagy teljesen manuálisan. Az OSM térképek előnye, hogy azok tartalmaznak laterális és longitudinális koordinátákat is, így a valós környezetek digitalizálásakor célszerű ezeket használni. A koordináta rendszerek átváltásáról már korábban esett szó. E feladat megvalósításában teljes mértékig a SUMO-ra hagyatkozunk. A hálózat megalkotásán és a koordinátatranszformáción kívül a forgalomszimulátor fő feladata a virtuális forgalom generálása. A SUMO lehetőséget nyújt arra, hogy a valós forgalmi áramlások jellemzői alapján forgalmat generáljunk. Ezt a forgalmat akár fix útvonalon, akár véletlenszerűen a teljes hálózaton át tudjuk vezetni. A SUMO hálózatban a közlekedési lámpák is definiálhatók. Az ezekkel kapcsolatos információk (forgalom, jelzők, jelzőlámpa-program) a TraCi interfészen keresztül lekérdezhetők, ill. ezen objektumok kezelése is a TraCi-n keresztül történik.

2.4. Az információ áramlása

Az eddigiekben ismertetett három részegység (Python szerver, Unity, SUMO) együttesen alkotja a szimulációs rendszert. A köztük lévő kapcsolat az 1. ábra szerint alakul. Az elemek közti kapcsolaton felül azonban fontos ismerni az információ áramlásának folyamatát is. A megjelenítéshez szükséges egy információs mező, azaz egy TCP üzenet, amely három részből áll. Az üzenet első része az EGO járművel – azaz a valós tesztjárművel – kapcsolatos információkból áll. A második rész a további objektumokkal kapcsolatos információk továbbítására szolgál, a harmadik pedig a közúti jelzőlámpákkal kapcsolatos információk továbbításáért felel. Az üzenet egyes részeihez tartozó információkat az 1. táblázat tartalmazza.

Minden objektumhoz tartozik egy azonosító (ID): egy x és egy y pozíció. Az objektumok z irányú elhelyezkedését a virtuális környezet viszonyai adják. Az EGO jármű

1. táblázat: A szimulációs keretrendszerben használt TCP üzenetek tartalma

EGO objektum	További objektumok	Közlekedési lámpák
ID (EGO)	ID	ID (junctuon – kereszteződés)
pos_x	pos_x	ID (lane index – sáv azonosító)
pos_y	pos_y	pos_x
sebesség	sebesség	pos_y
a mozgás iránya abszolút koordináta rendszerben	a mozgás iránya abszolút koordináta rendszerben	állapot (o, r, y, g)
kormányszög	féklámpa állapot (on-off)	
féklámpa állapot (on-off)	„sizeclass” (méret osztály)	

esetén az üzenet tartalmazza a jármű sebességét, állásszögét, elkormányzási szögét, valamint a féklámpájának állapotát. A féklámpa két állapotú rendszerként van definiálva: bekapcsolt, illetve kikapcsolt állapota létezik. A további objektumok esetében a kormányszög nem kerül kiküldésre, mivel azt a SUMO nem tartalmazza, a megjelenítést a többi információ alapján valósították meg. Alkalmaznak viszont egy „sizeclass” változót, amely az adott objektumot mérete szerint sorolja osztályokba. Ez alapján különböztethetők meg a gyalogosok a járművektől, valamint így van lehetőség a rövidebb és hosszabb járművek megjelenítésére is. A közlekedési lámpák esetében a lámpa azonosíthatósága érdekében két azonosító is továbbításra kerül. Az első az adott kereszteződés azonosítója, a második pedig a lámpához tartozó sáv azonosítója. A továbbított állapot négy fajta értéket vehet fel: o – off (kikapcsolt), r – red (piros), y – yellow (sárga), ill. g – green (zöld).

Az EGO jármű objektumról származó információk a korábban ismertetett információs láncon keresztül jutnak el a vizualizációs modulhoz. A tesztautót a CAN hálózatról fogadott információk alapján a Python szerveren belül – azaz a forgalomszimulációs szoftverben is – el kell helyezni, hogy a virtuális SUMO forgalom is érzékelje a valós tesztjár-

mű jelenlétét. Ehhez is felhasználásra kerül a koordinátatranszformációt megvalósító SUMO TraCi parancs:

„X,Y = traci.simulation.
convertGeo(Longitudinal, Lateral,
fromGeo=True)".

Az így kapott X és Y koordináta a jármű középpontját jelöli, amely a megjelenítő számára szükséges, a SUMO-ban értelmezett járműveket azonban az első lökhárító középpontjának koordinátájával definiálnak. Ezért a kapott koordinátákat az első lökhárítóra kell transzformálni a forgalomszimulátorba illesztés előtt. Ezután történik meg a teljes forgalom minden résztvevőjéről az információk lekérése. A pozíció értékeket ismét transzformálni szükséges, ezúttal a lökhárító középpontjából a járművek középpontjába. Ehhez lekérdezhető a járművek hossza. Az adott szcenárióban szereplő gyalogosokat szintén el kell helyezni a forgalomszimulációs modellben. Valódi dummy gyalogos esetén a mért pozíciója alapján kerül elhelyezésre, hasonlóan a valós tesztjárműhöz, virtuális dummy gyalogos esetén pedig az előre definiált útvonala alapján. Ezzel rendelkezésre áll az összes résztvevő minden tulajdonsága, ami szükséges a megjelenítés szempontjából. A valós objektumok külső TCP kapcsolaton keresztül érkező adatok, minden más virtuális objektumot a SUMO alapján definiálnak.

Lehetőség van a tesztkörnyezet valós tesztjármű nélküli felhasználására is. Ebben az esetben a csak virtuálisan létező EGO jármű irányítása nem külső objektum alapján, hanem valamilyen egyéb bemeneti értékekkel kerül irányításra. Ebben az esetben az információ, amely a tesztjárművet leírja, a vizualizációs modultól származik, ott történik meg a jármű irányítása. Így a Unity és a Python szerver között kétirányú kommunikáció jön létre a TCP kapcsolaton keresztül. Az így létrejövő üzenet alapján kerül aztán elhelyezésre a digitális jármű a forgalomszimulációban.

2.5. Szimulált szenzorok

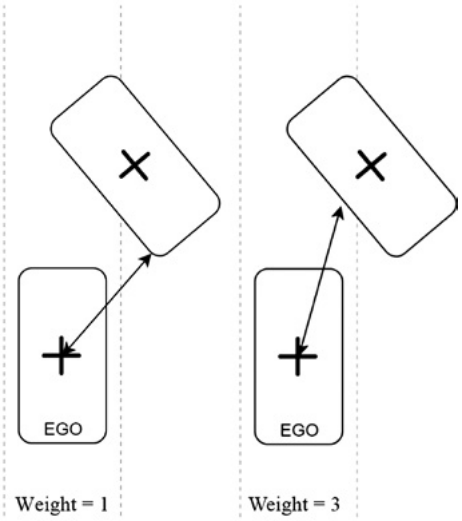
Ahhoz, hogy a tesztjármű érzékelje a virtuális környezetét, szükséges valamilyen érzékelési algoritmus megalkotása is. Ehhez az összes objektumról lekérdezett pozíció, állásszög, szélesség és hosszúság értékek kerültek felhasználásra. Ismerve az EGO jármű pontos pozícióját és haladási irányát, minden egyéb objektum lokalizálható, amely releváns módon a jármű útjában van. Egy ilyen alacsony szintű szenzormodell esetében, amely csak geometria távolságokkal dolgozik, és funkciója csak annyi, hogy minden járműről visszaadja a legközelebbi pontját, fontos volt egy olyan metódus megalkotása, amely nem csak euklideszi távolságok alapján ítéli meg a környezetét. Egy, a tesztjármű útjában ferdén belógó jármű esetén például elképzelhető, hogy a legközelebb eső pont a jármű űrszelvényéből kiesik, miközben más pontjai akadályozzák a tesztjárművet. Ezért a számításhoz felhasznált, releváns legközelebbi pont az euklideszi távolság ($d_{euklideszi}$) és a jármű haladási irányával α bezárt szögön) alapul:

$$d_{releváns}=d_{euklideszi}+|\alpha*w|$$

A bezárt szög w-vel való súlyozásával érhető el, hogy az érzékelt pontok közül a haladási irányba esők számítsanak legnagyobb súlyban. Ezek alapján az összes objektumra kapott releváns pontokat összegezve az összes szenzorirányban a legkisebb érték kerül továbbításra, azaz visszacsatolásra a jármű felé. A releváns legközelebbi pont értelmezését a 2. ábrán látható eset szemlélteti. Amennyiben minden járműről csak egy adatpontot adunk át szenzor információként, mindenképp szükséges foglalkozni azzal, hogy ez melyik pont legyen. Értelemszerűen adódik az EGO jármű középpontjához vagy haladási iránytól függően az első vagy a hátsó lökhárító középpontjához legközelebb eső pont kiválasztása. Ebben az esetben azonban nem garantálható, hogy a mozgás szempontjából legrelevánsabb adat kerül átadásra (lásd 2. ábra bal oldali része). Ezekben az esetekben fontosabb, hogy a haladási iránynak megfelelően olyan információt válasszunk ki, amely releváns módon adja vissza az érzékelt jármű pozícióját (lásd 2. ábra jobb oldala).

A megfelelő w súly meghatározása hangolással történik, de akár időben változó paraméter is lehet: a haladás irányától és a haladási sebességtől van értelme függővé tenni.

2. ábra A releváns legközelebbi pont meghatározása



2.6. Szimulált szenzorok Unity szoftverben

Ezen a ponton a szimulációs környezet csak a SUMO-ból származó alacsony szintű szenzoradat alapján alkalmas a környezet érzékelésére. Ez limitációkat jelent az érzékelés valóságosságán túl az érzékelt objektumok számában is. A rendszer így csak a forgalomszimulátorban definiált objektumokat képes érzékelni, miközben a virtuális világban lehetőség van további, állandó akadályok, környezeti elemek elhelyezésére is. Ezen limitációk áthidalására a Unity nyújtotta lehetőségek felhasználásával kifinomultabb szenzormodellek alkalmazására van lehetőség, amelynek megvalósítása a jövőbeli munka keretein belül történik meg. A kamera alapú rendszerekkel való tesztelés már megvalósításra került más szoftverekben (mint pl. Carla). A fejlettebb szenzormodellek alkalmazása LiDAR és RADAR szenzorokhoz központi szerepet játszik néhány létező szoftver esetében, pl. PreScan, CarMaker. Ugyanakkor a Unity játékmotor megadja a lehetőséget, hogy flexibilis fizikával rendelkező szenzormodelleket alkossunk. Így a Unity-n keresztül virtuális szenzorinformációk generálására is van lehetőség.

3. IRÁNYÍTÁSI LEHETŐSÉGEK

A megalkotott szimulációs keretrendszer lehetőséget biztosít arra, hogy különböző módokon használjuk fel azt egy automatizált/autonóm jármű fejlesztési szakaszaiban. A rendszer fő funkciója, hogy ViL és SciL tesztek elvégzését tegye lehetővé. Ezen felül azonban a rendszer strukturális felépítése lehetővé teszi, hogy valódi tesztjármű nélkül, csak a rendszer további elemeit használva szimulációkat hajtsunk végre. Ebben az esetben az EGO irányítása történhet a virtuális valóságban, vagy valamilyen egyéb külső algoritmus, szimuláció által generált kimenetek alapján.

3.1. Vehicle-in-the-Loop irányítás CAN interfészen keresztül

Az irányítás legkézenfekvőbb módja a jármű CAN hálózatán lévő jelek alapján való irányítás. Ennek elméleti hátterét a 2.2. fejezetben fejtettük ki részletesen. A szimulációs rendszerben az információ áramlása egy teljes, zárt kört ír le a virtuális környezet és a tesztjármű között. Kiindulási pont a jármű CAN hálózata, amelyből az információt a Python szerveren keresztül dolgozzák fel. Onnan az információ a forgalomszimulátorba, majd a vizualizációs modulba kerül. Ezzel párhuzamosan számíthatók a virtuális szenzorinformációk. Ezeket visszacsatolják a jármű CAN hálózatára. Ezek a szenzoradatokat tartalmazó üzenetek a tesztjárművön elhelyezett valódi szenzorok adatstruktúrájának megfelelő formátumban épülnek fel, így nincs szükség a járművet vezérlő szoftver átprogramozására. Megemlítendő továbbá, hogy ezek a szenzorinformációk már mint feldolgozott információk jelennek meg, nem pedig nyers mérési adatként. A szenzorok és a jármű közötti kommunikációs folyamatba ott csatlakozunk be, ahol a valódi szenzorok is a már feldolgozott, strukturált formában továbbított adatokat küldik ki. Az általunk generált, virtuális információkat hordozó CAN üzenetek tehát formailag, felépítésüket tekintve csak az azonosítójukban térnek el a valódi szenzorok üzeneteitől. Ennek eredményeként a vezérlő számítógép egyszerre tudja kezelni mind a

valós, mind a virtuális szenzorok információit: azokat logikai „VAGY" kapcsolattal kezelve minden esetben a megfelelő biztonsággal reagál az akadályok érzékelése esetén.

3.2. Irányítás billentyűzet vagy digitális kormány segítségével

Egy másik egyszerű – a programban kevés változtatást igénylő – irányítási lehetőség a digitális tesztjármű manuális irányítása billentyűzet vagy valamilyen digitális kimenetekkel rendelkező kormány és pedálok segítségével. Ennek a gyakorlati haszna elsősorban mint a rendszeren belüli hibakeresés jelenik meg. Ebben az esetben a szimuláció bemenetet a valós jármű helyett a digitális tesztjármű adja. A digitális jármű mozgásállapotát jellemző paraméterek kerülnek visszacsatolásra a Unity-ből a szerver modulba. Ezen információk alapján történik a tesztjármű forgalomszimulátorba való illesztése is. Minden más objektum kezelése az eddig ismertetett módon történik.

A jármű irányítása a Unity-ben definiált horizontális és vertikális tengelyek segítségével lehetséges. A Unity-ben való definiálás ebben az esetben azt takarja, hogy a billentyűzet nyíl billentyűi, valamint a „WASD" billentyűk lenyomásával ezen tengelyek értékeit tudjuk megváltoztatni. Ugyanez igaz a digitális kormányok esetében is, ezért minden olyan teszt, ami a billentyűzettel működik, kormánnyal is megvalósítható. A tengelyek értékei mindig -1 és +1 között mozognak. Ebben az esetben a jármű mozgatását a két tengely adta értékek segítségével lehet megvalósítani. Ilyen irányítás esetén ellenőrizhető a tesztjármű és a környezete közötti interakció megfelelő működése, ill. felkészíthető a környezet a valós járművel történő tesztelésre.

3.3. Irányítás önvezető algoritmusok segítségével

Az előző fejezetben bemutatott, billentyűzettel való irányítás hátránya, hogy a jármű mozgása nem tartalmaz semmilyen járműdinamikai modellt, így az nem reális. Az ilyen digitális bementekkel rendelkező irányítás esetében felmerülhet az igény, hogy a járművet egy algoritmus irányítsa, amire ilyen bemeneti paraméterekkel nehezen lenne lehetőség. Ezért célszerű olyan modellek megalkotása, amelyek tartalmaznak járműdinamikai paramétereket, valamint biztosítják, hogy a jármű irányítása a kivezérelt nyomaték és kormányszögérték segítségével is megoldható legyen. Ehhez biztosít kiindulási alapot a Unity „Standard Asset" ingyenesen elérhető programcsomagja. Ebben elérhető egy, a járművek irányításáért felelős kontroller kód. Ez a kontroller szintén a billentyűzet által generált jeleket használja fel a jármű irányítására, azonban a fizikai korlátokat figyelembe véve egy parametrizálható nyomatékokat használó irányítást és mozgást hoz létre. Ez a modell így már tartalmaz járműdinamikát, habár különösen a járműdinamikai szimulátorokhoz képest ez nem mondható részletesnek. A Unity ugyanakkor lehetőséget biztosít ezen modell továbbfejlesztésére is.

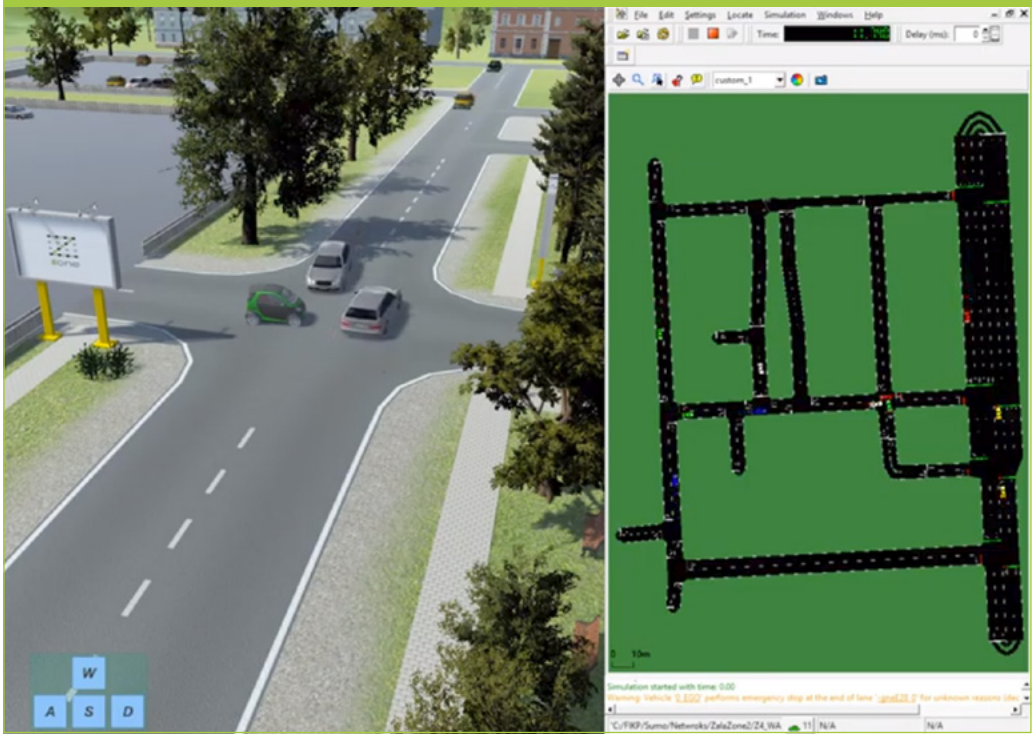
4. SZIMULÁCIÓK, TESZTEK ÉS EREDMÉNYEK

A megalkotott szimulációs környezetben több szimulációt és tesztet is sikerült elvégezni. Ezek egy része a rendszer működésének ellenőrzésére, funkciók fejlesztésére irányult, mások pedig konkrét szimulációs eseteken keresztül, valódi teszteket eredményeztek. Ezen tapasztalatok és eredmények igazolásul szolgálnak a megalkotott keretrendszer létjogosultságára, és irányt mutatnak annak továbbfejlesztésére.

4.1. Manuálisan irányított jármű tesztelése virtuálisan generált forgalomban

Ahhoz, hogy a szimulációs rendszert valódi tesztjárművel is kipróbáljuk, szükséges volt a rendszer működésének tesztelésére. Ezzel az olyan nem várt hibák kiküszöbölése volt a cél, amiket a valós tesztek során időigényes lett volna javítani. Ezért a virtuális forgalomba először manuálisan – esetünkben billentyűzettel – irányított járművet helyeztünk (lásd 3. ábra). Így meg tudtuk vizsgálni, hogy a rendszer logikai felépítéséből származik-e a rendszer működését akadályozó probléma. A tesztek során megbizonyosodtunk róla, hogy a virtuális forgalomba illesztett járművet a forgalom résztvevői látják, mozgásukat meg-

3. ábra: A szimulációs rendszer tesztelése billentyűzettel irányított tesztjárművel (https://youtu.be/dsQRD4rSqf4)



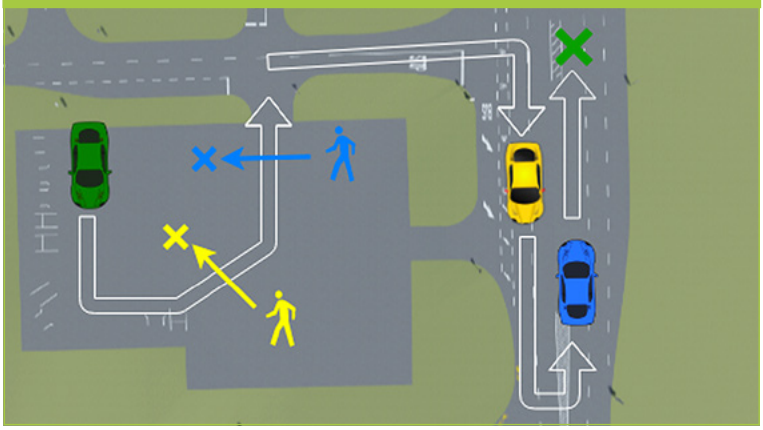
változtatják a beillesztett jármű érzékelésekor a SUMO beépített járműkövetési modelljének megfelelően, azaz teljes értékű forgalmi résztvevőnek tekintik azt. Megfigyelhető volt ugyanakkor, hogy a tesztjármű jelenléte több esetben nem valós viselkedést, főként vészfékezést váltott ki a forgalom többi szereplőjéből. Ez annak volt köszönhető, hogy a jármű mozgása nem volt teljesen valósághű (hiszen billentyűzettel irányítottuk), valamint annak is, hogy a tesztek során nem minden esetben követtük a forgalmi rend szabályait. A vészfékezéses reakció jól bizonyítja, hogy olyan esetekben is működőképes a rendszer, amikor nem minden a forgalmi rendben megszokott módon történik a szimuláció során.

4.2. Valós tesztek a ZalaZone járműipari tesztpályán

A megalkotott szimulációs környezet segítségével valós tesztek is végrehajtásra kerültek a zalaegerszegi ZalaZone járműipari tesztpálya Smart City zónájában [15]. Ezen tesztek során egy olyan SciL szcenáriót demonstráltunk, amelyben a valós tesztjárművön kívül szerepelt valós és virtuális dummy gyalogos, valamint virtuális és valós jármű is.

A 4. ábrán a valós teszt során végrehajtott szcenáriót mutatunk be. A valós tesztjárművet a zöld színű jármű jelképezi, a sárga színű gyalogos és jármű a virtuális objektumokat jelöli, a kék színűek pedig a valós objektumoknak felelnek meg. A szcenárió első szakaszában egy „valet parking" funkció került bemutatásra, amely során a jármű az első fehér színű nyilat követve külső hívásra, teljesen önvezető üzemmódban közlekedett. Az előre rögzített útvonalon való végighaladás során útját először egy virtuális gyalogos, majd később egy valós gyalogos dummy keresztezte. A szimulációs rendszer, valamint a jármű saját szenzorai alapján mindkét esetben megszakította útját

4. ábra: A ZalaZone-ban végrehajtott teszt sematikus ábrája



a jármű, azaz amikor a gyalogos a jármű elé lépett, az megállt, majd miután a gyalogos áthaladt előtte, tovább folytatta útját. Ezt követően a járművet járművezető segítségével a többsávos útra irányították, úgy, hogy közben a szimulációs rendszerben továbbra is létezett a digitális iker, a megjelenítő felületen követhető volt a virtuális valóságban való mozgása. Ezek után két „adaptív cruise control" funkciót bemutató szakasz következett. Először egy virtuális jármű követését valósítottuk meg, majd közvetlenül utána, egy valós járművel is sikerült demonstrálni ugyanezt.

A demonstráció során a szimulációs környezet mindvégig a vártak szerint működött. Az egyes elemek közötti kapcsolat stabil, és megfelelően gyors volt. A jármű mozgása a virtuális térben teljes mértékig reális volt, a mozgás gyakorlatilag teljesen valós időben történt. A szimuláció frekvenciája 10 és 20 ms közé tehető. Ezzel a sebességgel teljes mértékben garantálható, hogy alacsony sebességnél minden információ időben eljusson a tesztkörnyezethez, majd vissza a járműhöz. A szimuláció tapasztalatai alapján elmondható, hogy a keretrendszer használható önvezető járművek bizonyos funkcióival kapcsolatos tesztelés végrehajtásához. Esetünkben a virtuális objektumok érzékelése volt a fő eredmény. A virtuális gyalogos akadályként történő megjelenésére a fékezési és megállási manőver válasz megvalósításra került, valamint a virtuális járművel is sikeresen működött az adaptive cruise control funkció.

4.3. VR megvalósítási lehetőségek

A meglévő rendszer kommunikációjának strukturális kialakításából fakadóan további vizualizációs egységek rendszerhez csatlakoztatására van lehetőség. A már meglévő vizualizációs modell fő feladata a szimuláció követhetőségének biztosítása. Azonban a Unity lehetőséget nyújt arra is, hogy a virtuális valóságot egy VR eszközön keresztül tekinthessük meg, vagy adott esetben a szimulációkba bele is avatkozhassunk. Egy ilyen VR rendszerrel való kibővítés megvalósítható: a jelenleg is kiküldött üzenetet második kliensként fogadva a környezet egy VR eszközön is megjeleníthető. A Unity rendelkezik az Android eszközök támogatásával, így minden giroszkóppal rendelkező Android eszköz (Android API level 21 felett) becsatlakoztatható a szimulációba. A támogatott VR eszközök között szerepel a Google Cardboard, az Oculus, a Daydream, a Gear VR, HTV VIVE, Microsoft HoloLens, Playstation VR, valamint a Windows 10 is [16].

5. ÖSSZEGZÉS ÉS TOVÁBBI FELADATOK

Az eredeti célt, hogy lehetővé tegyük egy valós jármű virtuális környezetben való tesztelését, elértük. A megalkotott szimulációs rendszer ennél többre is képes: a forgalomszimuláció bevonásával jelentősen megnő a lehetséges tesztelési szcenáriók száma. A jelzőlámpákkal együtt valóságos városi közlekedési rendszer építhető fel. Így egy tesztjármű annak teljes szabályozó algoritmusával tesztelhető több olyan forgalmi szituációban, amelyre a valós forgalomban nem lenne lehetőség a szigorú szabályozások miatt. Az elkészített keretrend-

szer a validációs folyamatok támogatására is alkalmas lehet autonóm járműtechnológiák fejlesztése során. Amennyiben egy jármű a virtuális valóság alapján képes megfelelő döntéseket és beavatkozásokat hozni, úgy a sikeres működéssel kapcsolatban csak az érzékelési rétegről nem tehetünk megállapításokat. Egy olyan SciL szcenárió esetén, ahol több valós objektum is szerepel, azok megfelelő érzékelése már elegendő alapot adhat arra, hogy az érzékeléssel kapcsolatban is megállapíthassuk annak megfelelő működését.

Az általunk fejlesztett rendszerrel kapcsolatban kiemelendő, hogy nem csak elméleti eredményként valósult meg, hanem valós tesztek során is kipróbálásra került. Így nem csak elméleti és szimulációs tapasztalatokkal rendelkezünk, hanem a rendszer valós, fizikai működésével kapcsolatban is vannak információink. Ezen tesztek során láthattuk milyen előkészületek szükségesek a rendszer megfelelő működéséhez. A szerzett tapasztalatokra alapozva elmondhatjuk, hogy a kidolgozott rendszerfelépítés és megvalósítás egyaránt alkalmas ilyen tesztek elvégzésére. A tapasztaltak alapján minden lehetőség adott, hogy további fejlesztésekkel növeljük a rendszer teljesítményét, alkalmassá tegyük azt még több, szabadabban definiálható tesztelési szcenárió végrehajtására. Bizonyítékot nyert, hogy egy ilyen rendszerrel ténylegesen támogatható az autonóm járművek fejlesztési folyamata.

Mindazonáltal, hogy ténylegesen validációs eljárásokhoz lehessen felhasználni egy ilyen rendszert, még további fejlesztésekre van szükség. Fontos, hogy mind a kommunikáció [17], mind a szcenáriók leírása kövesse az azokra vonatkozó szabványokat [18]. Meg kell találni a megfelelő csatornát arra, hogy a különböző járműgyártók és azzal kapcsolatos fejlesztő cégek által tesztelt járművekkel milyen módon valósítható meg a kommunikáció. Ennek megfelelően a további fejlesztési munkák során a szabványos eljárások implementálására fókuszálunk. Másik fejlesztési irány pedig a virtuális szenzormodellek alkalmazásának megvalósítása a keretrendszerünkben.

KÖSZÖNETNYILVÁNÍTÁS

A cikk elkészítésével kapcsolatos köszönetnyilvánítás: EFOP-3.6.2-16-2017-00002: Autonóm járműrendszerek kutatása a zalaegerszegi autonóm tesztpályához kapcsolódóan - A projekt a Magyar Állam és az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósul meg.

FELHASZNÁLT IRODALOM

[1] T. Derenda, M. Zanne, M. Zöldy, Á. Török, Automatization in road transport: a review, Production Engineering Archives, vol. 20, pp. 3–7, 09 2018. DOI: https://doi.org/f9vw

[2] W. Huang, K. Wang, Y. Lv, and F. Zhu, Autonomous vehicles testing methods review, in 2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC), Nov 2016, pp. 163–168. DOI: https://doi.org/ggh2p7

[3] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun: Carla - an open urban driving simulator, Eprint: 1711.03938, arXiv, 2017. https://arxiv.org/abs/1711.03938

[4] Tass. (2019, Oct.) Prescan - simulation of adas & active safety (2019.2) https://tass.plm.automation.siemens.com/prescan

[5] Vires. (2019, Oct.) VTD - Virtual Test Drive. https://vires.com/vtd-vires-virtual-test-drive/

[6] Mathworks. (2019, Oct.) Matlab - Automated Driving Toolbox. https://www.mathworks.com/products/automateddriving.html

[7] Y. Laschinsky, K. von Neumann-Cosel, M. Gonter, C. Wegwerth, R. Dubitzky, and A. Knoll, Evaluation of an active safety light using virtual test drive within vehicle in the loop, in 2010 IEEE International Conference on Industrial Technology, March 2010, pp. 1119–1112. DOI: https://doi.org/fs5rzc

[8] C. Biurrun-Quel, L. Serrano-Arriezu, and C. Olaverri-Monreal, Microscopic driver-centric simulator: Linking Unity3D and SUMO, in Recent Advances in Information Systems and Technologies, Á. Rocha, A. M. Correia, H. Adeli, L. P. Reis, and S. Costanzo, Eds. Cham: Springer International Publishing, 2017, pp. 851–860. DOI: https://doi.org/f9vx

[9] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wießner, Microscopic traffic simulation using sumo, in 2018 21st International Conference on Intelligent Transportation Systems (ITSC), Nov 2018, pp. 2575–2582. DOI: https://doi.org/gfvjnh

[10] J. Kaths, B. Schott, and F. Chucholowski, Virtual testing by coupling high fidelity vehicle simulation with microscopic traffic flow simulation, Conference: Autonomous Vehicle Test & Development Symposium 2018, 06 2018.

[11] J. Kaths and S. Krause, Integrated simulation of microscopic traffic flow and vehicle dynamics, in IPG Apply & Innovate, 09 2016.

[12] F. Michaeler and C. Olaverri-Monreal, 3D driving simulator with VANET capabilities to assess cooperative systems: 3DSimVanet, in 2017 IEEE Intelligent Vehicles Symposium (IV), June 2017, pp. 999–1004. DOI: https://doi.org/f9v3

[13] T. Tettamanti, M. Szalai, S. Vass, and V. Tihanyi, Vehicle-In-the-Loop test environment for autonomous driving with microscopic traffic simulation, in 2018 IEEE International Conference on Vehicular Electronics and Safety (ICVES 2018), Madrid, Spain, 12-14. Sept. 2018, pp. 295–300. DOI: https://doi.org/f9v4

[14] Wegener, A., Piórkowski, M., Raya, M., Hellbrück, H., Fischer, S., Hubaux, J.P.: Traci: An interface for coupling road traffic and network simulators. In: Proceedings of the 11th Commu-nications and Networking Simulation Symposium, CNS '08, pp. 155–163. ACM, New York, NY, USA (2008). DOI: https://doi.org/c44qw8

[15] M. Horváth, Q. Lu, T. Tettamanti, A. Török, Zs. Szalay, Vehicle- In-The-Loop (VIL) and Scenario-In-The-Loop (SCIL) automotive simulation concepts from the perspectives of traffic simulation and traffic control, Transport and Telecommunication Journal, vol. 20, pp. 153–161, 04 2019. DOI: https://doi.org/f9v5

[16] (2019, Oct.) Unity user manual (2019.2) https://docs.unity3d.com/Manual/XR.html

[17] A. Dua, N. Kumar, and S. Bawa, A systematic review on routing protocols for vehicular ad hoc networks, Vehicular Communications, vol. 1, p. 33–52, 01 2014. DOI: https://doi.org/f9v6

[18] T. Thomsen and P. R. Mai, ASAM OpenSCENARIO - List of feauters and requirements, Jan. 2019.

Mixed reality development environment for self-driving cars

During the development of self-driving cars, testing possibilities are rather limited, and carrying them out can be difficult and costly. The solution of this problem can be provided by the so-called mixed reality technology, i.e., the application of systems creating a mixed reality. These systems provide an opportunity to create an environment in which real self-driving cars can be tested in virtual traffic. Systems based on mixed reality can greatly assist the research and development of self-driving cars and can also serve as a basis for the shaping of validation procedures.

Mixed-Reality-Entwicklungsumgebung für autonome Fahrzeuge

Während der Entwicklung autonomer Fahrzeuge sind die Testmöglichkeiten eher begrenzt, schwierig und kostenintensiv zu implementieren. Dieses Problem kann durch die sogenannte Mixed-Reality („gemischte Realität") Technologie, d.h. durch die Verwendung von Systemen, die Mixed-Reality realisieren. Diese Systeme bieten die Möglichkeit für die Schaffung einer Umgebung, in der reale autonome Autos im virtuellen Verkehr getestet werden können. Mixed-Reality-Systeme können bei der Forschung und Entwicklung autonomer Fahrzeuge eine große Hilfe gewährleisten und auch als Grundlage für die Entwicklung von Validierungsverfahren dienen.