Autonóm menetciklus modellezés, szimuláció és validálás 1:10-es méretarányú járműmodell platformon

Autonomous driving cycle modeling, simulation and validation on 1:10 scale vehicle model platform

FERENCZ Csanád[1], Dr. ZÖLDY Máté[1*]

[1]Department of Automotive Technologies, Faculty of Transportation Engineering and Vehicle Engineering, Budapest University of Technology and Economics, Hungary Budapest, Stoczek street. 6. Building J, Floor V. Postcode: 1111
[*]Corresponding author, E-mail: mate.zoldy@auto.bme.hu

Abstract

In the present research paper, the authors provide a comprehensive overview about the R&D possibilities and processes in the field of autonomous vehicle testing and validation, an exhaustive investigation concerning an autonomous vehicle driving cycle, by developing not only camera-based traffic sign and lane markings detection and tracking algorithms, but also the implementation and simulation of these, as well as the verification and validation procedures on a 1:10 scale vehicle model platform, realizing and reproducing thus a complete embedded system development life cycle.

Keywords: autonomous vehicles, traffic-sign detection, lane detection and tracking, testing, validation

Kivonat

Jelen kutatási cikkben a szerzők átfogó áttekintést nyújtanak az autonóm járművek tesztelése és validálása területén végzett K + F lehetőségekről és folyamatokról, autonóm jármű menetciklusra vonatkozó kimerítő vizsgálatról, nemcsak a kamerán alapuló táblafelismerési, illetve sávfelismerési és sávkövetési algoritmusok fejlesztése, hanem ezek implementálása és szimulációja, valamint az ellenőrzési és validálási eljárások által 1:10 méretarányú járműmodell-platformon egyaránt, megvalósítva és reprodukálva ezáltal egy teljes beágyazott rendszer-fejlesztési életciklust.

Kulcsszavak: autonóm járművek, sávfelismerés és sávkövetés, táblafelismerés, tesztelés, validálás

1. BEVEZETÉS

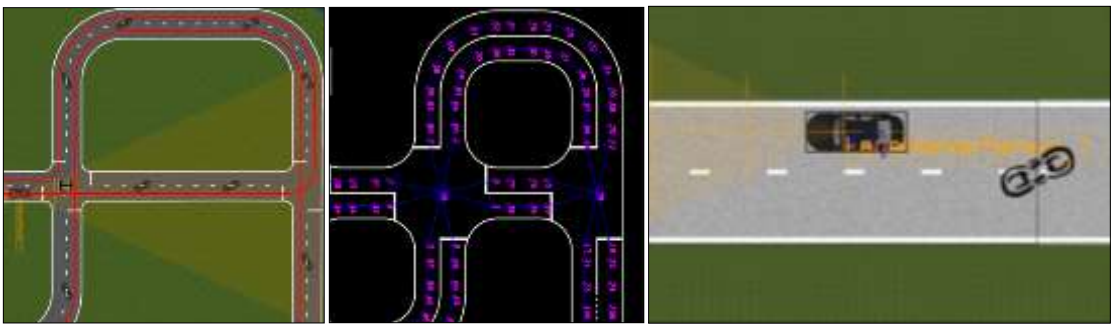
Napjaink járművei magasabb automatizálási szintje olyan forradalmi technológiákra támaszkodik, amelyeket korábbi módszerekkel nem lehet tesztelni és validáni. A jövőbeli közúti biztonság garantálásához úgyszintén forradalmi tesztelési és validálási módszerekre van szükség [2].

Jelen cikk egy ilyen módszer bemutatását célozza, kifejezetten egy virtuális környezetben kifejlesztett digitális kamera-alapú sáv- és tábla-felismerési algoritmus validálására és ellenőrzésére, autonóm vezetési ciklusokban végzett kísérleti teszteken keresztül, 1:10 méretarányú járműmodell-platformon megvalósítva egy beágyazott rendszer szoftverének a részeként.

2. VIRTUÁLIS SZIMULÁCIÓS KÖRNYEZET

A jelen fejezetben bemutatjuk a PreScan szimulációs környezetet és annak kapcsolatát a Simulinkkel, egy egyszerű sávfelismerő és sávkövető algoritmus mellett. Ezenkívül bemutatásra kerül egy globális állapotgép megvalósítása Simulinkben, amely szükséges a fejleszteni kívánt rendszerfunkciók virtuális és kísérleti teszteléséhez is egyaránt [2].

A PreScan szimulációs környezet segít fejleszteni, tesztelni és hibakeresni vezérlő algoritmusainkat, mielőtt feltöltenénk azokat a tényleges járműmodellre. Az *1. ábra* a felépített virtuális környezetet mutatja, amely lényegében megegyezik azzal, amelyet később a járműmodellek esetében alkalmaztunk.



1. ábra *A felépített PreScan virtuális környezet sávjelző érzékelővel*

A modellhez hozzáadódik egy sávjelző szenzor, amely a kamera és a képfeldolgozó algoritmust reprodukálja a valós járműmodellen és információt szolgáltat az úttesten a sávvonalak és a szkennelési vonalak metszéspontjaként az érzékelőhöz viszonyított sávvonalakról.

A MATLAB függvényben egy másodrendű polinomot illesztünk a bal és a jobb sáv adatpontjaihoz a *polyfit()* függvénnyel, tekintettel arra a tényre, hogy a valós jármű Simulink modellje egy olyan vektort kap, amely a kamera és a képfeldolgozó algoritmusok által detektált polinomgörbe együtthatóit tartalmazza. Ezt követően számít egy átlagos hibát a bal és jobb sáv vonalainak az adataiból és továbbítja kimeneti adatként a PID vezérlő felé. A sávjelző szenzor négy érzékelő vonal adatait továbbítja kimenetként, ezáltal relatív pontos görbéket kapva [3].

Figyelembe véve a globális állapotgépet, a Simulink Stateflow folyamatábrájának állapotgépét használjuk, ahol a bemeneti jelek és feltételek alapján a megfelelő kimenet létrehozható. Ha a kamera felismer egy stop táblát, látva a hozzá tartozó vonalat is, akkor a járművet 3 [s]-ig meg kell állítani, ennek a logikája a feltétel, a bemenetek a stop jel és a vonal, a kimenet a vezérlő parancs.

3. KÍSÉRLETI MÉRÉSEK ÉS TESZTELÉS

Az automatizált funkcióink és algoritmusaink virtuális környezetben és nyilvánosan elérhető adatkészleteken történő tesztelése mellett valós körülmények közötti kísérleteket is végeztünk. Erre a célra két mérési járműmodell platformot építettünk, amelyek digitális kamera-egységekkel vannak felszerelve, így valós körülmények között valós időben tesztelhetjük algoritmusainkat.

3.1. Mérési járműmodell platform

A kezdeti hardver-modell két fő része a két Audi RS4 1:10 méretarányú karosszériából, valamint azok alvázából áll. A további elektronikus alkatrészeket az alábbiakban részletezzük, míg a mérési járműmodellek végső formáját az *2. ábra* mutatja.

2. ábra *Járműmodell platform karosszéria, futómű és elektronikus komponensek*

Valószínűleg a legfontosabb elem a jelen tanulmány szempontjából a Raspberry Pi 4-B Modell/4GB kompakt egykártyás számítógép, amely a jármű akkumulátorával működik. Alapvetően fejlesztő-kártyaként használva ez a képfeldolgozást és a jármű irányítását is kezelni fogja, ezért a teljesítménykorlátozásokat szem előtt kell tartani. Ezenkívül egy STM32 Nucleo-64 fejlesztőkártyát is használunk alacsony szintű vezérléshez, közvetlenül mögé helyezve és összekapcsolva a Raspberry Pi nyomtatott áramköri kártyával.

A Raspberry Pi v2 kameramodul alapvetően a járműmodell látórendszere lesz, egy magas minőségű, 8 megapixeles Sony IMX219 képérzékelő, rögzített fókuszú lencsével. A felső felületén található kis

foglalatok rögzítik az egykártyás számítógéphez dedikált CSi interfész használatával, amelyet kifejezetten kamerákhoz való csatlakozáshoz terveztek [6].

A mérőjárművet egy 7,5 [V] DC villanymotor hajtja, emellett pedig egy 4,8 - 6 [V] szabványos analóg szervóval is rendelkezik a kormányszög irányításához kanyarodás közben. Továbbá egy kapacitív moduláris inkrementális kódolót is használunk, amely érzékeli a mechanikai mozgást (helyzetet, sebességet, távolságot, irányt) és elektromos jelekké alakítja. A jármű működéséhez szükséges energia modern LiPo (lítium-ion polimer) újratölthető akkumulátorokból (14,8 [V], 3800 [mAh]) származik, amelyek nemcsak egyértelműen nagyobb kapacitással rendelkeznek, mint a NiMH vagy NiCd újratölthető akkumulátorok, de lényegesen kisebb a súlyuk is.

Ezen felül szükség van még egy MOSFET H-híd motor meghajtóra, amely lehetővé teszi a nagy teljesítményű DC motor kétirányú vezérlését, széles 5,5 - 30 [V] feszültségtartományt támogatva, miközben folyamatos 15 [A] teljesítményt nyújt, illetve egy DC-DC villamosenergia-átalakítóra. Ez utóbbi segítségével elérhető az egyik feszültségszintű egyenáramból a másikba való átalakítás, aláramkörökbe juttatva, melyek mind saját, az akkumulátor által biztosított feszültségszinttől eltérő, feszültségszint-követelménnyel rendelkeznek.

3.2. Környezetérzékelési algoritmus

Egy robusztus sávfelismerő funkció létfontosságú szerepet játszik minden autonóm jármű esetében, de a jelen projekt esetén ez még fontosabb, mivel ez az egyetlen bemenet a környezet felől. A modell központi eleme a vezérlőszoftver, amelynek a teljes menetciklus folyamatot irányítania kell. A fejlesztési periódus során tradicionális képfeldolgozási módszereket alkalmaztunk, Python 3.x-et és OpenCV-t használva a szoftverfejlesztéséhez. Néhány, az alábbiakban leírt módszert implementáltunk és teszteltünk a tanulmány témáját tekintve a legmegfelelőbb algoritmus feltárása céljából [1].

Első próbálkozásként Hough transzformációt használtunk a Canny élkeresés segítségével a sávvonalak detektálására, majd ezeket a vonalakat szűrtük és feldolgoztuk, hogy meghatározzuk, melyik tartozik a bal és a jobb sávvonalhoz. Az algoritmus jól működött és jó eredményeket nyújtott, azonban csak egyenesekben volt hasznos, ezért olyan módszert kellett implementálni, amely képes kezelni a görbéket is [7].

Ezt követően a Sobel operátoron alapuló Sliding Windows metóduson volt a sor, amely egy ablakon belül egy bináris kép alsó felének hisztogramját véve feltárja azt a környéket, ahol a sávok kezdődnek, ezután a képet vízszintes szeletekre osztva, egy keresőablakot csúsztatva az egyes szeleteken a legmagasabb frekvenciájú területeket tárja fel a bal és a jobb sávon, majd egy másodrendű polinomot illesztve a *polyfit()* függvénnyel megkapható a vonalakon lévő legjobban illeszkedő görbe. A módszer nagyszerű eredményeket hozott, azonban számításigény szempontjából drága, ezért fenntarthatóbb algoritmust kellett találni [5].

Az algoritmus teljesítményigényének csökkentése érdekében az élérzékelést egyszerű fehér küszöbértékkel Hough transzformációra változtattuk. Ez a módszer soha nem érte el a 10 fps (képkocka per másodperc) sebességet, így annak ellenére, hogy robusztus módszer, a Hough transzformációval és a vonalak Hough-térben történő kezelésével kapcsolatos problémák miatt nem volt használható.

A különböző megoldásokat, azok előnyeit és hátrányait az *1. táblázat* mutatja be, amely alapján könnyen meghatározható a legjobb megoldás, nevezetesen a Canny élkeresés hisztogram analízissel.

Összehasonlítás a tesztelt sávfelismerési és sávkövetési algoritmusok között 1. táblázat

Sávfelismerési és sávkövetési algoritmus	**Előny**	**Hátrány**
Hough transzformáció - Canny élkeresés	Megfelelő működés, jó eredmények	Csak egyenes vonalak (sávok) esetében hasznos
Sliding Windows metódus - Sobel operátor	Megfelelő működés kanyarokban, egyenesekben	Nagy számításigény
Hough transzformáció - fehér küszöbérték	A világítás jobb szűrése	Soha nem ért el 10 fps-nél többet
Canny élkeresés - Hisztogram analízis	Robusztus, gyors és egyszerű	Nem tapasztaltunk

Ami a táblafelismerési funkciót illeti, a cél négyféle közlekedési tábla (fő/elsőbbségi út, stop, gyalogos átkelő és parkoló táblák) sikeres felismerése és osztályozása. A fejlesztett algoritmus színküszöbön, alakfelismerésen és klasszifikáción alapul. A táblák színe egészen más, mint a szokásos környezetük - piros, sárga és ilyen kék szín nem várható a térképek meghatározott területén. Miután a képet a megadott küszöbértékkel eltakarjuk, morfológiai zárást különféle típusú ellipszis és téglalap alakú szerkezeti elemekkel hajtunk végre a zaj csökkentése érdekében. A táblák különböző formájúak, kontúrkereséssel megtalálhatók. A kis tárgyak és zaj észlelésének elkerülése érdekében a kontúr minimális ívhossza állítandó be. Ezután az algoritmus a megtalált kontúrokat tipikus alakzatokba (háromszög, téglalap, ellipszis, kör stb.) összegyűjti, megrajzolja és osztályozza [6].

A fényérzékenység kérdésének leküzdése érdekében a sablonillesztés alkalmazható az észlelt tábla helyességének nagyobb bizonyosságának érdekében. A többi objektum és zaj észlelésének minimalizálása érdekében az úgynevezett érdekelt régiók módszert alkalmazzuk és a képet a feldolgozás elején levágjuk. Ez csökkenti a szükséges feldolgozási teljesítményigényt is [4].

Az alábbiakban bemutatott perspektivikus nézet továbbra is a szkript fontos része a felülnézet biztosítása és a torzítások elkerülése érdekében. A vonalszélek a függőleges fehér vonal alapján illesztődnek, a lineáris illesztés hisztogram csúcsait a kamera képe mutatja a *3. ábrán*.

3. ábra *Valós idejű kamera kép a közlekedési tábla és a sávfelismerő algoritmusok kimenetéről*

3.3. Járműirányítási rendszer

A kommunikációs stratégia alkalmazásának fő célja a megfelelő vezérlőjelek elküldése a Raspberry Pi-től az STM32 Nucleo-64 kártyához. Áttekintésként a Raspberry Pi felelős a magas szintű irányításért (képfeldolgozás, pozícionálás, kontroll), míg az STM32 Nucleo-64 feladata az egyenáramú motor és a kormányszervó vezérlése az előbbi irányítási parancsai alapján. A két fent említett mikroprocesszor közötti kapcsolat soros kommunikációs protokoll, fizikailag kábel. Karakterlánc formátumú üzenetek felelősek a parancs átadásáért, ezek a Simulink által küldött üzenetek C ++ kódban állítódnak össze minden 0,01 [s]-ban (100 [Hz]), adatokat írva a soros eszközre, soros kommunikációt létrehozva.

A sávkövetési funkció kontrollerével kapcsolatosan viszonylag egyszerű, de performáns megoldást választottunk, így a PID vezérlőt csak PD feltételekkel hoztunk létre a távolsághibára. Ezt követően egy másik PD vezérlővel együtt figyelembe vettük az irányhibát is, ami a jármű stabilabb mozgását eredményezte a kanyarokban [2].

A modell vezérlőrendszerének másik fontos eleme a globális útvonaltervezés és trajektória végrehajtás, egy Python-kód, amely a kiindulási és a kívánt véghelyzetek alapján kiszámítja a járműmodell megfelelő útvonalát. Az adott térkép gráf formátumú, koordinátákkal és élekkel rendelkező csomópontokat tartalmaz, amelyek deklarálják a csomópontok közötti kapcsolatokat. Minden csomópont rózsaszínű azonosítóval és kék élekkel, irányított nyilakkal rendelkezik. Ennek eredményeként az adott gráf egy irányított gráf, ami azt jelenti, hogy csak egy lehetőség van egy csomópontról a másikra lépni (lásd *1. ábra*).

A használt útvonaltervezési algoritmus a Dijkstra optimális legrövidebb út algoritmusa. Ennek a módszernek a kimenete a csomópontok azonosítója az útvonal sorrendjében. Az előfeldolgozó rész két fő változója maga a gráf, amelyen az útvonaltervezés végződik és az iránymátrix, amely a kereszteződéseknél az elfordulási lehetőségek specifikálására szolgál (-1 jobbra, 1 balra és 0 az egyenes mozgáshoz). Az adott bemenet megoldása a csomópont-azonosítókhoz rendelt koordináták a tervezett útvonal szerint, ezek gráf formátumban láthatók a térképen. Ha a jármű kereszteződés előtt áll, akkor az irányítási rendszer a kimeneti mátrix alapján fogja tudni a kormányzási parancsokat. Az utófeldolgozási szakasz végén van a függvény és a bemenetek meghívása, ezen felül lehetőség van köztes csomópont-azonosítók megadására is, így különböző útvonalak tervezhetők [8].

3.4. Robot operációs rendszer

A ROS, vagyis a robot operációs rendszer, egy rugalmas, nyílt forráskódú keretrendszer, amely átlátható robotszoftverek írására készült, könyvtárakat és eszközöket kínál a fejlesztők számára robotalkalmazások létrehozására.

A rendszer alapkomponensei a *node*-ok (csomópontok), ezek számítást végző folyamatok, írhatók C ++ vagy Python nyelven is. Egy robotnak minden feladat esetén különböző csomópontjai lesznek, ezért párhuzamosan fejleszthető és strukturált szoftverarchitektúrát eredményez. A csomópontok *topic* (tárgyakon) keresztül kommunikálhatnak egymással, egy busz kommunikációs protokoll, meghatározott üzenettípussal. Az interfészeket tárgyak segítségével definiálhatjuk, így a modulokat lecserélhetjük a tesztelési fázisokban. A meglévő algoritmusokat áthelyezését követően a ROS keretrendszerbe, a képfeldolgozó csomópont csak néhány egyszerű módosítást igényelt. Mivel a vezérlő a Simulinkben készül, első lépésként C ++ kódot kellett létrehozni a modellből. A létrehozott kódhoz ROS burkoló készült, így a ROS csomópont mögötti modell minden nehézség nélkül megváltoztatható [6].

Ezáltal elérhető a végső ROS-grafikon, melyben az összes szkript képes kommunikálni egymással és I/O állapotokat biztosítani, így létrehozva egy teljes globális kommunikációs rendszert. Jelenleg két fő csomópont van, a képfeldolgozó csomópont és az irányítási csomópont, valamint egy harmadik, paraméter-csomópont, amely a kontroller paramétereinek hangolására és a képfeldolgozásra használandó, kommunikál a dedikált hibakeresési tárgyakkal és tartalmaz néhány paraméter-beállítást. Ezek visszahívása után a megváltozott értékeket közzéteszi a különböző tárgyakban, és a vezérlő-képfeldolgozó csomópont ennek megfelelően megváltoztatja a paramétereket.

4. ÖSSZEGZÉS

A mai rendszerek komplexitása és a lehetséges forgalmi helyzetek sztochaszticitása új megközelítéseket igényel, különböző tesztelési szintekkel, validációs és jóváhagyási rétegekkel.

Tekintettel arra, hogy a jelen projekt keretein belül két meghatározó automatizált funkció, a sávfelismerés és sávkövetés, illetve a táblafelismerés is nem csak kifejlesztésre, hanem tesztelésre és ellenőrzésre is került járműmodell platformokon megvalósított vezetési ciklusok révén, a szerzők remélik, hogy új tesztelési és validálási módszertanok is alapulhatnak a bemutatott elképzeléseken, hogy további automatizált funkciók vagy akár összetett autonóm rendszerek is ellenőrizhetők és validálhatók lesznek ilyen beágyazott rendszerek segítségével.

KÖSZÖNETNYILVÁNÍTÁS

A tanulmányhoz az Új Széchenyi Terv keretein belül a „Tehetséggondozás és kutatói utánpótlás fejlesztése autonóm járműirányítási technológiák területén (EFOP-3.6.3- VEKOP-16-2017-00001)” projekt biztosított forrást. A kutatás az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósult meg.

IRODALMI HIVATKOZÁSOK

[1] Aradi Sz., Bécsi T., Gáspár P. *Experimental Vehicle Development for Testing Autonomous Vehicle Functions.* 10th International Conference on Mechatronic & Embedded Systems & Applications (MESA), 2014, 1-5.

[2] Ferencz Cs., Zöldy M. *Körforgalmi szituáció szimulációja és validálása rádióvezérlésű (RC) autonóm járművek segítségével.* 28th International Conference on Mechanical Engineering – OGÉT, 28, 2020, 206 - 210.

[3] Németh H., Háry A., Szalay Zs., Tihanyi V., Tóth B. *Proving Ground Test Scenarios in Mixed Virtual and Real Environment for Highly Automated Driving.* Mobilität in Zeiten der Veränderung, Springer, 2019, 198-210.

[4] Ramesh J., Rangachar K., Brian G. S. *Machine Vision*, McGraw-Hill, Inc., USA, 1995.

[5] *** *Advanced Lane Finding Using Sliding Window Search.* The world's leading software development platform - GitHub, https://github.com/charleswongzx/Advanced-Lane-Lines (Utolsó letöltés: 27.01.2021).

[6] *** *Competition Regulations 2019*. Bosch Future Mobility - The Challenge, https://www.boschfuturemobility.com/uploads/Competition_Regulations_2019.pdf (Utolsó letöltés: 20.02.2021).

[7] *** *Lane Detection Using Hough Transform.* The world's leading software development platform - GitHub, https://github.com/karasuno7/Lane-Detection-using- Hough-Transform (Utolsó letöltés: 01.02.2021).

[8] *** *Tutorial: Build a lane detector*. Towards Data Science, https://towardsdatascience.com/tutorial-build-a-lane-detector-679fd8953132 (Utolsó letöltés: 20.02.2021).