VIŠJA STROKOVNA ŠOLA ACADEMIA

MARIBOR

Primerjava učenja samostojne vožnje v simulatorjih CARLA in Udacity za začetnike

Kandidatka: Aspasija Cvetkoska

Vrsta študija: študentka izrednega študija

Študijski program: Informatika

Mentor predavatelj: mag. Ervin Schaff

Mentor v podjetju: mag. Valneja Stojčič Erat, dipl. inž. rač. inf.

Lektor: Ljiljana Mićović Struger, prof. slov. jez. in knjiž.

Maribor, 2024

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Podpisana Aspasija Cvetkoska, sem avtorica diplomskega dela z naslovom Primerjava učenja samostojne vožnje v simulatorjih CARLA in Udacity za začetnike, ki sem ga napisala pod mentorstvom mag. Ervina Schaffa.

S svojim podpisom zagotavljam, da:

- je predloženo delo izključno rezultat mojega dela,
- sem poskrbela, da so dela in mnenja drugih avtorjev, ki jih uporabljam v predloženi nalogi, navedena oz. citirana skladno s pravili Višje strokovne šole Academia Maribor,
- se zavedam, da je plagiatorstvo – predstavljanje tujih del oz. misli kot moje lastne kaznivo po Zakonu o avtorskih in sorodnih pravicah (Uradni list RS, št. 16/07 – uradno prečiščeno besedilo, 68/08, 110/13, 56/15 in 63/16 – ZKUASP); prekršek pa podleže tudi ukrepom Višje strokovne šole Academia Maribor skladno z njenimi pravili,
- skladno z 32.a členom ZASP dovoljujem Višji strokovni šoli Academia Maribor objavo diplomskega dela na spletnem portalu šole.

Ljubljana, september 2024 Podpis študentke:

ZAHVALA

Iskreno se zahvaljujem svojemu mentorju mag. Ervinu Schaffu za njegovo neprecenljivo pomoč in mentorstvo med nastajanjem mojega diplomskega dela. Njegovo vodenje in podpora sta bila ključnega pomena pri oblikovanju tega dela.

Globoko sem hvaležna tudi svoji družini in prijateljem, katerih neomajna podpora in spodbuda sta mi bili stalen vir motivacije.

Poleg tega bi se rada zahvalila podjetju, v katerem delam, za priložnosti in vire, ki so pomembno prispevali k uspešnemu dokončanju študija.

Zahvaljujem se vsem za stalno podporo in zaupanje vame.

POVZETEK

Diplomsko delo primerja učinkovitost dveh priljubljenih simulacijskih platform, CARLA in Udacity, s poudarkom na začetnikih na tem področju, z namenom, da bi usposobili modele umetne inteligence za aplikacije avtonomnih vozil. Ocenjena je tudi uporabnost Jupyter Notebook in Google Colab, dveh znanih programskih okolij, za ustvarjanje in testiranje algoritmov za samostojno vožnjo. Glavni cilji naloge so ugotoviti, katera razvojna okolja in simulacijska orodja so najboljša za neizkušene razvijalce ter kako optimizirati algoritme samoučenja za izboljšanje učinkovitosti in natančnosti modelov umetne inteligence v razmerah avtonomne vožnje.

Rezultati raziskave kažejo, da je bolj realističen, vendar zahtevnejši simulator CARLA primernejši za izkušene uporabnike in temeljite simulacije, medtem ko je začetnikom prijaznejši simulator Udacity, ki je dostopnejši zaradi preprostejšega vmesnika in manjših potreb po obdelavi. Podobno tudi Google Colab novim uporabnikom ponuja okolje, ki je enostavno za uporabo in učinkovito z viri, saj z uporabo infrastrukture v oblaku zagotavlja zanesljive računalniške vire, ne da bi bilo treba lokalno namestiti visokozmogljivo strojno opremo. Po drugi strani pa velja, da je Jupyter Notebook bolj koristen za izkušene razvijalce, ki potrebujejo več svobode in nadzora pri vzpostavljanju svojega razvojnega okolja.

Poleg tega raziskava potrjuje, da povečanje števila iteracij v ciklu algoritma samoučenja do določene mere poveča učinkovitost modelov umetne inteligence. Vendar pa po tej točki nadaljnje iteracije vodijo k zmanjševanju donosa, saj povečujejo porabo virov, ne da bi prinesle občutno izboljšanje učnih rezultatov. Ta opažanja izboljšujejo naše razumevanje kompromisa med učinkovitostjo učenja in učinkovitostjo računanja med učenjem modelov AV.

Glede na vse navedeno diplomsko delo ponuja pronicljive in koristne nasvete, ki bodo tako začetnikom kot tudi izkušenim uporabnikom pomagali pri izbiri najboljših orodij in razvojnih postopkov za avdiovizualne vsebine. Poleg tega podaja predloge za prihodnje študije, namenjene izboljšanju učinkovitosti in odpornosti modelov umetne inteligence v avtonomni vožnji.

Ključne besede: *avtonomna vozila, umetna inteligenca, CARLA, Udacity Simulator, Jupyter Notebook, Python.*

ABSTRACT

Comparison of Learning Autonomous Driving in CARLA and Udacity Simulators for Beginners

In order to train artificial intelligence models for autonomous vehicle applications, this thesis compares the performance of two popular simulation platforms, CARLA and Udacity, with a focus on beginners in the field. This study also evaluates the usefulness of Jupyter Notebook and Google Colab, two well-known software environments, for creating and testing self-driving algorithms. The main objectives of this study are to identify which development environments and simulation tools are best suited for inexperienced developers, and how to optimise self-learning algorithms to improve the performance and accuracy of UI models in autonomous driving situations.

The results of the study show that the more realistic but more complex CARLA simulator is more suitable for experienced users and in-depth simulations, while the more beginner-friendly Udacity simulator is more accessible due to its simpler interface and lower processing requirements. Similarly, Google Colab offers new users an easy-to-use and resource-efficient environment, using cloud infrastructure to provide reliable computing resources without the need to install high-performance hardware locally. On the other hand, Jupyter Notebook is considered more useful for experienced developers who need more freedom and control in setting up their development environment.

In addition, the study confirms that increasing the number of iterations in a self-learning algorithm cycle increases the performance of AI models to a certain extent. However, beyond this point, further iterations lead to diminishing returns, as they increase resource consumption without significantly improving learning outcomes. These observations improve our understanding of the trade-off between learning efficiency and computational efficiency during the learning of AV models.

In view of all the above, the thesis offers insightful and useful advice that will help both beginners and experienced users to choose the best tools and development processes for audiovisual content. Furthermore, it provides suggestions for future studies aimed at improving the performance and resilience of AI models in autonomous driving.

Keywords: *autonomous vehicles, artificial intelligence, CARLA, Udacity Simulator, Jupyter Notebook, Python.*

Kazalo vsebine

KAZALO SLIK

KAZALO TABEL

SEZNAM KRATIC

Kratica	Pomen v slovenščini	Pomen v angleščini
AV	Avtonomno vozilo	Autonomous Vehicle
ML	Strojno učenje	Machine Learning
CNN	Konvolucijsko nevronsko omrežje	Convolutional Neural Network
UAV	Brezpilotno letalo	Unmanned Aerial Vehicle
UI	Umetna inteligenca	Artificial Intelligence
VM	Virtualni stroj	Virtual Machine
TPU	Procesna enota za tenzorje	Tensor Processing Unit
GPU	Grafična procesna enota	Graphics Processing Unit
NLP	Obdelava naravnega jezika	Natural Language Processing
SVM	Metoda podpornih vektorjev	Support Vector Machine
PCA	Analiza glavnih komponent	Principal Component Analysis
PID	Proporcionalno-integralno-derivativni	Proportional-Integral-Derivative
CARLA	Avtomobil, ki se uči, kako se obnašati	CAR Learning to Act

R-CNN	Regijsko osnovana konvolucijska nevronska mreža	Region-based Convolutional Neural Network
YOLO	Pogledaš le enkrat	You Only Look Once

1 UVOD

Hiter napredek na področju tehnologij avtonomnih vozil (AV) in strojnega učenja je odprl številne možnosti za raziskave in razvoj. Namen tega diplomskega dela je prispevati k temu rastočemu področju s poudarkom na posebnih hipotezah povezanih s simulacijami AV in okolji za razvoj algoritmov.

Motivacija tega diplomskega dela je zagotoviti vpogled, ki bo začetnikom v pomoč pri premagovanju zapletenosti razvoja in simulacije AV. Ugotovitve bodo prispevale k širšemu razumevanju simulacijskih platform, okolij za razvoj algoritmov in tehnik optimizacije pri raziskavah AV.

1.1 Opis področja in opredelitev problema

Avtonomna vozila vključujejo samovozeče avtomobile in tehnologije, ki vozilom omogočajo samostojno delovanje. To interdisciplinarno področje združuje strojno učenje, umetno inteligenco, računalniški vid, senzorje in robotiko ter ustvarja sisteme, ki zaznavajo okolico, sprejemajo odločitve in nadzorujejo gibanje vozila. Ključni izzivi vključujejo realistično simulacijo za urjenje modelov umetne inteligence v različnih voznih razmerah ter ustvarjanje robustnih, uporabniku prijaznih okolij za hitro izdelavo prototipov in testiranje algoritmov AV.

Problemi, ki jih obravnava to diplomsko delo, so naslednji:

- Izbira simulacijske platforme

Izbira simulacijske platforme za usposabljanje modelov umetne inteligence za začetnike. Predvideva se, da je CARLA, simulator visoke verodostojnosti, zaradi svojega realističnega okolja in obsežnih funkcij učinkovitejši od simulatorja Udacity, vendar bolj kompliciran in zahtevnejši. Razumevanje optimalne platforme je ključno za začetnike, ki potrebujejo zanesljiva in dostopna orodja za razvoj svojih modelov.

- Razvojno okolje za algoritme

Primerjava učinkovitosti Jupyter Notebook in Google Colab za razvoj in testiranje AV algoritmov. Glede na njuno razširjeno uporabo v skupnosti strojnega učenja je bistveno ugotoviti, katera platforma bolje podpira začetnike pri pisanju, testiranju in odpravljanju napak v kodi.

- Optimizacija algoritmov za samostojno učenje

Ugotavljanje optimalnega števila zagonov za cikle algoritmov samoučenja v simulacijah AV. To vključuje določitev ravnovesja med računalniškimi viri in učinkovitostjo učenja, kar je ključnega pomena za začetnike, ki morajo v okviru omejenih časovnih in računalniških proračunov povečati svoje rezultate.

1.2 Namen, cilji in osnovne trditve

Namen tega diplomskega dela je opredeliti najučinkovitejša orodja in metode za začetnike na področju razvoja AV. Z vrednotenjem različnih simulacijskih platform, razvojnih okolij in optimizacijskih tehnik je namen tega diplomskega dela je zagotoviti praktična spoznanja in smernice, ki lahko novim razvijalcem/inženirjem olajšajo vstop na področje AV.

Diplomsko delo se bo osredotočilo na tri glavne cilje:

(1) primerjava programa CARLA in simulatorja Udacity, da bi določili učinkovitejšo platformo za učenje modelov umetne inteligence za začetnike,

(2) vrednotenje Jupyter Notebook in Google Colab, da bi določili boljše okolje za razvoj in testiranje algoritmov AV ter

(3) določitev optimalnega števila zagonov za cikle algoritmov za samoučenje v simulacijah AV.

Osnovne trditve, ki jih je treba raziskati, vključujejo hipoteze, da je program CARLA boljši od simulatorja Udacity za usposabljanje modelov, da je beležnica Jupyter za razvoj algoritmov učinkovitejša od programa Google Colab in da obstaja optimalno število iteracij za algoritme samoučenja, ki uravnotežijo učinkovitost in uporabo virov.

V tem diplomskem delu smo postavili naslednje hipoteze:

- H1: Udacity Simulator je bolj prijazen za začetnike pri treniranju modelov umetne inteligence za AV kot CARLA pri enostavnih samostojnih vožnjah.
- H2: Google Colab ima večjo podporo in je lažji za začetnike kot Jupyter Notebook.
- H3: Povečanje števila ponovitev v ciklu samoučenja algoritma v simulaciji AV pri enostavnih samostojnih vožnjah vodi k izboljšanju učinkovitosti algoritma.

- H4: Jupyter Notebook je učinkovitejši za razvoj in testiranje algoritmov umetne inteligence za AV pri enostavnih samostojnih vožnjah v primerjavi z Google Colab za začetnike.

1.3 Predpostavke in omejitve

To diplomsko delo temelji na ključnih predpostavkah: da bosta CARLA in simulator Udacity dostopna in delujoča ter da bosta Jupyter in Google Colab ostala stabilna za dosledno primerjavo. Predpostavlja tudi, da bodo podatkovne zbirke za usposabljanje in testiranje algoritmov reprezentativne za scenarije iz resničnega sveta.

Vendar se raziskava sooča z omejitvami, ki lahko vplivajo na njene rezultate. Razpoložljivost in velikost naborov podatkov lahko vplivata na celovitost simulacij in testiranja algoritmov. Omejena ustrezna literatura bi lahko omejila teoretično raziskovanje. Časovne omejitve zahtevajo zaključek v določenem obdobju, kar omejuje število poskusov. Omejitve virov, vključno z računalniško močjo in pomnilnikom, lahko omejijo obseg in zapletenost simulacij in izvajanje algoritmov. Nazadnje je lahko velikost vzorca za testiranje Jupyter Notebook in Google Colab omejena s časom in viri, kar vpliva na širino primerjalne analize.

1.4 Uporabljene raziskovalne metode

Diplomsko delo uporablja teoretično analizo in aplikativno raziskavo, da bi odgovorilo na zastavljena vprašanja. Prvi del obsega teoretično ozadje, ki raziskuje ključne koncepte, modele in metode pomembne za simulacijo in razvoj AV. Ta del vključuje pregled literature o simulacijskih platformah, razvojnih okoljih in tehnikah optimizacije za samoučenje AV.

Aplikativna raziskava vključuje podrobne analize primerov, diagnostiko in predlagane posege. V primerjalni analizi sta ocenjena simulator CARLA in simulator Udacity za usposabljanje modelov AV na podlagi meril, kot je uporabnost. Eksperimentalno testiranje primerja Jupyter Notebook in Google Colab s praktičnimi primeri, pri čemer se osredotoča na uporabnost, funkcionalnost in zmogljivost. Optimizacijske študije določajo optimalno število zagonov za algoritme za samostojno učenje, pri čemer je treba uravnotežiti učinkovitost učenja in računalniške vire. Praktični prikazi potrjujejo ugotovitve s scenariji iz resničnega sveta in ponujajo konkretne primere, ki podpirajo predlagane hipoteze.

2 UMETNA INTELIGENCA

Umetna inteligenca (UI) je področje, ki proučuje, kako omogočiti računalnikom izvajanje inteligentnih nalog, ki jih je v preteklosti lahko izvajal le človek (Huang, Huan, Xu, Zheng, & Zou, 2019).

Razvoj se je začel že več kot pred 70 leti. Začel se je leta 1943 z modelom umetnega nevrona, kar je privedlo do uradne predstavitve UI na konferenci v Dartmouthu leta 1956. V šestdesetih letih prejšnjega stoletja je zanimanje zanjo upadlo, vendar je v sedemdesetih letih prejšnjega stoletja z algoritmi povratnega širjenja in izboljšano računalniško močjo ponovno napredovala. Osemdeseta leta so prinesla splošno priznanje nevronskih mrež ter napredek na področju strojne opreme in interneta. V 21. stoletju se je uporaba UI razširila z mobilnim internetom, vrhunec pa je dosegla leta 2012 z globokim učenjem, ki je bistveno izboljšalo tehnologije prepoznavanja govora in vida (Zeng, Li , & Duan, 2012).

Umetna inteligenca se v zadnjih letih hitro razvija, zaradi česar so številna podjetja in organizacije optimistične, da jim lahko ta tehnologija pomaga pri reševanju številnih težav, ki so se do zdaj izkazale za nerešljive (Yogesh, in drugi, 2021). Zato naj bi obstajale neprimerljive priložnosti za številna področja uporabe in domene, zlasti sposobnost prepoznavanja vzorcev in korelacij v ogromnih količinah podatkov na ravni kompleksnosti, ki je za človeka nedoumljiva (Hu, Lu, Pan, Gong, & Yang, 2021).

Postala je vroča točka za znanstvene in tehnološke študije; velika podjetja, kot so Google, Microsoft in IBM, se posvečajo UI in jo uporabljajo na vse več področjih (Shi, in drugi, 2007).

2.1 Gonilne sile in tehnologije umetne inteligence

- Big Data – veliki podatki so bistveni za UI, saj znatno povečujejo stopnjo prepoznavanja in natančnost. Eksponentna rast podatkov, ki jo spodbuja internet stvari, zagotavlja obsežne, visoko-dimenzionalne podatkovne nize, potrebne za napreden razvoj UI (Chen & Lin, 2014).
- Algoritmi – tradicionalne metode prepoznavanja vzorcev so bile omejene z abstraktnostjo in natančnostjo. Algoritmi strojnega učenja, kot so nevronske mreže, so se zgledovali po človeškem učenju in lahko samodejno prepoznavajo vzorce v velikih zbirkah podatkov. Ti algoritmi omogočajo napredek v različnih aplikacijah umetne inteligence, vključno s

prepoznavanjem govora in slik (Zhang & Fu, Optimal Model for Patrols of UAVs in Power Grid under Time Constraints, 2021).

- Strojno učenje – izboljšuje zmogljivost z algoritmi, ki temeljijo na podatkih, in rešuje težave, kot so napovedovanje, grozdenje, razvrščanje in zmanjševanje razsežnosti. Vključuje nadzorovano učenje (npr. SVM in regresija), nenadzorovano učenje (npr. k-clustering in PCA), delno nadzorovano učenje (mešanica označenih in neoznačenih podatkov) in okrepljeno učenje (učenje z nagradami in dejanji) (Erhan, in drugi, 2010; Bose, 2017),.
- Obdelava naravnega jezika (NLP – angl. Natural Language Processing) – NLP omogoča računalnikom, da razumejo in obdelujejo človeški jezik. Vključuje naloge, kot so slovnična in semantična analiza, iskanje besedil, strojno prevajanje in pogovorni sistemi, ki računalnikom omogočajo učinkovito razumevanje in ustvarjanje človeškega jezika (Zhang, Xu, & Chen , Theoretical foundations and applications of cyber-physical systems: a literature review, 2020).
- Strojna oprema – globoko učenje (angl. Deep Learning), podmnožica strojnega učenja, temelji na zmogljivi strojni opremi, kot so grafični procesorji. Grafični procesorji NVIDIA pospešujejo globinsko učenje z obsežnimi vzporednimi izračuni, kar v primerjavi s klasičnimi procesorji znatno pospeši postopke usposabljanja (Makkar, in drugi, 2020; Zhao, in drugi, 2019).
- Računalniški vid – računalniški vid omogoča računalnikom, da interpretirajo in analizirajo vizualne informacije. Tehnike, kot so globoko učenje in konvolucijske nevronske mreže (CNN), se uporabljajo za naloge, kot je prepoznavanje obrazov in slik. Napredni modeli, kot sta Faster R-CNN in YOLO, ponujajo visoko natančnost in hitrost za analizo slik v realnem času in semantično segmentacijo (Tan, Hu, & Hanzo, 2019).

2.2 Uporaba umetne inteligence v različnih panogah

- Avtomobilska industrija – avtonomna vožnja je primer integracije umetne inteligence v avtomobilsko industrijo, ki uporablja senzorje in algoritme umetne inteligence za optimizacijo navigacije vozila (Li & Xu, 2001). Kitajska na tem področju napreduje vzporedno z razvojem v Evropi in ZDA. Pomembna mejnika sta Googlov prvi prototip

avtomobila brez voznika iz leta 2014 in Audijeve izboljšave UI iz leta 2017 (Došilović, Brčić, & Hlupić, 2018; Wang, Ma, Zhang, Gao, & Wu, 2018).

- Finančni trgi (»Trading«) – UI spreminja finance, saj se uporablja pri nadzoru tveganja, svetovanju, napovedovanju in bonitetnem ocenjevanju (Wu, Han, Wang, & Sun, 2020). Strojno učenje pomaga upravljati finančna tveganja, slediti potrebam strank in optimizirati naložbene strategije. Podjetja, kot je Alpaca, uporabljajo UI za učinkovito analizo grafov forex trgovanja (Khalaf, Mostafa, Mustapha, Mohammed, & Abduallah, 2019).
- Zdravstvo – UI pomaga pri medicinski diagnostiki, razvoju zdravil in odkrivanju raka (Ravì, in drugi, 2016). IBM-ov Watson na primer uporablja obsežne zbirke medicinskih podatkov za zagotavljanje natančnih diagnoz in zdravstvene pomoči (Esteva, in drugi, 2019).
- Trgovina na drobno – UI povečuje učinkovitost maloprodaje s tehnologijami, kot je "Just Walk Out" podjetja AmazonGo, ki uporablja senzorje in računalniški vid za upravljanje zalog in racionalizacijo nakupovalne izkušnje (Lu, in drugi, 2013). Umetna inteligenca izboljšuje tudi spletno prodajo in upravljanje zalog s priporočilnimi sistemi (Erokhin, 2019).
- Medijska industrija – platforme za ustvarjanje vsebin, ki jih poganja UI, hitro pripravljajo članke in upravljajo komunikacijo blagovnih znamk (Misra, in drugi, 2020). Ti sistemi analizirajo trende in javno mnenje ter tako učinkovito ustvarjajo in razširjajo vsebine (Haenlein & Kaplan, 2019).
- Pametna plačila – UI omogoča inovativne načine plačevanja, kot sta prepoznavanje glasu in obraza, kar zmanjšuje potrebo po fizičnih denarnicah (Farivar, Haghighi, Jolfaei, & Alazab, 2019). Tehnologije, kot je prepoznavanje obraza podjetja Alipay, izboljšujejo hitrost in varnost transakcij (Swamy & Sarojamma, 2020).
- Pametni domovi – sistemi pametnih domov združujejo različne gospodinjske naprave za nemoteno upravljanje in večje udobje. Glasovni pomočniki, kot so pametni zvočniki, imajo ključno vlogo pri upravljanju teh sistemov z glasovnimi ukazi, zaradi česar so pametni domovi uporabniku prijaznejši (Qela & Mouftah, 2012).

2.3 Umetna inteligenca v avtomobilski industriji

Avtomobilski sektor doživlja pomembne spremembe zaradi UI. Ko se govori o UI v povezavi z vozili, jo ljudje pogosto takoj povežejo s samovozečimi avtomobili, pri tem pa spregledajo,

da ima UI v resnici veliko globlji in širši vpliv na osnove avtomobilskega sektorja (Chai & Nizam, 2021).

Osnovna opredelitev AV pove, da gre za osebno vozilo, ki deluje samostojno brez človeške pomoči. AV, znana tudi kot samodejno vodena vozila, avtomobili brez voznika, vozila z avtopilotom ali vozila naslednje generacije, poganjajo avtomatizirani sistemi, ki lahko spremenijo prometni sistem z zmanjšanjem emisij in prometa ter s tem prihranijo gorivo, starejšim in invalidom omogočijo mobilnost ter s preprečevanjem nesreč preprečijo trke s smrtnim izidom. Standard SAE J3016 Združenja avtomobilskih inženirjev opredeljuje 6 ravni AV (Chai & Nizam, 2021).

Stopnja 0 je brez avtomatizacije in zahteva simulacije prometa in senzorskih sistemov. Stopnja 1 vključuje nadzor krmiljenja ali pospeševanja ter dodaja simulacije dinamike vozila in ultrazvočnih senzorjev. Stopnja 2 vključuje tako krmiljenje kot pospeševanje z dodatnim preskušanjem nadzora voznika in vmesnikov človek-stroj. Raven 3 omogoča pogojno avtonomno vožnjo in zahteva simulacije prometne infrastrukture in dinamičnih predmetov. Raven 4 vključuje visoko stopnjo avtomatizacije pod posebnimi pogoji, pri čemer so potrebne simulacije vremena, lidarja, kamere, radarja in kartiranja. Raven 5 pomeni popolno avtomatizacijo v vseh pogojih (Kaur, Taghavi, Tian, & Shi, 2021).

Glavni cilj AV je opravljati številne naloge, ki jih človeški voznik ne more opravljati, na primer ohranjanje zbranosti med utrujenostjo ali spanjem in natančnejše načrtovanje potovanj (Chai & Nizam, 2021).

Algoritmi, kot so globoke nevronske mreže, so zasnovani tako, da posnemajo načela možganov in se usposabljajo na obsežnih naborih podatkov za izvajanje različnih funkcij. Da bi omogočili inteligentno odločanje, inteligentni avtomobili združujejo tehnike umetne inteligence, kot so zaznavanje okolja, izdelava zemljevidov in načrtovanje poti z večplastnimi pomožnimi voznimi storitvami in drugimi funkcijami. Osredotoča se na to, kako se UI, strojno učenje in avtomatizirano krmiljenje uporabljajo v avtomobilih (Li, Cheng, Guo, & Qiu, 2018).

Potreba po inteligentnih avtomobilih zaradi gospodarskega razvoja hitro narašča. Skoraj vse države se poleg stalnega in hitrega povečevanja števila lastnikov vozil soočajo z resnimi težavami povezanimi z varnostjo v cestnem prometu, onesnaževanjem okolja in prometnimi zastoji. Med tem se letno število prometnih nesreč s smrtnim izidom povečuje, pri čemer večino teh nesreč povzročijo človeške napake. Predvideva se, da se bo število prometnih nesreč s smrtnim izidom povečalo, saj se bo število lastnikov avtomobilov še naprej povečevalo. Z

uporabo najsodobnejših metod UI lahko rešimo zgoraj opisane težave (Li, Cheng, Guo, & Qiu, 2018).

Poleg tehnoloških težav so med glavnimi ovirami za široko uporabo AV tudi spori glede odgovornosti. Čas, potreben za preusmeritev trenutnega voznega parka iz neavtonomnega v avtonomni sistem, odpor potrošnikov do predaje nadzora nad vozili, zaskrbljenost potrošnikov glede varnosti avtomobilov brez voznika, izvajanje pravnih okvirov in vladnih predpisov za avtomobile brez voznika, zaskrbljenost zaradi izgube delovnih mest v industriji cestnega prometa zaradi vožnje ter tveganje večje suburbanizacije zaradi lažje in hitrejše vožnje brez ustreznih javnih politik za preprečevanje širjenja mest (Li, Cheng, Guo, & Qiu, 2018).

Sedanja revolucija v informacijski tehnologiji spreminja zasnovo avtomobilov; tehnologija inteligentnih vozil spreminja vedenje ljudi pri vožnji, hkrati pa povečuje prometno varnost, zmanjšuje emisije in varčuje z energijo. To na novo opredeljuje načrtovanje prometa v občinah. Prihodnji inteligentni avtomobili bodo osredotočeni na energetsko učinkovitost, ohranjanje okolja, inteligenco, personalizacijo, varnost in udobje. Rast vgrajenih sistemov komunikacijskih tehnologij in zaznavanja bodo pomembni dejavniki napredka inteligentnih avtomobilov. Trenutno je asistenčna vožnja še vedno v ospredju razvoja tehnologije inteligentnih vozil. Čeprav bo trajalo nekaj časa, da bo dosegla najvišjo raven polavtomatske in popolnoma samodejne faze, bo tehnologija inteligentnih vozil hitro rasla in sčasoma povečala priljubljenost inteligentnih avtomobilov zaradi povečevanja inteligentne tehnologije, oblikovanja ustreznih zakonov in predpisov ter sprejemanja javnosti (Li, Cheng, Guo, & Qiu, 2018).

3 SIMULATORJI

Da bi samovozeče avtomobile usposobili za obvladovanje različnih pogojev, s katerimi se bodo verjetno srečali na javnih cestah, je nujno obsežno in strogo testiranje. Na javnih cestah je fizično testiranje tvegano, drago in ga običajno ni mogoče ponoviti. Za testiranje programske opreme za samovozeče avtomobile je na voljo veliko simulatorjev, ki imajo svoje prednosti in slabosti (Kaur, Taghavi, Tian, & Shi, 2021).

Najbolj realističen simulator je tisti, ki se najbolj približa resničnosti. To pa pomeni, da mora biti izredno natančen, ko gre za izračune na nižji ravni, kot je fizika avtomobila, in izredno celovit, ko gre za 3D virtualno okolje. Zato moramo najti ravnovesje med pristnostjo 3D prizora in preprostostjo dinamike vozila (Figueiredo, Rossett, Braga, & Reis, 2009).

Težava pri simulacijskem testiranju je, da je njegova učinkovitost odvisna od kakovosti uporabljenega simulatorja in stopnje, do katere simulirane okoliščine natančno odražajo dejanski svet (Kaur, Taghavi, Tian, & Shi, 2021).

3.1 CARLA (Car Learning to Act) Simulator – odprtokodni simulator za vožnjo v mestu

Simulator CARLA je bil od samega začetka razvit, da bi podpiral usposabljanje, izdelavo prototipov in potrjevanje modelov avtonomne vožnje, vključno z zaznavanjem in nadzorom. Edinstveno je, da je vsebina mestnih okolij, ki jo ponuja CARLA, tudi brezplačna. Vsebino je od začetka ustvarila posebna ekipa digitalnih umetnikov, ki so bili zaposleni v ta namen. Vključuje urbane načrte, številne modele vozil, stavbe, pešce, ulične znake itd. Simulacijska platforma podpira prilagodljivo nastavitev sklopov senzorjev in zagotavlja signale, ki jih je mogoče uporabiti za urjenje strategij vožnje, kot so koordinate GPS, hitrost, pospešek ter podrobni podatki o trkih in drugih prekrških (Dosovitskiy, Ros, Codevilla, Lopez, & Koltun, 2017). Določiti je mogoče številne okoljske pogoje, vključno z vremenom in dnevnim časom. Številni okoljski pogoji so prikazani na sliki 1.

Za prilagodljivost in realističnost grafike in fizikalnega modeliranja je bila razvita aplikacija CARLA. V pogonu Unreal Engine 4 je izveden kot odprtokodni sloj (Epic Games, brez datuma), ki omogoča razširitve skupnosti v prihodnosti. Pogon ponuja sodobno kakovost upodabljanja, realistično fiziko, temeljno razmišljanje NPC in mrežo združljivih vtičnikov. Pogon lahko brezplačno uporabljamo v nekomercialne namene. CARLA je simulacijski sistem

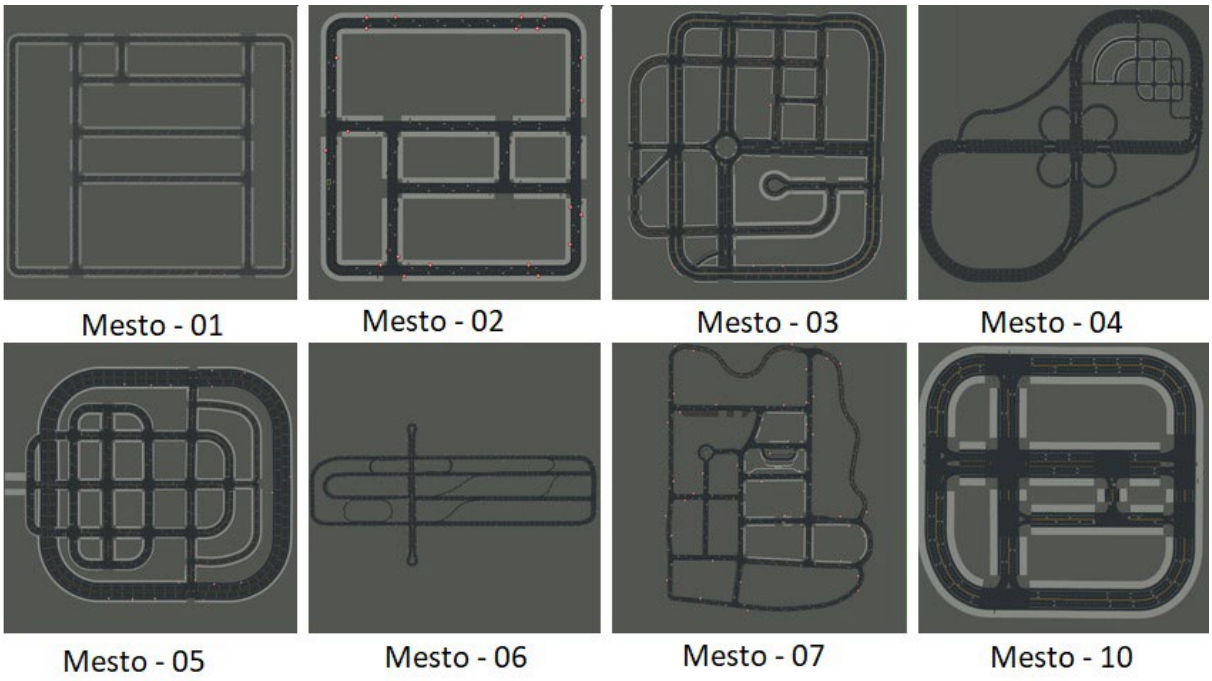
strežnik-odjemalec, kjer strežnik izvaja simulacijo in upodablja prizor, medtem ko client – API v Python upravlja interakcije prek vtičnic. Odjemalec strežniku pošilja ukaze (krmiljenje, pospeševanje, zaviranje) in meta ukaze (ponastavitev, spreminjanje nastavitev okolja, spreminjanje senzorjev). Okolje vključuje podrobne 3D-modele statičnih in dinamičnih predmetov, pri čemer je poudarek na uravnoteženju vizualne kakovosti in hitrosti upodabljanja z učinkovitimi geometrijskimi modeli in teksturami (Dosovitskiy, Ros, Codevilla, Lopez, & Koltun, 2017).

Knjižnica digitalnih vsebin vključuje štirideset vrst stavb, šestnajst modelov vozil in petdeset modelov pešcev. Urbana okolja v sistemu CARLA so ustvarjena z risanjem cest in pločnikov, ročnim postavljanjem statičnih objektov (kot so stavbe in prometni znaki) ter določanjem lokacij dinamičnih objektov. CARLA vsebuje osem mest, vsako z ne- in večplastnimi zemljevidi. Podpira tudi realistične ne-igralske akterje, saj uporabnikom omogoča nastavljanje kinematičnih parametrov in izvajanje krmilnikov za obnašanje vozil, vključno s sledenjem voznemu pasu in odločanjem v križiščih. Vozila in pešci lahko zaznajo in se izogibajo drug drugemu, uporabniki pa lahko vključijo napredne krmilnike za vozila (Malik, Khan, & El-Sayed, 2021).

Tabela 1: Opis vseh zemljevidov mest brez plasti, ki so na voljo v simulatorju CARLA

Mesto	Značilnosti.
Mesto – 01	Osnovni načrt mesta s križišči v obliki črke "T".
Mesto – 02	Podobno mestu 01, vendar manjše.
Mesto – 03	Najbolj zapleteno mesto s petpasovnim križiščem, krožiščem, neravninami, predorom in drugimi elementi.
Mesto – 04	Neskončna zanka z avtocesto in majhnim mestom.
Mesto – 05	Mesto s kvadratno mrežo, križiščem in mostom ter več voznimi pasovi v vsako smer.
Mesto – 06	Dolge avtoceste z veliko vhodi in izhodi z avtoceste.
Mesto – 07	Podeželsko okolje z ozkimi cestami, skednji in skoraj nobenim semaforjem.
Mesto – 10	Mestno okolje z različnimi okolji in bolj realističnimi teksturami.

Vir: (Lastni vir)



Slika 1: Ilustracija zemljevidov brez plasti v simulatorju CARLA

Vir: (Lastni vir)

3.1.1 Namestitev in konfiguracija

CARLA je prenosna in jo je mogoče namestiti v operacijska sistema Linux in Windows, saj deluje v pogonu Unreal Engine 4. Za optimalno delovanje zahteva posebne specifikacije strojne in programske opreme, čeprav lahko deluje tudi na nižjih specifikacijah z manjšo zmogljivostjo. Program CARLA lahko namestimo tako, da ga sestavimo iz izvorne kode ali z uporabo enostavnejše namestitve paketa, ki je na voljo v repozitoriju za izdajo, ki vključuje vse izdaje in digitalna sredstva. Po namestitvi lahko sistem CARLA deluje v dveh načinih: Samostojni način in Strežniški način. Koraki so opisani spodaj (Malik, Khan, & El-Sayed, 2021).

Postopek:

```
1. # Kloniranje repozitorija CARLA iz GitHuba
2.  git clone https://github.com/carla-simulator/carla cd carla
```

Gradnja programa CARLA iz izvorne kode (opcijsko):

```
1. # Poskrbimo, da smo v direktoriju Carla
2.  make PythonAPI
```

Namestitev odvisnosti za Python API:

```
1. pip install - r PythonAPI/Carla/dist/requirements.txt
```

Zagon sistema CARLA v samostojnem načinu:

```
1. # Navigacija v imenik CARLA
2. cd Carla
3.  # Zagon samostojnega načina
4. ./CarlaUE4.sh
```

Po mestu se premikamo s tipkovnico: Za vožnjo in raziskovanje simulacijskega okolja uporabljamo tipkovnico (na primer puščične tipke ali WASD).

Prilagodimo lahko nastavitve za CARLA:

```
1. nano CarlaSettings.ini
```

Zagon sistema CARLA v strežniškem načinu:

```
 1. # Namestimo odvisnosti odjemalca Python
 2. pip install -r PythonClient/requirements.txt
 3.
 4. # Zagon strežnika CARLA
 5. ./CarlaUE4.sh -carla-server
 6.
 7. # Zaženemo skripto primera odjemalca
 8. # Navigiramo v direktorij PythonClient
 9. cd PythonClient
10. # Zaženemo skripto primera odjemalca
11. python example.py
```

Tabela 2: Specifikacije strojne in programske opreme za CARLA

Zahteve	Strojna in programska oprema	Upoštevani stroški
Sistemske zahteve	CARLA lahko deluje v katerem koli 64-bitnem operacijskem sistemu	Linux – brezplačno Windows – 140 €–200 €/licenca macOS – stroški so samo za mac računalnike
Ustrezen grafični procesor	Strežnik zahteva vsaj 6 GB grafičnega procesorja, čeprav je priporočljiv 8 GB grafični procesor	Približno 100 € za 8 GB grafični procesor

Prostor na disku		CARLA potrebuje približno 20 GB prostora	90 – 150 € (16 GB + 4 GB ali 2 x 8 GB)
Python		CARLA za pisanje skript podpira program Python 3.5.x in Python 3.6.x	Brezplačno
Python module		Modula Pygame za ustvarjanje grafike neposredno s Pythonom in Numpy za odlično računanje	Brezplačno
Porti TCP		Zahtevana sta porta 2000 in 2001	Brezplačno
Ogrodje		Unreal	CARLA je open-source in je brezplačna
Ubuntu	Windows	Docker	Ker so vse te stvari vključene v računalnik, bo edini strošek odvisen od tega, ali bo potrebna nadgradnja. Zato je edini pogojni strošek licenca za Windows.
Ubuntu 16.04 ali 18.04	Windows 7 ali 8	Docker	
Grafični gonilniki NVIDIA >361.93	Update Graphics Drivers	NVIDIA-Docker2	
OpenGL 3.3	Visual Studio	NVIDIA Driver >= 390	
Dodatne odvisnosti	OpenGL 3.3 ali več ali DirectX 10 ali več	CARLA Simulator	
	Dodatne odvisnosti		

Vir: (Malik, Khan, & El-Sayed, 2021)

3.2 Udacity Simulator

Udacity ponuja spletni tečaj, ki z uporabo globokega učenja usposablja agenta za avtonomno vožnjo in vključuje odprtokodni tridimenzionalni simulator vožnje z enojnim prikazom. Pogled na zbiranje podatkov je z zadnjega dela vozila, fotografije z opombami pa so ustvarjene z

voznikovega zornega kota. Projekt poleg dveh vnaprej nameščenih skladb vključuje tudi edinstven modul za ustvarjanje skladb. Ta simulator nima dodatnih avtomobilov, ima bolj zapletene vizualne podobe in je manj prilagodljiv (Hsieh, 2017).

Simulator olajša naloge s svojo arhitekturo odjemalec-strežnik, uporabniku pa s prijaznim vmesnikom API za zbiranje in prenos podatkov. Deluje v dveh načinih: način usposabljanja in avtonomni način. V načinu usposabljanja simulator snema posnetke iz treh kamer (leva, srednja, desna) in jih povezuje s parametri, kot so hitrost, plin, kot krmiljenja in zaviranje. V avtonomnem načinu nastavitev odjemalec-strežnik omogoča komunikacijo podatkov v realnem času prek vmesnika API, kar usposobljenemu modelu omogoča uporabo podatkovnih tokov v živo za posodabljanje parametrov in odzivov vozila (Gupta, Upadhyay , Kumar, & Al-Turjman, 2021).

Slika 2: Udacity Simulator

3.2.1 Namestitev in konfiguracija

3.2.1.1 Ubuntu

Če ga želimo uporabljati, najprej namestimo različico Unity3D 5.5.1f1, saj je simulator ustvarjen s to različico. Prenesemo Unity3D in sledimo korakom:

Namestimo zahtevane odvisnosti:

```
1. sudo apt install gconf-service lib32gcc1 lib32stdc++6 libc6-i386 libgconf-2-4
npm
```

Namestimo paket Unity3D:

```
1. sudo dpkg -i ~/Downloads/unity-editor_amd64-5.5.1xf1Linux.deb
```

Odpravimo vse manjkajoče odvisnosti:

```
1. sudo apt --fix-broken install
```

Omogočamo shranjevanje velikih datotek (LFS) za sistem Git:

```
1. curl -s https://packagecloud.io/install/repositories/github/git-
lfs/script.deb.sh | sudo bash
```

Kloniramo repozitorij simulatorja:

```
1. git clone https://github.com/udacity/self-driving-car-sim.git
```

- Zagon simulatorja:

Simulator naredimo izvršljiv:

```
1. sudo chmod +x ~/Downloads/beta_simulator_linux/beta_simulator.x86_64
```

Simulator zaženemo z mesta prenosa:

```
1. ~/Downloads/beta_simulator_linux/beta_simulator.x86_64
```

Za namenski grafični način zaženemo simulator z:

```
1. RUN_GRAPH=true ~/Downloads/beta_simulator_linux/beta_simulator.x86_64
```

Iz simulatorja izstopimo z Alt + F4.

3.2.1.2 Windows

Če želimo namestiti simulator v operacijskem sistemu Windows, prenesemo datoteko zip iz github-a z razpoložljivimi datotekami, jo razširimo in zaženemo izvršilni program. Za namestitev sledimo korakom:

Repozitorij kloniramo s sistemom Git LFS:

```
1. git lfs install
```

Kloniramo repozitorij:

```
1. git clone https://github.com/udacity/self-driving-car-sim.git
```

Namestimo program Unity (če še ni nameščen):

- Delo s skriptami in gradnja skladb:

Najdemo skripte za uporabniški vmesnik in vtičnike:

```
1. Assets/1_SelfDrivingCar/Scripts
```

Najdemo skripte za interakcije z avtomobili:

```
1. Assets/Standard Assets/Vehicle/Car/Scripts
```

Zgradimo novo progo:

```
1. Assets/RoadKit/Prefabs
```

Tabela 3: Specifikacije strojne in programske opreme za Udacity Simulator

Zahteve	Windows minimum	Windows maximum	Ubuntu minimum	Ubuntu maximum	Upoštevani stroški
Operacijski sistem (OS)	Windows 7/8/10	Windows 10	Ubuntu 16.04 ali novejši	Ubuntu 18.04 ali novejši	Win – 140 €– 200 €/licenca Lin – brezplačno macOS – stroški računalnika
Procesor	Intel Core i5-2500K ali enakovreden procesor AMD	Intel Core i7 ali AMD Ryzen 7	Intel Core i5 ali enakovreden procesor AMD	Intel Core i7 ali AMD Ryzen 7	Intel Core i5 – od 230 € naprej. Intel Core i7 – od 250 € naprej.
Spomin	8 GB RAM	16 GB RAM	8 GB RAM	16 GB RAM	8GB RAM – od 20 € naprej. 16GB RAM – od 50 € naprej.
Grafika	NVIDIA GeForce GTX 670 ali AMD Radeon HD 7870	NVIDIA GeForce GTX 1060 ali AMD Radeon RX 580	NVIDIA GeForce GTX 670 ali AMD Radeon HD 7870	NVIDIA GeForce GTX 1060 ali AMD Radeon RX 580	NVIDIA GeForce GTX 670 – 415 € AMD Radeon HD 7870 – 240 € NVIDIA GeForce GTX 1060 – 500 € AMD Radeon RX 580 – 350 €

DirectX	Version 11	Version 12	N/A	N/A	
Pomnilnik	10 GB razpoložljivega prostora	10 GB razpoložljivega prostora	10 GB razpoložljivega prostora	10 GB razpoložljivega prostora	Cena je odvisna od izbrane grafične kartice
Unity različica	Unity 5.5.1f1	Unity 5.5.1f1	Unity 5.5.1f1	Unity 5.5.1f1	Personal – brezplačno Pro – od 2,040.00 $/leto Industry – od 4,950.00 $ /leto
Dodatno	Git LFS za velike vrednosti	Posodobljeni grafični gonilniki	Git LFS za velike vrednosti	Posodobljeni grafični gonilniki	Ni dodatnih stroškov

Vir: (Lastni vir)

3.3 Udacity Simulator vs CARLA

CARLA je primerna za napredne raziskave in razvoj, saj omogoča zelo natančno simulacijo kompleksnih metropolitanskih območij, različnih vremenskih razmer in dinamičnih predmetov. Zagotavlja zelo prilagodljivo in vsestransko platformo, vendar zahteva veliko računalniške moči. Omogoča široko interakcijo s številnimi ogrodji za robotiko in strojno učenje. Za programom CARLA stoji živahna skupnost, ki zagotavlja bogato dokumentacijo in pomoč. Zaradi svoje fine vizualne podobe in zapletenih tekstur, ki povečujejo vizualni realizem, je kot nalašč za zapleteno testiranje algoritmov in potrjevanje sistemov (Li, in drugi, 2024).

Simulator samovozečega avtomobila Udacity pa je namenjen predvsem za izobraževalne namene. V primerjavi s simulatorjem CARLA ima preprostejša okolja in manj dinamičnih elementov. Simulator je enostavnejši za vzpostavitev in uporabo, saj se osredotoča na preprosto interakcijo in osnovne koncepte samovozečega avtomobila. Ima bolj omejene možnosti integracije in je manj zahteven z vidika računalniških virov. Čeprav so njegova podpora skupnosti in posodobitve bolj omejene, je simulator Udacity primeren za uvodno učenje in izobraževalne vaje s poenostavljeno grafiko in teksturo. Njegova prilagodljivost je prav tako omejena, saj se osredotoča na vnaprej določene scenarije in ne na obsežno prilagajanje (Li, in drugi, 2024).

Glede na vse to je simulator CARLA zaradi velike natančnosti in številnih funkcij primernejši za napredne študije, medtem ko je simulator Udacity zasnovan z mislijo na poučevanje in poudarja preprostost uporabe in osnovne ideje.

Tabela 4: Primerjava simulatorjev CARLA in Udacity Self-Driving Car Simulator

Aspekt	CARLA	Udacity Simulator
Link na install page	CARLA Quick Start	Udacity Installation
Verodostojnost simulacije	Visok realizem s podrobnimi mestnimi okolji, vključno z različnimi vremenskimi razmerami in dinamičnimi predmeti.	V primerjavi z igro CARLA je manj podrobna, z enostavnejšimi okolji in manj dinamičnimi elementi.
Enostavno vključevanje	Podpira integracijo z različnimi okvirji za strojno	Osnovne možnosti integracije, osredotočen

	učenje in robotiko. Dobro dokumentirani API-ji in močna skupnost za podporo.	predvsem na izobraževalno uporabo z omejeno razširljivostjo.
Uporabnost	Bolj zapletena nastavitev in konfiguracija, vendar ponuja obsežno prilagajanje in napredne funkcije.	Lažje ga je nastaviti in uporabljati, zasnovan je za izobraževalne namene s poudarkom na enostavni interakciji.
Podpora skupnosti	Aktivna in dobro podprta skupnost z obsežnimi viri in dokumentacijo. Redne posodobitve in izboljšave.	Omejena podpora skupnosti in manj posodobitev, vzdržuje se predvsem v izobraževalne namene.
Zmogljivost in sistemske zahteve	Zahteva znatne računalniške vire, zlasti za simulacije visoke verodostojnosti s podrobnimi okolji.	Na splošno so manj zahtevni; primerni so za izobraževalne namene in simulacije, ki zahtevajo manj virov.
Aplikacije	Primeren je za široko paleto raziskovalnih in razvojnih aplikacij, vključno z naprednim testiranjem algoritmov in potrjevanjem sistemov.	Zasnovan je predvsem za izobraževalne in uvodne namene s poudarkom na osnovnih konceptih samovozečega avtomobila in simulacij.
Grafika in vizualna podoba	Visokokakovostna grafika s podrobnimi teksturami in realističnimi vizualnimi elementi.	Poenostavljena grafika in teksture z manjšim poudarkom na vizualnem realizmu.
Prilagodljivost in razširljivost	Zelo prilagodljiv, saj uporabnikom omogoča ustvarjanje prilagojenih okolij in scenarijev.	Omejena prilagodljivost, osredotočeno na vnaprej določene scenarije in izobraževalne vaje.

Vir: (Lastni vir)

4 KNJIŽNICE IN RAZVOJNA OKOLJA

Beležnica Jupyter zagotavlja interaktivno programsko okolje, ki združuje kodo z besedilom markdown, kar pomaga pri učenju in poučevanju. Zmožnost izvajanja kode v kosih in takojšnje povratne informacije omogočajo lajšanje napak in razumevanje zapletenih konceptov, kot so nevronske mreže (Menke, Homberg, & Koch, 2023).

Beležnico je mogoče gostiti na GitHubu, kar omogoča enostaven dostop in sodelovanje. Uporabniki lahko klonirajo repozitorije, jih prilagajajo in ponovno vključujejo spremembe, kar omogoča nenehne izboljšave in sodelovanje skupnosti. Funkcije GitHuba, kot je „Issue Tracker“, dodatno izboljšujejo interakcijo in povratne informacije uporabnikov (Taieb, 2018).

Google Colaboratory (Colab) platforma je dostopen način za zagon beležnic Jupyter brez potrebe po lokalni namestitvi. Ponuja računalniške vire v oblaku, vključno z grafičnimi procesorji in procesorji TPU, zaradi česar je primerna za umetno inteligenco in globoko učenje. Glavna pomanjkljivost je, da podatkov in nastavitev ni mogoče shraniti, ko se primerek strežnika zapre (Nelson & Hoover, 2020).

4.1 Jupyter Notebook

Najbolj priljubljena platforma za interaktivno pismeno programiranje je beležnica Jupyter (Shen, 2014). Njen namen je bil olajšati dokumentiranje, izmenjavo in ponovitev analize podatkov. Od leta 2013, ko je sistem začel delovati, je bilo v GitHubu zbranih več kot 9 milijonov beležnic (Parente, 2020).

Jupyter izhaja iz IPythona in poleg Pythona podpira različne programske jezike, kot so Julia, R, JavaScript in C. Poleg kode in besedila omogoča tudi prepletanje različnih vrst bogatih medijev, vključno s slikami, videom in celo interaktivnimi gradniki, ki združujejo HTML in JavaScript (Perez & Granger, 2007).

Odprtokodna aplikacija Jupyter Notebook služi kot virtualni laboratorijski zvezek za podporo podatkov, kode, delovnih postopkov in vizualizacij raziskovalnega procesa. Njena strojno in človeško berljiva narava spodbuja znanstveno sodelovanje in interoperabilnost. Te beležnice je mogoče shraniti v spletnih skladiščih in jih povezati z drugimi raziskovalnimi artefakti, vključno s kodo, članki, delovnimi tokovi, priročniki za tehnike in podatkovnimi zbirkami (Randles, Pasquetto, Golshan, & Borgman, 2017).

Vendar je ta oblika vse bolj tarča kritik zaradi spodbujanja nezaželenih navad, ki povzročajo nepredvideno vedenje in jih ni mogoče ponoviti (Xie, 2018). Med glavnimi kritikami so skrita stanja, nepričakovan vrstni red izvajanja z razdrobljeno kodo ter slabe prakse pri poimenovanju, različicah, testiranju in modularizaciji kode. Poleg tega oblika beležnice ne kodira odvisnosti knjižnic s pripetimi različicami, zaradi česar je težko (in včasih nemogoče) reproducirati beležnico. Te kritike potrjujejo prejšnje delo, ki je poudarilo negativen vpliv pomanjkanja najboljših praks programskega inženirstva (Wilson, in drugi, 2014) v programski opremi za znanstveno računalništvo glede ločevanja skrbi (Hursch & Lopes, 1995), testov in vzdrževanja (Neglectos, 2018).

4.2 Google Colab

Čeprav je bil program Colab ustvarjen za lažjo izmenjavo ponovljivih poskusov in opisov tehnik med raziskovalci na področju umetne inteligence in znanosti o podatkih, so ugotovili, da je odlično orodje tudi za izobraževalne namene. Glavna prednost je v tem, da lahko učenci z dovolj procesorske moči interaktivno izvajajo napredne pristope umetne inteligence, saj lahko uporabljajo inštruktorjeve delovne zvezke v skupni rabi. Tako uporabnikom ni treba individualno konfigurirati programskih paketov in odvisnosti (Nelson & Hoover, 2020).

Notebooke delujejo v virtualnih strojih (VM – Virtual Machine), ki temeljijo na operacijskem sistemu Linux in jih vzdržuje in zagotavlja Google. Ti VM omogočajo izvajanje izračunov s centralnimi procesnimi enotami (CPU – Central Processing Unit) ali pospešeno z uporabo specializiranih grafičnih procesorjev in tenzorskih procesnih enot (TPU – Tensor Processing Unit). Vsak VM ima za posamezno sejo na voljo različno strojno opremo, čeprav so običajno na voljo vrhunski grafični procesorji NVIDIA (K80, T4 ali P100), 8-12 GB pomnilnika RAM in 50–70 GB prostega prostora na trdem disku VM. Notebooke Colab so zasnovani za interaktivno uporabo in ne za daljše preiskave. Zato se VM po izteku časa mirovanja prekinejo in imajo 12-urno omejitev seje (Nelson & Hoover, 2020).

Google Colab deluje kot npr. Google Docs in omogoča uporabnikom, da skupaj delajo na istem Notebook-u. TensorFlow, Matplotlib in Keras so le nekatere od ključnih knjižnic za strojno učenje in umetno inteligenco, s katerimi je program Colaboratory predhodno konfiguriral izvajalne sisteme Python 2 in 3. Po določenem času se VM pod izvajalnim časom zapre, vse uporabniške nastavitve in podatki pa izginejo. Kljub temu pa beležnica ostane nedotaknjena, informacije pa se lahko s trdega diska virtualnega stroja prenesejo na uporabnikov račun Google

Drive. Nazadnje, ob popolni konfiguraciji prej omenjene programske opreme, ta Googlova storitev ponuja izvajanje s pospeševanjem z grafičnim procesorjem. Google Cloud služi kot gostiteljska platforma za infrastrukturo Google Colaboratory (CARNEIRO, in drugi, 2018).

4.3 Google Colab vs Jupyter Notebook

Jupyter Notebook in Google Colab sta priljubljeni interaktivni programski orodji, ki imata vsaka svoje prednosti in slabosti. Ker Jupyter Notebook uporabnikom omogoča, da popolnoma prilagodijo svoj delovni prostor, je odlično orodje za lokalno delo. To vključuje možnost ohranjanja in spreminjanja določenih parametrov, kar olajša preprosto repliciranje operacij. Poleg tega se Jupyter povezuje s številnimi platformami, kot je GitHub, in ponuja široko podporo za več programskih jezikov, kar spodbuja sodelovanje in izmenjavo raziskovalnih dosežkov.

Nasprotno pa je Google Colab odličen vir za vse, ki želijo izkoristiti zmogljivosti v oblaku, ne da bi za to potrebovali zapletene lokalne nastavitve. V Colabu so na voljo močni računski viri, kot so enote za obdelavo tenzorjev (TPU) in grafične procesne enote (GPU), kar je zelo koristno za globoko učenje in druge najsodobnejše tehnike umetne inteligence. Uporabnikom se ni treba ukvarjati z nameščanjem potrebnih knjižnic ali skrbeti za strojno opremo, saj deluje v oblaku.

Google Colab prinaša veliko prednosti, vendar ima tudi slabosti. Glavna je, da se ob zaprtju navideznega stroja izgubijo podatki in nastavitve, saj okolja ni mogoče shraniti med sejami. Dolgotrajnejše raziskave lahko zaradi tega postanejo zahtevne, zato bodo potrebne pogoste varnostne kopije na Googlovem disku ali v drugi spletni shrambi. Za izboljšanje ponovljivosti in dolgoročnega vodenja projektov ponuja beležnica Jupyter popoln nadzor nad delovnim okoljem vključno z različicami knjižnic in drugimi spremenljivkami.

Medtem ko se Google Colab odlikuje po dostopnosti, enostavnosti uporabe in močnih virih v oblaku, zaradi česar je idealen za hitro testiranje in izdelavo prototipov, beležnica Jupyter ponuja večjo prilagodljivost in vzdržljivost za dolgoročne projekte. Posebne zahteve uporabnika, vključno s tistimi povezanimi z računsko močjo, ponovljivostjo, sodelovanjem in želenim okoljem pogosto določajo, katera možnost je najboljša.

Tabela 5: Primerjava lastnosti Google Colab in Jupyter Notebook

Lastnost	Google Colab	Jupyter Notebook

Okolje	Storitev v oblaku	Lokalno ali na strežniku
Dostop	Zahteva internetno povezavo	Lahko se uporablja brez povezave
Računalniški viri	Zagotavlja brezplačen dostop do grafičnih procesorjev in procesorjev TPU	Zanaša se na lokalno ali strežniško strojno opremo
Nastavitev in konfiguracija	Nastavitev ni potrebna, na voljo so vnaprej konfigurirane knjižnice	Zahteva ročno nastavitev in konfiguracijo
Sodelovanje	Sodelovanje v realnem času, podobno kot v Googlovih dokumentih	Sodelovanje prek skupnih datotek ali nadzora različic (npr. GitHub)
Trajanje seje	Omejeno na 12-urne seje, ponastavitev VM izgubi podatke	Trajne seje s popolnim nadzorom nad okoljem
Podprti jeziki	Predvsem Python (podpira druge z dodatnimi nastavitvami)	Podpira več jezikov, vključno z jeziki Python, R, Julia itd.
Shranjevanje podatkov	Začasno, podatke je treba ročno shraniti v Google Drive ali druge storitve v oblaku	Lokalno shranjevanje s trajnimi datotečnimi sistemi
Reproduktibilnost	Omejeno zaradi ponastavitve VM, potrebna je ponovna namestitev paketov	Visoka, saj je mogoče okolja v celoti nadzorovati in reproducirati
Idealni primer uporabe	Hitro prototipiranje, poskusi globokega učenja, izobraževalni nameni	Dolgoročni projekti, zapleteni delovni tokovi, popoln nadzor nad okoljem

Upoštevani stroški	V brezplačni različici programa Colab je dostop do grafičnih procesorjev zelo omejen Colab Pro – 11,28 €/mesec Colab Pro+ – 51,54 €/mesec Colab Enterprise – Plačilo po uporabi	Uporaba je prosto dostopna

Vir: (Lastni vir)

5 UVOD V SAMOUČENJE ALGORITMA

5.1 Strojno učenje

Znanstveno preučevanje statističnih modelov in tehnik, ki jih računalniški sistemi uporabljajo za izvajanje določenih nalog brez izrecnega programiranja, je znano kot strojno učenje ali krajše ML (Machine Learning). Eden od razlogov, zakaj je spletni iskalnik, kot je Google, tako dober vsakič, ko ga uporabimo za iskanje po internetu, je, da ima algoritem, ki se nenehno uči, kako razvrščati spletna mesta. Prediktivna analitika, obdelava slik, podatkovno rudarjenje in druge uporabe teh algoritmov so le nekatere. Glavna prednost strojnega učenja je sposobnost algoritmov, da samodejno opravljajo naloge, ko se naučijo, kako ravnati s podatki (Mahesh, 2018). Na kratko si oglejmo nekaj najpogosteje uporabljenih algoritmov v ML (Nasteski, 2017):

- Nadzorovano učenje (Supervised Learning): vključuje algoritme, ki se učijo iz označenih podatkov za napovedovanje rezultatov. Različni algoritmi ustvarijo funkcijo, ki preslika vhodne podatke v želene izhodne podatke. Ena od standardnih formulacij naloge nadzorovanega učenja je problem klasifikacije: uporabnik se mora naučiti (približati obnašanje) funkcije, ki prikazuje vektor v enega od več razredov, tako da preuči več vhodno-izhodnih primerov funkcije.
- Nenadzorovano učenje (Unsupervised Learning): pri tem se osredotoča na algoritme, ki delajo z neoznačenimi podatki in iščejo skrite vzorce.
- Delno nadzorovano učenje (Semi-Supervised Learning): združuje tako označene kot neoznačene podatke.
- Učenje z okrepitvijo (Reinforcement Learning): algoritem se nauči, kako naj ravna glede na opazovanje sveta. Vsako dejanje ima določen vpliv na okolje, okolje pa zagotavlja povratne informacije, ki usmerjajo učni algoritem.
- Večopravilno učenje (Multitask Learning): cilj je sočasno reševanje več nalog z izkoriščanjem podobnosti med njimi.
- Učenje v skupinah (Ensemble Learning): vključuje združevanje več modelov za izboljšanje splošne učinkovitosti, pri čemer se preučujejo metode, kot sta Boosting in Bagging.

5.2 Globoko učenje

Globoko učenje omogoča računalniškim modelom, ki so sestavljeni iz več slojev obdelave, da se naučijo predstavitev podatkov z več ravnmi abstrakcije. Te metode so bistveno izboljšale stanje na področju prepoznavanja govora, vizualnega prepoznavanja predmetov, zaznavanja predmetov in številnih drugih področjih, kot sta odkrivanje zdravil in genomika. Z uporabo tehnike povratnega širjenja, s katero se predlagajo spremembe notranjih parametrov stroja, ki se uporabljajo za izračun predstavitve v vsaki plasti na podlagi predstavitve v prejšnji plasti. Globinsko učenje odkriva kompleksno strukturo znotraj obsežnih podatkovnih nizov. Rekurentne mreže so osvetlile zaporedne podatke, kot sta besedilo in glas, medtem ko so globoke konvolucijske mreže pomembno napredovale pri obdelavi slik, videa, govora in zvoka (Lecun, Bengio, & Hinton, 2015). V nasprotju s splošnim strojnim učenjem se pri globokem učenju uporablja kaskada slojev nelinearnih procesnih enot za ekstrakcijo in spreminjanje funkcij. S hierarhično predstavitvijo podatkov, kjer se značilnosti višje ravni ustvarjajo iz informacij nižje ravni, omogoča računalnikom učenje (Hao, Zhang, & Ma, 2016).

Globoke arhitekture so na voljo v številnih različicah, za predstavitev različnih virov podatkov pa se lahko uporabljajo različne strukture. Predstavljajo nabor modelov nevronskih omrežij, zasnovan za samodejno učenje in pridobivanje lastnosti iz podatkov prek več plasti, ki omogoča naloge, kot so prepoznavanje slik, obdelava naravnega jezika in kompleksno odločanje. Ključne arhitekture vključujejo konvolucijske nevronske mreže za prostorske podatke, rekurentne nevronske mreže in transformatorje za zaporedne podatke ter generativne adverzijske mreže za generiranje novih vzorcev podatkov. Konvolucijske nevronske mreže se na primer najpogosteje uporabljajo za prepoznavanje slik, rekurentne nevronske mreže pa bolje delujejo pri zaporednih aplikacijah, kot je prepoznavanje glasu ali rokopisa (Hao, Zhang, & Ma, 2016).

5.3 Definicija samoučenja v kontekstu avtonomnih vozil

Aplikacije UI v avtomobilskem sektorju segajo precej dlje od razvoja, inženiringa, logistike, proizvodnje, oskrbovalne verige, uporabniške izkušnje, trženja, prodaje, poprodajnih storitev in storitev mobilnosti. UI je ključ do nove prihodnosti glede vrednosti za avtomobilsko industrijo (Hofmann, Neukart, & Bäck, 2017).

V zadnjem času so bili predstavljeni številni testni projekti s samovozečimi avtomobili. Vsem tem eksperimentalnim projektom je skupno, da se pri nekaterih nalogah vožnje, kot so načrtovanje poti, zavedanje okolja in celo upravljanje volana, uporabljajo metode, ki temeljijo

na globokem učenju. Z uspešno predstavitvijo avtonomnih prototipov, ki jih poganja globoko učenje, se avtomobilska industrija postopoma preusmerja od izdelave in predstavitve prototipnih vozil k serijski proizvodnji. Danes je glavni izziv, kako nevronske mreže spraviti v serijsko proizvodnjo avtomobilov na način, ki bo skladen z varnostnimi zahtevami (Rao & Frtunikj, 2018).

Natančno zaznavanje drugih avtomobilov na cesti z uporabo računalniškega vida je zahtevna tema, ki je v zadnjih dvajsetih letih pritegnila veliko pozornosti (Sun, Bebis, & Miller, 2006). Ceste, po katerih vozijo avtomobili, so dinamične, z nenehno spreminjajočo se osvetlitvijo in ozadjem. Ker vsak avtomobil na cesti pogosto vozi v istem trenutku, se velikost in lokacija vozila v slikovni ravnini, ki jo zajame fotoaparat, razlikujeta. Velikost, barva in oblika vozila se lahko v vsakdanjih voznih razmerah zelo razlikujejo. V literaturi je že več kot deset let veliko raziskav o sledenju in zaznavanju vozil. V prejšnjih raziskavah je bilo za odkrivanje vozil uporabljenih več umetno ustvarjenih značilnosti (Capparuccia, Renato, & Marchitto, 2007).

6 ALGORITMI ZA SAMOSTOJNO VOŽNJO

Razvoj algoritmov za samovozeče avtomobile je že na najosnovnejši ravni zahteven podvig, ki vključuje vrsto senzorjev, nadzornih shem in računalniških pristopov. Poudarek pri »enostavnih« ali vstopnih samovozečih avtomobilih bi bil verjetno na nadzorovanih poteh, parkiriščih ali testnih stezah in ne na zapletenih situacijah v resničnem svetu (Badue, in drugi, 2021).

6.1 Algoritmi zaznavanja

Zajemanje pomeni razumevanje okolja okoli avtomobila s pomočjo različnih senzorjev. Pri osnovnem samovozečem avtomobilu se algoritmi zaznavanja lahko osredotočijo na zaznavanje preprostih ovir in oznak voznega pasu (Rosique, Navarro, Fernández, & Padilla, 2019).

- Fuzija senzorjev: združitev podatkov iz več senzorjev (npr. kamer, LIDAR, ultrazvočnih senzorjev) za ustvarjanje celovitega pogleda na okolje. Pri preprostejših aplikacijah lahko to vključuje le kamero in nekaj ultrazvočnih senzorjev (Rosique, Navarro, Fernández, & Padilla, 2019).
- Zaznavanje predmetov: uporaba osnovnih tehnik računalniškega vida ali predhodno usposobljenih modelov strojnega učenja za zaznavanje predmetov, kot so pešci, druga vozila ali statične ovire. Tehnike vključujejo zaznavanje robov, zaznavanje madežev in naprednejše metode, kot so konvolucijske nevronske mreže (Rosique, Navarro, Fernández, & Padilla, 2019).
- Zaznavanje voznih pasov: algoritmi za zaznavanje oznak voznih pasov na cesti, pri čemer se pogosto uporabljajo tehnike računalniškega vida, kot je Houghova transformacija za prepoznavanje ravnih črt ali krivulj (Rosique, Navarro, Fernández, & Padilla, 2019).

6.2 Algoritmi za lokalizacijo

Pri lokalizaciji gre za določanje natančnega položaja avtomobila v danem okolju (Lu, Ma, Smart, & Yu, 2021).

- Lokalizacija na podlagi GPS: pri osnovnih samovozečih avtomobilih lahko GPS zagotovi grobo oceno lokacije avtomobila. Vendar je to morda treba dopolniti z drugimi metodami zaradi nenatančnosti GPS, zlasti v mestnih okoljih (Lu, Ma, Smart, & Yu, 2021).

- Mrtvo določanje: ta tehnika ocenjuje trenutni položaj avtomobila na podlagi njegovega prejšnjega položaja, smeri in hitrosti. Pogosto se uporablja v povezavi s podatki GPS (Lu, Ma, Smart, & Yu, 2021).
- Hkratna lokalizacija in kartiranje: čeprav je ta tehnika naprednejša, jo je mogoče poenostaviti za enostavne aplikacije za samovozeče avtomobile. Vključuje izdelavo zemljevida neznanega okolja in hkratno spremljanje lokacije avtomobila v tem okolju, pri čemer se običajno uporabljajo metode LIDAR ali metode, ki temeljijo na vidu (Lu, Ma, Smart, & Yu, 2021).

6.3 Algoritmi za načrtovanje

Načrtovanje vključuje določitev poti, ki naj jo vozilo opravi od trenutnega položaja do cilja (Ming, Y., Li, Y., Zhang, Z., & Yan, W., 2021).

- Načrtovanje poti: pri enostavnih samovozečih avtomobilih lahko načrtovanje poti vključuje preproste algoritme, kot sta A* ali Dijkstrov algoritem za iskanje najkrajše poti na vnaprej začrtanem območju (Ming, Y., Li, Y., Zhang, Z., & Yan, W., 2021).
- Načrtovanje obnašanja: to vključuje odločanje o tem, katera dejanja naj vozilo izvede, na primer upočasnitev, ustavitev ali prehitevanje. Pri osnovnih aplikacijah lahko preproste scenarije obravnavajo sistemi, ki temeljijo na pravilih (npr. stroji končnih stanj) (Ming, Y., Li, Y., Zhang, Z., & Yan, W., 2021).
- Načrtovanje poti: ko je pot določena, načrtovanje trajektorije vključuje ustvarjanje gladke vozne poti, ki se izogne oviram in upošteva cestne predpise. V ta namen se lahko uporabijo polinomske metode (Ming, Y., Li, Y., Zhang, Z., & Yan, W., 2021).

6.4 Nadzorni algoritmi

Algoritmi za nadzor upravljajo gibanje vozila in zagotavljajo, da vozilo nemoteno in varno sledi načrtovani poti (Zulu & John, 2014).

- Proporcionalno-integralno-derivativni (PID – Proportional-Integral-Derivative) krmilniki: ti se običajno uporabljajo za krmiljenje, pospeševanje in zaviranje v enostavnejših aplikacijah za samovozeče voznike. Krmilnik PID nenehno izračunava vrednost napake in prilagaja krmilne vhode, da bi to napako čim bolj zmanjšal, s čimer pomaga vzdrževati želeno pot (Zulu & John, 2014).

- Modelno napovedno krmiljenje (MPC – Model Predictive Control): pri naprednejši metodi gre za napovedovanje prihodnjega obnašanja vozila in ustrezno optimizacijo njegove trajektorije. Čeprav je MPC bolj zapleten, ga je mogoče prilagoditi za preprostejša, omejena okolja (Zulu & John, 2014).

6.5 Algoritmi za izogibanje oviram

Izogibanje oviram zagotavlja, da vozilo ne trči v predmete na svoji poti (Chen, Peng, & Grizzle, 2018).

- Potencialna področja: ta tehnika vključuje ustvarjanje „potencialnega polja“, v katerem ovire vozilo odbijajo, cilj pa ga privlači, kar vozilu omogoča nemoteno navigacijo okoli ovir (Chen, Peng, & Grizzle, 2018).
- Reaktivne metode: za lažja okolja je mogoče uporabiti preproste reaktivne metode, kot je „Ustavi in počakaj“ (ustavi se, ko zazna oviro, in počakaj, dokler se ta ne odstrani) (Chen, Peng, & Grizzle, 2018).

6.6 Varnost in varnostni mehanizmi

Varnost je najpomembnejša pri vsakem sistemu za samovozeče voznike. Tudi preprosti sistemi potrebujejo osnovna varovala pred okvarami (Patel, 2021).

- Zaviranje v sili: osnovno zaznavanje ovir v kombinaciji z enostavnim zavornim sistemom lahko pomaga preprečiti trke (Patel, 2021).
- Redundanca: več senzorjev za isto funkcijo (npr. dve kameri za zaznavanje voznega pasu) poveča zanesljivost (Patel, 2021).

6.7 Algoritmi za simulacijo in testiranje

Pred uvedbo algoritma za samovozečo vožnjo v resničnem svetu se opravi obsežno testiranje v simulacijskih okoljih (Schöner, 2018).

- Simulatorji: programsko opremo, kot so CARLA, Gazebo, Udacity ali celo preprostejši po meri izdelani simulatorji, je mogoče uporabiti za testiranje algoritmov v različnih nadzorovanih okoljih (Schöner, 2018).

6.8 Simulacija samovozečega avtomobila z uporabo globokega učenja

6.8.1 Konvolucijsko nevronsko omrežje (CNN)

Samovozeči avtomobil v tem diplomskem delu je bil opremljen s tehnologijo CNN zaradi njenih dobrih zmogljivosti prepoznavanja slik in vzorcev. Poleg učenja upravljanja avtomobila se je CNN naučil tudi, v kakšni situaciji je treba uporabiti določen kot krmiljenja. CNN je potreboval izjemno dolgo obdobje, da je končal svoje usposabljanje. Nevronska mreža v tej študiji je za vsako epoho potrebovala od tri do štiri minute, da se je usposobila v desetih epohah. 1 h za 30 epoh in 2 h za 50 epoh. Metodologija omrežja, znana kot model usposabljanja Nvidia, je podrobneje pojasnjena v nadaljevanju.

6.8.2 Zbiranje podatkov

Zbrati moramo dovolj podatkov za usposabljanje predlaganega modela. Simulator Udacity ponuja možnost, da v načinu simulacijskega usposabljanja ustvarimo lasten nabor podatkov. Kote krmiljenja posname simulator, slike leve, desne in sredinske strani pa leva, desna oziroma sredinska kamera. S puščicami se uravnava hitrost vozila. Ustvarita se datoteka csv in mapa z vsemi slikami, ki se pozneje uporabita za usposabljanje.

Kot obračanja je pozitivno število med 0 in 1, če se avtomobil obrača v desno, in negativno število med 0 in -1, če se obrača v levo. Ko vozilo vozi naravnost, je kot zavijanja enak nič. Avtomobil ima največjo hitrost 30 in lahko doseže katero koli število med 0 in 30 (simulator nima enote).

6.8.3 Obdelava podatkov

Zbrane podatke, tj. fotografije, pred učenjem modela predhodno obdelamo. Med predobdelavo se slike obrežejo, da se odstranita nebo in sprednji del avtomobila. Slike se nato pretvorijo iz RGB v YUV in pomanjšajo, da ustrezajo vhodni obliki modela. To se izvede, ker RGB ni najboljša preslikava za čutno zaznavanje. Ko gre za kodiranje in zmanjšanje pasovne širine, so barvni prostori YUV bistveno učinkovitejši od RGB.

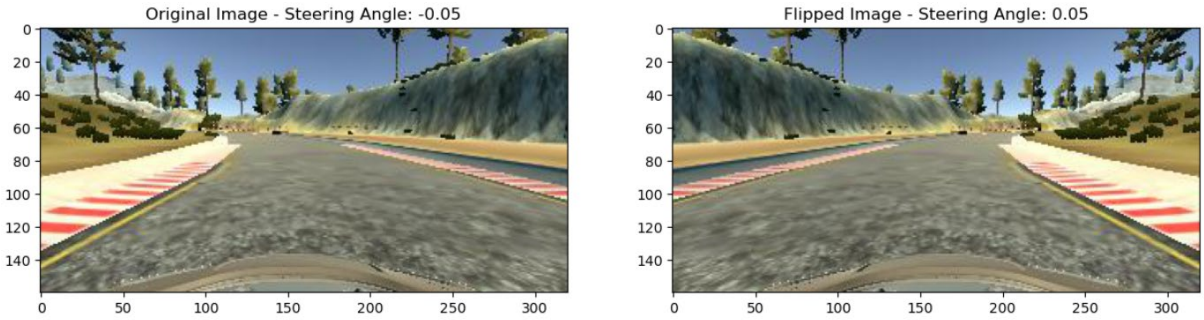
6.8.4 Usposabljanje

Pri usposabljanju omrežja so bili upoštevani številni vidiki. Ti so vključevali strukturo modela, vrsto razširitve, ki se je uporabila v učnem nizu in nepristranski kot obračanja. Za usposabljanje omrežja je bilo uporabljenih 10, 30 in 50 vadbenih epoh, podatki pa so bili naključno razdeljeni na vadbene in validacijske množice. Izguba pri potrjevanju se izračuna na koncu epohe, medtem ko se izguba pri usposabljanju izračuna med epoho. Manjša validacijska izguba bi pomenila izboljšano delovanje vozila, kar bi privedlo do daljših potovalnih razdalj in manjšega števila trkov, če sploh.

Da bi razumeli vpliv različnih razširitev in kako lahko te povzročijo pretirano ali premajhno prilagajanje ali razkrijejo, da je nabor podatkov nereprezentativen, je bil ustvarjen tudi graf, ki prikazuje izgube pri usposabljanju in potrjevanju za vsako omrežje. Preučene so bile razširitve slik z obračanjem, pomikanjem, povečavo, svetlostjo, brez razširitve, naključno razširitvijo ter kombinacijo razširitve z obračanjem in svetlostjo.

6.8.4.1 Obrnjena slika (flipped image)

Pri razširitvi s flipom se slika obrne čez os Y, kot obrata pa spremeni znak iz „+“ v „-“ in obratno. Na sliki 3 je primer, ki prikazuje kot zasuka 0,05154746, ki na spremenjeni sliki postane - 0,05154746.

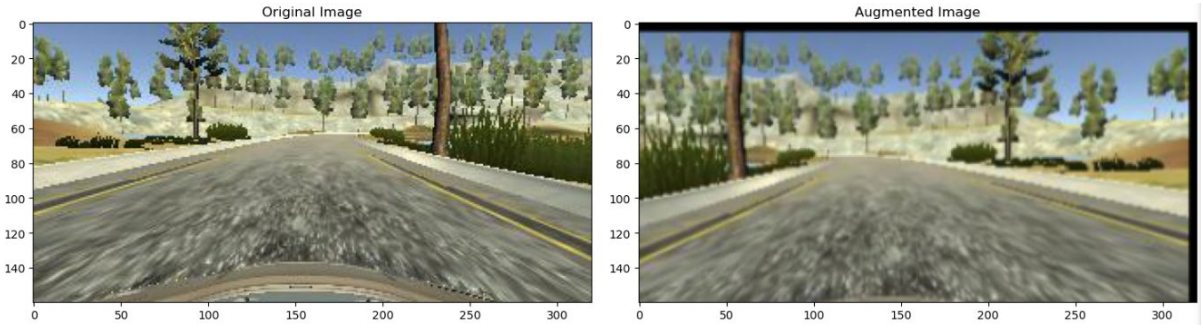


Slika 3: Obrnjena slika (flipped image)

(Vir: Lasten)

6.8.4.2 Augmentirana slika

Povečanje s panoramo je neke vrste fina transformacija. V študiji sta bila kot argumenta za afinno funkcijo uporabljena 10-odstotni translacijski premik v levo in desno, naključno. Na sliki 4 je primer.

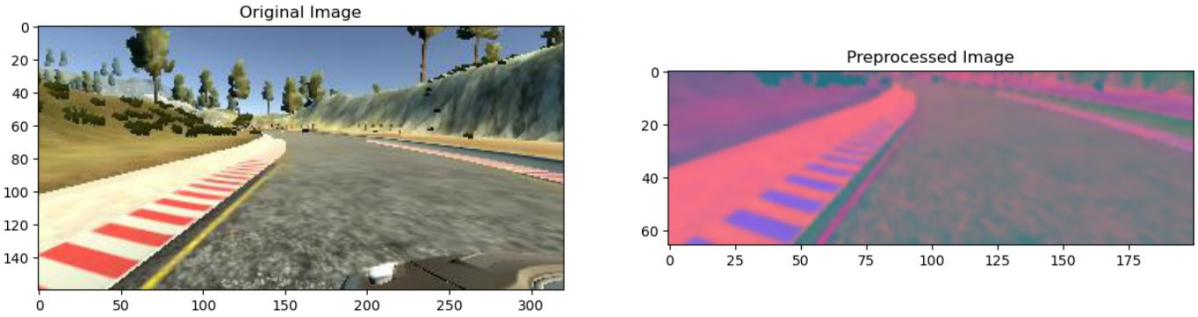


Slika 4: Augmentirana slika

(Vir: Lasten)

6.8.4.3 Preprocesirana slika

Izvirna slika se naloži s poti do datoteke, nato pa se uporabi funkcija predobdelave za spremembo slike, na primer za spremembo velikosti, normalizacijo ali zmanjšanje šuma. Obe sliki se nato prikažeta na eni sliki z dvema podpoglavjema: prvo podpoglavje prikazuje izvirno sliko, drugo podpoglavje pa predobdelano sliko. Ta vizualna primerjava pomaga oceniti učinke korakov predobdelave na neobdelane vhodne slike.

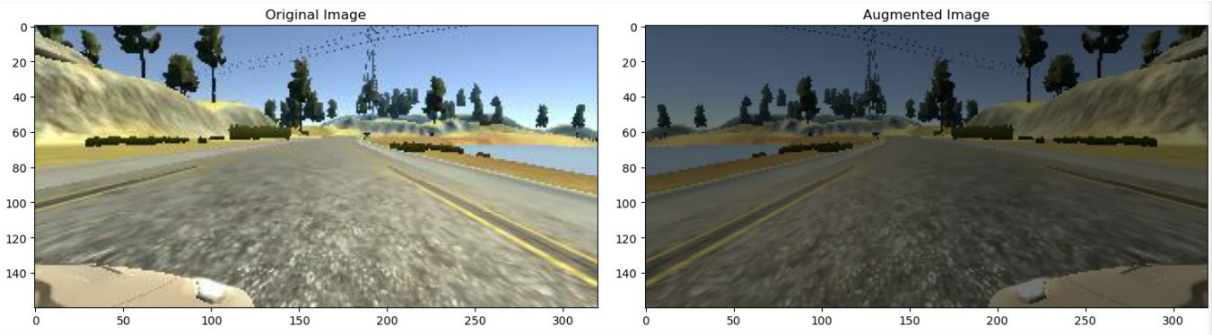


Slika 5: Preprocesirana slika

(Vir: Lasten)

6.8.4.4 Povečanje in/ali zmanjševanje svetlosti

Povečanje svetlosti izvirno sliko naključno osvetli ali zatemni in omrežje ustrezno izpostavi povečani sliki. Na sliki 6 je primer povečanja svetlosti, kjer je spremenjena slika temnejša.



Slika 6: Zmanjšan nivo svetlosti slike

(Vir: Lasten).

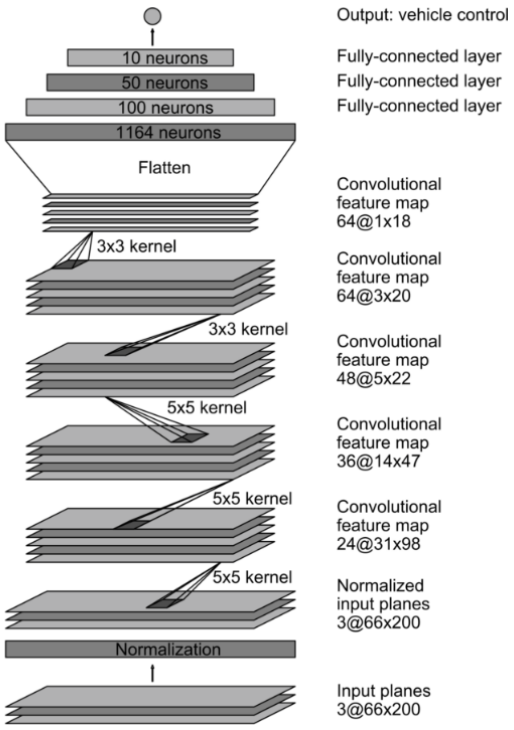
6.8.4.5 Brez povečanja

Originalne fotografije, ki jih je posnel simulator Udacity, so edine, ki so bile uporabljene za učenje algoritma, ki je bil učen brez kakršne koli razširitve. Po razširitvah so fotografije podvržene predobdelavi za zmanjšanje neželenih popačenj in izboljšanje atributov slike. Obrezani so odvečni elementi, kot so pokrov motorja avtomobila in območja slike, ki ne vključujejo ceste. Na sliki 5 je izvirna slika obrezana na osi Y od 135 do 160 in od vrednosti 0 do 60. Poleg obrezovanja je bilo opravljenih še nekaj sprememb za izboljšanje slike.

Slike YUV so bile poslane v omrežje v modelu Nvidia. Zato je bil v tej preiskavi uporabljen tudi sistem barvnega kodiranja YUV. Za glajenje slike in zmanjšanje šuma je bila uporabljena Gaussova zameglitev. Kot je razvidno iz predobdelane slike na sliki 5, je bila velikost slike nato zmanjšana za 66 na 200, da je ustrezala velikosti vhoda v modelu Nvidia.

6.8.5 Model usposabljanja (Model Nvidia)

Pri tem je bil uporabljen model Nvidia, ki je uporaben model za kloniranje vedenja. Arhitektura modela je bila pridobljena iz publikacije „End to End Learning for Self-Driving Car“ (Bojarski, in drugi, 2016).



Slika 7: Arhitektura CNN

(Vir: https://developer.nvidia.com/blog/deep-learning-self-driving-cars/)

Ker so bili podatki že normalizirani, normalizacija pri ustvarjanju modela ni bila potrebna. Prvi sloj konvolucije prejme normalizirane podatke. Slika 7 prikazuje, da ima prva konvolucijska plast velikost jedra 5 x 5 in 24 filtrov. Pri velikosti jedra 5 x 5 ima druga konvolucijska plast 36 filtrov. Tretja plast ima jedro velikosti 3 x 3 in 48 filtrov. Pri 64 plasteh in velikosti jedra 3 x 3 sta četrta in peta plast enaki. Dolžina koraka jedra med premikanjem po sliki se imenuje podvzorec. Za pospešitev izračuna so bili prvi trije sloji izvedeni z dolžino koraka 2 x 2, nato pa je bila uporabljena aktivacijska funkcija „elu“ (eksponentna linearna enota). Izbrana je bila dolžina koraka enega piksla, saj četrta in peta raven ne zahtevata izpuščanja pikslov. Izhod iz prejšnje konvolucijske plasti prejme raven „flatten“, ki ga pretvori v eno samo enodimenzionalno polje. Za izravnalnim slojem so nameščene štiri debele plasti s 100, 50, 10 in 1. Zadnji sloj zagotavlja predvideni kot krmiljenja za samovozeči avtomobil, medtem ko imajo prejšnji trije sloji enako aktivacijsko funkcijo kot „elu“.

Tabela 6: Sekvencijski model po Nvidia standardih

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 31, 98, 24)	1,824
conv2d_1 (Conv2D)	(None, 14, 47, 36)	21,636
conv2d_2 (Conv2D)	(None, 5, 22, 48)	43,248
conv2d_3 (Conv2D)	(None, 1, 18, 64)	76,864
flatten (Flatten)	(None, 1152)	0
dense (Dense)	(None, 100)	115,300
dense_1 (Dense)	(None, 50)	5,050
dense_2 (Dense)	(None, 10)	510
dense_3 (Dense)	(None, 1)	11

Total params: 264,443 (1.01 MB)
Trainable params: 264,443 (1.01 MB)
Non-trainable params: 0 (0.00 B)
None

(Vir: Lasten).

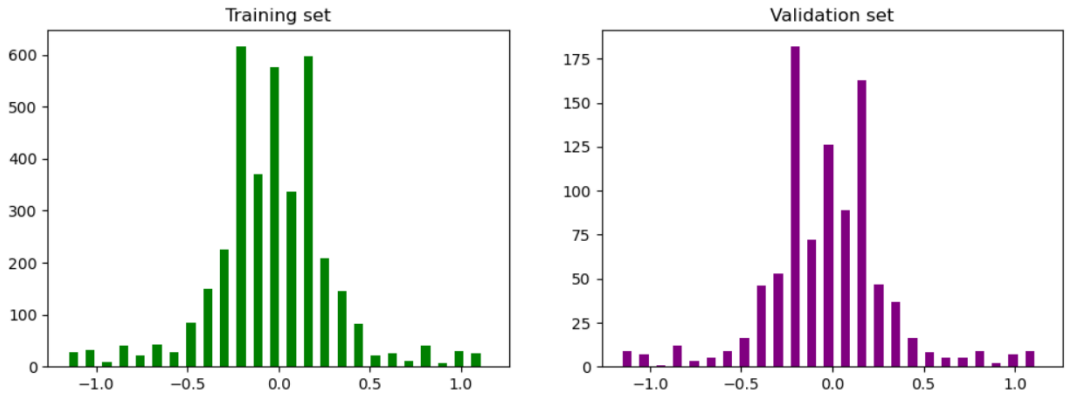
Model je sekvencijski, kar pomeni, da se vsak sloj linearno navezuje na naslednjega. Model je sestavljen iz več konvolucijskih slojev, ki jim sledijo popolnoma povezani (gosti) sloji, kar je značilna arhitektura za CNN, ki se uporabljajo pri nalogah prepoznavanja ali razvrščanja slik.

Sestavljajo ga štiri plasti *Conv2D*, ki postopoma zmanjšujejo prostorske dimenzije vhoda, hkrati pa povečujejo število zemljevidov funkcij. Temu sledi plast *Flatten*, ki pretvori 2D zemljevide značilk v 1D vektor, ki se nato prenese skozi vrsto popolnoma povezanih plasti *Dense*. Ti gosti sloji postopoma zmanjšujejo razsežnost podatkov, kar vodi do enega samega izhodnega nevrona, kar nakazuje, da je model zasnovan za naloge binarne klasifikacije ali regresije.

Povzetek modela vsebuje informacije o izhodni obliki in številu parametrov, ki jih je mogoče trenirati za vsako plast. Skupaj je 264 443 parametri, ki jih je mogoče natrenirati.

6.8.5.1 Usposabljanje in testiranje modelov

Za učenje smo uporabili več kot 3751 slik, za testiranje našega modela pa manj kot polovico slik. Slika 8 prikazuje število kotov krmiljenja v naboru podatkov za usposabljanje.



Slika 8: Število kotov krmiljenja v naboru podatkov za usposabljanje (usposabljanje in validacija)

(Vir: Lasten)

Z uporabo Pythonovega modula matplotlib smo izdelali graf, kot je prikazan na sliki 8. Število krmilnih stopinj v testni zbirki podatkov je prikazano vizualno.

Pri usposabljanju modelov smo uporabili vrednosti 10, 30 in 50 za epohe (epochs), epohe_na_korak (epochs_per_step) in korake_preverjanja (validation_steps). Rezultati učenja so navedeni v poglavju » 6.8.6. Primerjava rezultatov pri različnem številu ciklov samoučenja«.

6.8.5.2 Rezultati

Simulacija in konzola s predhodno predvidenim kotom krmiljenja sta prikazani na sliki 9. Večina kotov krmiljenja, ki jih je napovedal naš model, je bila točna. Vozilo je samostojno končalo progo.

Slika 9:Samovozeči avtomobil v simulatorju Udacity.

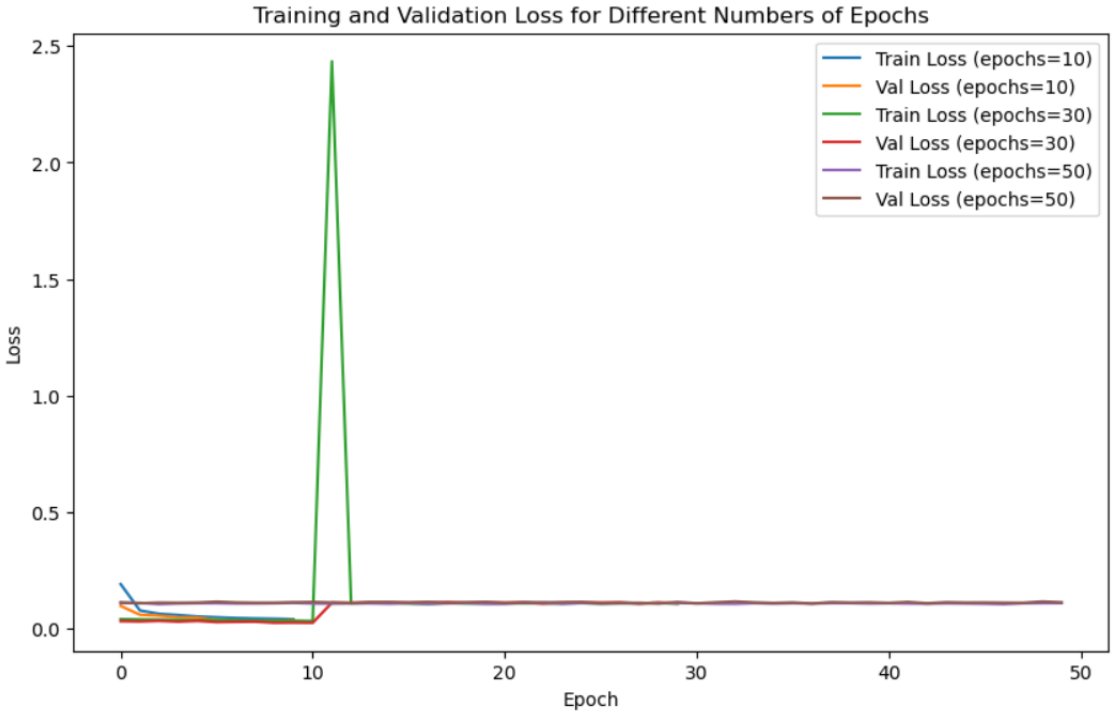
(Vir: Lasten).

6.8.6 Primerjava rezultatov pri različnem številu ciklov samoučenja

Postopek usposabljanja modela lahko izvedemo večkrat z različnim številom epoh usposabljanja, da preverimo hipotezo, ki primerja rezultate za različno število ciklov samoučenja. Za vsako izvedbo bo predstavljeno ločeno število ciklov samoučenja. Ko je teh več izvedb končanih, lahko primerjamo merila uspešnosti (kot sta izguba in natančnost), da preverimo, kako je število epoh učenja vplivalo na splošno uspešnost modela.

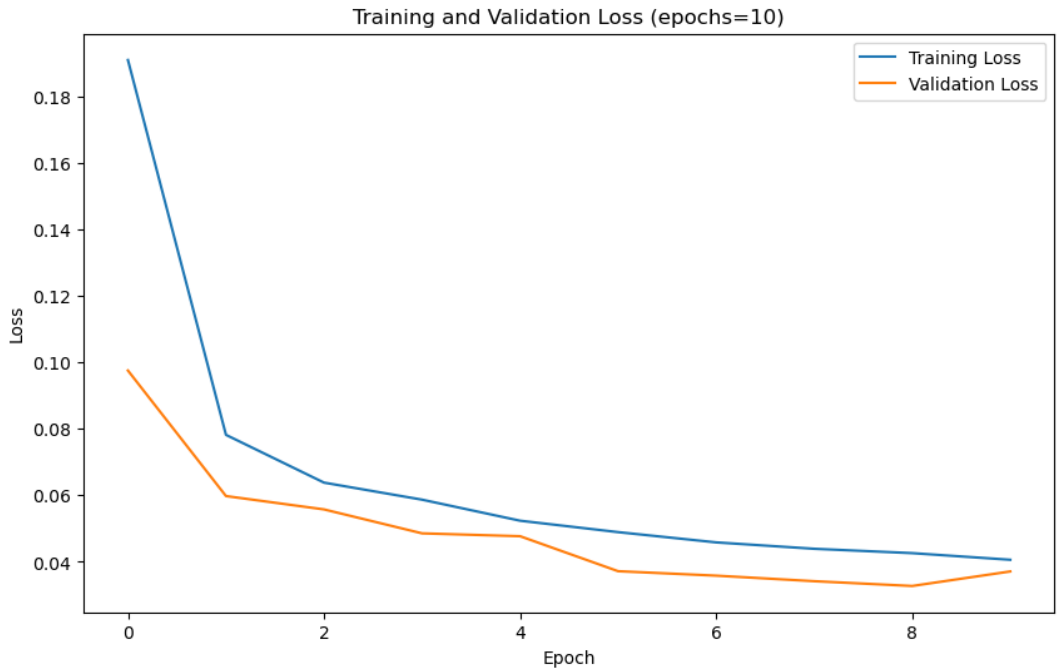
V našem primeru seznam epochs_list določa različna števila epoh, ki se uporabijo za učenje, na primer [10, 30, 50], kar pomeni, da bo model učen trikrat z 10, 30 in 50 epohami. Seznam, imenovan histories, je ustvarjen za shranjevanje objektov zgodovine, ki jih je vrnila metoda model.fit() za vsako izvedbo usposabljanja. Ti objekti zgodovine vsebujejo podrobne informacije o postopku usposabljanja, vključno z izgubo pri usposabljanju in potrjevanju ter natančnostjo za vsako epoho.

V zanki for se iterira nad vsako vrednostjo v seznamu epochs_list in za vsako iteracijo se model usposobi z določenim številom epoh. Po vsakem usposabljanju se objekt zgodovine doda na seznam zgodovine, kar omogoča enostaven dostop in primerjavo rezultatov vseh usposabljanj, da se oceni vpliv različnega števila epoh na uspešnost modela.



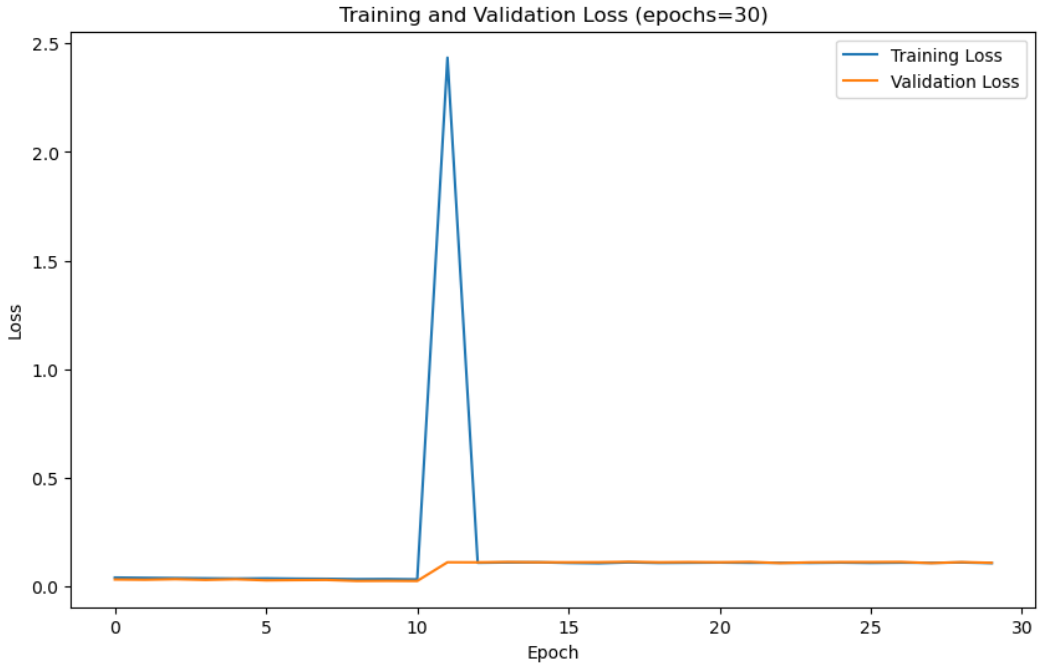
Slika 10: Skupni graf različnih ciklusov učenja

(Vir: Lastni)



Slika 11: Graf učenja epoch 10

(Vir: Lastni)



Slika 12: Graf učenja epoch 30

(Vir: Lastni)

Slika 13: Graf učenja epoch 50

(Vir: Lastni)

Hipoteza tega poskusa je primerjava rezultatov za različno število ciklov samoučenja (epoch), da bi ugotovili, kako število iteracij usposabljanja vpliva na uspešnost in sposobnost posploševanja modela namenjenega uporabi v avtonomnih vozilih.

Analiza rezultatov:

- Epohe = 10:

Model izkazuje stalno zmanjševanje izgub pri usposabljanju in potrjevanju, kar pomeni, da se učinkovito uči iz podatkov. Obstaja dobro ravnovesje med zmanjšanjem izgube pri usposabljanju in ohranjanjem nizke izgube pri potrjevanju.

To nakazuje, da z 10 epohami model ni pretirano opremljen in je sposoben dobro posplošiti na neznane podatke. Hipoteza, da lahko določeno število ciklov usposabljanja zagotovi optimalno delovanje, je tu podprta, saj model kaže dobro delovanje z minimalnim usposabljanjem.

- Epohe = 30:

Model se sprva še naprej izboljšuje, kot kažejo padajoče vrednosti izgube, vendar pa se okoli 12. epohe pojavi močan skok v izgubi pri usposabljanju, ki povzroči nestabilnost. Po tem skoku se izgube pri učenju in potrjevanju ponovno stabilizirajo, vendar proces učenja kaže znake morebitne nestabilnosti ali občutljivosti na nekatere vidike podatkov za učenje.

Ta rezultat kaže, da lahko razširitev števila ciklov samoučenja na 30 v proces učenja vnese nepričakovano vedenje ali anomalije. Izbruh lahko pomeni, da je model naletel na težave z določenimi podatki ali hiperparametri, kar je začasno vplivalo na izgubo.

Vendar model sčasoma ponovno pridobi stabilnost, kar pomeni, da lahko podaljšanje števila epoh sicer poveča možnosti učenja, vendar lahko privede tudi do nestabilnosti. To delno podpira hipotezo, saj kaže, da ima lahko več epoh različne učinke na učenje modela.

- Epohe = 50:

Pri 50 epohah izgube pri učenju in potrjevanju močno nihajo, kar pomeni, da model težko vzdržuje stalno učinkovitost. Povečano nihanje, zlasti pri izgubi potrjevanja, kaže, da se model verjetno preveč prilagaja učnim podatkom, saj jih večkrat vidi v številnih epohah.

Ta nihanja izgub kažejo na zmanjševanje donosnosti povečanja števila ciklov samoučenja po določeni točki. Model ne kaže stalnega izboljšanja, temveč postane bolj nestabilen in deluje nedosledno na podatkih za preverjanje.

Ta rezultat kaže, da lahko preveliko število epoh škoduje sposobnosti posploševanja modela, kar potrjuje hipotezo, da obstaja optimalno število ciklov samoučenja, nad katerim se uspešnost ne izboljša in se lahko celo poslabša.

- Zaključek v povezavi s hipotezo:

Hipoteza je bila primerjati uspešnost pri različnem številu ciklov samoučenja in določiti optimalno število za usposabljanje modela. Rezultati podpirajo hipotezo, saj kažejo, da:

- spodnje epohe (10): vodijo do stabilnega in učinkovitega učenja, kar nakazuje, da bi manjše število epoh lahko zadostovalo za optimalno delovanje;
- srednje število epoh (30): povzročijo nekaj nestabilnosti, vendar lahko še vedno privedejo do učinkovitega učenja. To pomeni, da lahko modelu koristi zmerno povečanje števila epoh, vendar obstaja tveganje nestabilnosti;
- večje število epoh (50): povzročijo znatno pretirano prilagajanje in nestabilno učenje, kar potrjuje, da preveliko število epoh vodi v zmanjševanje donosnosti in lahko poslabša učinkovitost modela na nevidnih podatkih.

7 RAZPRAVA

Raziskava, predstavljena v tem delu, se osredotoča na primerjavo dveh priljubljenih simulacijskih platform - CARLA in Udacity - za usposabljanje modelov UI v aplikacijah AV, zlasti za začetnike. Raziskava ocenjuje tudi učinkovitost dveh razvojnih okolij, Jupyter Notebook in Google Colab, za razvoj in testiranje algoritmov AV, hkrati pa optimizira algoritme samoučenja za aplikacije AV. V poglavju o razpravi bodo obravnavane posledice teh ugotovitev, izzivi, na katere smo naleteli in morebitna področja za prihodnje raziskovanje.

7.1 Vpliv ugotovitev

O prednostih in slabostih simulatorjev CARLA in Udacity za usposabljanje modelov umetne inteligence v avtonomnih vozilih je mogoče pridobiti veliko informacij. Simulator CARLA kot zelo bogat s funkcijami in realističen simulator zagotavlja stabilno okolje za oblikovanje in preskušanje modelov umetne inteligence v različnih voznih scenarijih. Za začetnike pa bi lahko bila njegova zapletenost in večje zahteve glede obdelave podatkov težavna. Simulator Udacity pa, kljub temu da je manj realističen, ponuja bolj dostopno okolje z manj vstopnimi ovirami, zaradi česar je primeren za začetnike, ki se šele spoznavajo z izdelavo AV.

Ugotovitve kažejo, da bi bil za izobraževalne namene ali začetne faze razvoja modela zaradi svoje preprostosti in enostavne uporabe koristnejši simulator Udacity. Nasprotno pa je lahko simulator CARLA primernejši za napredne uporabnike ali za faze, v katerih sta podrobno testiranje in realizem ključnega pomena.

Kontrast med Google Colabom in beležnico Jupyter v smislu razvojnih okolij pokaže na kompromise med dostopnostjo in vsestranskostjo. Ker je Jupyter Notebook lokalno okolje, omogoča uporabnikom večje možnosti prilagajanja, vendar zahteva lokalno procesorsko moč. Google Colab pa ponuja brezplačen dostop do zmogljivih grafičnih procesorjev v oblaku, ki lahko močno izboljšajo razvojni proces – zlasti pri računsko zahtevnih dejavnostih, kot je usposabljanje modelov globokega učenja.

V delu je bilo ugotovljeno, da bi lahko začetnikom bolj koristila uporaba storitve Google Colab zaradi njene dostopnosti in razpoložljivosti računalniških virov, ki zmanjšujejo potrebo po lokalnih visokozmogljivih računalniških napravah. Ko pa razvijalci pridobijo več izkušenj, lahko postaneta prilagodljivost in nadzor, ki ju zagotavlja beležnica Jupyter, bolj dragocena, zlasti v strokovnih ali raziskovalnih okoljih.

7.2 Izzivi in omejitve

Študija je vključevala številne težave, ki jih je treba upoštevati pri prihodnjih raziskavah. Ena glavnih pomanjkljivosti je bila razpoložljivost in obseg naborov podatkov uporabljenih za usposabljanje in ocenjevanje modelov umetne inteligence. Zanesljivi modeli AV in natančna simulacija voznih okoliščin v resničnem svetu so odvisni od realističnih in obsežnih zbirk podatkov. Na posplošljivost ugotovitev bi lahko vplivala odvisnost študije od majhnih naborov podatkov.

Dodatno težavo predstavljajo računalniški viri, ki so potrebni za izvajanje modelov globokega učenja in zahtevnih simulacij, zlasti v okolju CARLA. Raziskovalci in razvijalci, ki nimajo dostopa do visokozmogljivih računalniških virov, lahko pri svojem delu naletijo na omejitve zaradi vse večjih potreb po grafičnih procesorjih in pomnilniških virih.

Poleg tega so časovne omejitve omejile število poskusov in iteracij, ki jih je bilo mogoče izvesti med raziskavo. Ta omejitev je morda vplivala na zanesljivost rezultatov optimizacije, ki se nanašajo na algoritme samoučenja.

7.3 Priporočila za prihodnje raziskave

Za temeljitejšo primerjavo bi bilo treba v prihodnjih študijah razmisliti o raziskavi širšega nabora razvojnih okolij in simulacijskih platform. Poleg tega so potrebna bolj realistična, a hkrati dostopna simulacijska okolja, ki lahko povežejo vrzel med obsežnimi zmogljivostmi za izkušene razvijalce in uporabnostjo za začetnike.

Prihodnje raziskave se lahko osredotočijo tudi na oblikovanje in izboljšanje algoritmov za samoučenje, ki so izdelani posebej za aplikacije AV. Pričujoče delo predlaga idealno število iteracij za cikle samoučenja; vendar pa bi lahko z več raziskavami raziskali prilagodljive algoritme, ki dinamično spreminjajo parametre učenja kot odziv na povratne informacije in delovanje v realnem času.

Razširitev obsega zbirk podatkov na bolj raznolike scenarije vožnje, vremenske razmere in geografske lokacije bi povečala zanesljivost modelov umetne inteligence, razvitih s temi simulacijskimi platformami. To ne bi izboljšalo le natančnosti in zanesljivosti modelov, temveč bi pomagalo tudi pri razvoju sistemov AV, ki so bolj prilagodljivi različnim okoljem v resničnem svetu.

8 SKLEP

V tem diplomskem delu smo se osredotočili na oceno, kako dobro simulacijski platformi - CARLA in Udacity - usposabljata modele umetne inteligence za aplikacije AV, zlasti za uporabnike začetnike. Namen te študije je bil tudi oceniti uporabnost dveh razvojnih platform za testiranje in razvoj algoritmov AV: Jupyter Notebook in Google Colab. Drugi cilj študije je bil poiskati najboljše konfiguracije za algoritme samoučenja v simulacijah AV. Na podlagi temeljitega pregleda in testiranja ugotavljamo, da so bili cilji, ki smo si jih zastavili na začetku, v bistvu doseženi.

Začetni cilj je bil najti boljšo simulacijsko platformo med CARLA in Udacity za usposabljanje začetnih modelov umetne inteligence. Glede na to analizo ima vsaka platforma edinstvene prednosti in slabosti. Platforma CARLA je zaradi svoje velike realističnosti in širokih zmogljivosti primernejša za zapletene simulacije in zahtevno testiranje modelov. Vendar zaradi svoje zapletenosti in visokih zahtev glede obdelave morda ni najboljša možnost za začetnike. Po drugi strani pa simulator Udacity, čeprav manj realističen, ponuja bolj dostopen in intuitiven vmesnik, zaradi česar je boljša možnost za začetnike, ki se učijo osnov AV ustvarjanja. Tako je bil dosežen cilj primerjave teh platform in ugotavljanja, ali so primerne za začetnike ali ne.

Drugi cilj je bil ugotoviti, katero okolje - beležnica Jupyter ali Google Colab - bolje omogoča ustvarjanje in testiranje algoritmov AV. Študija je pokazala, da je Google Colab zaradi svoje zasnove v oblaku in lahko dostopnih računalniških virov boljša možnost za začetnike, zlasti tiste, ki nimajo dostopa do visoko zmogljive lokalne strojne opreme. Čeprav beležnica Jupyter Notebook ponuja več svobode in nadzora, bi bila za zahtevnejše uporabnike, ki potrebujejo posebne nastavitve, lahko primernejša. Tako je bil z ugotavljanjem prednosti in slabosti posameznih nastavitev dosežen tudi ta cilj.

Tretji cilj je bila optimizacija algoritmov samoučenja za aplikacije AV – natančneje ugotoviti, koliko iteracij učnih ciklov je idealnih. Raziskava je potrdila, da povečanje števila iteracij poveča učinkovitost učnih algoritmov do določenega praga, po katerem omejitve virov povzročijo zmanjšanje donosnosti. Ta rezultat prispeva k cilju optimizacije ciklov samoučenja z uravnoteženjem uporabe računalniških virov z učinkovitostjo učenja.

Raziskava je temeljila na štirih osnovnih hipotezah:

H1: Udacity Simulator je bolj prijazen za začetnike pri treniranju modelov umetne inteligence za AV kot CARLA pri enostavnih samostojnih vožnjah.

Ta hipoteza je bila potrjena. Simulator Udacity se je zaradi preprostejšega vmesnika in manjših računalniških zahtev izkazal za uporabniku prijaznejšega in dostopnejšega za začetnike. Model CARLA je sicer zmogljivejši, vendar je bolj primeren za napredne uporabnike.

H2: Google Colab ima večjo podporo in je lažji za začetnike kot Jupyter Notebook.

Tudi ta hipoteza je bila potrjena. Google Colab je zaradi svoje postavitve v oblaku in razpoložljivosti zmogljivih računalniških virov bolj praktična izbira za začetnike, zlasti za tiste, ki nimajo lokalnih računalniških virov.

H3: Povečanje števila ponovitev v ciklu samoučenja algoritma v simulaciji AV pri enostavnih samostojnih vožnjah vodi k izboljšanju učinkovitosti algoritma.

Ta hipoteza je bila delno potrjena. V študiji je bilo ugotovljeno, da se s povečevanjem števila iteracij sicer sprva izboljša učinkovitost, vendar obstaja optimalna točka, po kateri dodatne iteracije ne povečajo bistveno učinkovitosti in lahko namesto tega povzročijo neučinkovitost zaradi omejenih virov.

H4: Jupyter Notebook je učinkovitejši za razvoj in testiranje algoritmov umetne inteligence za AV pri enostavnih samostojnih vožnjah v primerjavi z Google Colab.

Ta hipoteza je bila zavrnjena. Ugotovitve kažejo, da ima Google Colab za začetnike več prednosti zaradi infrastrukture v oblaku in enostavne uporabe, zato je primernejši za začetne faze razvoja.

Raziskava je privedla do več ključnih ugotovitev, ki so dragocene tako za začetnike kot za izkušene strokovnjake na področju razvoja AV:

Primernost platforme: V tem primeru je platforma Udacity zaradi svoje preprostosti primernejša za začetnike, medtem ko je CARLA primernejša za napredne uporabnike, ki zahtevajo visoko stopnjo realizma in podrobne simulacije.

Razvojna okolja: Google Colab je ugodnejši za začetnike, saj ponuja preprosto uporabo in zmogljive vire v oblaku, medtem ko beležnica Jupyter zagotavlja več nadzora in prilagodljivosti, kar lahko koristi izkušenim uporabnikom.

Optimizacija učnih algoritmov: Obstaja optimalno število iteracij za algoritme za samostojno učenje, ki uravnoteži računsko učinkovitost in uporabo virov. Nad to optimalno točko se povečanje učinkovitosti zmanjša, kar poudarja pomen upravljanja virov v simulacijah AV.

Na podlagi ugotovitev raziskave so predlagani naslednji predlogi:

Za začetnike: Priporočljivo je začeti s simulatorjem Udacity in okoljem Google Colab. Ti orodji zagotavljata manj zapleteno in bolj dostopno vstopno točko v razvoj AV, kar začetnikom omogoča, da se osredotočijo na temeljno učenje, ne da bi jih preobremenili s tehničnimi zapleti.

Za napredne uporabnike: Napredni uporabniki naj razmislijo o prehodu na simulator CARLA in beležnico Jupyter, ko se njihove spretnosti in potrebe razvijajo. Realistično okolje v simulatorju CARLA in prilagodljiva nastavitev Jupytra ponujata robustnejšo platformo za testiranje zapletenih scenarijev in razvoj naprednih algoritmov.

Optimizacija algoritmov: Nadaljnje raziskave bi morale raziskati prilagodljive algoritme samoučenja, ki dinamično prilagajajo svoje parametre glede na uspešnost v realnem času, kar zagotavlja optimalno uporabo računalniških virov in povečuje učinkovitost učenja.

Razširitev podatkov: Prihodnje študije bi morale vključevati širši nabor podatkovnih nizov, ki odražajo različne vozne pogoje in okolja. To bo povečalo robustnost in posplošljivost modelov umetne inteligence usposobljenih s temi simulatorji.

Cilji tega diplomskega dela so bili učinkovito doseženi, večina postavljenih teorij pa je bila preverjena. Rezultati zagotavljajo pomembne nove informacije o tem, kako dobro delujejo razvojna okolja in simulacijske platforme za aplikacije AV, zlasti za začetnike. Ta raziskava omogoča bolj premišljeno odločanje na tem področju, saj zagotavlja uporabne nasvete, ki pomagajo tako neizkušenim kot izkušenim razvijalcem pri premagovanju izzivov, povezanih z razvojem AV tehnologije. Prihodnje študije bi morale temeljiti na teh odkritjih, da bi izboljšale instrumente in tehnike, ki so na voljo za ustvarjanje avtonomnih vozil, kar bi zagotovilo nenehne inovacije in napredek v tem hitro razvijajočem se sektorju.

9 LITERATURA

Badue, C., Guidolini, R., Carneiro, R. V., Azevedo, P., Cardoso, V. B., Forechi, A., . . . F. De Souza, A. (2021). Self-driving cars: A survey.

Bojarski, M., Testa, D. D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., . . . Zieba, K. (2016). End to End Learning for Self-Driving Cars. *arXiv*.

Bose, B. K. (2017). Artificial Intelligence Techniques in Smart Grid and Renewable Energy Systems—Some Example Applications. *Proceedings of the IEEE*, 2262-2273.

Bostrom, N., & Yudkowsky, E. (2014). The ethics of artificial intelligence. V K. Frankish, & W. M. Ramsey, *The Cambridge Handbook of Artificial Intelligence* (str. 316-344). Cambridge: Cambridge.

Bučar, F. (2012). *Rojstvo države.* Ljubljana: Didakta.

Capparuccia, R., Renato, D., & Marchitto, E. (2007). Integrating support vector machines and neural networks. *Neural Networks*, 590-597.

CARNEIRO, T., DA NÓBREGA, R. V., NEPOMUCENO, T., BIAN, G.-B., DE ALBUQUERQUE, V. H., & FILHO, P. P. (2018). Performance Analysis of Google Colaboratory as a Tool for Accelerating Deep Learning Applications. *IEEE Xplore*, 61677-61685.

Chai, T. Y., & Nizam, I. (2021). IMPACT OF ARTIFICIAL INTELLIGENCE IN AUTOMOTIVE INDUSTRIES TRANSFORMATION. *International Journal of Information System and Engineering*.

Chen, X. W., & Lin, X. (2014). Big Data Deep Learning: Challenges and Perspectives. *IEEE Access 2*, 514-525.

Chen, Y., Peng, H., & Grizzle, J. (2018). Obstacle Avoidance for Low-Speed Autonomous Vehicles With Barrier Function. *IEEE Transactions on Control Systems Technology*, 194-206.

Došilović, F. K., Brčić, M., & Hlupić, N. (2018). Explainable artificial intelligence: A survey. *IEEE*, 210-215.

Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., & Koltun, V. (2017). CARLA: An Open Urban Driving Simulator. *Proceedings of Machine Learning*.

Epic Games. (brez datuma). Pridobljeno iz Unreal Engine 4: https://www.unrealengine.com.

Erhan, D., Bengio, Y., Courville, A., Manzagol, P. A., Vincent, P., & Bengio, S. (2010). Why Does Unsupervised Pre-training Help Deep Learning? *Journal of Machine Learning Research*, 625-660.

Erokhin, S. D. (2019). A review of scientific research on artificial intelligence. *Systems of Signals Generating and Processing in the Field of on Board Communications*, 1-4.

Esteva, A., Robicquet, A., Ramsundar, B., Kuleshov, V., DePristo, M., Chou, K., . . . Dean , J. (2019). A guide to deep learning in healthcare. *Nature Medicine*, 24–29.

Farivar, F., Haghighi, M. S., Jolfaei, A., & Alazab, M. (2019). Artificial Intelligence for Detection, Estimation, and Compensation of Malicious Attacks in Nonlinear Cyber-Physical Systems and Industrial IoT. *IEEE transactions on industrial informatics*, 2716-2725.

Figueiredo, M. C., Rossett, R. J., Braga, R., & Reis, L. P. (2009). An Approach to Simulate Autonomous Vehicles in Urban Traffic Scenarios. *IEEE Xplore*, 1-6.

Gupta, M., Upadhyay , V., Kumar, P., & Al-Turjman, F. (2021). Deep Learning Implementation of Autonomous. *Research Square*.

Haenlein , M., & Kaplan, A. (2019). A Brief History of Artificial Intelligence: On the Past, Present, and Future of Artificial Intelligence. *California Management Review*, 5-14.

Hao, X., Zhang, G., & Ma, S. (2016). Deep Learning. *International Journal of Semantic Computing*, 417-439.

Hofmann, M., Neukart, F., & Bäck, T. (2017). Artificial Intelligence and Data Science in the Automotive Industry. *arXiv*.

Hsieh, W. (2017). *First Order Driving Simulator.* Berkeley: University of California.

Hu, Q., Lu, Y., Pan, Z., Gong, Y., & Yang, Z. (2021). Can AI artifacts influence human cognition? The effects of artificial autonomy in intelligent personal assistants. *International Journal of Information Management*, 56.

Huang, B., Huan, Y., Xu, L. D., Zheng, L., & Zou, Z. (2019). Automated trading systems statistical and machine learning methods and hardware implementation: a survey. *Enterprise Information Systems*, 132-144.

Hursch, W., & Lopes, C. (1995). *Separation of concerns.* Boston, Massachusetts: Northeastern University.

Kaur, P., Taghavi, S., Tian, Z., & Shi, W. (2021). A Survey on Simulators for Testing Self-Driving Cars. *IEEE, 2021 Fourth International Conference on Connected and Autonomous Driving (MetroCAD)*, 62-70.

Khalaf, B. A., Mostafa, S. A., Mustapha, A., Mohammed, M. A., & Abduallah, W. M. (2019). Comprehensive Review of Artificial Intelligence and Statistical Approaches in Distributed Denial of Service Attack and Defense Methods. *IEEE Access*, 51691-51713.

Lecun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, 436-444.

Li, H. X., & Xu, L. D. (2001). Feature space theory — a mathematical foundation for data mining. *Knowledge-Based Systems*, 253-257.

Li, J., Cheng, H., Guo, H., & Qiu, S. (2018). Survey on Artificial Intelligence for Vehicles. *Automotive Innovation*, 2-14.

Li, Y., Yuan, W., Zhang, S., Yan, W., Shen, Q., Wang, C., & Yang, M. (2024). Choose Your Simulator Wisely: A Review on Open-source Simulators for Autonomous Driving. *IEEE Transactions on Intelligent Vehicles*, 4861 - 4876.

Lu, F., Yamamoto, K., Nomura, L. H., Mizuno, S., Lee, Y., & Thawonmas, R. (2013). Fighting game artificial intelligence competition platform. *IEEE 2nd Global Conference on Consumer Electronics (GCCE)*, 320–323.

Lu, Y., Ma, H., Smart, E., & Yu, H. (2021). Real-Time Performance-Focused Localization Techniques for Autonomous Vehicle: A Review. *IEEE Transactions on Intelligent Transportation Systems*, 6082-6100.

Mahesh, B. (2018). Machine Learning Algorithms - A Review. *International Journal of Science and Research (IJSR)*, 381-386.

Makkar, A., Garg, S., Kumar, N., Hossain, S. M., Ghoneim, A., & Alrashoud, M. (2020). An Efficient Spam Detection Technique for IoT Devices Using Machine Learning. *IEEE Transactions on Industrial Informatics*, 903-912.

Malik, S., Khan, M. A., & El-Sayed, H. (2021). CARLA: Car Learning to Act — An Inside Out. *International Workshop on Smart Communication and Autonomous Driving*, 742-749.

Menke, J., Homberg, S., & Koch, O. (2023). Introduction to artificial intelligence and deep learning using interactive electronic programming notebooks. *ARCH Pharm*, 1-10.

Ming, Y., Li, Y., Zhang, Z., & Yan, W. (2021). A Survey of Path Planning Algorithms for Autonomous Vehicles. *SAE Int. J. Commer. Veh.*, 97-109.

Misra, N. N., Dixit, Y., Al-Mallahi, A., Bhullar, M. S., Upadhyay, R., & Martynenko, A. (2020). IoT, Big Data, and Artificial Intelligence in Agriculture and Food Industry. *IEEE Internet of Things Journal*, 6305-6324.

Myers, G., Badgett, T., & Sandler, C. (2011). *The Art of Software Testing.* Hoboken: Wiley Online Library.

Nasteski, V. (2017). An overview of the supervised machine learning methods. *Horizons*, 51-62.

Neglectos. (02. 04 2018). *A Preliminary Analysis on the Use of Python Notebooks*. Pridobljeno iz Bitergia: https://bitergia.com/blog/opensource/a-preliminary-analysis-on-the-use-of-python-notebooks/

Nelson, M. J., & Hoover, A. K. (2020). Notes on Using Google Colaboratory in AI Education. *Association for Computing Machinery*, 533–534.

Parente, P. (2020). *parente/nbestimate*. Pridobljeno iz GitHub: https://github.com/parente/nbestimate

Patel, A. H. (2021). Exploiting Adaptation Behavior of an Autonomous Vehicle to Achieve Fail-Safe Reconfiguration. V K. D. Berns, *Commercial Vehicle Technology.* Wiesbaden: Springer Vieweg.

Perez, F., & Granger, B. E. (2007). IPython: A System for Interactive Scientific Computing. *Computing in Science & Engineering*, 21-29.

Perkel, J. M. (30. 10 2018). *Nature*. Pridobljeno iz Why Jupyter is data scientists' computational notebook of choice: https://www.nature.com/articles/d41586-018-07196-1

Qela, B., & Mouftah, H. T. (2012). Observe, Learn, and Adapt (OLA)—An Algorithm for Energy Management in Smart Homes Using Wireless Sensors and Artificial Intelligence. *IEEE Trans. Smart Grid*, 2262–2272.

Randles, B. M., Pasquetto, I. V., Golshan, M. S., & Borgman, C. L. (2017). Using the Jupyter Notebook as a Tool for Open Science: An Empirical Study. *2017 ACM/IEEE Joint Conference on Digital Libraries (JCDL)*, 1-2.

Rao, Q., & Frtunikj, J. (2018). Deep learning for self-driving cars: chances and challenges. *Association for Computing Machinery*, 35-38.

Ravì, D., Wong, C., Deligianni, F., Berthelot, M., Perez, J. A., Lo, B., & Yang, G. Z. (2016). Deep Learning for Health Informatics. *IEEE journal of biomedical and health informatics*, 4-21.

Rosique, F., Navarro, P. J., Fernández, C., & Padilla, A. (2019). A Systematic Review of Perception System and Simulators for Autonomous Vehicles Research. *Sensors*.

Schöner, H. (2018). Simulation in development and testing of autonomous vehicles. V M. R. Bargende, *Internationales Stuttgarter Symposium.* Wiesbaden: Springer Vieweg.

Shen, H. (2014). Interactive notebooks: Sharing the code. *Nature* , 515(7525):151-2.

Shi, Z., Huang, Y., He, Q., Xu , L., Liu, S., Qin, L., . . . Zhao, L. (2007). MSMiner—a developing platform for OLAP. *Decision Support Systems*, 2016-2028.

Sun, Z., Bebis, G., & Miller, R. (2006). Monocular Precrash Vehicle Detection: Features and Classifiers. *IEEE TRANSACTIONS ON IMAGE PROCESSING*, 2019-2034.

Swamy, A. K., & Sarojamma, B. (2020). Bank transaction data modeling by optimized hybrid machine learning merged with ARIMA. *Journal of Management Analytics*, 624-648.

Taieb, D. (2018). *Thoughtful Data Science: A Programmer's Toolset for Data Analysis and Artificial Intelligence with Python, Jupyter Notebook, and PixieDust.* Packt Publishing.

Tan, L. T., Hu, R. Q., & Hanzo, L. (2019). Twin-Timescale Artificial Intelligence Aided Mobility-Aware Edge Caching and Computing in Vehicular Networks. *IEEE Transactions on Vehicular Technology*, 3086-3099.

Wang, J., Ma, Y., Zhang, L., Gao, R. X., & Wu, D. (2018). Deep learning for smart manufacturing: Methods and applications. *Journal of Manufacturing Systems*, 144-156.

Wilson, G., Aruliah, D., Brown, C., Hong, N., Davis, M., Guy, R., . . . Wilson, P. (2014). Best Practices for Scientific Computing. *PLOS Biology*, 1-7.

Wu, H., Han, H., Wang, X., & Sun, S. (2020). Research on Artificial Intelligence Enhancing Internet of Things Security: A Survey. *IEEE Access*, 153826-153848.

Xie, Y. (10. 09 2018). *The First Notebook War*. Pridobljeno iz Yihui.org: https://yihui.org/en/2018/09/notebook-war/

Yogesh, K. D., Hughes, L., Ismagilova, E., Aarts, G., Coombs, C., Crick, T., . . . Lal, B. (2021). Artificial Intelligence (AI): Multidisciplinary perspectives on emerging challenges, opportunities, and agenda for research, practice and policy. *International Journal of Information Management*, 55.

Zeng, L., Li , L., & Duan, L. (2012). Business intelligence in enterprise computing environment. *Information Technology and Management*, 297-310.

Zhang, C. (2020). Research on the Economical Influence of the Difference of Regional Logistics Developing Level in China. *Journal of Industrial Integration and Management*, 205-223.

Zhang, C., Xu, X., & Chen , H. (2020). Theoretical foundations and applications of cyber-physical systems: a literature review. *Library Hi Tech*, 95-104.

Zhang, C., & Fu, W. (2021). Optimal Model for Patrols of UAVs in Power Grid under Time Constraints. *International Journal of Performability Engineering*, 103-113.

Zhao, R., Yan, R., Chen, Z., Mao, K., Wang, P., & Gao, R. X. (2019). Deep learning and its applications to machine health monitoring. *Mechanical Systems and Signal Processing*, 213-237.

Zulu, A., & John, S. (2014). A Review of Control Algorithms for Autonomous Quadrotors. *Open Journal of Applied Sciences*, 547-556.

10 PRILOGE

Vse potrebne kode, ki sem jih naredila za to diplomsko delo in analizo, so vključene in oddane na USB ključku.