Usporedba rada algoritma za autonomnu vožnju zasnovanog na procjeni kuta zakreta upravljača vozila u različitim simulatorima za razvoj algoritama autonomne vožnje

Dumančić, David

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij

USPOREDBA RADA ALGORITMA ZA AUTONOMNU VOŽNJU ZASNOVANOG NA PROCJENI KUTA ZAKRETA UPRAVLJAČA VOZILA U RAZLIČITIM SIMULATORIMA ZA RAZVOJ ALGORITAMA AUTONOMNE VOŽNJE

Diplomski rad

David Dumančić

Osijek, 2022.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 18.09.2022.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime Pristupnika:	David Dumančić
Studij, smjer:	Diplomski sveučilišni studij Automobilsko računarstvo i komunikacije
Mat. br. Pristupnika, godina upisa:	D-46ARK, 11.10.2020.
OIB studenta:	49516408454
Mentor:	Izv. prof. dr. sc. Mario Vranješ
Sumentor:	,
Sumentor iz tvrtke:	David Mijić
Predsjednik Povjerenstva:	Prof. dr. sc. Marijan Herceg
Član Povjerenstva 1:	Izv. prof. dr. sc. Mario Vranješ
Član Povjerenstva 2:	Izv.prof.dr.sc. Ratko Grbić
Naslov diplomskog rada:	Usporedba rada algoritma za autonomnu vožnju zasnovanog na procjeni kuta zakreta upravljača vozila u različitim simulatorima za razvoj algoritama autonomne vožnje
Znanstvena grana diplomskog rada:	**Umjetna inteligencija (zn. polje računarstvo)**
Zadatak diplomskog rada:	U ovom radu, potrebno je pronaći jedan od suvremenih algoritama autonomne vožnje zasnovan na procjeni kuta zakreta upravljača vozila (engl. steering angle prediction) i testirati ga u dva različita simulatora za razvoj algoritama autonomne vožnje. Ako je postojeće rješenje već testirano u jednom od simulatora, potrebno je pronaći dodatni simulator sličnih karakteristika kako bi se rezultati usporedili. Potrebno je analizirati kolike su razlike u točnosti procjene kuta zakreta upravljača vozila između rezultata predstavljenih u znanstvenom radu gdje je algoritam predstavljen i rezultata u simulatorima, te ukoliko su razlike velike, pronaći uzrok (npr. trening podaci ne odgovaraju okruženju
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 1 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	18.09.2022.

Potvrda mentora o predaji konačne verzije rada:	*Mentor elektronički potpisao predaju konačne verzije.*
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 22.09.2022.

Ime i prezime studenta:	David Dumančić
Studij:	Diplomski sveučilišni studij Automobilsko računarstvo i komunikacije
Mat. br. studenta, godina upisa:	D-46ARK, 11.10.2020.
Turnitin podudaranje [%]:	5

Ovom izjavom izjavljujem da je rad pod nazivom**: Usporedba rada algoritma za autonomnu vožnju zasnovanog na procjeni kuta zakreta upravljača vozila u različitim simulatorima za razvoj algoritama autonomne vožnje**

izrađen pod vodstvom mentora Izv. prof. dr. sc. Mario Vranješ

i sumentora ,

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD

Prema svim pokazateljima i trendovima, automobilska industrija je trenutno na pragu mobilne revolucije. Od nastanka izuma automobila, svako vozilo i svaki njegov dio su napretkom tehnologije velikom brzinom napredovali u dizajnu, razvoju i proizvodnji, kako bi se zadovoljile potrebe ljudi, povećala sigurnost i ostvarili ekološki standardi današnjice. Jedini element cestovnog prometa koji je ostao donekle isti je vozač, sa svojim ljudskim čimbenikom. Prema istraživanju objavljenom u [1], posljednji događaj u lancu uzroka sudara, tj. kritični razlog sudara u 94 % (+/- 2%) slučajeva je upravo ljudska pogreška. Stoga je jasno da je smanjivanje ili uklanjanje ljudskog čimbenika sljedeći veliki korak u automobilskoj industriji. Sigurnost svih sudionika u prometu je primarni cilj, no automatizacija vožnje bi pridonijela i u drugim segmentima, poput smanjenja emisija štetnih plinova, smanjena gužvi u prometu, povećanja ekonomičnosti vozila i povećanja mobilnosti za podzastupljene i ugrožene zajednice.

Trenutno postoje razni napredni sustavi za pomoć vozaču u vožnji (*engl. Advanced Driver Assistance Systems - ADAS*). Korištenjem dostupnih senzora, npr. RADAR-a, LIDAR-a i video kamera, uz programska rješenja implementirana na čipu ugrađenom u vozilo, rješavaju se sigurnosno-kritični problemi poput prepoznavanja pješaka, upozorenja o napuštanja vozne trake, prepoznavanja prometnih znakova ili automatskog kočenja u nuždi. Takvi sustavi predstavljaju osnovu za autonomnu vožnju. Potpuno autonomno vozilo može se definirati kao vozilo koje ima sposobnost promatranja svoje okoline i koje može obavljati funkcije vožnje bez angažiranja vozača. Vrlo je važno postići da sve značajke vožnje rade jednako dobro u svim vremenskim uvjetima, na različitim cestama i u različitim okolinama u stvarnom vremenu. U slučaju kad čovjek vozi, on se primarno oslanja na vizualne podražaje iz okoline. Stoga postoji rastuće uvjerenje među znanstvenicima i tvrtkama da će autonomna vožnja postati moguća uz korištenje samo sustava kamera u vozilu i strojnog učenja (engl. *Machine Learning*). Strojno učenje je metoda korištenja algoritma da analizira ulazne podatke, uči iz podataka kako da uspješno izvrši zadatak te ga kasnije na osnovu naučenog znanja uspješno izvršava.

Razvoj algoritama autonomne vožnje i njihovo testiranje bili bi vrlo skupi postupci kada bi se od samog početka razvoja koristila realna vozila i realna okruženja. Zbog toga se u početnoj fazi razvoja različitih algoritama autonomne vožnje vrlo često koriste simulatori, u kojima je danas moguće postići vrlo vjernu simulaciju stvarnog svijeta. Upravo zbog toga danas su razvijeni brojni simulatori za razvoj autonomne vožnje, a upravo su oni jedna od glavnih tema ovog rada. U sklopu ovog rada stoga će biti analizirani različiti postojeći simulatori za razvoj algoritama autonomne

vožnje. Glavni je cilj ovog rada uspješno implementirati i testirati jedan postojeći algoritam za autonomnu vožnju zasnovan na procjeni kuta zakreta upravljača vozila u različitim simulatorima i vrednovati njegove performanse. Potrebno je i ispitati različite pogodnosti koje navedeni simulatori pružaju za razvoj i testiranje algoritama autonomne vožnje.

U drugom poglavlju rada pobliže je objašnjen problem procjene kuta zakreta upravljača vozila i dan je osvrt na postojeća rješenja koja implementiraju procjenu kuta zakreta upravljača u različitim simulatorima. Također je dan osvrt na simulatore korištene za razvoj algoritama autonomne vožnje. U trećem poglavlju opisani su implementacija odabranog algoritma i cjelokupni proces treniranja, stvaranja testnih scenarija, kao i upravljanje vozilom unutar simulatora. U četvrtom poglavlju opisan je proces testiranja predloženog algoritma u simuliranom okruženju. U posljednjem poglavlju se nalazi zaključak.

2. PROBLEM PROCJENE KUTA ZAKRETA UPRAVLJAČA VOZILA

Upravljanje vozilom izvodi se slanjem upravljačke naredbe kojom se manipulira brzinom i smjerom kretanja vozila. Promjenu brzine vozila određuje položaj papučica gasa i kočnice, dok smjer kretanja vozila određuje kut zakreta upravljača. U ovom radu je obrađeno upravljanje smjerom vozila zasnovano na procjeni kuta zakreta upravljača vozila. Složenost problema procjene kuta zakreta upravljača vozila leži u varijabilnosti vanjskih čimbenika, kao što su izgled okoliša, vremenski uvjeti, gustoća prometa i geometrijske naravi ceste. Električno upravljanje zakretom upravljača vozila (engl. *drive by wire*), na koje se nadovezuje procjena kuta zakreta upravljača, zahtjeva najveću razinu integriteta u automobilskoj sigurnosti, tzv. ASIL - D (engl. *Automotive Safety Integrity Level*) prema ISO26262 standardu [2] koji regulira funkcionalnu sigurnost (engl. *functional safety*) cestovnih vozila. Samim time je predočena složenost problema procjene kuta zakreta upravljača. Rješavanje istog može biti vrhunac u području naprednih sustava za pomoć vozaču (engl. *Advanced Driver Assistance Systems* – ADAS), a kao jedan od presudnih problema autonomne vožnje, problem procjene kuta zakreta upravljača vozila je privukao pažnju istraživača i proizvođača u automobilskoj industriji.

Algoritmi za procjenu kuta zakreta upravljača vozila općenito koriste dva različita pristupa za rješavanje navedenog problema: pristup zasnovan na tradicionalnom računalnom vidu (engl. *Computer vision*) i pristup zasnovan na dubokom učenju (engl. *Deep learning*). Oba pristupa kao ulazni podatak u algoritam koriste slike snimljene kamerom montiranom na prednjoj strani vozila, dok kao izlaz daju procjenu kuta zakreta upravljača vozila.

2.1. Pregled postojećih rješenja za procjenu kuta zakreta upravljača vozila

2.1.1. Rješenja zasnovana na računalnom vidu

Algoritmi zasnovani na tradicionalnom računalnom vidu općenito provode postupke izvlačenja značajki, detekcije rubova ili regija ceste i računanja kuta zakreta upravljača vozila. U radu [3] je predstavljena hibridna metoda za računanje kuta zakreta upravljača vozila s ciljem da zadrži vozilo u središtu vozne trake. Korištena procedura je podijeljena u više faza. Prva faza uključuje otkrivanje rubova i morfoloških operacija. Zatim je određena središnja točka koja određuje koliko vozilo mora skrenuti da bi ostalo u središtu vozne trake, računanjem Euklidske udaljenosti i korištenjem trigonometrijskih operacija. Ulazni podaci su nepravilnog intenziteta, tj. neujednačena razlika kontrasta čini izvlačenje rubova neučinkovitim. Za rješavanje takvih situacija je korištena Houghova transformacija za detekciju linija. Predloženo rješenje je pokazalo

bolje ponašanje od postojećih rješenja koja su na temelju informacija gradijenta na slici izdvajali rubove primjenjujući prag, što je otežavalo proces otkrivanja linije u smanjenom osvjetljenju. Prijašnja rješenja su zahtijevala dvije vidljive linije, dok navedeni algoritam aproksimira virtualnu liniju (koju nije detektirao) na temelju jedne vidljive linije. Osim toga, u radu je reducirana složenost izračuna što je dovelo do smanjenog vremenskog kašnjenja. Iako su ostvarili napredak u radu pri smanjenom osvjetljenju, algoritam pokazuje lošije ponašanje u otežanim vremenskim uvjetima (kiša i magla). Zaključci su izvedeni iz testiranja na video sekvencama iz stvarnog svijeta snimljenim po dnevnoj i noćnoj vožnji pri 30 okvira po sekundi, a da je pri tom rezolucija okvira 480x640. U radu nije dan podatak o točnom broju video okvira.

U radu [4] autori predstavljaju novi pristup za računanje kuta zakreta upravljača vozila na temelju računalnog vida. Koncept se sastoji se od izvlačenja područja vozne trake, izračuna kuta zakreta upravljača vozila i procjene budućeg zakreta upravljača vozila. Izvlačenje područja vozne trake se vrši GMM-EM (engl. *Gaussian mixture model – GMM, expectation – maximization - EM*) metodom grupiranja sličnih elemenata slike na temelju maksimalne vjerojatnosti. Rubovi na području definiranih područja su detektirani Cannyjevim operatorom. Nakon detektiranja rubova, slika se dijeli na pet dijelova čija veličina ovisi o rezoluciji slike. Najniži dijelovi slike se koriste za izračun orijentacije vozila s obzirom na trenutni video okvir, jer daje postojeću orijentaciju vozila u tom trenutku (neposredna blizina promatrane regije). Drugi isječak slike može se koristiti za procjenu kuta zakreta upravljača vozila (daleko područje) kojim vozilo treba skrenuti s trenutne orijentacije, kako bi održao svoju putanju u središtu vlastite vozne trake. Izvlačenjem područja vozne trake iz slike se definira kut zakreta upravljača vozila. Točnije, kut zakreta upravljača vozila je određen kao devijacija točke sjecišta dviju granica područja vozne trake i orijentacije vozila (središta slike). Ukoliko je vidljiva samo jedna granica unutar vidnog polja, druga granica se definira kao posljednji stupac elemenata slike na drugoj strani. U slučaju da algoritam nije detektirao niti jednu granicu vozne trake, vozilo zadržava svoj smjer kretanja do trenutka identificiranja novih granica. Predloženi algoritam učinkovito računa kut zakreta upravljača vozila uz ne prevelike zahtjeve sklopovlja. Uz to, izračun kuta zakreta upravljača vozila se prenosi na sljedeći trenutak vremena, što je važno za glatke i sigurne promjene putanje vozila, no pristup nije zasad primjenjiv na oštre i strme zavoje. Predstavljeno rješenje je testirano u Gazebo [5] simulatoru i slikama iz stvarnog svijeta. Unutar simulatora su stvorene dvije testne sekvence koje se sastoje od ravnih i zakrivljenih dionica. Granice staze predstavljaju zidovi stvoreni u simulatoru, a same staze ne sadrže vizualne oznake ili različitu teksturu u odnosu na okolinu. U prvoj testnoj sekvenci, vozilo je uspješno upravljalo kroz zadanu kružnu stazu, vozeći se središtem zadanog voznog

područja omeđenog s dva zida. Za drugu testnu sekvencu je izgrađena kružna staza koja sadrži jedan zid s unutarnje strane. Vozilo je uspješno prošlo cijelu stazu na sigurnoj udaljenosti (3.5 metra od zida). Test na slikama iz stvarnog svijeta je proveden na *Udacity* skupu podataka gdje je rješenje postiglo lošije rezultate od rješenja zasnovanih na dubokom učenju.

2.1.2. Rješenja zasnovana na dubokom učenju

Duboko učenje je vrsta strojnog učenja koje oponaša način na koji ljudi stječu određene vrste znanja. Rješenja zasnovana na dubokom učenju pokušavaju protumačiti svijet analizirajući kontekst predviđene scene, s naglaskom na bitne objekte i promatrajući ih na hijerarhijskim razinama. Prilikom analiziranja scene, duboko učenje je otpornije na promjenu uvjeta okoline. Međutim, u dubokom učenju umjetne neuronske mreže zahtijevaju velik broj ulaznih podataka iz kojih uče kako bi postigle prikladnu preciznost. Upravljanje vozilom je složen zadatak, i kao takvog ga je teško napisati u obliku egzaktnog algoritma tj. niza instrukcija. Umjetne neuronske mreže su pokazale obećavajuće sposobnosti i prilagodljivost u raznolikim okruženjima s velikom razinom šuma, poput prepoznavanja rukopisa [6] i govora [7]. Drugim riječima, umjetne neuronske mreže mogu prepoznati složene interakcije između značajki, što je korisno i za autonomnu vožnju u promjenjivim okruženjima.

Projekt ALVINN (engl. *Autonomous Land Vehicle In a Neural Network*) [8] je 1989. iskoristio prednosti neuronskih mreža s ciljem autonomne vožnje. Kao ulazne podatke neuronska mreža uzima slike s prednje kamere vozila i udaljenosti određene laserskim daljinomjerom, a izlaz iz mreže daje smjer u kojem bi se vozilo trebalo kretati kako bi slijedilo cestu. Dizajniran je u obliku plitke mreže s tri sloja i koristi prilagodbu koeficijenata u mreži unazad, koristeći gradijentnu metodu. Rad je demonstrirao da *end-to-end* trenirana mreža, tj. mreža u kojoj model uči samostalno sve korake između početne ulazne slike i konačnog izlaznog rezultata, može upravljati vozilom na javnim cestama. S tom spoznajom i napretkom tehnologije koja je omogućila bržu obradu i obradu većeg broja podataka, predloženo je rješenje nazvano *PilotNet*. Autori su u radu [9] predložili arhitekturu nove konvolucijske neuronske mreže (engl. *Convolutional neural network – CNN*) i nju su trenirali da mapira neobrađene (engl. *raw data*) elemente slike (engl. *pixel*) s jedne prednje kamere izravno u upravljačke naredbe. Za razliku od prijašnjih CNN rješenja, *PilotNet* je išao dalje od samog prepoznavanja uzoraka. Naučili su mrežu cijeli proces potreban za upravljanje automobilom. Predstavljeni algoritam je u stanju prepoznati potrebne značajke iz prometnog okruženja, poput linija, rubova kolnika i objekata, dok ignorira one koje nisu važne. Prema autorima, potrebno je više rada kako bi se poboljšala robusnost mreže, uz to potrebno je unaprijediti metode za vizualizaciju koraka obrade unutar mreže. Testiranje mreže se

provelo kroz dva koraka, prvo u simulaciji, a zatim u testovima na cesti. Autori su kreirali vlastiti simulator koji se sastoji od video sekvenci iz stvarnog svijeta. Poznato je da su video sekvence trajale 3 sata, što je ekvivalentno 160 prijeđenih kilometara, ali u radu nije predstavljen podatak o rezultatima testiranja na video sekvencama ili simulatoru. Na temelju testova na stvarnoj cesti, izmjerili su autonomiju od 98%, tj. pokazali su da je mreža u 98% vremena mogla ispravno upravljati vozilom. *PilotNet* algoritam je jedan od najpoznatijih i najrasprostranjenijih suvremenih algoritama za procjenu kuta upravljača vozila. Iako nema objavljenu korištenu bazu podataka, dostupna je njegova provjerena i jednostavna arhitektura. Postoji velik broj radova koji implementiraju *PilotNet* arhitekturu mreže koristeći različite biblioteke za strojno učenje, a time je dostupan velik broj objavljenih programskih kodova i popratne dokumentacije za lakši razvoj. Zbog svega navedenog, *PilotNet* je odabran za implementaciju u različitim simulatorima u sklopu ovog diplomskog rada i stoga je detaljnije predstavljen u potpoglavlju 2.2.

CNN je arhitektura dubokog učenja koja se sastoji od više slojeva. Zbog svoje učinkovitosti u rješavanju problema zasnovanih na obradi slike, iznimno je važna za razvoj algoritama autonomne vožnje. Prema preglednom radu [10], CNN se koristi u 79% radova vezanih uz procjenom kuta zakreta upravljača vozila. Velik broj radova je potvrdio visoku razinu autonomije. Kako autori *PilotNet-a* nisu prezentirali rezultate testiranja u simulatoru, u radu [11] su predstavili detaljnu implementaciju CNN mreže procjenu kuta zakreta upravljača vozila u TORCS [12] simulatoru. Skup podataka za treniranje je umjetno generiran u simulatoru, a sadrži deset sati vožnje na dvjema različitim stazama. Pri tom je zaključeno da mreža pokazuje bolje ponašanje ukoliko su trening podaci skupljeni na stazi koja ima veći postotak zavoja u odnosu na ravne dionice. Evaluacija rješenja je također izvršena u TORCS simulatoru. Koristeći *PilotNet* arhitekturu mreže, postigli su autonomiju od 96.62 % na cestama s više voznih traka i autonomiju od 89.02 % na cestama s jednom voznom trakom. Iako je model treniran na podacima sa staze u višestrukim oznakama voznih traka, pokazao se vrlo dobar u drugim stazama s jednom trakom u kojoj se boja i tekstura ceste razlikuje od staze na kojoj je učen. U radu [13] manja CNN mreža je trenirana pomoću video isječaka stvarne vožnje s ceste iz *Comma.ai* [14] skupa podataka. Za svaki okvir, predviđeni kut upravljanja se uspoređuje sa stvarnom vrijednošću (engl. *Ground truth*). Trenirani model je testiran na dvjema video sekvencama koje sadrže 25 tisuća video okvira. Za svaki okvir procijenjeni zakret upravljača vozila je uspoređen sa stvarnom vrijednošću. Prema autorima, korištenje razlike stvarnog kuta stvarnog vozača i predviđenog kuta mreže u svrhu evaluacije je zapravo upitna metoda. Stvarni kut upravljanja nije globalno optimalan. Stvarni vozač ne može održavati vozilo cijelo vrijeme u sredini vozne trake. Sve dok je vozilo u prometnoj

traci, predviđeni kutovi mreže su u redu i ne moraju biti u potpunosti isti kao kod stvarnog vozača, tj. kretanje vozila i upravljačke kontrole su kontinuirane, dakle ocjenjivanje kadar po kadar nije primjereno, što zagovara korištenje simulatora za testiranje procjene kuta zakreta upravljača vozila. Prema [15] mreže poput *PilotNeta* uglavnom neovisno procjenjuju kut zakreta upravljača za svaki video kadar, što je u suprotnosti s uobičajenim shvaćanjem principa kontinuirane vožnje. Predloženi model radi na temelju kombinacije prostornih i vremenskih značajki, korištenjem trenutnih slika s kamere i promatranjem prošlih stanja vozila. Autori ovog rada su to postigli postavljanjem povratnih jedinica na odgovarajućim slojevima duboke mreže, točnije postavljanjem LSTM (engl. *Long Short-Term Memory*) ćelija i konvolucijskog LSTM sloja. Mreža se sastoji od četiri prostorno-vremenske konvolucije. Svaki kut zakreta upravljača je određen trenutnim stanjem i prethodnim stanjima koje model pamti. CNN-LSTM se uz procjenu kuta zakreta upravljača koristi i za istovremenu procjenu brzine vozila. Jedan okvir slike s prednje kamere vozila nije dovoljan za preciznu procjenu brzine vozila, jer je izazovno dobaviti vremensku informaciju u procesu vožnje. Kombinacija strukture CNN-LSTM s višestrukim pomoćnim zadacima, poput semantičke segmentacije i detekcije objekata, pozitivno utječu na odluke u samostalnoj vožnji. Autori navedenog rada tvrde da je obuka iz video sekvenci stvarnog svijeta ključna za osiguranje sigurnosti vozila prilikom postavljanja modela u stvarnim vozilima. Međutim, iako podaci prikupljeni iz TORCS simulatora dolaze u velikoj količini, ne sadrže vizualni šum karakterističan za slike iz stvarnog svijeta. Vizualno okuženje se prikazuje tehnikama računalne grafike, a time ozbiljno odstupaju od scenarija stvarne vožnje. Uz to, ponašanje drugih agenata u TORCS simulatoru ne reprezentira stvarno ponašanje sudionika u prometu. Rješenje stoga nije testirano u simulatoru.

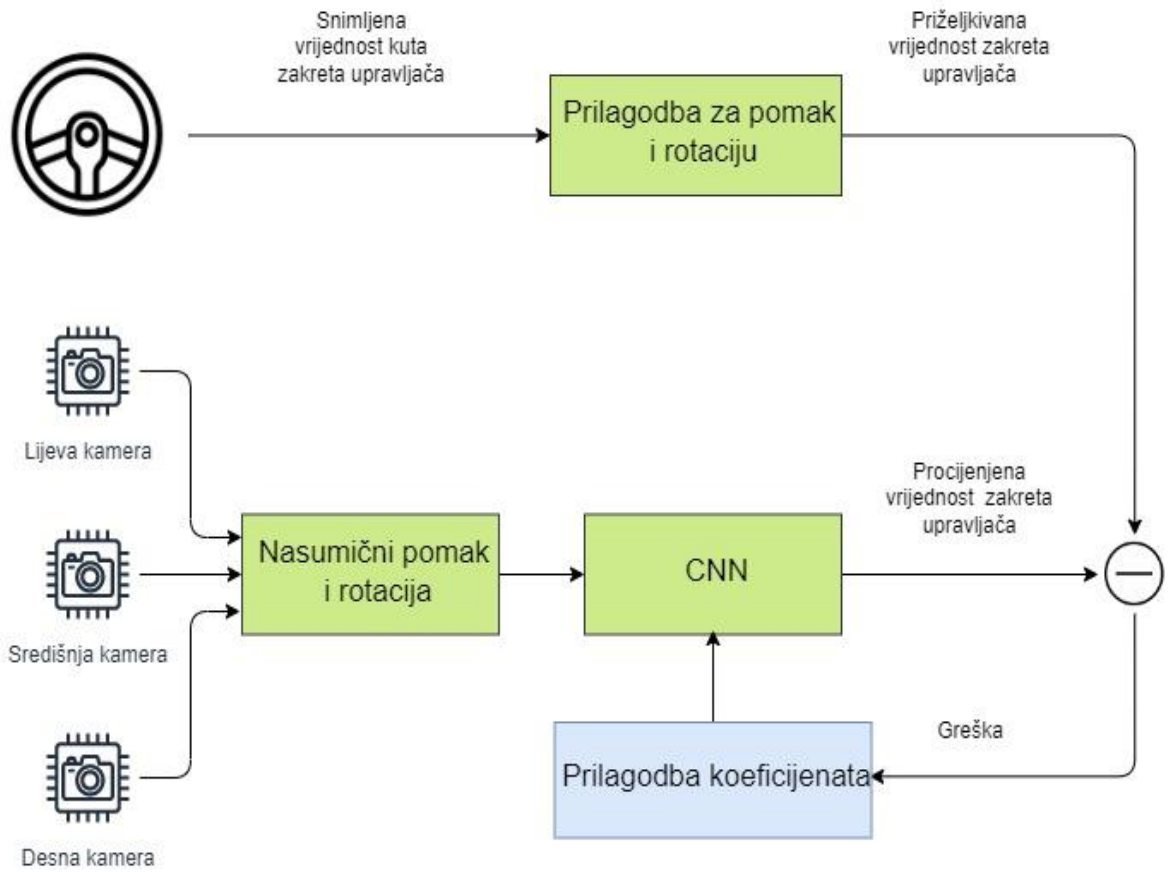
2.1.3. Rješenja zasnovana na podržanom učenju

U svim navedenim primjerima nadziranog učenja, željeni izlazni podatak mreže odgovara ulaznoj varijabli, koja je pridružena kao oznaka tijekom procesa treniranja. Međutim, postoji napredak i u drugoj grani strojnog učenja za rješavanje navedenog problema. Podržano učenje (engl. *Reinforcement learning*) je vrsta strojnog učenja u kojoj agent međusobno djeluje s okolinom, bez ikakvih prijašnjih informacija, kako bi pronašao najbolji opis ponašanja za zadani zadatak. Prilikom treniranja, agent izvodi radnju u određenom vremenskom intervalu s početnim stanjem i ponašanjem. Rezultat akcije mijenja stanje agenta. Na temelju procjene promjene stanja dodjeljuje se nagrada. Proces se ponavlja, a na temelju iskustva agent uči potrebno ponašanje za svladavanje zadatka. Sposobnost samoučenja vozila koje djeluje s okolinom je jedna od najizrazitijih osobina predložene metode. Takva sposobnost se može primijeniti za rješavanje

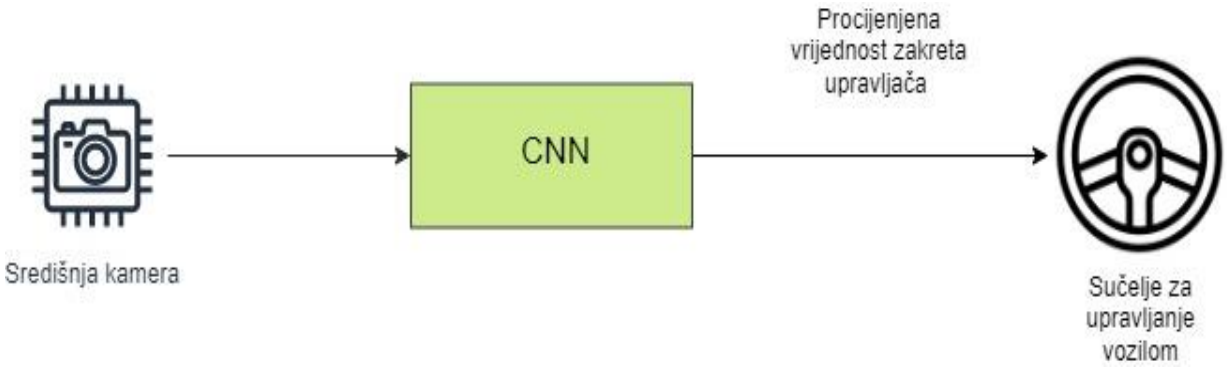
problema koji se suočava s neizvjesnim stanjem okoline. Upravo je to predstavljeno u radu [16], gdje je korištenje podržanog učenja usmjereno na provjeru je li moguće upravljati vozilom u simulatoru, a zatim koristiti isti model u stvarnom svijetu. Za simulacijsko okruženje je korišten CARLA simulator [17]. Model koristi RGB sliku, semantičku segmentaciju i navigacijske naredbe više razine (prati traku, skreni lijevo, skreni desno, idi ravno) kao ulazne podatke. U simulaciji je agent nagrađen za praćenje referentne putanje, a paralelnim treniranjem (korištenjem više agenata u isto vrijeme) model je skupio oko 100 godina iskustva u simuliranoj vožnji. Autori su testirali trenirani model u scenarijima iz stvarnog svijeta. U radu su postigli uspješan prijenos ponašanja iz simulatora u stvarno vozilo, međutim dokazali su da dobro ponašanje vozila u simulatoru nije garancija dobrog ponašanja u stvarnom svijetu.

2.2. PilotNet algoritam za procjenu kuta zakreta upravljača vozila

Kao dio cjelovite programske podrške za autonomnu vožnju, tvrtka NVIDIA je stvorila sustav zasnovan na CNN, poznat kao *PilotNet*. CNN se zasniva na nizu sekvencijalno povezanih konvolucijskih slojeva, između kojih se nalaze aktivacijske funkcije. Sustav procjenjuje kutove zakreta upravljača vozila za zadanu sliku ceste ispred vozila. Cilj mreže je ukloniti potrebu pisanja „ručnih“ pravila i stvaranje sustava koji uči promatranjem. Skup podataka za treniranje (engl. *dataset*) čine slike s prednje strane vozila (snimljene kamerom montiranom na prednjoj strani vozila i usmjerenom prema naprijed) uparene s vremenski sinkroniziranim kutovima zakreta upravljača dobivenim za vrijeme vožnje stvarnog vozača. Kutovi zakreta upravljača su prikupljeni pristupanjem CAN (engl. *Controller Area Network*) sabirnici vozila, te se zapisuju u obliku $1/r$, gdje je r polumjer najmanjeg kruga koji vozilo može napraviti s potpuno zakrenutim upravljačem. Treniranje mreže, koristeći isključivo ulazne podatke dobivene za vrijeme stvarne vožnje, nije dovoljno jer mreža mora naučiti kako se oporaviti od grešaka u vožnji. Nije prihvatljivo da vozač prilikom snimanja skupa podataka namjerno griješi u stvarnom prometu. Stoga se koriste tri kamere koje istovremeno snimaju prostor ispred vozila i na taj način proširuju skup podataka za treniranje. Dodatne slike oponašaju vozilo s različitim pomakom od središta vozne trake i različitom orijentacijom u odnosu na smjer ceste. Na slici 2.1. prikazan je blok dijagram sustava za treniranje korištenog u [9]. Nakon što je mreža jednom naučena kako procijeniti zakret upravljača vozila iz ulaznih podatka, može procijeniti vrijednost kuta zakreta upravljača vozila koristeći samo središnju kameru (Slika 2.2.) [9].



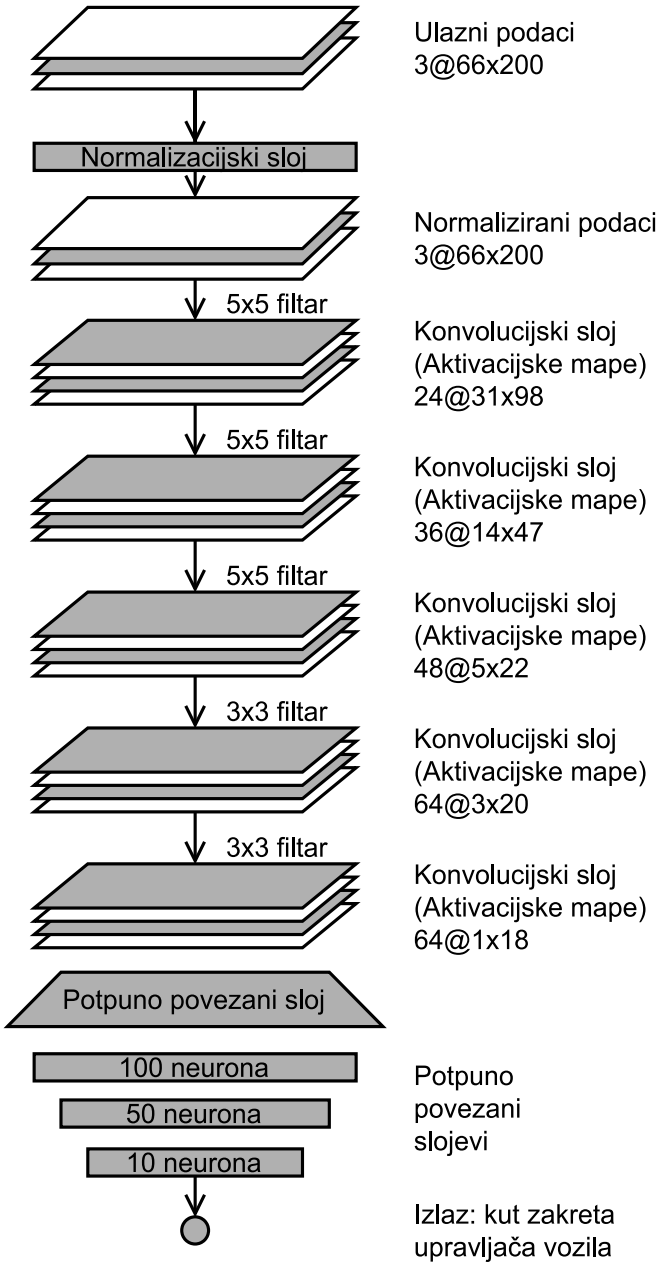
Sl. 2.1. Postupak treniranja neuronske mreže [9]



Sl. 2.2. Upravljanje zakretom upravljača vozila nakon što je mreža (CNN) naučena [9]

Arhitektura CNN sastoji se od 9 slojeva (Slika 2.3) [9], uključujući sloj normalizacije, 5 konvolucijskih slojeva i 3 potpuno povezana sloja, što ukupno daje oko 27 milijuna veza i 252219

tisuća parametara. Ulazni podatak mreže je YUV slika rezolucije 200x66 elemenata slike (pošto slika ima 3 kanala, može se vidjeti da je dimenzija ulaznog kanala označena kao 3@66x200).



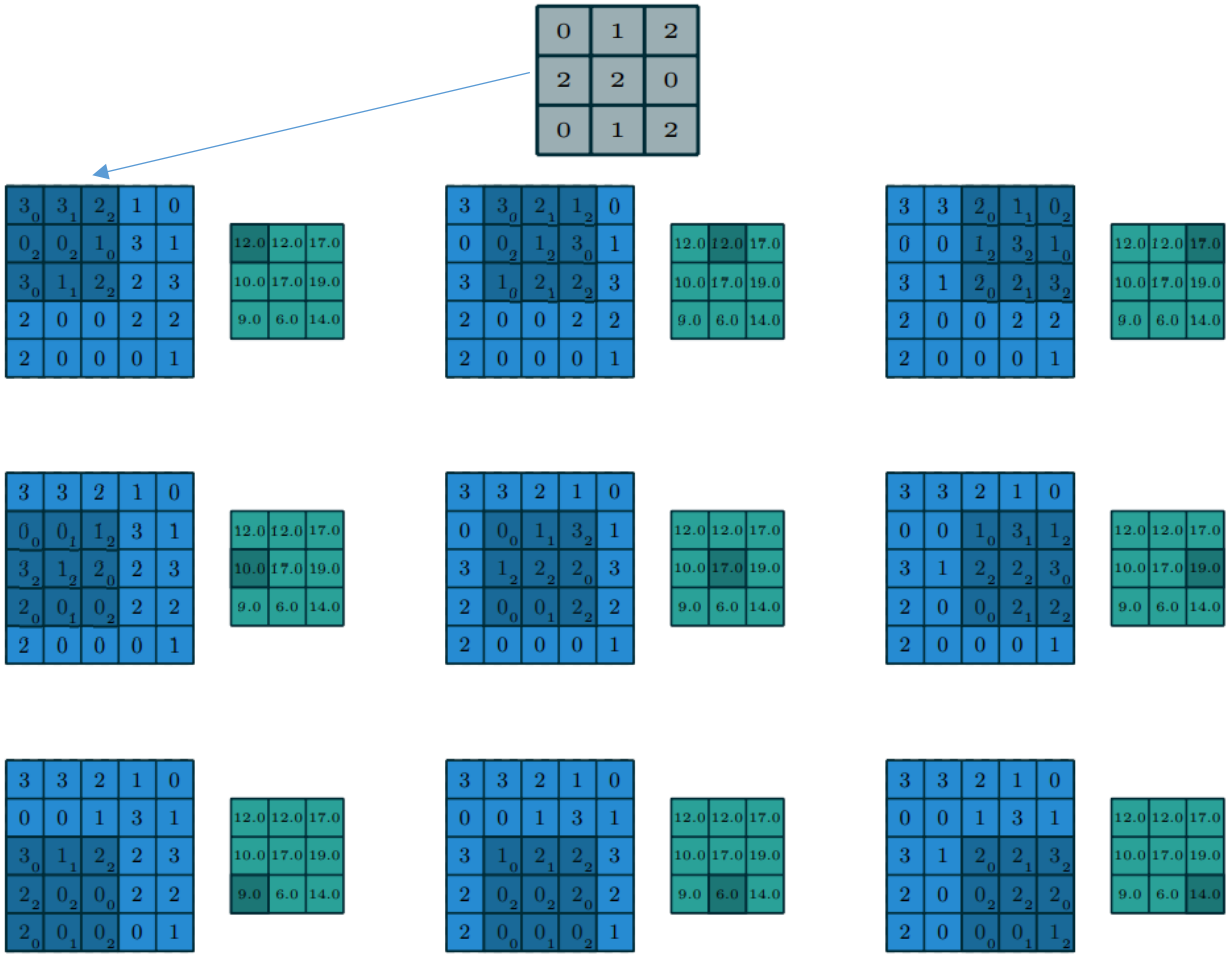
Sl. 2.3. Arhitektura konvolucijske neuronske mreže[9]

Treniranje je prilagođavanje navedenih parametara mreže pronalaskom minimuma kriterijske funkcije, koja ukazuje kolika je pogreška procjene mreže. Prvi sloj mreže izvodi normalizaciju slike tj. pretvaranje zapisa svakog elementa slike iz cjelobrojnog raspona [0,255] u broj s pomičnim zarezom u rasponu [0,1]. Zatim je definiran niz konvolucijskih slojeva, koji su

dizajnirani za izvlačenje značajki primjenom filtara. Korištenjem konvolucijskog sloja se zadržava prostorna struktura ulaznih podataka. Matematička formulacija 2D konvolucije je dana prema izrazu (2-1), gdje **x** predstavlja matricu ulazne slike koja konvoluira s filtrom **h** da bi se dobila nova matrica **y**, pri čemu ona predstavlja izlaznu aktivacijsku mapu.

$$y[i,j] = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} h[m,n] * x\,[i-m, j-n] \tag{2-1}$$

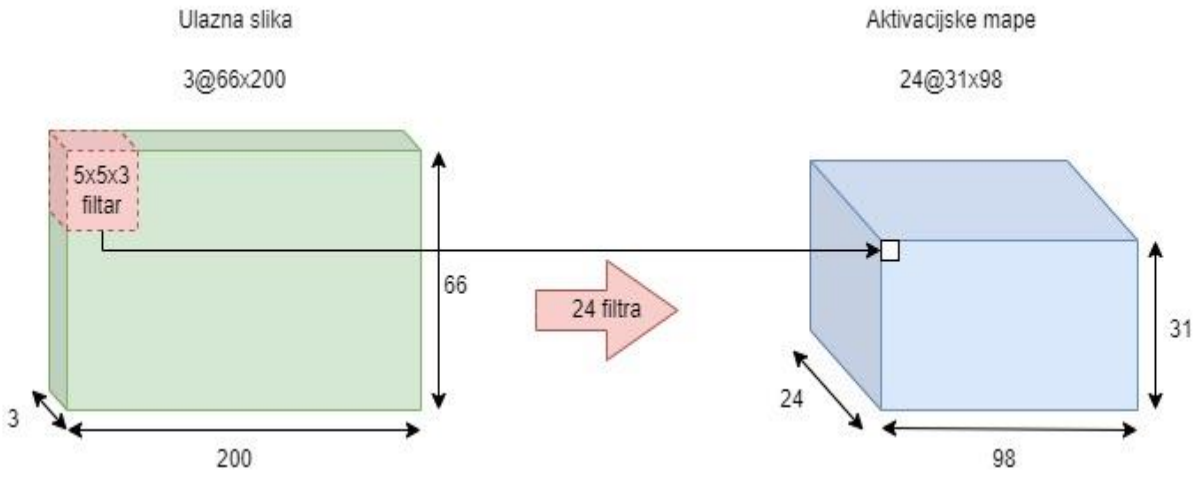
Na slici 2.4 prikazan je primjer 2D konvolucije.



Sl. 2.4. Računanje izlaznih vrijednosti konvolucije [16]

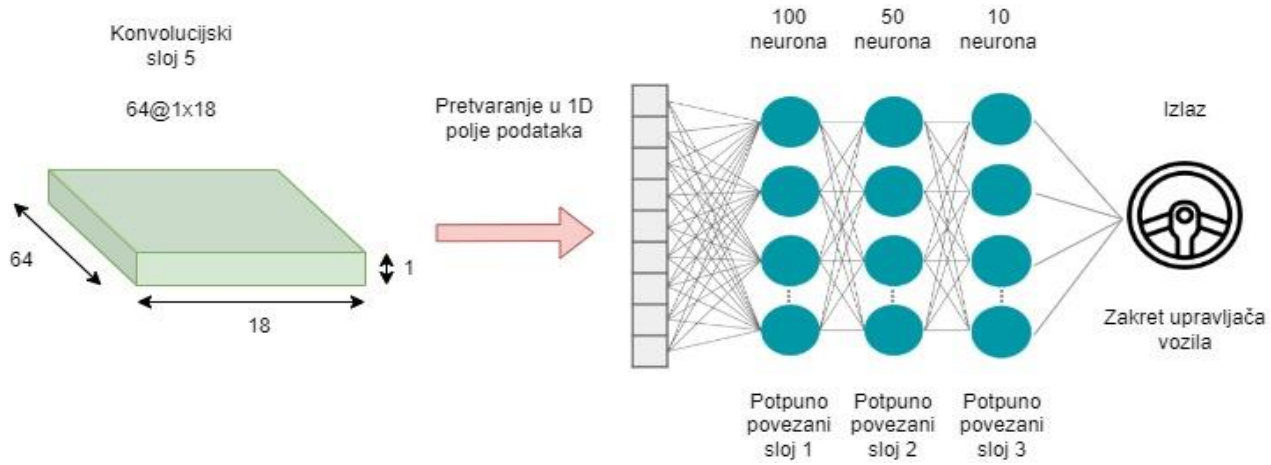
Za ovaj ilustrativni primjer, neka je matrica ulazne slike rezolucije 5x5 (na slici obojano plavom bojom) na koju se primjenjuje matrica filtra veličine 3x3 (na slici obojano sivom bojom). Izlazna aktivacijska mapa (na slici obojano zelenom bojom) predstavlja dvodimenzionalnu matricu koja sadrži odziv filtra na pojedinom dijelu ulazne slike. Izračunava se umnožak svakog elementa filtra

i elementa ulazne matrice koji se preklapaju. Njihov zbroj predstavlja rezultat na trenutnoj lokaciji u izlaznoj matrici. U slučaju *PilotNeta* korišteni su filtri veličine 5x5 s korakom (engl. *Stride*) 2x2 u prva tri konvolucijska sloja i filtri veličine 3x3 u posljednja dva konvolucijska sloja s korakom 1x1. Korak definira za koliko se filtar pomiče po aktivacijskoj mapi prilikom svakog izračuna. Sami ulaz u prvi konvolucijski sloj ima određenu dubinu zbog triju komponenata slike (Y,U,V). Shodno tomu, potrebno je koristiti i filtar jednake dubine (konkretno ovdje 5x5x3). Svaki filtar koji se primjenjuje na podatke daje odziv u obliku dvodimenzionalne aktivacijske mape. U konvolucijskom sloju se primjenjuje veći broj filtara istih dimenzija, a primjena svakog filtra rezultira jednom 2D aktivacijskom mapom. Tako npr. prvi konvolucijski sloj *Pilotnet-a* koristi 24 filtra dimenzije 5x5x3. Rezultat su 24 aktivacijske mape sa smanjenom prostornom veličinom 31x98, u zapisu 24@31x98 (Slika 2.5). Prostorna veličina je ovdje smanjena zbog primjene koraka 2x2.



Sl. 2.5. Prvi konvolucijski sloj *PilotNet* arhitekture i njegov izlazni volumen

Izlazni podaci nakon pet konvolucijskih slojeva pretvaraju se u jednodimenzionalno polje podatka (engl. *Flatten*). Stvoreni vektor podataka pritom služi kao ulaz u potpuno povezani dio predložene neuronske mreže, gdje je svaki čvor izravno povezan sa svakim čvorom u prethodnom i sljedećem sloju. Potpuno povezani dio *PilotNet* mreže se sastoji od takva tri potpuno povezana sloja (Slika 2.6). Izlaz iz mreže je vrijednost zakreta upravljača vozila u obliku (1/r), gdje je r radijus zakreta upravljača vozila.



Sl. 2.6. Pretvaranje 2D polja podataka u 1D polje te prikaz strukture potpuno povezanog dijela mreže *PilotNet* mreže

2.3. Pregled postojećih simulatora za razvoj algoritama autonomne vožnje

Istraživanje autonomne vožnje u realnom urbanom okolišu je ograničeno velikim infrastrukturnim troškovima i logičkim poteškoćama sustava za testiranje u fizičkom svijetu. Uz to, neki scenariji vožnje su opasni za testiranje na stvarnim prometnicama. Alternativni pristup je u ranim fazama razvoja rješenja trenirati ili potvrditi strategije vožnje unutar simulatora, a tek u kasnijim fazama prijeći na skuplje testiranje u realnom scenariju. Problem kod testiranja u simulatoru je taj što su rezultati testiranja ovisni o kvaliteti korištenog simulatora i o tome koliko dobro stvoreni scenarij simulira stvarni okoliš.

U radu [18] autori su koristili *CarND Udacity* [19] simulator pri snimanju skupa podataka za treniranje i testiranje *PilotNet* mreže. Pokazalo se da je generiranje podataka u simulatoru vrlo učinkovito. Jednostavno je stvoriti veliki skup podataka za eksperiment, a time su izbjegli skupljanje podataka iz stvarnog svijeta, što zahtijeva više resursa. *Udacity* je simulator otvorenog koda zasnovan na *Unity* alatu, a sadrži dva načina rada: prikupljanje podataka i autonomna vožnja. Pri zadanim postavkama u načinu rada prikupljanja podataka bilježe se tri slike s prednje strane vozila zajedno s pripadajućim kutovima zakreta upravljača. U autonomnoj vožnji se upravlja vozilom od strane aplikacijskog programskog sučelja (engl. *Application programming interface-API*). Sadrži samo dvije mape za testiranje autonomne vožnje, uz to ceste na implementiranim mapama ne sadrže realne kolničke oznake za detaljniju analizu ponašanja vozila. *Udacity* simulator je dizajniran da ne koristi mnogo RAM-a i CPU-a, a time ima sveukupno nisku upotrebu resursa sustava, što ga čini odličnim za rad na manjim projektima s ograničenim računalnim resursima. Međutim, ne sadrži različite vremenske uvjete i postavke scene. Najveća mana

navedenog simulatora je upravo manjak realizma, što znači da model treniran na skupu podataka iz simulatora vjerojatno ne bi radio dobro u drugom stvarnom ili simuliranom okruženju. Osim navedenih problema, simulator nema detaljnu i jasnu dokumentaciju, što zahtjeva znanje rada s *Unity* razvojnim okruženjem u slučaju promjene i osnovnih parametara (npr. postavki kamere).

Autori iz rada [20] su također koristili simulator za snimanje skupa podataka. Postavljeno je simulacijsko okruženje u komercijalnom CARSIM [21] simulatoru. CARSIM predstavlja vozilo kao skup više objekata, kako bi točno reproducirao fiziku vozila (kinematika, usklađenost ovjesa, sila trenja između gume i podloge, svojstva momenta) kao odgovor na kontrole vozača ili automatizacije. Simulator sadrži 99 različitih senzora (kamera, RADAR, LIDAR, itd.) za razvoj naprednih sustava za pomoć vozaču i autonomne vožnje. Podržava velik broj objekata s nezavisnim lokacijama i kretanjima, a uključuje: pješake, vozila, bicikliste, životinje i druge objekte od interesa za ADAS sustave. Od uvjeta okoline moguće je modificirati one aerodinamičke i efekte vjetra. CARSIM je specijaliziran za simulaciju dinamike vozila zbog svoje velike kolekcije vozila i mogućnosti podešavanja niza parametara simulacije, ali razvoj autonomne vožnje zahtjeva više faktora, poput kompleksnih uvjeta okruženja. CARSIM ne podržava različite vremenske uvjete u sceni što je uz komercijalnu licencu najveći nedostatak.

Gazebo je jedan od najpoznatijih simulacijskih platformi otvorenog koda u robotici i popratnim područjima. Modularni dizajn dopušta implementaciju raznih modela senzora i opisa ponašanja objekata. Korištenjem raznih proširenja omogućuje korištenje dinamičkog učitavanja modela, korištenja kamera, LIDAR-a, GPS-a, IMU-a, i RADAR-a. Prednost takvog popularnog i besplatnog simulatora je velika i aktivna Internet zajednica, a uz to Gazebo je uključen u ROS (engl. *Robot Operating System*) paket tj. skup biblioteka i raznih programskih rješenja otvorenog koda koji omogućuju stvaranje robotskih aplikacija. S navedenim mogućnostima, jasno je da je pronašao mjesto i u razvoju rješenja autonomne vožnje, ali kreiranje velikih i kompleksnih okruženja je vrlo zahtjevno. Velike virtualne scene, u odnosu na simulatore zasnovane na *Unreal Engine* i *Unity* razvojnom okruženju, imaju manjak realnosti prilikom prikazivanja. Simulatori poput *Udacity, CARSIM ili Gazebo* simulatora imaju svoje prednosti, ali za razvoj percepcijskih algoritama simulator mora prikazivati fotorealističnu reprezentaciju okoliša. Videoigre vezane uz vožnju danas nude realna okruženja. Osim komercijalnih i besplatnih simulatora, u svrhe istraživanja su korištene i igre poput *Grand Theft Auto V* za generiranje skupova podatka. Međutim, to zahtijeva uređivanje datoteka igrice, što može prekršiti korisničku licencu. Osim toga nemoguće je podržati senzore osim kamere i deterministički kontrolirati ostala vozila i pješake u prometu.

CARLA i AirSim [22] su simulatori otvorenog koda su napravljeni na bazi *Unreal Enginea,* alata razvijenog za stvaranje računalnih igrica, što rezultira visokom razinom realizma. Napravljeni su kao istraživačke platforme za podržano učenje i skupljanje umjetnih skupova podataka za strojno učenje. Time su i najprikladniji za testiranje funkcionalnosti koje nude autonomna vozila zbog ugrađenih senzora i automatiziranih značajki kao što su percepcija, mapiranje, lokalizacija i upravljanje vozilom. Oba simulatora omogućuju upravljanje vozilom od strane aplikacijskog programskog sučelja, što omogućuje implementaciju algoritma za autonomnu vožnju u simulator. Kako su oba simulatora zasnovana na *Unreal Engineu*, moguće je koristiti iste testne sekvence u kojima se testira algoritam. Zbog navedenih prednosti koje pružaju CARLA i AirSim simulatori, odabrani su za implementaciju *PilotNet* algoritma u sklopu ovog diplomskog rada i stoga su detaljnije predstavljeni u dijelovima 2.3.1 i 2.3.2., a uočene pogodnosti i nedostatke koje su pružili tijekom testiranja opisani su u 4. poglavlju.

2.3.1. CARLA simulator

CARLA (engl. *Car Learning to Act*) je simulator urbane vožnje otvorenog koda. Simulator je izrađen na temelju *Unreal Enginea 4* i pruža realistično grafičko okruženje. Razvijen je za potrebe skupljanja skupa podataka, izrade prototipova i validacije modela autonomne vožnje, uključujući metode opažanja i upravljanja. Okruženje je sastavljeno od 3D modela statičkih objekata poput zgrada, vegetacije, prometnih znakova, infrastrukture. Osim statičkih elemenata sadrži i dinamičke elemente poput vozila i pješaka. Sadrži različite postavke urbanih naselja, kao i mnoštvo modela vozila. Implementirani su različiti vremenski uvjeti i načini osvjetljenja. Razlikuju se po položaju i boji zračenja neba, kao i raspršivanju svjetlosti, atmosferskoj magli, naoblakama i oborinama. U korištenoj verziji, simulator podržava tri načina predefiniranih osvjetljenja povezana s dobima dana:

- podne,
- predvečer (zalazak sunca),
- noć.

Međutim, sadrži i 9 predefiniranih različitih vremenskih uvjeta koji se kombiniraju s načinima osvjetljenja, a pritom se razlikuju po naoblaci, razini oborina i količini lokvi na cesti:

- unaprijed zadani vremenski uvjeti (engl. *default*),
- vedro vrijeme,
- oblačno vrijeme,
- jaka kiša,

- umjerena kiša,
- slaba kiša,
- mokro i oblačno vrijeme,
- mokro.

Svi predefinirani vremenski uvjeti stvoreni kombinacijom više parametara u određenim rasponima:

- naoblaka [0,100] (pri čemu je 0 vedro nebo, a 100 potpuno prekriveno oblacima)
- padaline [0,100] (vrijednost intenziteta kiše, pri čemu je 0 bez kiše, a 100 jaka kiša)
- naslage padalina [0,100] (pri čemu je 0 ništa, a 100 cesta potpuno prekrivena vodom)
- intenzitet vjetra [0,100] (kontrolira jačinu vjetra gdje je 0 bez vjetra, a 100 jak vjetar)
- azimut sunca [0,360] (0 je početna točka u sferi koju određuje *Unreal Engine,* a 360 završna)
- kutna visina sunca [-90,90] (-90 odgovara ponoći, a 90 podnevu)
- gustoća magle [0,100] (koncentracija magle koja utječe samo na RGB kameru)
- udaljenost magle [0,+∞] (udaljenost na kojoj se magla generira u odnosu točku gledišta)
- vlaga [0,100] (intenzitet vlažnosti koji utječe samo na RGB kameru)
- opadanje magle [0, +∞] (što je vrijednost veća, to će magla biti gušća i teža, a time će dosezati manje visine)
- Rayleigheva skala raspršenja [0, +∞] (kontrolira interakciju svjetlosti s malim česticama poput molekula zraka. Ovisno o valnoj duljini svjetlosti, što rezultira plavim nebom danju ili crvenim nebom navečer)
- Mie ljestvica raspršenja [0, +∞] (Kontrolira interakciju svjetlosti s velikim česticama poput peludi ili onečišćenja zraka što rezultira maglovitim nebom s aureolama oko izvora svjetlosti)
- Pješčana oluja [0,100] (Određuje jačinu vremena oluje s prašinom).

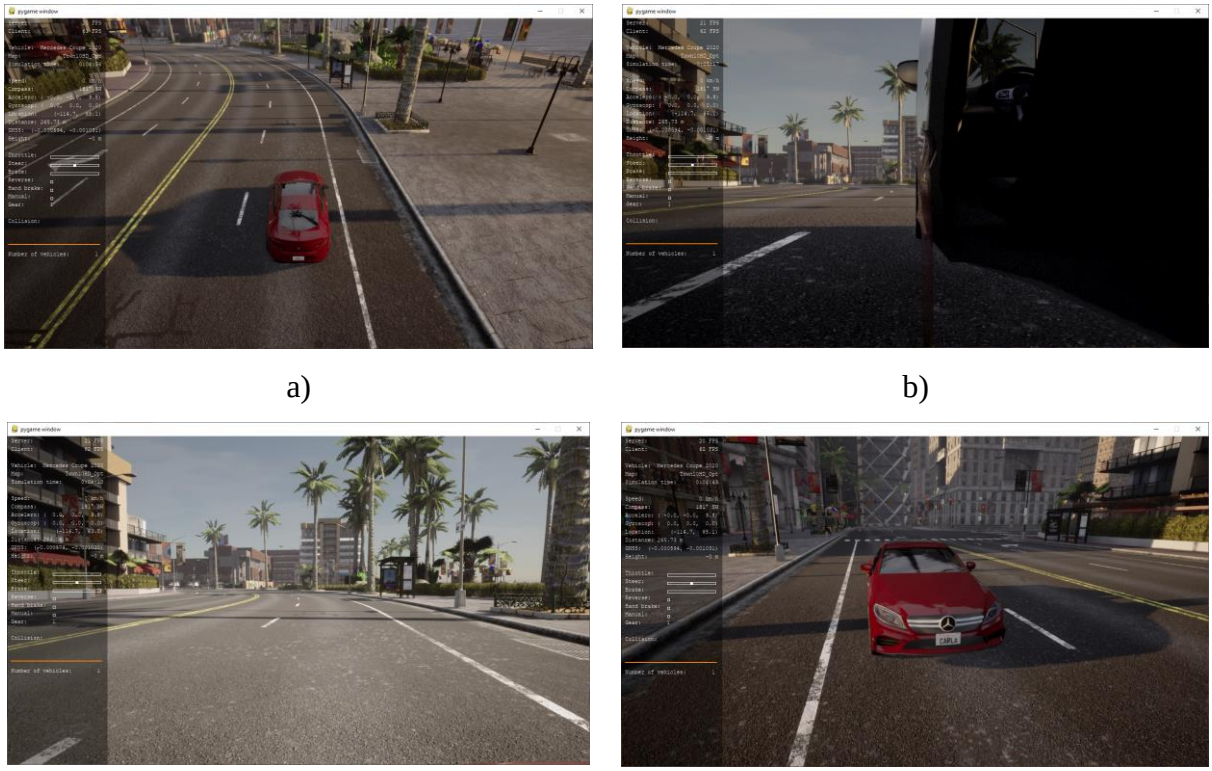
Svaki od navedenih parametara djeluje neovisno o ostalima, tako npr. povećanje količine padalina neće automatski stvoriti lokve niti će se promijeniti vlažnost ceste. Korištenjem API-ja koje pruža CARLA simulator, moguće je primijeniti vremenske uvjete prilagođene potrebama, koji nisu na popisu predefiniranih vremenskih uvjeta poput izražene guste magle po danu ili pješčane oluje. Na slici 2.7 prikazan je primjer različitih vremenskih uvjeta i osvjetljenja u CARLA simulatoru.

a) b)

c) d)

Sl. 2.7. Primjeri različitih vremenskih uvjeta i osvjetljenja CARLA simulatora u ugrađenoj mapi Town10: (a) vedro vrijeme u podne, (b) kišno vrijeme predvečer, (c) vedro vrijeme po noći, (d) kišno vrijeme po noći

Simulator podržava postavljanje različitih senzora koji se pojavljuju i na stvarnim vozilima. Po potrebi pružaju signale koji se koriste za treniranje strategije vožnje. Simulator daje sliku s RGB kamere i omogućava korištenje brojnih drugih senzora, poput LIDAR-a, RADAR-a i pseudo senzora koji pružaju dubinsku sliku, te omogućava korištenje implementiranog optičkog toka i semantičke segmentacije. Parametri kamere sadrže 3D lokaciju i orijentaciju u odnosu na koordinatni sustav vozila. Klijent može odrediti te parametre, broj kamera i njihovu vrstu. Po zadanim postavkama postoji veći broj pogleda kamere prilikom vožnje. Primjeri različitih perspektiva prilikom upravljanja vozila u CARLA simulatoru prikazani su na slici 2.8.



a) b)

c) d)

Sl. 2.8. Primjeri različitih perspektiva kamere u CARLA simulatoru: (a) pogled na vozilo odozgo, (b) pogled s bočne strane vozila, (c) pogled s prednje strane vozila, (d) pogled na vozilo s prednje strane

Osim senzora, CARLA simulator pruža podatke o stanju vozila i njegovoj usklađenosti s okolinom i prometnim propisima. Mjerenja agenta prikazuju lokaciju vozila, brzinu, ubrzanje, zakret kuta upravljača vozila itd. Sva mjerenja imaju važnu ulogu u treniranju i evaluaciji vožnje. Simulator je napravljen u obliku skalabilne klijent – poslužitelj arhitekture. Poslužitelj je odgovoran za sve povezano sa samom simulacijom: generiranje objekata, repliciranje fizike krutih tijela (sudari, trenje između guma i podloge ceste, dinamika ovjesa vozila itd.), osvježavanje stanja okruženja i agenta. Klijentska strana se sastoji od velikog broja modula koji kontroliraju logiku agenta i scene, a uz to postavlja i uvjete okruženja. To je postignuto korisničkim CARLA aplikacijskim programskim sučeljima.

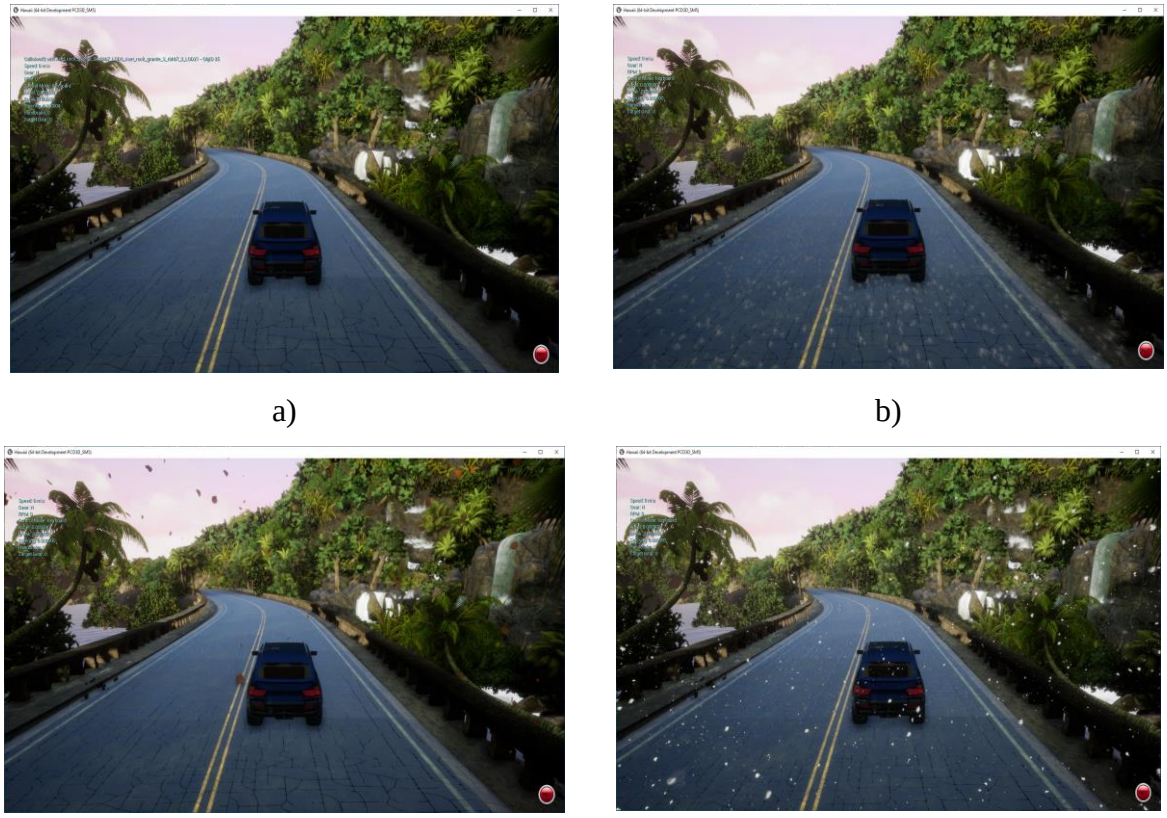
2.3.2. AirSim simulator

Microsoft AirSim je simulator otvorenog koda za autonomne sustave. Platforma je namijenjena za eksperimentiranje s rješenjima zasnovanim na umjetnoj inteligenciji, dubokim učenjem, računalnim vidom i podržanim učenjem. AirSim također izlaže API-je za dohvaćanje podataka iz simulacije i slanje upravljačkih naredbi za vozila u simulaciji. Napravljen je na *Unreal*

Engine platformi, koja pruža realistično okruženje u smislu fizičke i vizualne simulacije. Iako je prvo implementirana bespilotna letjelica kao promatrano autonomno vozilo, simulator je dizajniran od početka da bude proširiv na druge vrste vozila. S postojećom fizikom kretanja, simulator u novijim verzijama podržava i vožnju jedne vrste vozila. Primjenjivi su razni vremenski uvjeti. Svaki vremenski uvjet je definiran postotkom kojeg se prilagođava u vremenskom izborniku:

- kiša (0 % - 100%),
- vlažnost ceste (0 % - 100%),
- snijeg (0 % - 100%),
- količina snijega na cesti (0 % - 100%),
- padajuće lišće (0 % - 100%),
- količina lišća na cesti (0 % - 100%),
- prašina,
- magla.

Na slici 2.9 prikazan je primjer različitih vremenskih uvjeta u AirSim simulatoru.



a) b)

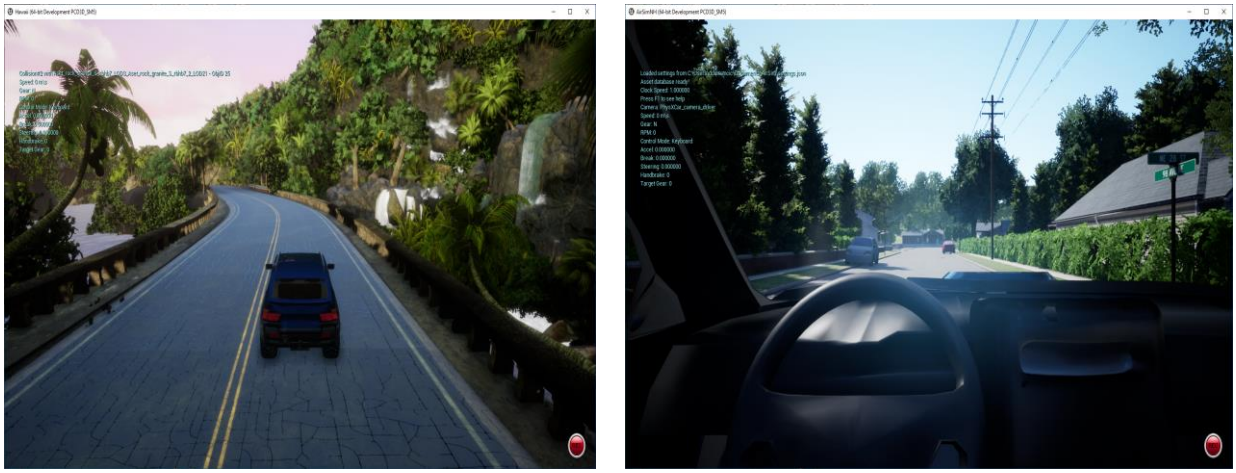
c) d)

Sl. 2.9. Primjeri različitih vremenskih uvjeta i osvjetljenja u AirSim simulatoru: (a) vrijeme bez padalina, (b) padanje kiše, (c) padanje lišća, (d) padanje snijega

Vremenski uvjeti se kombiniraju sa smjerom puhanja vjetra, čiji je vektor smjera definiran u koordinatnom sustavu (*x*,*y*,*z*). Valja napomenuti da se koordinatni sustav razlikuje od onog korištenog od strane *Unreal Enginea* (*z* koordinata suprotnog predznaka). Unutar konfiguracijske datoteke se navode željeni senzori, a podržava rad sljedećih senzora:

- kamera,
- barometar,
- jedinica za mjerenje inercije (engl. *inertial measurement unit - IMU*),
- globalni položajni sustav (engl. *global positioning system - GPS*),
- magnetometar,
- jedinica za mjerenje udaljenosti,
- LIDAR.

Što se tiče ugrađenih prikaza perspektive prilikom upravljanja, AirSim pruža pogled odozgo, pogled iz kokpita i ručno upravljanje pozicijom kamere putem tipkovnice. Primjer mogućih perspektiva kamere prikazan je na slici 2.10.



a) b)

Sl. 2.10. Primjeri različitih perspektiva kamere u AirSim simulatoru: (a) zadani pogled na vozilo odozgo, (b) pogled iz kokpita vozila

2.3.3. Usporedba analiziranih simulatora

U ovom dijelu uspoređene su prednosti i nedostaci koje pružaju analizirani simulatori, a zatim su detaljnije uspoređena dva odabrana simulatora: Carla i Airsim. Tablica 2.1 prikazuje usporedbu analiziranih simulatora.

Tablica 2.1 Osnovne karakteristike analiziranih simulatora

	Udacity [19]	CARSIM [21]	CARLA [17]	Gazebo [5]	AirSim [22]
Podržava samo kameru ili i dodatne senzore?	Samo kamera	Kamera i dodatni senzori	Kamera i dodatni senzori	Kamera i dodatni senzori	Kamera i dodatni senzori
Podržava različite vremenske uvjete?	Ne	Ne	Da	Ne	Da
Ugrađena prometna infrastruktura?	Da	Da	Da	Samo ručno stvaranje	Da
Različiti dinamički objekti?	Da	Da	Da	Ne	Da
Otvorenost koda	Da	Ne	Da	Da	Da

Odabrani Carla i Airsim simulatori su se pokazali najprikladniji za testiranje *PilotNet* mreže zbog svojih jedinstvenih funkcija koje nude za razvoj rješenja autonomne vožnje i problema kuta zakreta upravljača vozila. Među njima su mogućnost dohvaćanja slike s prednje kamere vozila i zapisivanje njihovih rezultata za daljnju obradu te mogućnost upravljanja vozilom slanjem upravljačkih naredbi putem API-ja. Iako oba simulatora podržavaju različite vremenske uvjete, CARLA simulator nema ugrađene uvjete za snijeg, što predstavlja veliki nedostatak za razvoj autonomne vožnje. Nasuprot tome, AirSim ima implementirane navedene uvjete. Međutim, nema mogućnost promjene doba dana i promjene postavke položaja sunca u odnosu na mapu, što onemogućuje vožnju u različitim uvjetima osvjetljenja (noćna vožnja, vožnja predvečer, itd.), a uz to se može dogoditi da se sjene na cesti generiraju stalno na istim mjestima. U AirSim simulatoru po zadanim postavkama postoji samo jedno vozilo za korištenje, dok je za korištenje drugih potrebno stvoriti vlastito u nekom od razvojnih okruženja, dok CARLA sadrži velik broj ugrađenih različitih vozila. Oba simulatora imaju mogućnost generiranja većeg broja vozila na mapi, ali AirSim ima nedostatak što nema ugrađeni autopilot poput CARLA simulatora. Na taj način je potrebno definirati ponašanja svih stvorenih vozila, dok u CARLA simulatoru ugrađeni autopilot radi samostalno bez dodavanja novih funkcionalnosti. Jedna od najvećih prednosti navedenih simulatora otvorenog koda je opsežna dokumentacija i Internet zajednice koje aktivno rade na razvoju simulatora. Oba simulatora su zasnovana na bazi *Unreal Enginea*, što je i jedan od razloga odabira ovih simulatora, sa svrhom pokretanja istih testnih scenarija u obama simulatorima.

3. IMPLEMENTACIJA ODABRANOG ALGORITMA AUTONOMNE VOŽNJE ZASNOVANOG NA PROCJENI KUTA ZAKRETA UPRAVLJAČA VOZILA U RAZLIČITIM SIMULATORIMA

U ovom poglavlju je opisana implementacija odabranog *PilotNet* algoritma u CARLA i AirSim simulatorima, s ciljem usporedbe mogućnosti koje pružaju odabrani simulatori. Potrebno je procijeniti kut zakreta upravljača vozila na temelju slike s vozila koju pružaju simulatori, a zatim i poslati procijenjeni kut zakreta u obliku upravljačke naredbe vozila. Opisani su alati koji su omogućili implementaciju algoritma, a zatim i sam proces implementacije algoritma u simulatore. U posljednjem potpoglavlju su dane upute za korištenje.

3.1. Opis alata korištenih pri implementaciji algoritma za upravljanjem modelom vozila zasnovanog na procjeni kuta zakreta upravljača vozila

3.1.1. Keras

Izvorni kod implementiranog *PilotNet* algoritma je napisan u *Python* programskom jeziku. Zbog svoje jednostavnosti koja omogućuje učinkovito istraživanje i izradu prototipova, za rad s umjetnim neuronskim mrežama dobro je koristiti biblioteku *Keras*. Služi kao programsko sučelje visoke razine za *Tensorflow2* biblioteku koja sadrži sve potrebne funkcije za duboko učenje, što uključuje rad s CNN. Time omogućuje jednostavno stvaranje i korištenje potrebnih slojeva, aktivacijskih funkcija, optimizacijskih funkcija i korištenje mnoštva alata za rad sa slikovnim i tekstualnim podacima. Ključne strukture podataka koje se koriste su model i sloj (engl. *layer*). Stvaranje modela se sastoji od nekoliko koraka:

1. Definiranje mreže: u ovom koraku, definiraju se različiti slojevi modela i veze između njih. Keras podržava sekvencijalnu i funkcionalnu vrstu modela.
2. Prevođenje mreže (engl. *Compile*): prijevod programskog koda u oblik razumljiv računalu. U Kerasu metoda *model.compile()* obavlja tu funkciju. Definira se kriterijska funkcija koja računa gubitke modela, funkcija optimizacije koja smanjuje gubitke mreže i metrika koja se koristi za određivanje točnosti ponašanja modela.
3. Prilagodba mreže: nakon što je model preveden, koristi se *model.fit()* funkcija za treniranje modela na određenom skupu podataka.
4. Provjera mreže: odnosi se na provjeru iznosa greške u modelu nakon prilagodbe.

5. Predviđanje mreže: koristi se *model.predict()* funkcija za procjenu izlazne vrijednosti modela na zadanom ulaznom podatku. Ulazni podatak u implementiranoj *PilotNet* mreži je slika s kamere vozila, dok je izlazna vrijednost kut zakreta upravljača vozila.

3.1.2. Unreal Engine 4

Unreal Engine 4 je 3D alat za izradu i razvoj ponajprije videoigara. Predstavljen je 2014. godine, a pri njegovom izlasku je bila potrebna komercijalna licenca za izradu projekta. Danas više nije potrebna i alat je u potpunosti besplatno dostupan. Napretkom industrije računalnih igara, virtualna okruženja su dosegla razinu visokog realizma. Platforma sadrži velik broj komponenata i popratnih alata koje dijele mnoge igre, a danas mnoge igre pokušavaju što vjernije prenijeti zakone fizike, mehanike kretanja i sami vizualni prikaz stvorenih objekata u svoje modele. Zbog svog širokog raspona mogućnosti pronašao je svoju primjenu i u simulatorima. Upravo CARLA i AirSim su zasnovani na *Unreal Engineu*, što pruža mogućnost pokretanja simulatora unutar istog razvojnog okruženja, a time i usklađivanje uvjeta za vožnju i pripadajućih virtualnih okoliša.

3.2. Tijek procesa implementacije algoritma za upravljanje modelom vozila zasnovanog na procjeni kuta zakreta upravljača vozila

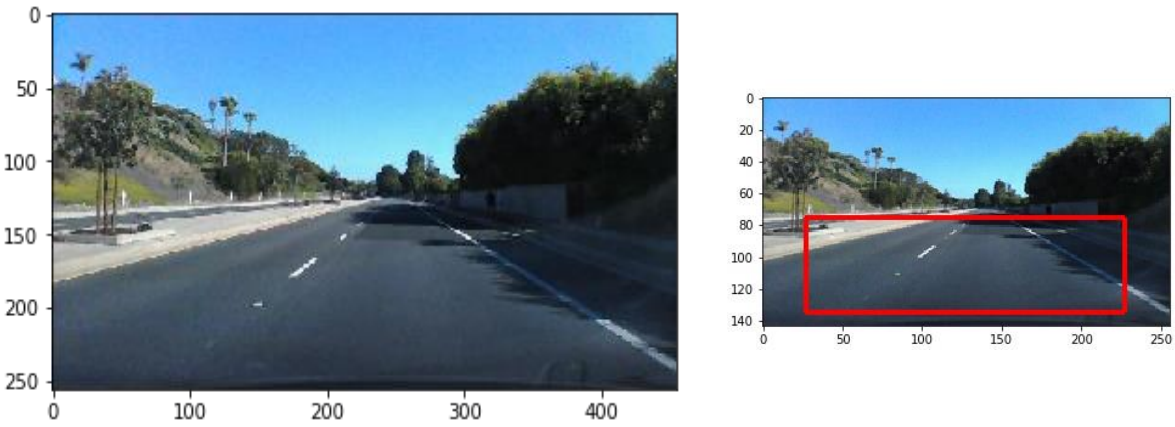
3.2.1. Opis korištenog skupa podataka i načina augmentacije skupa podataka

Za potrebe treniranja *PilotNet* mreže korišten je *Sully Chen driving* skup podataka otvorenog pristupa [9]. Sastoji se od 109394 slike prikupljene s kamere montirane na prednjoj strani vozila prilikom vožnje, uparenih s vremenski sinkroniziranim kutovima zakreta upravljača snimljenih od strane vozača na stvarnim prometnicama. Rezolucija izvornih slika iz skupa podataka je 455x256 elemenata slike. Skup podataka sadrži velik broj slika koje su nepotrebne za učenje mreže (npr. snimljene prilikom čekanja na semaforu) te kao takve nisu korištene prilikom treniranja. Time je smanjen skup podataka na 88458 slika. Prije samog treniranja slike su smanjene na veličinu 256x144 elemenata slike, korištenjem *resize()* funkcije koju pruža *OpenCV* biblioteka. Primjer programskog koda koji implementira smanjivanje rezolucije slike prikazan je na slici 3.1.

```
for image_name in image_names:
    im=cv2.imread(image_name)
    im=cv2.resize(im,(256,144),interpolation=cv2.INTER_AREA)
    imArr = np.asarray(im)
```

Sl. 3.1. Programskog kod korišten za smanjenje rezolucije ulazne slike

Korištena je *INTER_AREA* interpolacija koja uzima u obzir prostorne odnose elemenata slike, a osim toga kao ulaz u mrežu se koristi samo područje interesa (engl. *Region of interest - ROI*) veličine 200x66 elemenata slike. Izdvajanjem područja interesa iz slike smanjuje se količina podataka potrebna za treniranje modela, a time se smanjuje i vrijeme treniranja. Također će spriječiti da se model fokusira na nebitne značajke u okruženju (npr. planine, drveće itd.). Na slici 3.2 prikazan je primjer izvorne slike i smanjene slike s označenim područjem interesa iz skupa podataka. Crveni okvir prikazuje odabrano područje interesa.



a) b)

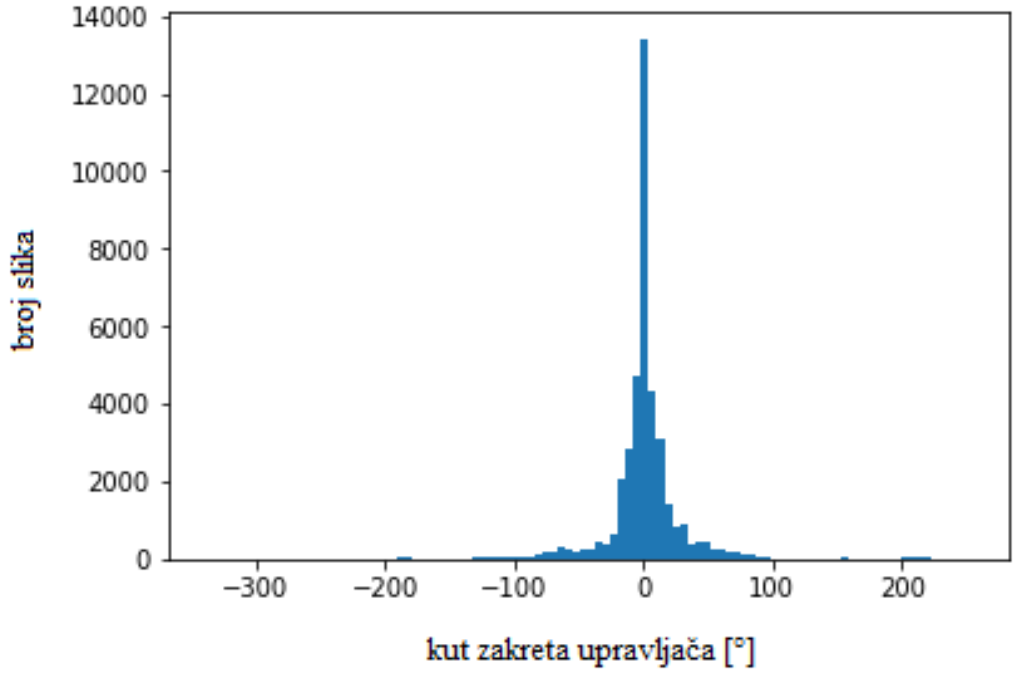
Sl. 3.2. Primjer slike iz skupa podataka: (a) izvorna slika, (b) smanjena slika s označenim područjem interesa

Oznake kutova zakreta upravljača su zapisane u *rec.txt* datoteci u obliku: naziv slike, zakret kuta upravljača (Slika 3.3). Izvorne oznake upravljača vozila su zapisane u rasponu [-250°,250°].

```
ImageName,Steering
  0.jpg,0.000000
  1.jpg,0.000000
  2.jpg,0.000000
  3.jpg,0.000000
  4.jpg,0.000000
  5.jpg,0.000000
  6.jpg,0.000000
  7.jpg,0.000000
  8.jpg,0.000000
  9.jpg,0.000000
 10.jpg,0.000000
...
```

Sl. 3.3. Isječak iz *rec.txt* datoteke u kojem su određenim slikama pridruženi pripadni kutovi zakreta upravljača vozila.

Na slici 3.4 prikazan je histogram distribucije kutova zakreta upravljača vozila za bazu podataka nakon odbacivanja nepotrebnih slika.



Sl. 3.4. Histogram distribucije kuta zakreta upravljača vozila

Vidljivo je da prevladavaju mali kutovi zakreta (blizu nule), jer se u većini vožnje prilikom prikupljanja baze podataka vozilo kretalo ravno. Model treniran na takvom skupu podataka niske standardne devijacije ne bi imao sposobnost korigiranja pogrešne putanje kretanja i svladavanja težih zavoja. Za bolje rezultate treniranja, potrebno je normalizirati zakrete upravljača vozila. Korištenjem funkcija koje pruža *pandas* biblioteka učitavaju se podaci razvrstani u stupce odvojene zarezom (Slika 3.3). Čitanjem svake linije tekstualne datoteke stvara se privremeni okvir podataka. Svaki kut zakreta upravljača se normalizira na vrijednost od 0 do 1, a zatim se zapisuju u rječnik s nazivom slike kao ključ elementa rječnika. Zakret 0 predstavlja maksimalni zakret u lijevo, zakret 1 predstavlja maksimalni zakret u desno.

Treniranje mreže s podacima samo stvarnog vozača za ovo rješenje nije dovoljno. Pri normalnoj vožnji, kut upravljanja gotovo je uvijek jednak nuli. Postoji velika neuravnoteženost u snimljenim podacima i trenirani model će uvijek predvidjeti nulu, a vozilo neće moći skrenuti. Od iznimne je važnosti da mreža nauči kako se oporaviti od pogrešaka. Ulazni skup podataka (88435 slika) je stoga proširen s dodatnim slikama koje nastoje prikazati vozilo u različitim pomacima od središta prometne trake i različite orijentacije u odnosu na cestu. Slike pomaka vozila u odnosu na

sredinu prometne trake mogu se usvojiti iz lijeve i desne kamere vozila, ali korišteni izvorni skup podataka sadrži slike samo jedne kamere iz središta vozila. Augmentacijom skupa podataka je nadomješten nedostatak bočnih kamera vozila. Ukoliko je izvorni kut zakreta upravljača manji od 30 stupnjeva, na uparenu sliku se primjenjuje postupak augmentacije. Kopira se, a zatim i izmjenjuje izvorna slika i pripadajući zakret upravljača. Korištenjem funkcija iz OpenCV biblioteke, za svaku takvu sliku se stvaraju četiri izmijenjene slike koje simuliraju promjenu točke gledišta kamere na cestu. Korištene su metode translacije slike s popunjavanjem elemenata slike i translacije slike bez popunjavanja. Svaka metoda stvara lijevu i desnu inačicu zakrenute slike s kamere vozila, a time se zapisuju i dvije nove oznake zakreta upravljača vozila (zakret u desno i zakret u lijevo).

Kod translacije slike s popunjavanjem elemenata slike, pseudo slučajni broj s pomičnim zarezom definira omjer translacije slike i utječe na iznos novih oznaka, što znači da će za veći omjer translacije biti zapisani veći kutovi zakreta upravljača vozila. Omjer definira horizontalni pomak lijevo i desno u odnosnu na ukupnu širinu slike. Nova oznaka desnog zakreta upravljača je definirana formulom (3-1), dok je nova oznaka lijevog zakreta upravljača definirana u formuli (3-2). U obje formule je pseudo slučajni broj koji definira omjer translacije u intervalu [0,0.6]. Interval je određen eksperimentalno.

$$nova\ oznaka\ 1\ = izvorna\ oznaka\ + \frac{1}{4}(slučajni\ broj * (2 * slučajni\ broj + 1) \quad (3\text{-}1)$$

$$nova\ oznaka\ 2\ = izvorna\ oznaka - \frac{1}{4}(slučajni\ broj * (2 * slučajni\ broj + 1) \quad (3\text{-}2)$$

U formuli (3-1) *nova oznaka 1* predstavlja zakret upravljača na koji je dodan iznos određen slučajnim brojem koji je ujedno i omjer translacije, a predstavlja zakret upravljača vozila u desno. *Nova oznaka 2 p*redstavlja zakret upravljača vozila umanjen za isti iznos i predstavlja zakret upravljača vozila u lijevo u odnosu na izvorni zakret upravljača vozila. *Izvorna oznaka* predstavlja zakret upravljača vozila izvorne slike iz skupa podataka. *Slučajni broj* stvara generator slučajnih varijabli u intervalu [0,0.6]. Programski kod za implementaciju translacije slike s popunjavanjem elemenata slike prikazana je na slici 3.5. Za generiranje slučajnih brojeva korištena je funkcija *random.uniform()* koja daje brojeve ravnomjerne distribucije iz zadanog intervala.

```
def fill(img, h, w): #popunjavanje slike
    img = cv2.resize(img, (455, 256), cv2.INTER_CUBIC)
    return img


def translation_fill_shift(img, ratio=0.0): #translacija
    h, w = img.shape[:2]
    to_shift = w*ratio
    if ratio > 0:
        img = img[:, :int(w-to_shift), :]
    if ratio < 0:
        img = img[:, int(-1*to_shift):, :]
    img = fill(img, h, w)
    return img

for i in range(1, current_df.shape[0] - 1):
    if (current_df.iloc[i-1]['Steering'] < 30.0) and
(current_df.iloc[i-1]['Steering'] > -30.0):
      rand=random.uniform(0,0.6) #definiranje omjera translacije slike
      ratio=(rand*(2*rand+1))/2

      shift_label_left=[steering_angle+ratio] #definiranje nove oznake
      shift_label_right=[steering_angle-ratio] #definiranje nove oznake

      shift_left=horizontal_shift(img,rand) #translacija izvorne slike
      (zakret lijevo)
      shift_right=horizontal_shift(img,-rand) #translacija izvorne
      slike (zakret desno)

      cv2.imwrite(shift_left_filepath,shift_left) #zapisivanje nove
      slike
      cv2.imwrite(shift_right_filepath,shift_right) #zapisivanje nove
      slike

      all_mappings[image_filepath] = (current_label) #mapiranje izvorne
      slike i oznake
      all_mappings[shift_left_filepath]=(shift_label_left) #mapiranje
      novonastale slike i oznake
      all_mappings[shift_right_filepath]=(shift_label_right) #mapiranje
      novonastale slike i oznake
```

Sl. 3.5. Primjer koda za implementaciju translacije slike s popunjavanjem elemenata slike

Translacijom slika s popunjavanjem elemenata slike, u skup podataka je dodano 155966 slika s pripadajućim vrijednostima zakreta upravljača vozila. Primjeri novonastalih slika i pripadajućih zakreta upravljača vozila prikazani su na slici 3.6. Prilikom translacije s popunjavanjem elemenata slike dolazi do geometrijskog izobličenja slike. Novonastale slike imitiraju slike s vozila koje je zakrenuto u odnosu na cestu.

Izvorna oznaka = 0.5 *Nova oznaka 1* = 0.8 *Nova oznaka 2* = 0.2

a) b) c)

Sl. 3.6. Primjer translacije slike s popunjavanjem elemenata slike: (a) izvorna slika, (b) lijeva perspektiva, (c) desna perspektiva

Korištenjem regije interesa ostaje po 28 elemenata slike sa svake strane izvorne slike (256 – 200 = 56, tj. 28 na lijevoj strani slike i 28 na desnoj), što omogućuje pomak slike za maksimalno isti broj elemenata slike bez geometrijskog izobličenja. Translacija slike bez popunjavanja elemenata slike koristi se kako bi novonastale slike imitirale slike prikupljene s vozila na određenoj udaljenosti od središnje linije. Kod translacije slike bez popunjavanja elemenata slike također je definiran pseudo slučajni broj, ali u rasponu [15, 28]. Taj broj definira broj elemenata slike za koji se slika pomiče u oba smjera. Nova oznaka koja predstavlja zakret u desno definirana je formulom (3-3), dok je nova oznaka zakreta u lijevo definirana formulom (3-4):

$$nova\ oznaka\ 1\ = izvorna\ oznaka\ + \frac{slučajni\ broj}{400} \tag{3-3}$$

$$nova\ oznaka\ 2\ = izvorna\ oznaka\ - \frac{slučajni\ broj}{400} \tag{3-4}$$

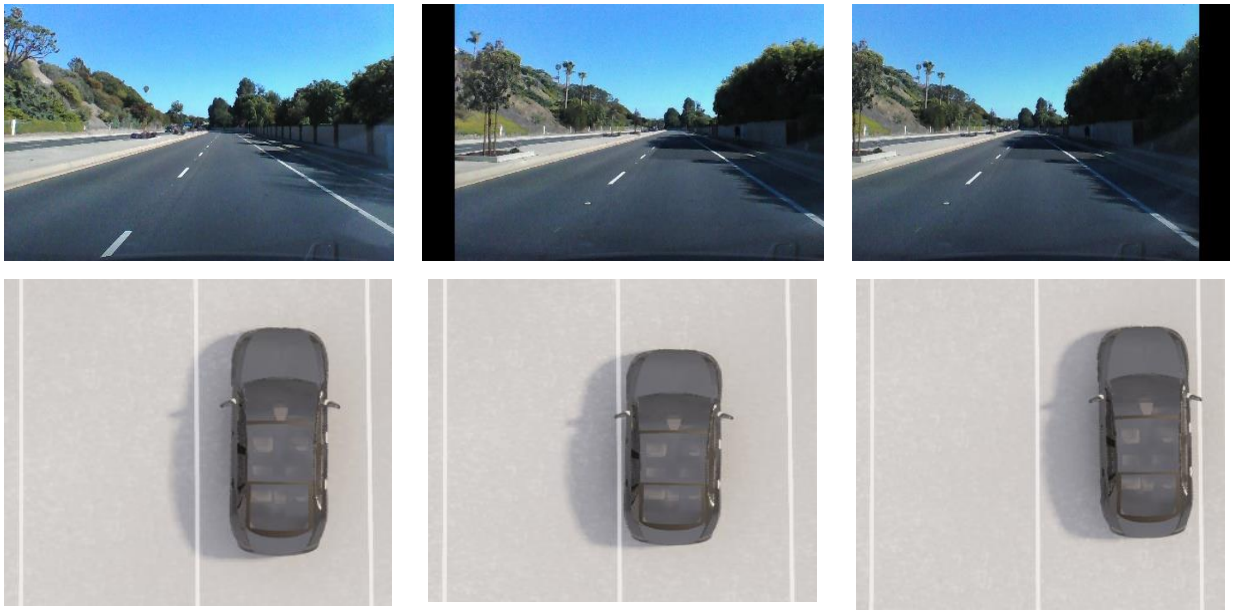
U formuli (3-3) *nova oznaka 1* predstavlja zakret upravljača vozila augmentirane slike u desno, dok *nova oznaka 2* u formuli (3-4) predstavlja zakret upravljača vozila u lijevo. *Izvorna oznaka* predstavlja zakret upravljača vozila izvorne slike iz skupa podataka. *Slučajni broj* je pseudo slučajni broj kojeg stvara generator slučajnih varijabli u intervalu [15,28].

Programski kod za implementaciju translacije slike bez popunjavanja elemenata prikazana je na slici 3.7.

```
def translation_shitf(image, ratio=0.0):
    height, width = image.shape[:2]
    random_bright=random.uniform(0.2,1)
    T = np.float32([[1, 0, ratio], [0, 1, 0]])
    img_translation = cv2.warpAffine(image, T, (width, height))
    return img_translation

for i in range(1, current_df.shape[0] - 1):
    if (current_df.iloc[i-1]['Steering'] < 30.0) and
(current_df.iloc[i-1]['Steering'] > -30.0):

      translation_offset=random.uniform(15,28) #definiranje pomaka
      translacije

      trans_label_left=[steering_angle+(translation_offset/400)]
      #definiranje nove oznake
      trans_label_right=[steering_angle-
      (translation_offset/400)]#definiranje nove oznake
      trans_left=translation_shitf(img,translation_offset)#translacija
      izvorne slike
      trans_right=translation_shitf(img,-
      translation_offset)#translacija izvorne slike

      cv2.imwrite(trans_left_filepath,trans_left)#zapisivanje nove
      slike
      cv2.imwrite(trans_right_filepath,trans_right)#zapisivanje nove
      slike

      all_mappings[trans_left_filepath]=(trans_label_left)#mapiranje
      novonastale slike i oznake
      all_mappings[trans_right_filepath]=(trans_label_right)#mapiranje
      novonastale slike i oznake
```

Sl. 3.7. Primjer koda za implementaciju translacije slike bez popunjavanja elemenata slike

Translacijom izvornih slika bez popunjavanja elemenata slike, skup podataka je proširen s još 155966 slika. Primjeri novih slika i pripadajućih kutova zakreta upravljača vozila prikazani su na slici 3.8.



Izvorna oznaka = 0.5 *Nova oznaka 1* = 0.57 *Nova oznaka 2* = 0.43

a) b) c)

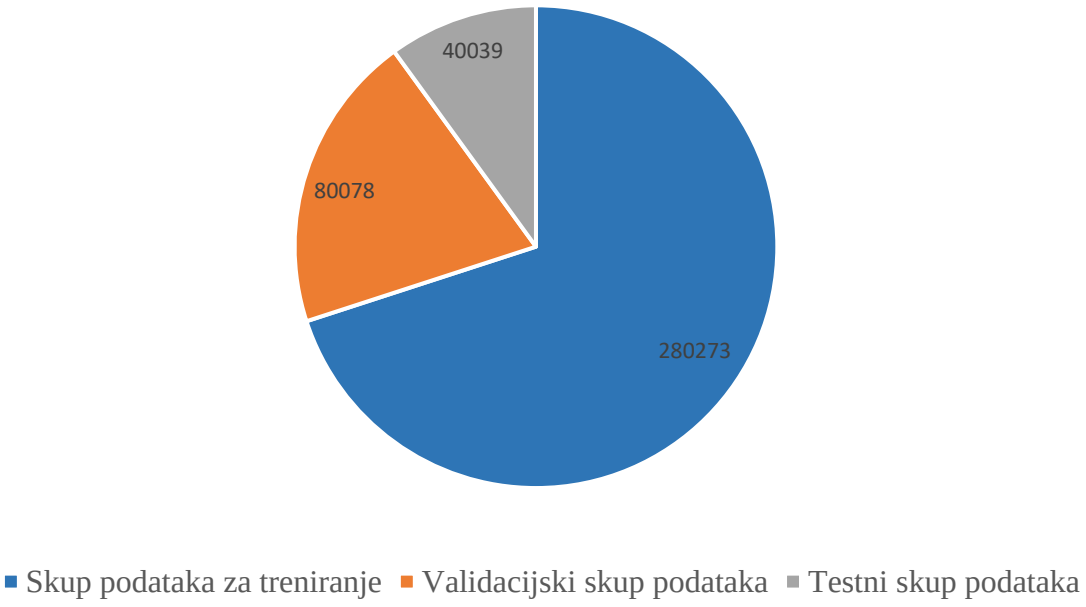
Sl. 3.8. Primjer translacije slike bez popunjavanja elemenata slike: (a) izvorna slika, (b) lijeva perspektiva, (c) desna perspektiva

Dodatno, ponašanje treniranog modela bi trebalo biti istoliko u različitim uvjetima osvjetljenja. U ranim eksperimentima se pokazalo da se vozilo različito pozicionira u prometnoj traci pri različitim uvjetima osvjetljenja. Kako bi se to što je moguće više i postiglo, uređene su slike na kojima je već primijenjena translacija bez popunjavanja elemenata slike. Učitavanjem RGB slike i pretvaranjem u HSV prostor boja, a zatim i promjenom V komponente, promijenjene su svjetline slika. Na slici 3.9 prikazan je primjer izvorne i zatamnjene slike. Skup podataka nije dalje proširivan kako se ne bi pojavljivali isti video okviri u skupu podataka za treniranje i skupu podataka za validaciju.

a) b)

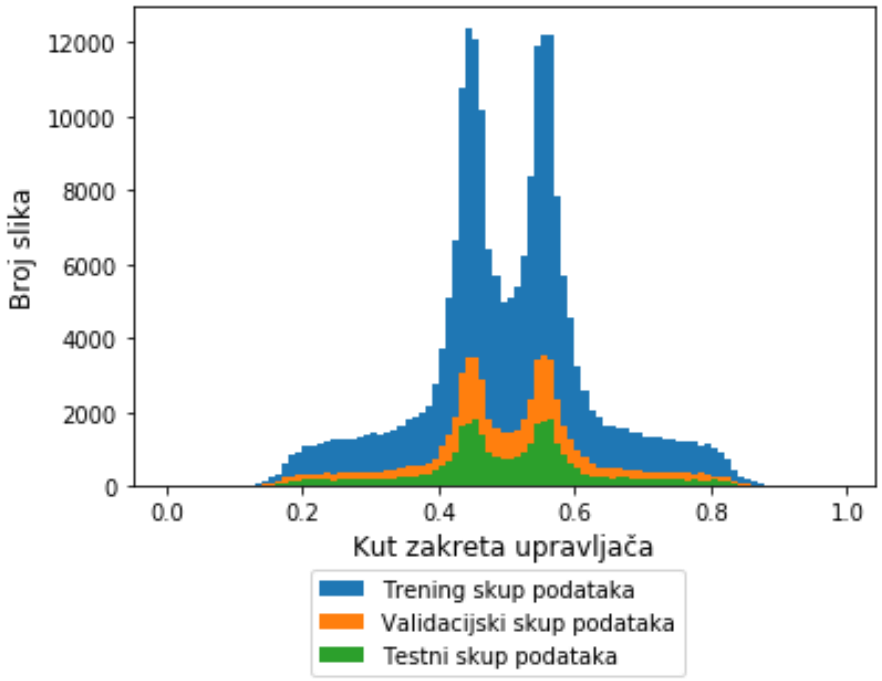
Sl. 3.9. Primjer zatamnjene slike: (a) izvorna slika, (b) uređena slika

Skup odabranih podataka je sadržavao 77983 slika za koje je zakret bio manji od 30 stupnjeva. Time je skup odabranih podataka (88458) proširen sa 155966 augmentiranih slika s popunjavanjem i 155966 augmentiranih slika bez popunjavanja elemenata slike. Zatim su uređene slike bez popunjavanja na način da je promijenjena izvorna svjetlina slika, bez dodavanja novih slika u skup podataka. Na kraju, korišteni skup podataka broji ukupno 400390 slika s popratnim kutovima zakreta upravljača vozila. Proširenjem skupa podataka uvedena je raznolikost podataka iz kojih model uči strategije vožnje, a rezultira boljim pozicioniranjem vozila u prometnoj traci samostalnim ispravljanjem pogrešnih putanja kretanja kroz zavoj. Nadalje, konačni skup podataka se sprema u komprimirani oblik najprikladniji za treniranje. Korišteni hijerarhijski format datoteke (engl. *hierarchical data format – HDF5*) .h5 je idealno rješenje za rad s velikom količinom podataka bez pisanja u memoriju odjednom. Konačni skup podataka je podijeljen na trening (*train.h5*), validacijski (*eval.h5*) i testni skup podataka (*test.h5*) u omjeru 70:20:10. Slika 3.10 prikazuje kružni graf s podjelom podataka u korištenom skupu.



Sl. 3.10. Grafički prikaz podjele podataka u korištenom skupu

Histogram kutova zakreta upravljača vozila za konačni skup podataka može se vidjeti na slici 3.11.



Sl. 3.11. Histogram distribucije oznaka zakreta upravljača vozila u konačnom skupu podataka

Sve datoteke se spremaju u direktorij *cooked_data*, a svaka .h5 datoteka se sastoji od 3 dijela:

- slika : *NumPy* polje koje sadrži podatke o slici,
- oznaka : *NumPy* polje podataka koje sadrži kutove zakreta upravljača vozila,
- metapodaci : *NumPy* polje podataka koje sadrži podatke o datotekama (iz kojeg direktorija dolaze itd.)

Konačni skup podataka dan je kao elektronički prilog P.3.1 ovog rada.

3.2.2. Opis postupka treniranja modela algoritma za autonomnu vožnju zasnovanog na procjeni kuta upravljača vozila

Ovaj dio sadrži detaljni opis implementacije *PilotNet* CNN arhitekture u Keras programskom sučelju. U radu sa slikovnim podacima, računalno je zahtjevno učitati cijeli skup podataka u memoriju. Međutim, kako bi bila što veća propusnost, Keras nudi koncept generatora slikovnih podataka (engl. *image data generator*) koji osigurava iterativno čitanje podataka s diska u dijelovima i generira nizove podataka s mogućnošću augmentacije u stvarnom vremenu. Premda sadrži standardne ugrađene transformacije za slike, klasa generatora podataka vožnje (engl. d*rive data generator*) proširuje ugrađene funkcije generatora slikovnih podataka kako bi bile korisne u primjeni stvaranja modela autonomne vožnje, npr. horizontalni okret slike u ugrađenoj funkciji generatora slikovnih podataka ne mijenja i predznak iznosa zakreta upravljača, dok generator

podataka vožnje mijenja. Dodijeljenom putanjom do *cooked_data* direktorija učitane su .h5 datoteke komprimiranih skupova podataka za treniranje i evaluaciju. Instancirani trening generator i evaluacijski generator su objekti klase generatora podataka vožnje, a skupovi podataka s pripadajućim oznakama su im delegirani kao parametri funkcije *flow()*. Fukncija *flow()* uzima polja slikovnih podataka uz pripadajuće oznake te stvara nizove podataka (engl. *Batch*). Izuzev skupa podataka, oznaka i veličine niza podataka, generatori mogu primati razne parametre poput zastavice treba li miješati podatke, putanje direktorija spremanja podataka, formata spremljenih podataka, postotka odbacivanja podataka s vrijednošću zakreta upravljača vozila 0 i koordinata pravokutnika koji predstavlja područje interesa. Kako je objašnjeno u dijelu 3.2.1., skup podataka je prethodno augmentiran, pa samim time nije dodatno proširen koristeći ugrađene metode koje pruža generator podataka vožnje. S obzirom na obradu izvornog skupa podataka i provedenu augmentaciju, konačni skup podataka je dobro balansiran, pa za treniranje modela nema ni odbačenih podataka s vrijednošću zakreta upravljača vozila 0. Prilikom inicijalizacije generatora podataka vožnje definirani su veličina niza podataka (engl. *Batch size*) koja prima 64 uparene slike s oznakama zakreta upravljača vozila i koordinate točaka pravokutnika područja interesa. Područje interesa je veličine 200x66 elemenata slike s definiranim točkama (*x1*=78, *x2*=143, *y1*=27, *y2*=226). Na slici 3.12 nalazi se isječak programskog koda za inicijalizaciju generatora podataka vožnje.

```
data_generator = DriveDataGenerator(rescale=1./255.)

train_generator = data_generator.flow\
(train_dataset['image'],train_dataset['label'],\
batch_size=batch_size,roi=[78,143,27,226])

eval_generator = data_generator.flow\
(eval_dataset['image'],eval_dataset['label'],batch_size=
batch_size, roi=[78,143,27,226])
```

Sl. 3.12. Inicijalizacija generatora podataka vožnje

Prije samog treniranja modela, arhitektura predložene *PilotNet* mreže je implementirana u Keras programskom sučelju (Slika 3.13).

```
image_input_shape = sample_batch_train_data[0].shape[1:]
pic_input = Input(shape=image_input_shape)

#Definiranje konvolucijske neuronske mreže
img_stack = Conv2D(24, (5, 5), name="conv1", strides=(2, 2),
padding="valid", activation=activation,
kernel_initializer="he_normal")(pic_input)
img_stack = Conv2D(36, (5, 5), name="conv2", strides=(2, 2),
padding="valid", activation=activation,
kernel_initializer="he_normal")(img_stack)
img_stack = Conv2D(48, (5, 5), name="conv3", strides=(2, 2),
padding="valid", activation=activation,
kernel_initializer="he_normal")(img_stack)

img_stack = Dropout(0.5)(img_stack)

img_stack = Conv2D(64, (3, 3), name="conv4", strides=(1, 1),
padding="valid", activation=activation,
kernel_initializer="he_normal")(img_stack)
img_stack = Conv2D(64, (3, 3), name="conv5", strides=(1, 1),
padding="valid", activation=activation,
kernel_initializer="he_normal")(img_stack)

img_stack = Flatten(name = 'flatten')(img_stack)

img_stack = Dense(100, name="fc2",
activation=activation,kernel_initializer="he_normal")(img_stack)
img_stack = Dense(50, name="fc3", activation=activation,
kernel_initializer="he_normal")(img_stack)
img_stack = Dense(10, name="fc4", activation=activation,
kernel_initializer="he_normal")(img_stack)
img_stack = Dense(1, name="output", activation = out_activation,
kernel_initializer="he_normal")(img_stack)

adam = Adam(lr=learning_rate, beta_1=0.9, beta_2=0.999, epsilon=1e-08,
decay=0.0)

model = Model(inputs=[pic_input], outputs=img_stack)
model.compile(optimizer=adam, loss='mse')
```

Sl. 3.13. Primjer definiranja *PilotNet* konvolucijske neuronske mreže u Keras programskom sučelju

Svaki konvolucijski sloj (u Kersu instanca klase Conv2D) predstavljen je brojem filtara, veličinom filtra, korakom pomjeranja filtra i vrstom popunjavanja (engl. *padding*). Ostatak dostupnih argumenata će poprimiti zadanu vrijednost (engl. *default)*. Instanca konvolucijskog sloja stvara konvolucijski kernel koji konvoluira s ulazom sloja kako bi proizveo izlazni tenzor. Objekt klase *Dense* generira uobičajeni potpuno povezani sloj neuronske mreže. Na svaki sloj mreže se

primjenjuje ReLU (engl. *Rectified Linear Unit*) aktivacijska funkcija, izuzev posljednjeg izlaznog sloja u kojem se primjenjuje sigmoidna aktivacijska funkcija koja osigurava izlaz u rasponu [0,1]. Osim već definiranih slojeva *PilotNet* arhitekture, važno je napomenuti da je implementirana *dropout* metoda regularizacije prilikom treninga mreže, kako bi se spriječilo pretjerano usklađivanje mreže na trening podatke. Nasumično odbacuje čvorove mreže zajedno s njegovim poveznicama nakon trećeg konvolucijskog sloja. Točnije, u svakoj iteraciji treniranja nasumično postavlja čvorove na vrijednost 0 s definiranim postotkom učestalosti. Oni čvorovi koji nisu postavljeni na 0 povećavaju se za 1/(1-stopa učestalosti) tako da je zbroj svih ulaza nepromijenjen. Ova efikasna tehnika regularizacije značajno smanjuje pretjerano usklađivanje na podatke (engl. *overfitting*) i pruža znatno poboljšane performanse konačnog modela. Tijekom prevođenja modela funkcijom *model.compile()* inicijalizirani su optimizator i kriterijska funkcija gubitaka (engl. *Loss function*). Optimizator implementira Adam algoritam, stohastičku gradijentnu metodu koja se zasniva na adaptivnoj procjeni stope učenja. Adam algoritam koristi procjenu momenata gradijenta prvog i drugog reda kako bi izračunala individualne stope učenja za različite parametre neuronske mreže. Računa stopu učenja na temelju zadanih argumenata funkcije, a zadana početna stopa učenja predstavlja gornju granicu, što implicira da svaka stopa učenja varira od 0 (nema ažuriranja) do zadane stope učenja (maksimalna stopa). Treniranje se može definirati i kao postupak minimiziranja kriterijske funkcije koja evaluira skup parametara treniranog modela. Iznos kriterijske funkcije za implementirani model je definiran kao srednja kvadratna pogreška (engl. *Mean squared error – MSE*) između zakreta upravljača vozila kojeg daje trenirani model i stvarnog (referentnog, poznatog) zakreta dobivenog vožnjom vozača koji je zabilježen u skupu podataka za treniranje.

Za navedeni model su definirani sljedeći hiperparametri:

- stopa učenja = 0.001,
- veličina niza podataka (enlg. *batch size*) = 64
- broj epoha = 200

Međutim, nisu svi konstantni. Korištenjem funkcija povratnih poziva (engl. *Callback*) hiperparametri se prilagođavaju tijekom procesa treniranja. Definirani su:

- Smanjivanje stope učenja – *ReduceLROnPlateau()* povratni poziv smanjuje stopu učenja ukoliko se iznos pogreške na validacijskom skupu podataka ne smanjuje u definiranom broju epoha.

- Ranije zaustavljanje treniranja – usprkos činjenici da je definiran veliki broj epoha, kako bi bila smanjena mogućnost pretjeranog usklađivanja na podatke, *EarlyStopping()* povratni poziv zaustavlja treniranje ukoliko se iznos pogreške na validacijskom skupu podataka počne povećavati.

Definirani su i dodatni povratni pozivi za bolje razumijevanje i praćenja procesa treniranja poput:

- Spremanja kontrolnih modela u nekom intervalu – *ModelCheckpoint()*.
- Bilježenja napretka modela nakon svake epohe – *CSVLogger()*.

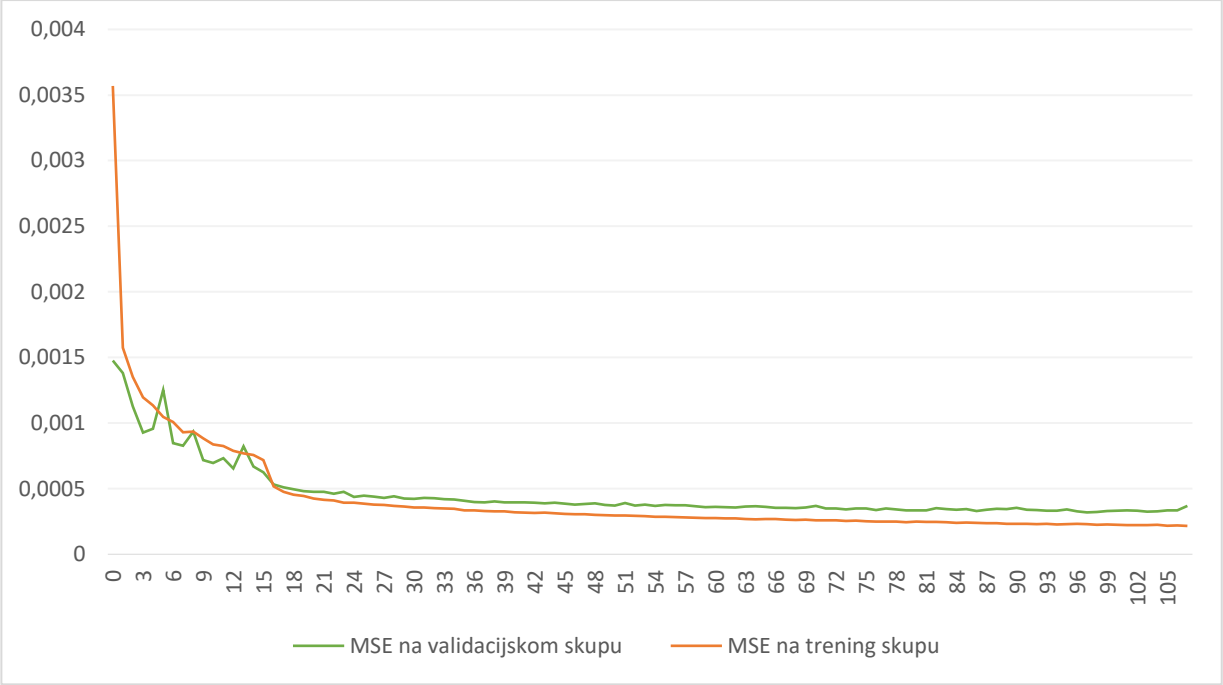
Primjeri programskog koda u kojem se definiraju funkcije povratnih poziva prikazan je na slici 3.14.

```
plateau_callback = ReduceLROnPlateau(monitor='val_loss', factor=0.1,
patience=5, min_lr=0.0001, verbose=1)

checkpoint_filepath = os.path.join(MODEL_OUTPUT_DIR, 'models',
'{0}_model.{1}-{2}.h5'.format('model', '{epoch:02d}',
'{val_loss:.7f}'))

checkAndCreateDir(checkpoint_filepath)

checkpoint_callback = ModelCheckpoint(checkpoint_filepath,
save_best_only=True, verbose=1)

csv_callback = CSVLogger(os.path.join(MODEL_OUTPUT_DIR,
'training_log.csv'))

early_stopping_callback = EarlyStopping(monitor='val_loss',
patience=10, verbose=1)

callbacks=[plateau_callback, csv_callback, checkpoint_callback,
early_stopping_callback, TQDMNotebookCallback(),TensorBoard('logs')]
```

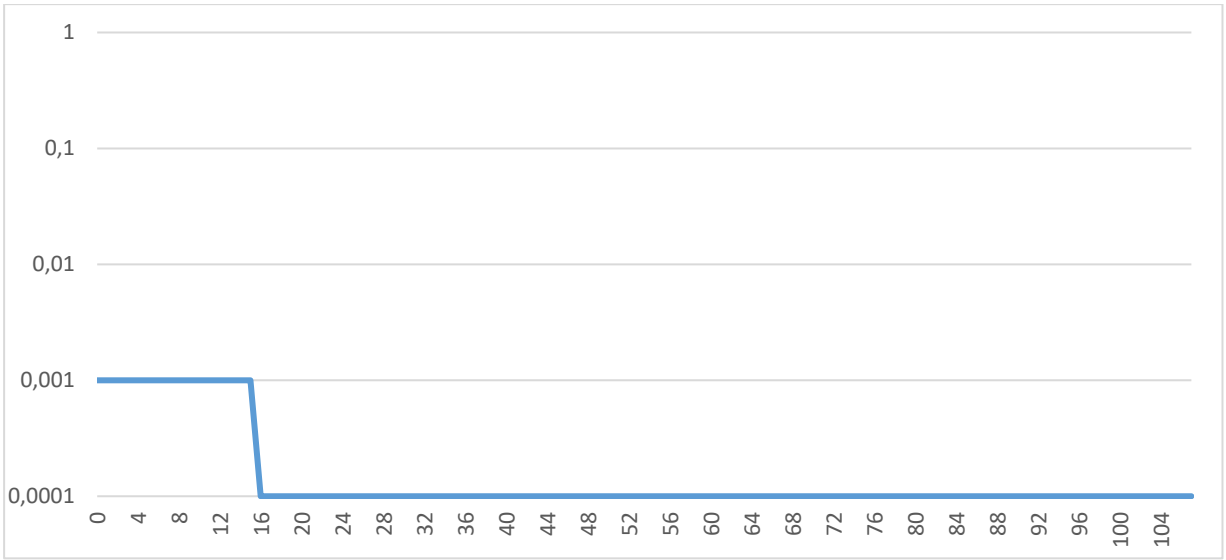
Sl. 3.14. Primjeri definiranja funkcija povratnih poziva prilikom treniranja mreže

Pozivanjem funkcije *model.fit()* je izvršeno treniranje. Automatski je zaustavljeno nakon 107 epoha i vremenskog trajanju od 79 sati, kada je srednja kvadratna pogreška na validacijskom skupu podataka počela pokazivati tendenciju rasta, tj. pretjerano usklađivanje modela na skup podataka za učenje. Model spremljen u 97. epohi daje najmanju srednju kvadratnu pogrešku na validacijskom skupu podataka i samim time je korišten kao konačni model. Upravo će taj model biti testiran na testnom skupu podataka. Na slici 3.16 prikazan je graf promjene kriterijske funkcije

kroz epohe, a na slici 3.15 graf promjene stope učenja kroz epohe. Zelenom bojom označena je promjena kriterijske funkcije na validacijskim podacima, a narančastom bojom na trening podacima.



Sl. 3.15. Graf promjene kriterijske funkcije (MSE) kroz epohe treninga mreže

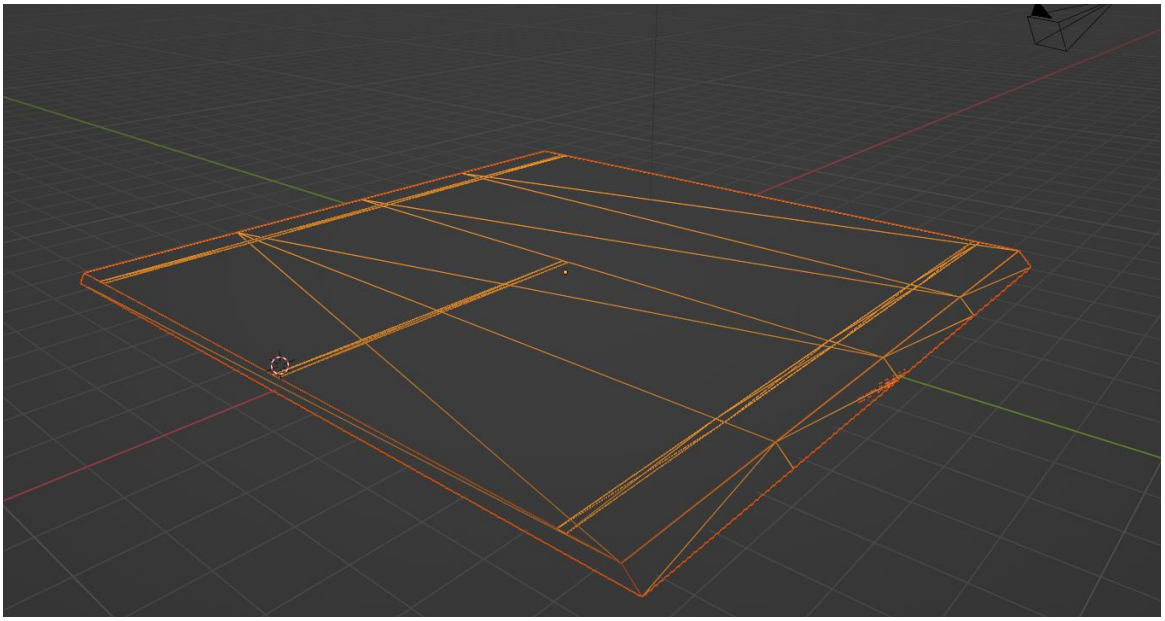


Sl. 3.16. Graf promjene stope učenja kroz epohe treninga mreže

Spremljeni model je dan kao elektronički prilog P.3.2 ovog rada.

3.2.3. Opis postupka stvaranja testnih sekvenci u Unreal Engine 4 razvojnom okruženju

Naučenu *PilotNet* mrežu potrebno je testirati u odabranim simulatorima koristeći raznolike postavke scene, npr. različite konfiguracije ceste, različita osvjetljenja i vremenske uvjete. Međutim, kako bi bilo moguće što pouzdanije usporediti rezultate koje model postiže u obama simulatorima, ključno je u maksimalnoj mogućoj mjeri uskladiti uvjete testiranja. Svaki simulator posjeduje različite implementacije vremenskih uvjeta i veći broj različitih testnih scena. Stoga su dizajnirane testne sekvence primjenjive u obama simulatorima. Kao osnovni element modeliran je jedan segment ceste koristeći *Blender* alat (Slika 3.17).

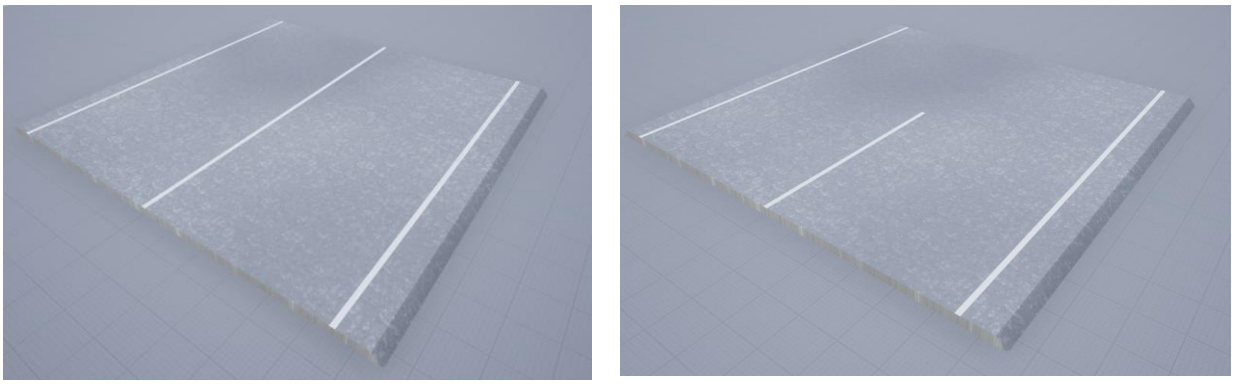


Sl. 3.17. Primjer primitivne mreže modela segmenta ceste u Blender alatu

Prva faza se sastoji od povezivanja primitivne mreže (engl. *mesh*) te se oblikuje izgled osnovnog modela. Segment ceste se sastoji od obrisa linija i rubova ceste, a na njih se zatim dodaju zadane (engl. *default*) teksture. Definirane teksture nemaju specifična svojstva, tek nakon uvoza komada ceste u *Unreal Engine* projekt CARLA simulatora, na mjestu definiranih tekstura su primijenjeni različiti materijali iz resursa koje pruža simulator. Materijal iz *Unreal Engine* razvojnog okruženja je sredstvo koje se može primijeniti na primitivnu mrežu za promjenu vizualnog izgleda objekta. Na visokoj razini, vjerojatno je najlakše zamisliti materijal kao "boju" koja se nanosi na objekt. Ali čak i to može dovesti u zabludu, budući da materijal doslovno definira vrstu površine od koje

se čini da je objekt napravljen. Moguće je definirati njegovu boju, koliko je sjajan, proziran i još mnogo toga. Korištenjem materijala (beton i bijela linijska oznaka) formirane su dvije vrste cesta:

- Cesta s punom bijelom razdjelnom crtom (Slika 3.18 (a))
- Cesta s isprekidanom bijelom razdjelnom crtom (Slika 3.18 (b))



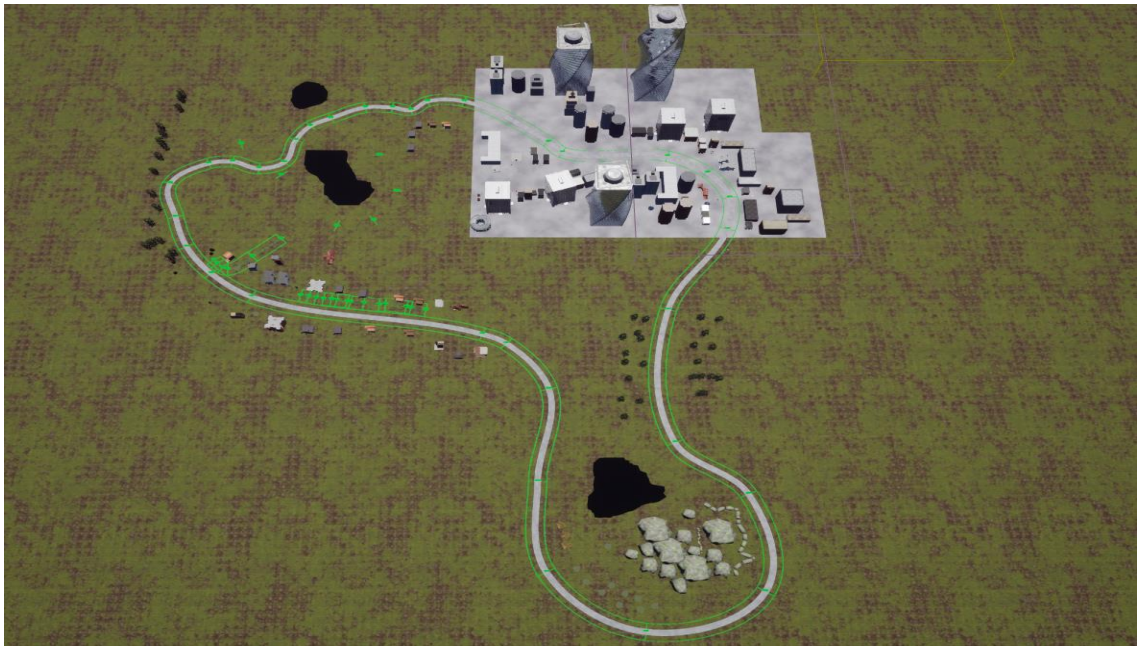
a) b)

Sl. 3.18. Prikaz uvezenih segmenata ceste u *Unreal Engineu*: (a) cesta s punom bijelom razdjelnom crtom, (b) cesta s isprekidanom bijelom razdjelnom crtom

U *Unreal Engine-u* moguće je stvarati virtualni okoliš velikih dimenzija koristeći alat *landscape* za uređivanje terena i reljefa. Dodavanjem početnog sloja stvorena je podloga na koju se dodaje dizajnirana cesta zajedno s već ugrađenim dostupnim objektima koje pružaju simulatori (npr. rasvjetni stupovi, ograde, zgrade, znakovi, itd.). U postavkama podloge dodani su materijali trave i pločnika kako bi što vjernije dočarali vangradski i gradski okoliš. Oblik ceste definiran je fleksibilnom krivuljom (engl. *Spline*) koja je fiksirana na odabranim kontrolnim točkama. Kontrolne točke su ručno postavljene, dok ih razvojno okruženje automatski spaja mrežom (engl. *Mesh*) kojoj se u postavkama pridodaje dizajnirani izgled ceste. Završno su dodani objekti kojima je cilj povećati realnost okruženja poput zgrada, kuća, raslinja, rasvjetnih stupova, ograda pored ceste itd. Korištenjem navedenih metoda za modeliranje virtualnog okoliša načinjene su tri različite kružne staze na kojima je izvršeno testiranje treniranog *PilotNet* algoritma za autonomnu vožnju:

- Mapa Test1 (Slika 3.19)
- Mapa Test2 (Slika 3.20)
- Mapa Test3 (Slika 3.21)

Sl. 3.19. Izgled mape Test1 načinjene u *Unreal Engine* alatu za testiranje implementiranog algoritma autonomne vožnje u obama simulatorima



Sl. 3.20. Izgled mape Test2 načinjene u *Unreal Engine* alatu za testiranje implementiranog algoritma autonomne vožnje u obama simulatorima

Sl. 3.21. Izgled mape Test3 načinjene u *Unreal Engine* alatu za testiranje implementiranog algoritma autonomne vožnje u obama simulatorima

Sve ceste sadrže dvije prometne trake. U tablici 3.1 nalaze se osnovne dimenzije prikazanih cesta.

Tablica 3.1. Dimenzije cestovnih staza za tri testne mape

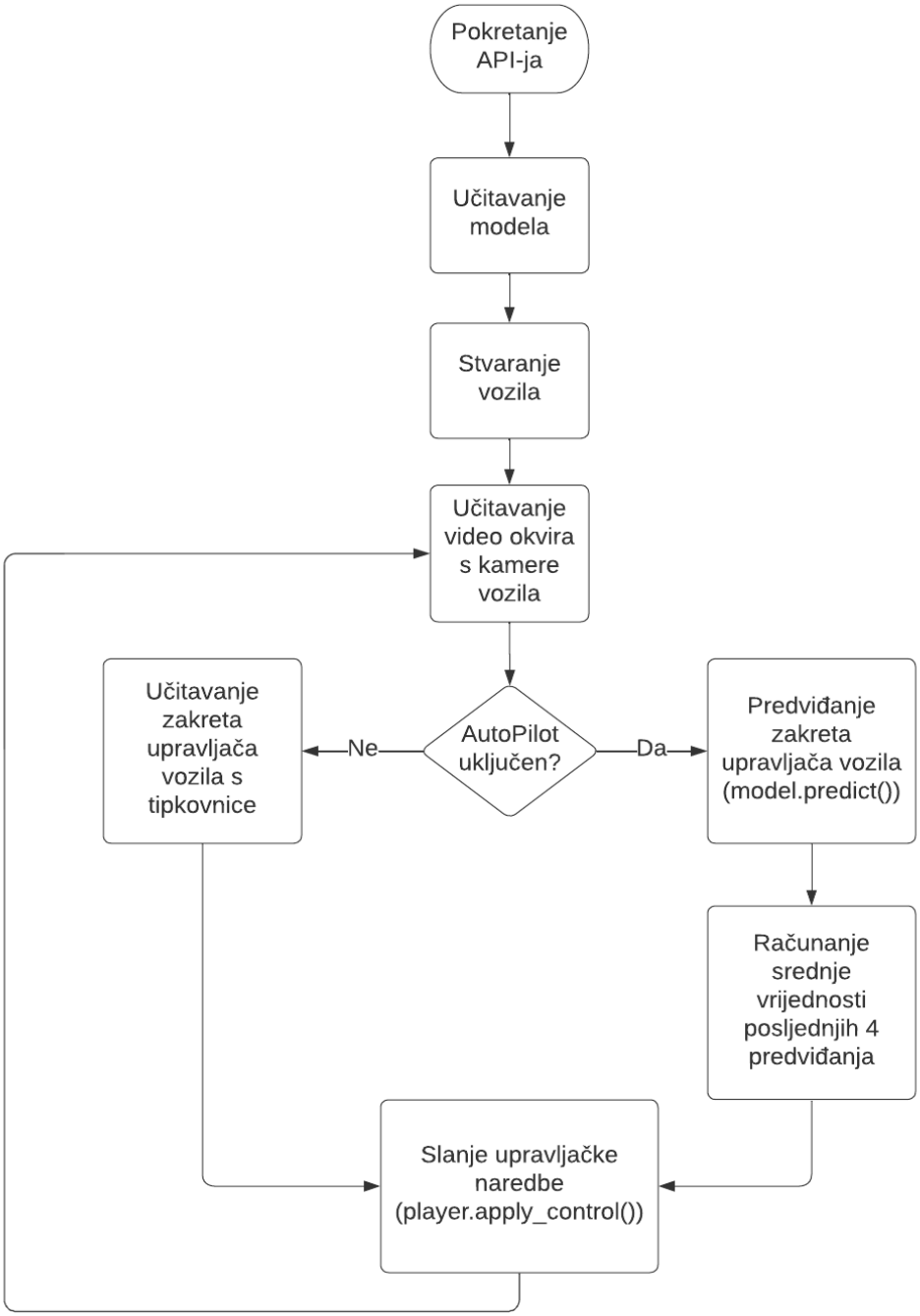
Naziv mape	**Ukupna duljina staze**	**Širina vozne trake**
Test1	3.14 km	3.3 m
Test2	2.7 km	3 m
Test3	1.8 km	4 m

Kreirane mape su dane kao elektronički prilog P.3.3, P.3.4 i P.3.5 ovog rada za korištenje u CARLA simulatoru. Kreirane mape za korištenje u AirSim simulatoru dane su kao elektronički prilog P.3.6, P.3.7 i P.3.8.

3.2.4. Implementacija algoritma za upravljanje modelom autonomnog vozila zasnovanog na procjeni kuta zakreta upravljača vozila u CARLA simulatoru

Kao baza za implementaciju algoritma koji upravlja modelom je API otvorenog koda s nazivom *manual_control.py*. Nalazi se u direktoriju s primjerima programskog koda koje pruža CARLA simulator, a omogućuje upravljanje vozilom koristeći tipkovnicu računala. Upravljanje

vozilom se izvršava na klijentskoj strani simulatora. Zadržane su sve funkcije koje pruža izvorni API uz potrebne nadogradnje. Dodane su funkcije potrebne za procjenu zakreta upravljača vozila na temelju slike s prednje kamere vozila i funkcije za upravljanje vozilom. Na slici 3.22 prikazan je dijagram toka *drive_model_carla.py* algoritma za upravljanje vozilom u CARLA simulatoru.



Sl. 3.22. Dijagram toka upravljačkog algoritma za CARLA simulator

Učitavanje odabranog modela se izvodi funkcijom *load_model()* koju pruža *Keras*, a prima putanju do spremljenog modela. Biblioteka *carla* pruža sve potrebne funkcije za konfiguraciju simulatora i upravljanje vozilom. Unutar klase *KeyboardControl*, koja rukuje s ulazima tipkovnice, dodan je uvjet unutar funkcije koja se bavi raščlanjivanjem aktiviranih akcija. Pritiskom na tipku „e“ mijenja se stanje varijable *autopilot*, što označava treba li model svojom procjenom upravljati vozilom ili to radi korisnik ručno putem tipkovnice (Slika 3.23). Prilikom pokretanja simulatora, autonomna vožnja je isključena, a prvim pokretanjem započinje mjerenje vremena i prijeđenog puta vozila, što je potrebno u kasnijim fazama testiranja.

```
elif event.key == K_e:

                    if (world.autopilot==False):
                        world.hud.notification("PilotNet On")
                        world.autopilot=True

                        if(self.human_intervention_count==0):
                            self.test_start= time.time()
                            _distance = 0
                        self.intervention_end=time.time()

                    else:
                        self.human_intervention_count=
self.human_intervention_count + 1
                        world.hud.notification("PilotNet Off")
                        world.autopilot=False
                        self.intervention_start=time.time()
```

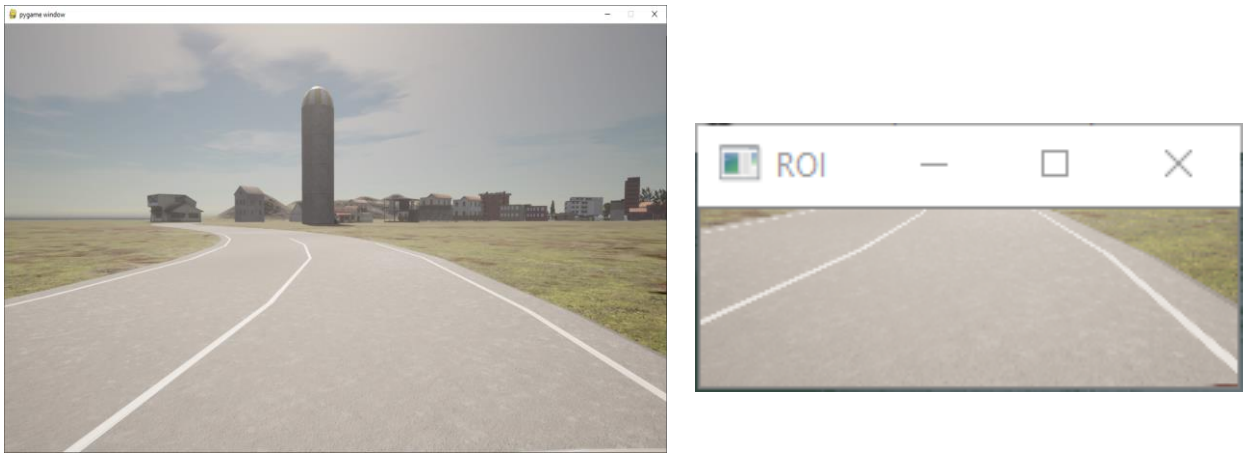
Sl. 3.23. Primjer programskog koda za uključivanje i isključivanje autonomne vožnje

Kamere i senzori su dodani definiranjem u postavkama klijenta koje se šalju prilikom svakog pokretanja. Postavke kamere je moguće modificirati popunjavanjem *CameraManager* Python klase ili učitavanjem INI konfiguracijske datoteke. Na slici 3.24 prikazani su korišteni definirani parametri kamere iz *CameraManagera*.

```
if not self._parent.type_id.startswith("walker.pedestrian"):
            self._camera_transforms = [
                (carla.Transform(carla.Location(x=1.9, y=-0.3, z=1)
,carla.Rotation(pitch=-2.85,yaw=-7)), Attachment.Rigid)
]
#parametri kamere
```

Sl. 3.24. Primjer programskog koda za uređivanje parametara kamere

Transform klasa predstavlja kombinaciju lokacije i rotacije bez skaliranja. Lokacija opisuje točku u koordinatnom sustavu cjelovitog virtualnog okoliša. Opis rotacije je u odnosu na objekt prema sustavu osi *Unreal Enginea*. Slike s kamere se šalju od strane servera kao instance klase *carla.Image* koja sadrži atribute horizontalnog vidnog polja (engl. *Field of view*) u stupnjevima, visinu slike, širinu slike i neobrađene podatke niza bajtova koji predstavljaju 32-bitnu BGRA sliku. Na dohvaćenoj slici se otklanja alfa kanal koji predstavlja stupanj prozirnosti boje. Takva slika izvorne rezolucije 1280x720 elemenata slike se ispisuje na *Pygame* platformu, odnosno interaktivni zaslon simulatora (Slika 3.25 (a)). Nadalje, slika iz simulatora se sprema u *numpy* polje podataka te se smanjuje na rezoluciju 256x144. U međuspremnik (engl. *Buffer*) se spremaju elementi slike koji su određeni definiranim područjem interesa (200x66) (Slika 3.25 (b)) i kao takvi se predaju CNN modelu kao ulazni podatak za koji se vrši procjena zakreta upravljača vozila. Područje interesa se prikazuje na zaslon korištenjem funkcije *cv2.imshow()*. Također, izvršena je normalizacija elemenata slike koji se predaje mreži iz cjelobrojnog raspona [0,255] u broj s pomičnim zarezom u rasponu [0,1].



a) b)

Sl. 3.25. Primjer (a) slike koja se ispisuje na interaktivni zaslon, (b) slike koja se predaje *PilotNet* mreži za procjenu zakreta upravljača vozila

Programski kod za dohvaćanje slike s kamere vozila koje pruža CARLA simulator prikazan je na slici 3.26.

```
image.convert(self.sensors[self.index][1])
array = np.frombuffer(image.raw_data, dtype=np.dtype("uint8"))
array = np.reshape(array, (image.height, image.width, 4))
array = array[:, :, :3]
surfacearray=array[:,:,::-1]
array = array[::5, ::5, ::-1]
self.surface = pygame.surfarray.make_surface(surfacearray.swapaxes(0,
1))
image_buf[0]=array[78:144,27:227,0:3].astype(float)
image_buf[0] /= 255
```

Sl. 3.26. Primjer programskog koda za dohvaćanje i obradu slike s kamere vozila

Slika iz međuspremnika predaje se funkciji *model.predict()* koja vraća procjenu zakreta upravljača vozila kao oblik broja s pomičnim zarezom (engl. *Float*) u rasponu [0-1]. CARLA simulator iznos zakreta upravljača definira u rasponu [-1,1], a prilikom pretvorbe u traženi raspon iznos procjene se množi s kalibracijskim koeficijentom 0.68:

$$zakret\ upravljača\ vozila\ = 0.68 * (procjena\ modela * 2 - 1) \quad (3\text{-}3)$$

Koeficijent kompenzira različite karakteristike vozila iz simulatora u odnosu na vozilo s kojim je sniman skup podataka. Postoje razlike i između samih vozila unutar simulatora: stvarni zakret u simulatoru ovisi o korištenom vozilu, odnosno o maksimalnom zakretu kotača, promjeru kotača i stupnju prigušenja (engl. *Damping rate*). Kalibracijski koeficijent je aproksimiran eksperimentalnom metodom u simulatoru za vozilo Mercedes Coupe 2020. Izračunati zakret upravljača vozila se sprema u red (engl. *Queue*) koji pohranjuje posljednje četiri procjene. Kao i u slučaju kalibracijskog koeficijenta, broj procjena koje se spremaju u red je određen eksperimentalnom metodom na kontrolnoj mapi koja sadrži jednu ravnu dionicu, jedan blagi zavoj i jedan oštri zavoj. Novi podaci zakreta upravljača zapisuju se na začelje, dok se brišu stari podaci s čela liste (engl. *First in, first out – FIFO).* Računa se prosječna vrijednost reda i predaje metodi *Carla.Vehicle.apply_control()* koja primjenjuje upravljačku naredbu za kretnju vozila u sljedećem vremenskom okviru. Usrednjavanjem vrijednosti zakreta upravljača vozilo uglađenije prolazi zadane zavoje i smanjuje neželjene lateralne oscilacije na ravnim dionicama. Na slici 3.27 vidljiv je graf koji prikazuje usporedbu procijenjene vrijednosti zakreta upravljača vozila i implementirane usrednjene vrijednosti zakreta upravljača pri prolasku jednog zavoja ceste po broju okvira slika.

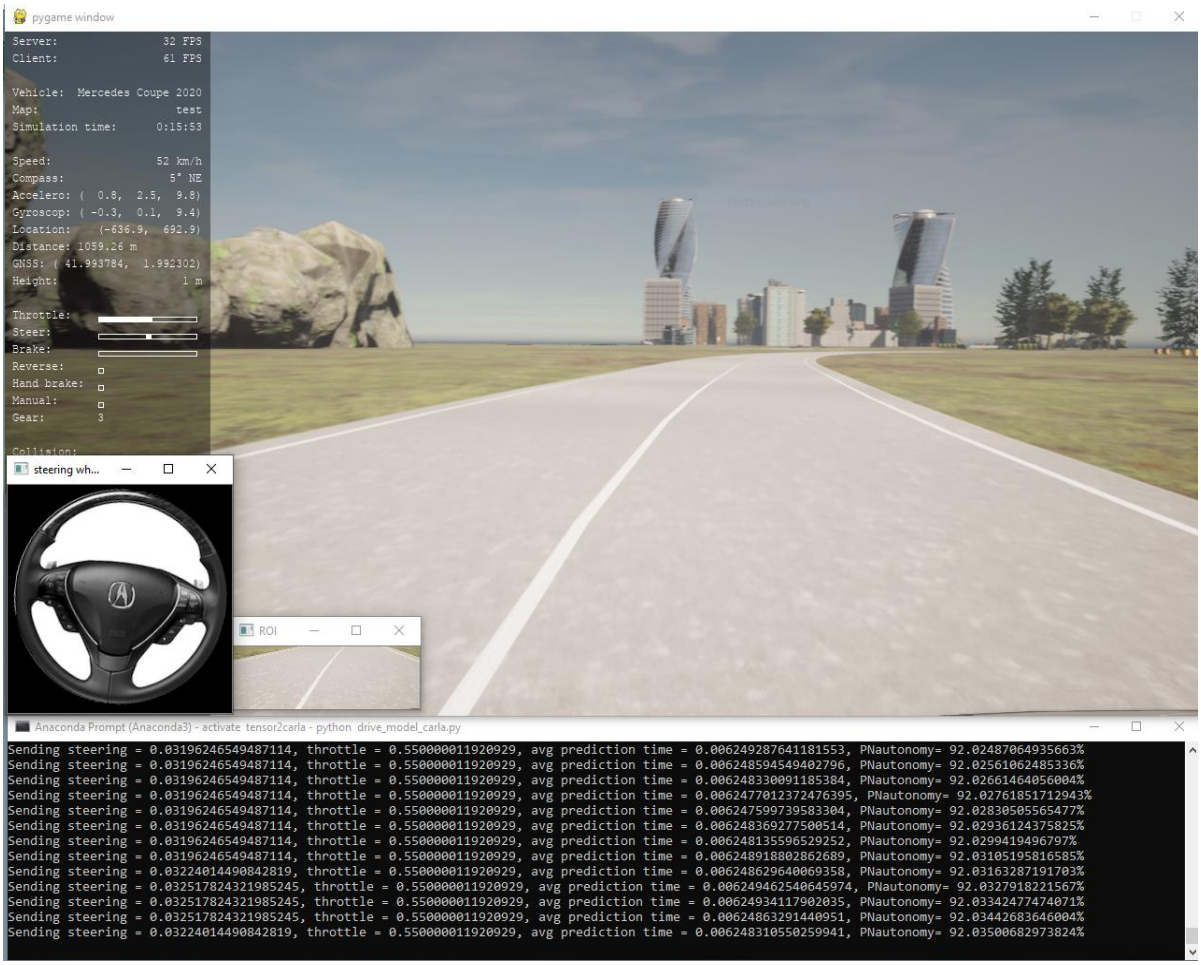
Sl. 3.27. Graf procijenjene vrijednosti zakreta upravljača vozila *PilotNet* mreže i implementirane usrednjene vrijednosti zakreta upravljača pri prolasku jednog zavoja ceste po broju okvira slika

Implementacija najvažnijih dijelova fukncije *_auto_drive()* koja rukuje predviđanjem modela i upravljanjem zakretom upravljača vozila u simulatoru prikazana je na slici 3.28.

```
def _auto_drive(self):
    model_output = model.predict([image_buf]) #procjena zakreta
upravljača vozila na temelju slike iz međuspremnika

    q1.get() #odbacivanje elementa iz reda
    q1.put(0.68*float((model_output[0][0]*2.0)-1)) #pretvorba i
dodavanje novog elementa u red

    received_output=KeyboardControl.queue_ave(q1) #računanje srednje
vrijednosti reda

    self._control.steer =received_output #definiranje zakreta
upravljača vozila
    world.player.apply_control(self._control)#slanje upravljačke
naredbe simulatoru
```

Sl. 3.28. Primjer programskog koda za upravljanje zakretom upravljača vozila u CARLA simulatoru

Uz navedene funkcionalnosti, skripta *drive_model_carla.py* u konzolu ispisuje poslane naredbe za upravljanje vozilom, prosječno trajanje procjene modela, autonomiju vozila i prijeđeni put u metrima. Uz grafičko sučelje simulatora predočeni su i ilustracija kuta zakreta upravljača vozila te prikaz područja interesa u stvarnom vremenu (Slika 3.29).

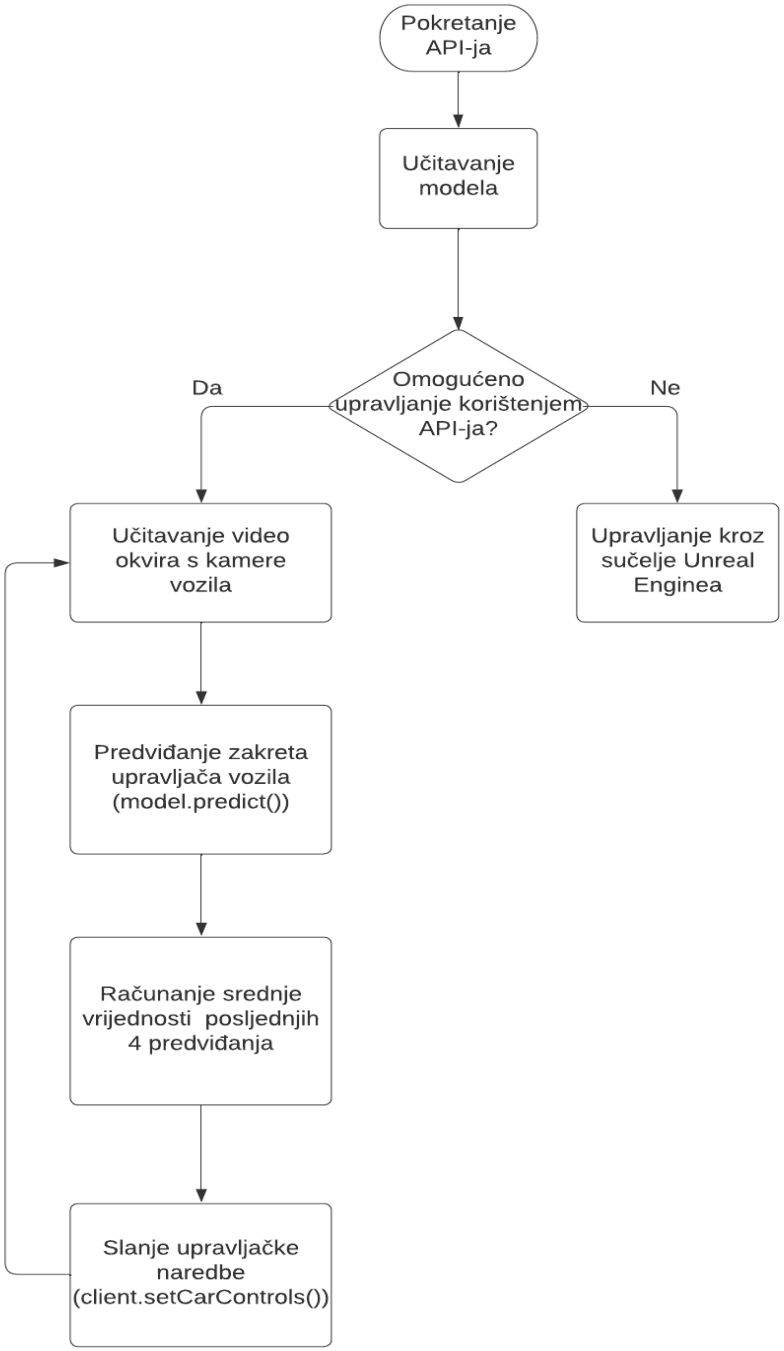


Sl. 3.29. Izgled grafičkog sučelja u CARLA simulatoru s pripadajućom ilustracijom zakreta upravljača vozila i prikazom područja interesa u stvarnom vremenu

Skripta za upravljanje vozilom u CARLA simulatoru (*drive_model_carla.py*) dana je kao elektronički prilog P.3.9 ovog rada.

3.2.5. Implementacija algoritma za upravljanje modelom autonomnog vozila zasnovanog na procjeni kuta zakreta upravljača vozila u AirSim simulatoru

Iako se radi o konceptualno različitom simulatoru, premisa je ista. Dohvaćanjem slike s vozila iz simulatora, treniranom modelu CNN se pruža ulazni podatak za procjenu kuta upravljanja vozila. Na slici 3.30 prikazan je dijagram toka *drive_model_airsim.py* algoritma za upravljanje vozilom u AirSim simulatoru.



Sl. 3.30. Dijagram toka upravljačkog algoritma za AirSim simulator

Kako se koristi isti trenirani model CNN, učitavanje se izvršava na identičan način: dodavanjem putanje do spremljenog modela funkciji *load_model()* koju pruža *Keras*. U AirSim simulatoru po zadanim postavkama API nije zadužen za stvaranje vozila. Uz to, korisnik ima potpunu kontrolu nad vozilom. Korišteno vozilo je u *Unreal Engineu* definirano kao trenutni „igrač“, te su u njemu

već implementirane naredbe za stvaranje vozila i upravljanje vozilom korištenjem perifernog ulaznog uređaja (tipkovnica, komandna palica ili upravljač). Međutim, AirSim izlaže API koji omogućuje programsku komunikaciju s vozilom u simulaciji. Predavanje upravljačkih mogućnosti klijentu prikazano je na slici 3.31.

```
client = airsim.CarClient()
client.confirmConnection()
client.enableApiControl(True)
car_controls = airsim.CarControls()
print('Connection established!')
```

Sl. 3.31. Primjer programskog koda za uspostavljanje komunikacije između klijenta i simulatora

Objekt klase *CarControls* sadrži atribute za upravljanje vozilom u simulatoru (iznos razine pritiska papučice gasa, kut zakreta upravljača, iznos razine pritiska papučice kočnice itd.). Važno je napomenuti da u isto vrijeme nije moguće upravljati vozilom putem API-ja i perifernim ulaznim uređajem u simulatoru. Metoda *client.enableApiControl* ovisno o primljenoj Booleovoj varijabli omogućuje kontrolu API-ja nad vozilom. Kako bi bila omogućena ljudska intervencija po potrebi, implementirana je funkcija koja mijenja stanje API upravljača pritiskom na definiranu tipku *Page Up* (Slika 3.32).

```
def on_press(key):
    try:

        if key == keyboard.Key.page_up and
client.enableApiControl(False):
            client.enableApiControl(True)
            inc_human_intervention_count()

        if key == keyboard.Key.page_up and
client.enableApiControl(True)::
            client.enableApiControl(False)
```

Sl. 3.32. Primjer programskog koda za aktivaciju i deaktivaciju autonomne vožnje

Sve postavke simulatora definirane su u konfiguracijskoj datoteci *settings.json*, između ostalog i parametri kamere koji su prilagođeni koordinatnom sustavu AirSim simulatora (Slika 3.33).

```
"Cameras": {
        "MyCamera1": {
          "X": 1.9, "Y": -0.3, "Z": -1,
          "Pitch": -2.85, "Roll": 0, "Yaw": -7,
      "CaptureSettings": [
        {
          "FOV_Degrees": 90
        }
      ]
        },
```

Sl. 3.33. Primjer definiranih parametara kamere iz konfiguracijske datoteke

Dohvaćanje slike s definirane kamere *MyCamera1* definirano je u funkciji *get_image()*, koja zahtjeva sliku od AirSim poslužitelja u .png formatu, a zatim sprema u *NumPy* polje cjelobrojnih 8-bitnih brojeva bez predznaka. Povratna vrijednost funkcije su elementi RGB slike u definiranom rasponu područja interesa (Slika 3.34).

```
def get_image():
      image_response=client.simGetImage("MyCamera1",AirSimImageTyp
e.Scene)
      image1d = np.fromstring(image_response, dtype=np.uint8)
      image_bgr=cv2.imdecode(image1d,cv2.IMREAD_COLOR)
      image_rgb= cv2.cvtColor(image_bgr, cv2.COLOR_BGR2RGB)
      return image_rgb[78:144,27:227,0:3].astype(float)
```

Sl. 3.34. Primjer programskog koda za dohvaćanje i obradu slike s kamere vozila

Unutar beskonačne petlje dohvaćena slika iz međuspremnika se predaje *model.predict()* funkciji kako bi trenirani model procijenio iznos zakreta upravljača. Također se procijenjeni iznos sprema u red, a srednja vrijednost četiriju posljednjih procjena se predaje *car_controls* objektu koji ažurira sljedeće stanje upravljača vozila. Eksperimentalno je utvrđeno da AirSim ima jako malu osjetljivost, te je potrebno pojačati predviđeni zakret upravljača vozila. Poslana naredba zakreta upravljača vozila definirana je kao:

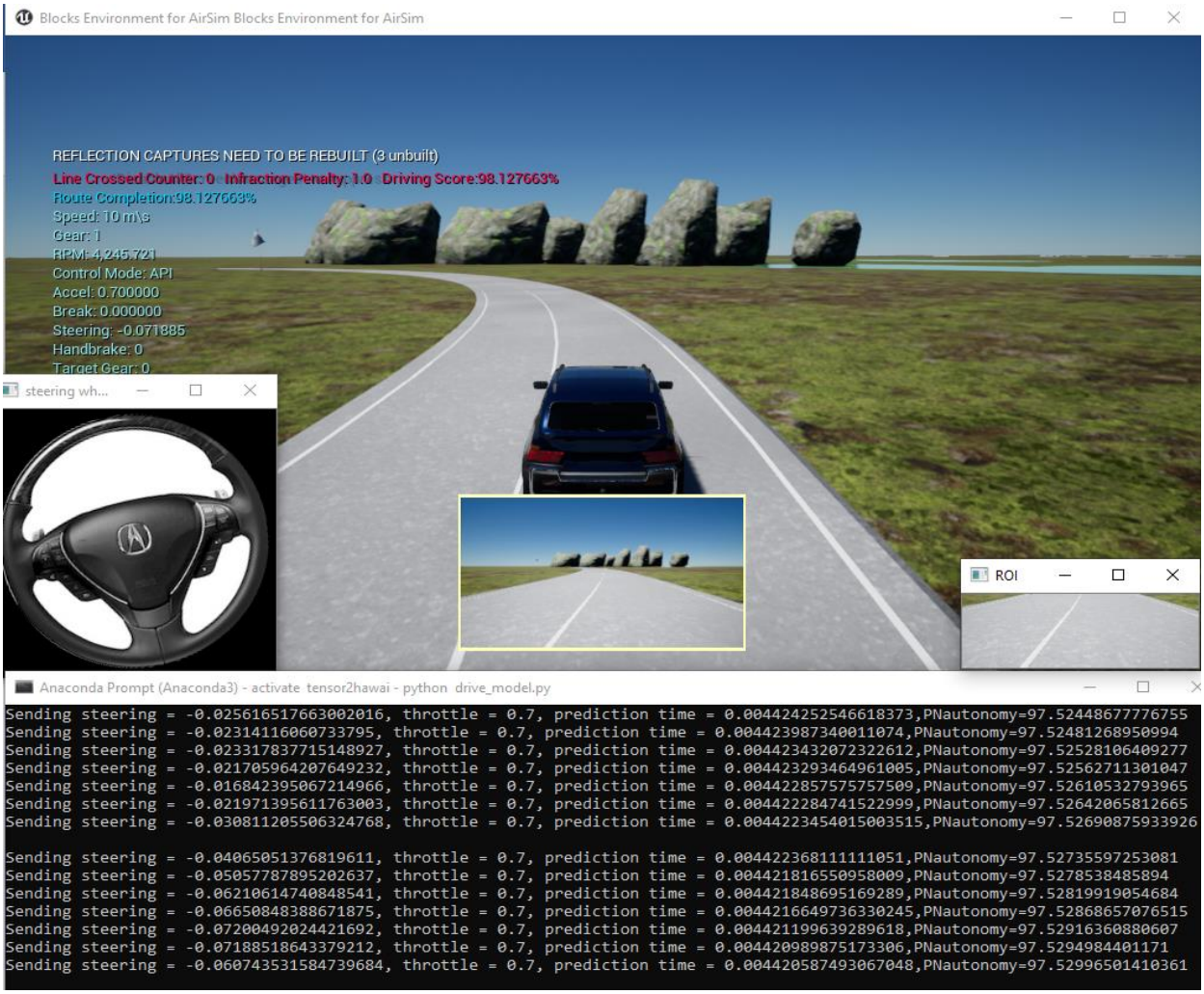
$$zakret\ upravljača\ vozila\ = 1.8 * (procjena\ modela * 2 - 1) \quad (3\text{-}3)$$

Procjena modela je u obliku broja s pomičnim zarezom (engl. *Float*) u rasponu [0-1]. AirSim simulator iznos zakreta upravljača definira u rasponu [-1,1]. Nakon pretvorbe procijenjenog zakreta upravljača vozila u raspon [-1,1] zakret se množi s koeficijentom 1.8 zbog male osjetljivosti zakreta upravljača vozila u AirSim simulatoru (npr. za isti zavoj kalibracijske mape vozilo u CARLA simulatoru uspješno prolazi sa zakretom upravljača vozila 0.20, dok vozilo u AirSim simulatoru prolazi uspješno isti zavoj sa definiranim zakretom upravljača vozila 0.54). Množenjem pretvorenog zakreta upravljača vozila s koeficijentom 1.8, u slučajevima kada je procjena modela veća od 0.78, zakret upravljača vozila izlazi iz raspona [-1,1]. Zato je važno napomenuti da je zakret upravljača programskim kodom ograničen na [-1,1]. Procijene modela apsolutne vrijednosti veće od 0.78 su definirane kao maksimalni zakret upravljača vozila. Na slici 3.35. prikazan je isječak koda za slanje upravljačke naredbe zakretanja upravljača vozila.

```
image_buf[0] = get_image()
image_buf[0] /= 255 # Normalizacija
model_output = model.predict([image_buf])
q1.get()
q1.put(float((model_output[0][0]*2.0)-1))
received_output=queue_ave(q1)
received_output=1.8*received_output
if (received_output>1):
    received_output=1
if(received_output<-1):
    received_output=-1

car_controls.steering =received_output
# Ažuriranje sljedećeg stanja vozila
client.setCarControls(car_controls)
```

Sl. 3.35. Primjer programskog koda za upravljanje zakretom upravljača vozila u AirSim simulatoru

Skripta *drive_model_airsim.py* također u konzolu ispisuje poslane naredbe za upravljanje vozilom, prosječno trajanje procjene modela i postotak autonomije vozila. Simulator ima ugrađenu funkcionalnost prikaza prozora sa slikom prednje kamere. Dodani su prikazi područja interesa i ilustracija upravljača vozila. Na slici 3.36 prikazan je izgled grafičkog sučelja u AirSim simulatoru s pripadajućom ilustracijom zakreta upravljača vozila i prikazom područja interesa u stvarnom vremenu. Skripta za upravljanje vozilom u CARLA simulatoru (*drive_model_airsim.py*) dana je kao elektronički prilog P.3.10 ovog rada.



Sl. 3.36. Izgled grafičkog sučelja u AirSim simulatoru s pripadajućom ilustracijom zakreta upravljača vozila i prikazom područja interesa u stvarnom vremenu

3.3. Upute za pokretanje algoritma za upravljanje modelom autonomnog vozila zasnovanog na procjeni kuta zakreta upravljača vozila

3.3.1. Upute za pokretanje algoritma u CARLA simulatoru

CARLA simulator je moguće preuzeti i pokrenuti u *build* ili *release* verziji. *Build* se odnosi na samostalni program generiran nakon pretvaranja izvornog koda u izvršni kod koji se može pokrenuti na računalu. S druge strane, *release* je distribucija konačne verzije aplikacije. U ovom poglavlju opisano je pokretanje algoritma na *build* verziji, kako bi simulator bilo moguće pokrenuti u *Unreal Engine* razvojnom okruženju. *Release* sadrži sve potrebne funkcionalnosti, ali u tom slučaju nema mogućnosti modificiranja korištenih mapa i stvaranja novih koje služe kao testne sekvence. Za pokretanje *builda* potreban je Visual Studio 2019 uz dodatne elemente:

- Windows 8.1 SDK,
- X64 Visual C++ Toolset,
- .NET framework 4.6.2.

Verzija 0.9.13 CARLA simulatora zahtijeva modificiranu verziju *Unreal Enginea* 4.26, koji sadrži potrebne zakrpe specifične za korišteni simulator (*CarlaUnreal*). Preuzima se upisivanjem naredbe u terminalu:

git clone --depth 1 -b carla https://github.com/CarlaUnreal/UnrealEngine.git .

Korištena je verzija Pythona 3.6.13. Od drugih zahtijeva CARLA simulatora, korištenjem terminala i Python instalacijskog paketa instalirane su sljedeće biblioteke:

- *future,*
- *numpy,*
- *pygame,*
- *matplotlib,*
- *open3d,*
- *Pillow.*

Zatim se pokreću konfiguracijske skripte *Setup.bat* i *GenerateProjectFiles.bat*. Prevođenje stvorenog programskog rješenja *UE4.sln* izvodi se u *Visual Studio* razvojnom okruženju, a sadrži informacije o korištenom razvojnom okruženju, parametre i sve projekte paketa uz njegove reference. Tek tada je moguće pokrenuti *Unreal Engine*.

Instalacija samog simulatora se izvršava kloniranjem službenog repozitorija projekta naredbom:

git clone https://github.com/carla-simulator/carla

Skripta *Update.bat* ažurira sve korištene značajke simulatora. Prevođenje *Python* aplikacijskog programskog sučelja za interakciju s klijentom je potrebno samo pri prvom pokretanju CARLA simulatora, a provodi se naredbom:

make PythonAPI

Sljedeća naredba prevodi i pokreće *Unreal Engine* razvojno okruženje, a poziva se pri svakom pokretanju servera:

make launch

U konačnici, otvara se razvojno okruženje gdje je moguće manipulirati svim dostupnim objektima i značajkama simulatora. Pokretanje željene mape se izvodi dvostrukim klikom na ikonicu razine (engl. *level).* Sve mape se nalaze u direktoriju *Content/Carla/Maps.* Simulacija se pokreće pritiskom na gumb *Play* u sučelju razvojnog okruženja.

Za ispravan rad skripte za autonomnu vožnju, nazvane *drive_model_carla.py*, korištene su sljedeće biblioteke:

- *Keras 2.1.2,*
- *Tensorflow-gpu 1.6.0,*
- *Numpy 1.19.5,*
- *OpenCV 4.2.0.32,*
- *H5py 2.10.0,*
- *Carla*.

Skripta se pokreće iz terminala, uz prethodno pokrenutu simulaciju, naredbom:

python putanja/do/skripte/drive_model_carla.py

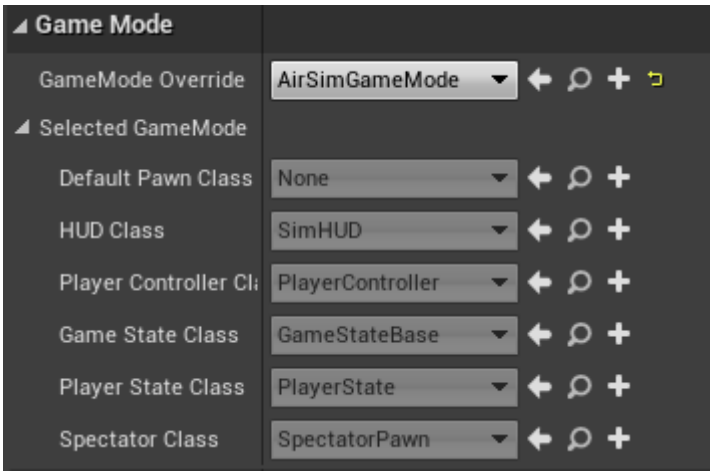
3.3.2. Upute za pokretanje algoritma u AirSim simulatoru

Za pokretanje AirSim simulatora korištena je ista verzija *Unreal Enginea 4.26* zbog mogućnosti migriranja svih dostupnih resursa između korištenih simulatora. Prvo je potrebno klonirati repozitorij AirSim simulatora dostupan na:

git clone https://github.com/Microsoft/AirSim.git

Unutar AirSim direktorija nalazi se *build.cmd* skripta koja sadrži potrebne naredbe za instalaciju pripremljenih proširenja *Unreal Engine* razvojnog okruženja. Pokretanjem skripte iz terminala stvara se *VisualStudio* programsko rješenje *Blocks.sln.* Struktura rješenja sadrži projekt koji izvođenjem automatski generira potrebne datoteke projekta, ažurira pakete i pokreće prazno okruženje simulatora u *Unreal Engineu.* Prije samog pokretanja poželjno je u postavkama odabrati verziju 4.26 *Unreal Enginea,* a prilikom pokretanja konfiguracija rješenja je postavljena na *Develop Editor* i platformu x64. Projekt sadrži sve potrebne funkcionalnosti simulatora, dodatno je potrebno ručno dodati željene mape u otvoreni projekt. U otvorenom *Unreal Engine* projektu simulacija se pokreće odabirom mape i pritiskom na gumb *Play*. U slučaju dopremanja mapa iz

CARLA simulatora, za svaku mapu je neophodno u postavkama okruženja promijeniti način igre na AirSim način (Slika 3.37).



Sl. 3.37. Primjer korištenih postavki okruženja u AirSim simulatoru

Pokretanje skripte za autonomnu vožnju u AirSim simulatoru zahtijeva sljedeće biblioteke:

- *Keras 2.1.2,*
- *Tensorflow-gpu 1.6.0,*
- *Numpy 1.19.5,*
- *OpenCV 4.2.0.32,*
- *H5py 2.10.0,*
- *Airsim.*

Pokreće se putem terminala naredbom:

python putanja/do/skripte/drive_model_airsim.py

4. EVALUACIJA RADA ALGORITMA ZA PROCJENU KUTA ZAKRETA UPRAVLJAČA I UPRAVLJANJE MODELOM VOZILA U DVAMA SIMULATORIMA

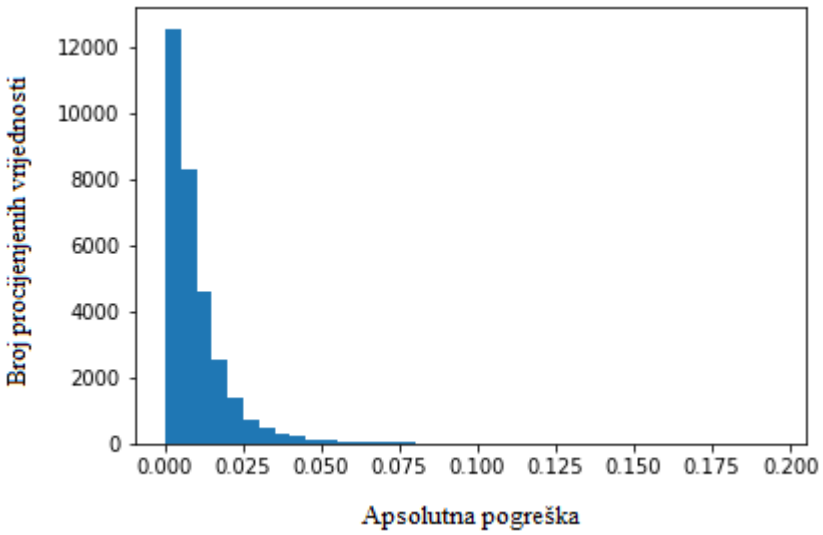
U ovom poglavlju opisan je postupak evaluacije rada implementiranog *PilotNet* algoritma za procjenu kuta zakreta upravljača. Testiranje algoritma je izvršeno na računalu s operacijskim sustavom Windows 10, procesorom Intel i7-6700 3.40 GHz, 16 GB RAM-a i grafičkom karticom NVIDIA GeForce GTX 1060 6 GB.

4.1. Testna baza podataka iz stvarnog svijeta i testne mape iz simulatora

Testirano je ponašanje modela na slikama iz stvarnog svijeta i na testnim sekvencama iz simulatora. Testni skup podataka se sastoji od 40036 slika i pripadajućih iznosa zakreta upravljača vozila. Korišteni testni skup podataka je opisan u dijelu 3.2.1. Testiranje u simulatoru je izvedeno na mapama napravljenim u *Unreal Engineu* (Test1, Test2 i Test3), a detaljnije su opisane u dijelu 3.2.3.

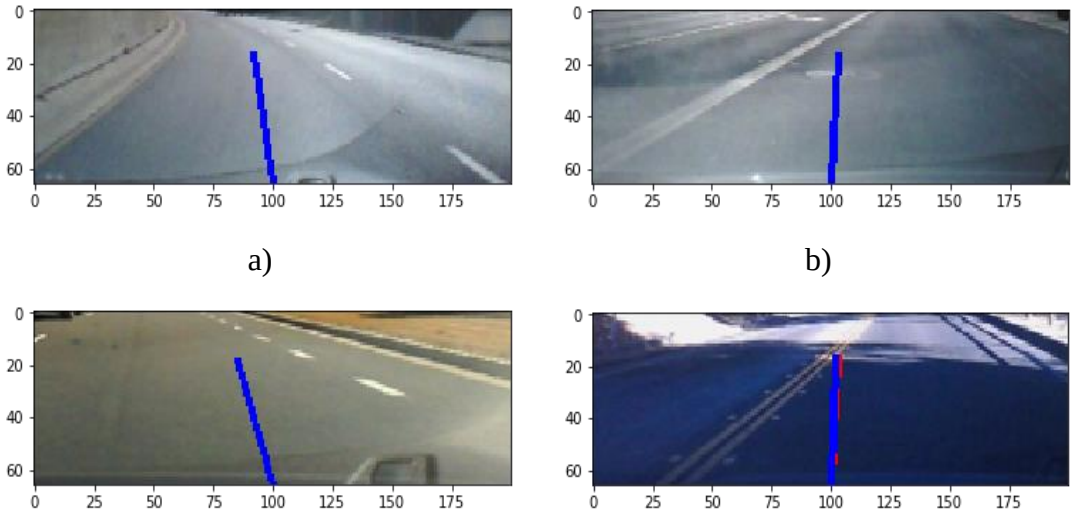
4.2. Opis testiranja algoritma za procjenu kuta zakreta upravljača vozila na testnom skupu iz stvarnog svijeta

Upravljanje vozilom je kontinuirana i kompleksna radnja koja ovisi o većem broju faktora i uvjeta. Uspoređivanje stvarnog i procijenjenog kuta upravljanja na bazi jednog okvira nije optimalan pristup. Putanja kretanja autonomnog vozila se može razlikovati od putanje stvarnog vozača, što ne znači da takvo ponašanje nije ispravno. Međutim, testiranje na testnom skupu podataka može poslužiti kao test zdravog razuma (engl. *Sanity test*). Cilj je utvrditi da osnovne funkcionalnosti nisu ugrožene i da je procjena modela istinita i racionalna. Uspoređuje se procijenjena vrijednost zakreta upravljača sa stvarnom (u vožnji zabilježenom) vrijednošću. Prema autorima u radu [23] prag tolerancije pogrešku kuta zakreta upravljača vozila je 6°, a kako je korišteni skup podataka normaliziran na vrijednosti [0,1], dobrom procjenom se smatra vrijednost koja odstupa za manje od 0.012 od stvarne vrijednosti kuta zakreta upravljača vozila, što je u slučaju 78,8 % testnih video okvira. Slika 4.1 prikazuje histogram distribucije apsolutnih pogrešaka na testnom skupu podataka.



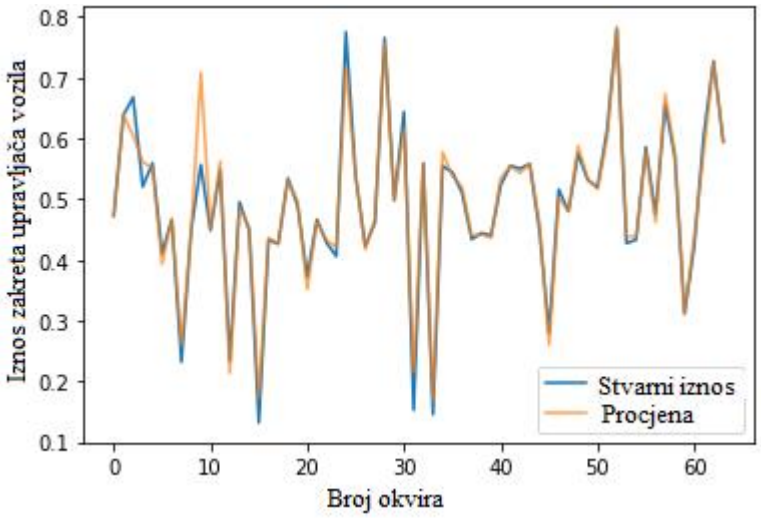
Sl. 4.1. Histogram distribucije apsolutnih pogrešaka procijenjenih vrijednosti u odnosu na stvarnu vrijednost kutova zakreta upravljača vozila

Na slici 4.2 prikazani su primjeri dobre procjene kuta zakreta upravljača vozila. Plavom bojom označena je procjena kuta zakreta upravljača vozila od strane modela, a crvenom bojom označena je stvarna vrijednost kuta zakreta upravljača vozila.



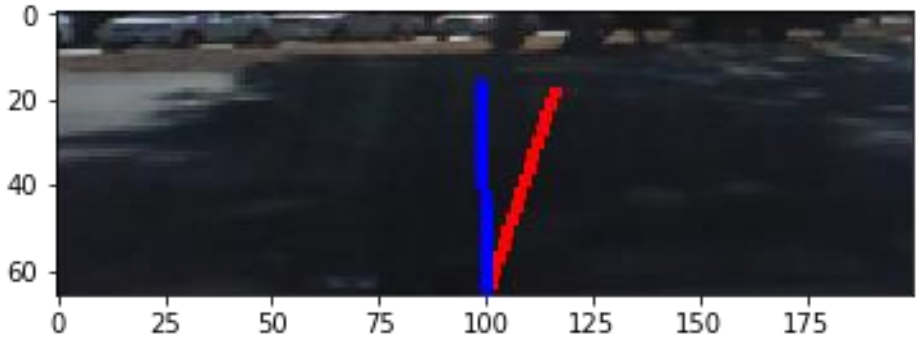
Sl. 4.2. Primjeri točne procjene kuta upravljača vozila: (a) Stvarna vrijednost = [0.39471077], Procijenjena vrijednost = [0.38861808], Apsolutna greška = [0.00609269], (b) Stvarna vrijednost = [0.54844718], Procijenjena vrijednost = [0.55157864], Apsolutna greška = [0.00313146], (c) Stvarna vrijednost = [0.29168488], Procijenjena vrijednost = [0.29327068], Apsolutna greška = [0.00158579], (d) Stvarna vrijednost = [0.54844718], Procijenjena vrijednost = [0.54963297], Apsolutna greška = [0.00118579]

Slika 4.3 prikazuje graf usporedbe procijenjenih vrijednosti modela (označeno narančastom bojom) i stvarnih vrijednosti kuta zakreta upravljača (označeno plavom bojom) u 64 nasumično odabrana video okvira iz testnog skupa podataka.



Sl. 4.3. Graf usporedbe procijenjene vrijednosti modela i stvarnog zakreta upravljača u 64 nasumično odabrana video okvira iz testnog skupa podataka

Trenirani model *PilotNet* mreže u 78,9 % slučajeva uspješno predviđa prihvatljiv zakret upravljača vozila, no postoje izolirane situacije gdje pokazuje osjetljivost na različite uvjete u vožnji. U navedenom primjeru sa slike 4.3, jedina veća apsolutna pogreška (0.16) vidljiva je na desetom video okviru (Slika 4.4), gdje model nije prepoznao potrebne značajke za prihvatljivu procjenu kuta zakreta upravljača vozila, uslijed slabe vidljivosti zbog nastanka sjene na cesti.



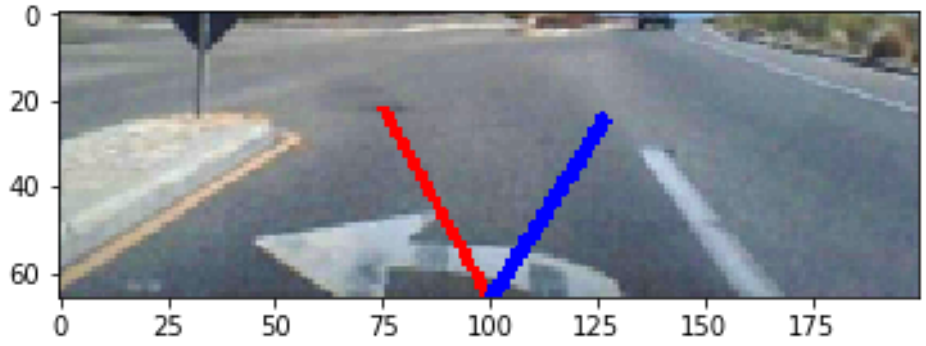
Sl. 4.4. Primjer pogrešne procjene kuta zakreta upravljača vozila

Na promatranom testnom skupu podataka je izvršeno mjerenje standardnih statističkih metrika. U tablici 4.1 su navedeni iznosi maksimalne (MAXAE), minimalne (MINAE), srednje (MAE) i medijalne apsolutne pogreške (MedAE) i iznosi maksimalne (MAXSE), minimalne (MINSE), srednje (MSE) i medijalne kvadratne pogreške (MedSE).

Tablica 4.1. Rezultati testiranja na testnom skupu podataka

MAXAE	**MINAE**	**MAE**	**MedMAE**	**MAXSE**	**MINSE**	**MSE**	**MedMSE**
0.794	6.04e-7	0.010	0.006	0.631	3.65e-13	3.40e-4	4.41e-05

Na osnovu rezultata iz tablice 4.1 vidljivo je da model načelno uspješno procjenjuje zakret upravljača vozila sa srednjom i medijalnom apsolutnom pogreškom manjom od zadanog praga tolerancije (0.012). Najveće odstupanje procijenjenog zakreta upravljača vozila prikazano je na slici 4.5. U primjeru na slici vidljivo je kako se znatno razlikuje iznos zakreta upravljača kojeg daje trenirani model i stvarnog (referentnog, poznatog) zakreta dobivenog vožnjom vozača koji je zabilježen u skupu podataka za treniranje. Iz primjera se može zaključiti da trenirani model nije primjenjiv u situacijama na cesti gdje smjer kretanja vozila definiraju oznake na cesti ili kada postoji više prometnih traka u kojima je dozvoljeno kretanje, što je specifično za raskrižja. Uostalom, model na temelju video okvira ne može odlučiti o prestrojavanju i skretanju na raskrižjima. Za kompleksnije odluke potrebna je interakcija s drugim ADAS sustavima i drugim senzorima poput *GPS-a*.



Sl. 4.5. Primjer pogrešne procjene kuta zakreta upravljača vozila na raskrižju

4.3. Opis testiranja algoritma za procjenu kuta zakreta upravljača vozila u simulatoru

Testiranje u svakom simulatoru se sastoji od tri različita testa na stvorenim mapama koristeći sve dostupne vremenske uvjete koje pružaju simulatori. Korištene su mape opisane u dijelu 3.2.3, a svaki testni scenarij je predstavljen vožnjom vozila čiji zakret upravljača određuje model trenirane *PilotNet* mreže. Svaki scenarij se izvodi na jednom krugu staze i pri jednom tipu vremenskog uvjeta. Cilj testiranja je utvrditi u kojem postotku ukupnog vremena vožnje model CNN može uspješno upravljati vozilom, a to će se mjeriti mjerom koja se naziva *autonomija*. Mjera uspješnosti se provodi brojanjem ljudskih intervencija prilikom vožnje. Ljudska intervencija je postupak preuzimanja kontrole nad vozilom, vraćanje vozila u središte prometne trake i ponovno pokretanje autonomne vožnje. Takve intervencije se pojavljuju kada središte simuliranog vozila odstupi od središta prometne trake za više od jednog metra. Pretpostavka je da bi u stvarnoj vožnji ljudska intervencija trajala oko 6 sekundi [8]. Kao i pretpostavka trajanja ljudske intervencije, izračun autonomije je preuzet iz rada [8], a predstavljen je izrazom:

$$autonomija = \left(1 - \frac{broj\ ljudskih\ intervencija * 6}{proteklo\ vrijeme\ vožnje}\right) * 100 \tag{4-1}$$

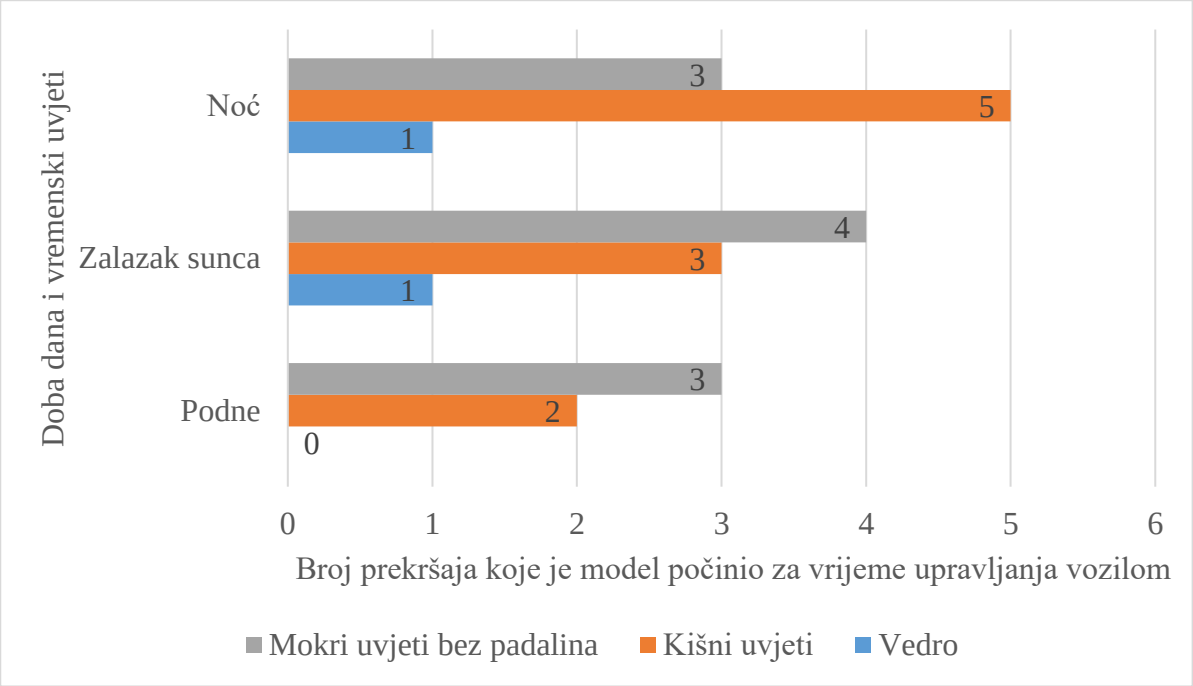
Testovi su organizirani na način da *PilotNet* algoritam upravlja vozilom jedan krug na stazi za svaki od odabranih vremenskih uvjeta i doba dana. Vozilo se kreće konstantnom brzinom od 50 km/h. Prilikom vožnje je mjeren broj ljudskih intervencija. Za prvi test su korišteni svi dostupni vremenski uvjeti iz svakog simulatora, kako bi bile uspoređene mogućnosti koje svaki simulator pruža. Uz to, testirano je ponašanje vozila kojim upravlja *PilotNet* algoritam u odnosu na različite vremenske uvjete. Testiranje je provedeno u obama simulatorima na mapi Test1. U tablici 4.2 nalaze se mjereni rezultati iz CARLA simulatora.

Tablica 4.2. Rezultati testiranja na mapi Test1 u CARLA simulatoru

Vrsta razdjelne linije	**Vremenski uvjeti i doba dana**	**Broj ljudskih intervencija**	**Broj ljudskih intervencija/ km**	**Autonomija [%]**
Puna linija	Zadani (Vedro, dan)	0	0,00	100,00

Puna linija	Vedro, podne	0	0,00	100,00
Puna linija	Vedro, zalazak sunca	1	0,32	97,32
Puna linija	Oblačno, noć	3	0,96	91,96
Puna linija	Oblačno, podne	0	0,00	100,00
Puna linija	Oblačno, zalazak sunca	0	0,00	100,00
Puna linija	Jaka kiša, noć	4	1,27	89,29
Puna linija	Jaka kiša, zalazak sunca	1	0,32	97,32
Puna linija	Jaka kiša, podne	0	0,00	100,00
Puna linija	Kiša, zalazak sunca	3	0,96	91,96
Puna linija	Kiša, noć	5	1,59	86,61
Puna linija	Kiša, podne	2	0,64	94,64
Puna linija	Lagana kiša, noć	5	1,59	86,61
Puna linija	Lagana kiša, podne	3	0,96	91,96
Puna linija	Lagana kiša, zalazak sunca	4	1,27	89,29
Puna linija	Mokro, oblačna noć	6	1,91	83,93
Puna linija	Mokro, oblačno podne	5	1,59	86,61
Puna linija	Mokro, oblačno pri zalasku sunca	6	1,91	83,93
Puna linija	Mokro, noć	3	0,96	91,96
Puna linija	Mokro podne	3	0,96	91,96
Puna linija	Mokro, zalazak sunca	4	1,27	89,29
Puna linija	Vedro, noć	1	0,32	97,32

Iz predstavljenih rezultata u tablici 4.2 uočljivo je da algoritam zadovoljavajuće obavlja zadatak vožnje. U 22 kruga na stazi (69 km) model je 92,82 % provedenog vremena na stazi uspješno upravljao vozilom unutar dopuštenih granica. Pri tome je vozilo 59 puta odstupilo od središta prometne trake za više od jednog metra, što bi u prosjeku značilo 0.85 ljudskih intervencija po kilometru u stvarnoj vožnji. Budući da je test izvršen u simulatoru, nije bilo potrebe za ručnim pozicioniranjem vozila. Model je konstantno samostalno ispravljao putanju vozila. Slika 4.8 prikazuje graf s brojem prekršaja u trima različitim dobima dana u kombinaciji s različitim vremenskim uvjetima.



Sl. 4.8. Grafički prikaz broja prekršaja u različitim dobima dana i pri različitim vremenskim uvjetima u CARLA simulatoru na mapi Test1

Očekivano, u slučaju vedrih vremenskih uvjeta model koji upravlja vozilom je najmanje puta odstupio od središta prometne trake. Tada je najbolja vidljivost okruženja kojeg snima kamera s prednje strane vozila. Uostalom, cijeli skup podataka za treniranje je snimljen u sličnim uvjetima vožnje. Kišni i mokri uvjeti *PilotNet* mreži predstavljaju najveći izazov za procjenu zakreta upravljača, kako zbog nedostatka takvih primjera u skupu podataka za učenje, tako i zbog smanjene vidljivosti, promjene izgleda teksture ceste i refleksije svjetlosti u lokvama vode pri zalasku sunca. U noćnim uvjetima smanjene vidljivosti algoritam je upravljao vozilom bliže središnjoj razdjelnoj liniji, a direktna posljedica toga je i veći broj ljudskih intervencija. U mokrim

uvjetima bez padalina, model je pokazao lošije ponašanje u odnosu na noćnu vožnju. Kombinacija niske visine sunca i odsjaja sunčevih zraka je negativno utjecala na rad algoritma.

Test je proveden u AirSim simulatoru koristeći isto testno okruženje (mapa Test1). Rezultati testa prikazani su u tablici 4.3

Tablica 4.3. Rezultati testiranja na mapi Test1 u AirSim simulatoru

Vrsta razdjelne linije	**Vremenski uvjeti i doba dana**	**Broj ljudskih intervencija**	**Broj ljudskih intervencija/km**	**Autonomija[%]**
Puna linija	Vedro, dan	5	1,59	89,21
Puna linija	Kiša, dan	5	1,59	89,21
Puna linija	Snijeg, dan	5	1,59	89,21
Puna linija	Padajuće lišće, dan	5	1,59	89,21
Puna linija	Prašina, dan	12	3,82	74,10
Puna linija	Magla, dan	2	0,64	95,68

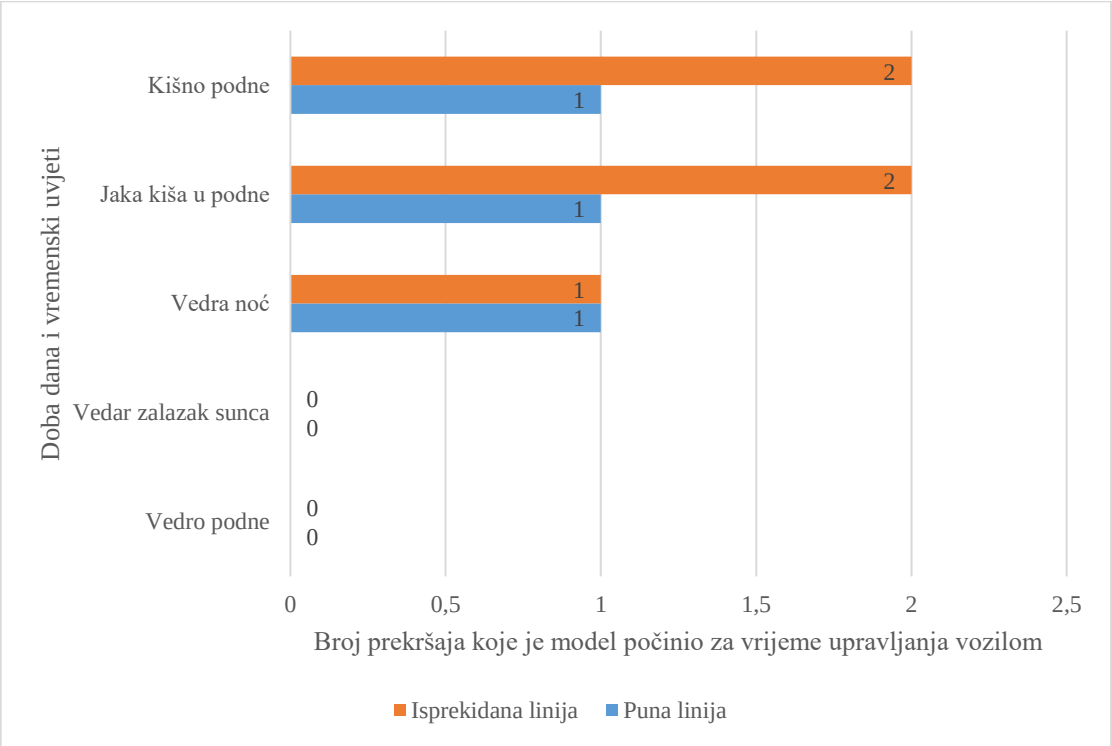
Premda je algoritam ispitan na istoj konfiguraciji staze, AirSim pruža manji broj različitih vremenskih uvjeta i ne omogućava noćnu vožnju. Provedeni test u AirSim simulatoru je na kraju pokazao lošije rezultate u odnosu na CARLA simulator. Model je pokazao autonomiju od 87,77 %, što je 0.95 ljudskih intervencija po kilometru više u odnosu na test u CARLA simulatoru. Algoritam je dao iste rezultate u idealnim vremenskim uvjetima i svim vremenskim uvjetima s padalinama (kiša, snijeg, padajuće lišće). Generirane padaline od strane simulatora se stvaraju iznad vozila, a pri kretanju vozila su manje vidljive na kameri vozila. Manjak realizma implementiranih padalina u simulatoru uskratio je mogućnost detaljnije analize navedenih vremenskih uvjeta. Međutim, algoritam je pokazao drugačija ponašanja u slučajevima prašine (pješčana oluja) i magle. Iako su oba uvjeta smanjila vidljivost, model je iznenađujuće najuspješnije upravljao vozilom u uvjetima magle. Iz prikazane vožnje u svim uvjetima može se zaključiti da je vozilo bolje pozicionirano u voznoj traci prilikom prolaska kroz gradski dio staze, zbog slične boje i teksture podloge pločnika uz cestu i same ceste. Time algoritam bolje prepoznaje značajke koje predstavljaju voznu traku ograničenu linijama, umjesto fokusiranja na rub kolnika. Isto ponašanje je algoritam pokazao i tijekom vožnje u uvjetima magle, gdje pri smanjenoj vidljivosti i preglednosti nije vidljiva okolina ceste.

Drugim testom provjerava se indiferentnost modela na različite vizualne identitete ceste, tj. različite vrste razdjelne središnje linije na cesti. Testiranje je izvršeno koristeći mapu Test2, kombinirajući pune razdjelne linije i isprekidane razdjelne linije ceste. Testiranje u CARLA simulatoru je izvršeno za pet odabranih različitih vremenskih uvjeta za obje vrste razdjelnih linija. Rezultati testiranja su prikazani u tablici 4.4.

Tablica 4.4. Rezultati testiranja na mapi Test2 u CARLA simulatoru

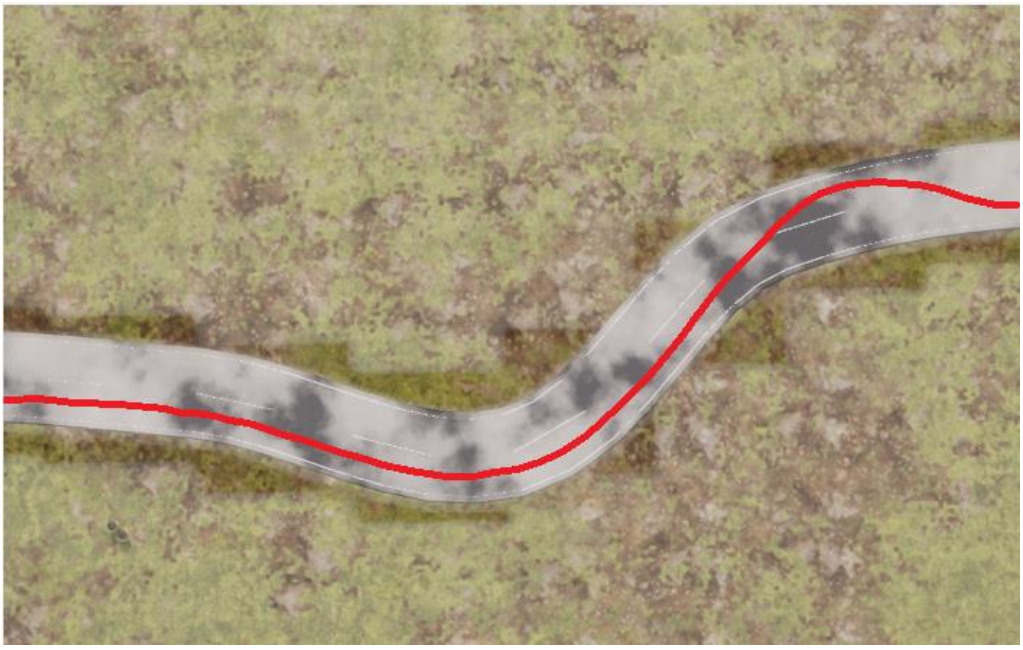
Vrsta razdjelne linije	**Vremenski uvjeti i doba dana**	**Broj ljudskih intervencija**	**Broj ljudskih intervencija/km**	**Autonomija[%]**
Puna linija	Vedro, podne	0	0,00	100,00
Puna linija	Vedar, zalazak sunca	0	0,00	100,00
Puna linija	Vedro, noć	1	0,32	97,06
Puna linija	Jaka kiša, podne	1	0,32	97,06
Puna linija	Umjerena kiša, podne	1	0,32	97,06
Isprekidana linija	Vedro, podne	0	0,00	100,00
Isprekidana linija	Vedro, zalazak sunca	0	0,00	100,00
Isprekidana linija	Vedro, noć	1	0,32	97,06
Isprekidana linija	Jaka kiša, podne	2	0,64	94,06
Isprekidana linija	Umjerena kiša, podne	2	0,64	94,06

Iz predočenih rezultata je vidljiva potvrda pretpostavke iz testova provedenih na mapi Test1: vremenski uvjeti utječu na točnost procjene zakreta upravljača vozila, a time i broj ljudskih intervencija prilikom autonomne vožnje. Vizualizirani podaci broja prekršaja u odnosu na različite vremenske uvjete i korištenu vrstu razdjelne linije prikazani su na slici 4.9.



Sl. 4.9. Grafički prikaz broja prekršaja u odnosu na različite vremenske uvjete i korištene različite vrste razdjelne linije na cesti u CARLA simulatoru na mapi Test2

Promjenom vrste razdjelne linije u isprekidanu liniju, algoritam autonomne vožnje je pokazao neželjeno lateralno kretanje u prometnoj traci (kretanje lijevo-desno; krivudanje), što se manifestiralo povećanjem prosječnog broja ljudskih intervencija po kilometru za 0.13 ljudskih intervencija po kilometru (0.19 na punoj liniji i 0.32 na isprekidanoj liniji). Vozilo se tijekom vožnje uz isprekidanu liniju približava središtu kolnika, ukoliko je u području interesa trenutnog video okvira praznina između dviju isprekidanih linija. Kombinacijom lošijih vremenskih uvjeta i isprekidane razdjelne linije, u jednom krugu je algoritam zahtijevao i ručno postavljanje vozila u ispravnu prometnu traku. Uslijed oštrog zavoja vozilo je prešlo razdjelnu liniju i nastavilo vožnju u krivoj prometnoj traci, što je zahtijevalo ručno ponovno pozicioniranje vozila. Na slici 4.10. prikazan je isječak zavoja iz scenarija u kojem je nastupio gubitak kontrole autonomnog vozila.



Sl. 4.10. Primjer zavoja iz mape Test2 na kojem nastupa gubitak kontrole vozila s označenom putanjom gibanja

Isti test proveden je i u AirSim simulatoru. Rezultati testiranja prikazani su u tablici 4.5.

Tablica 4.5. Rezultati testiranja na mapi Test2 u AirSim simulatoru

Vrsta razdjelne linije	**Vremenski uvjeti i doba dana**	**Broj ljudskih intervencija**	**Broj ljudskih intervencija/km**	**Autonomija[%]**
Puna linija	Vedro, dan	4	1,27	89,33
Puna linija	Kiša, dan	4	1,27	89,33
Puna linija	Snijeg, dan	4	1,27	89,33
Puna linija	Padajuće lišće, dan	4	1,27	89,33
Puna linija	Prašina, dan	14	4,46	62,67
Puna linija	Magla, dan	2	0,64	94,67
Isprekidana linija	Vedro, dan	18	5,73	51,79
Isprekidana linija	Kiša, dan	18	5,73	51,79
Isprekidana linija	Snijeg, dan	18	5,73	51,79

Isprekidana linija	Padajuće lišće, dan	18	5,73	51,79
Isprekidana linija	Prašina, dan	19	6,05	49,11
Isprekidana linija	Magla, dan	2	0,64	94,64

Prikazani rezultati iz AirSim simulatora potvrđuju prikazano u CARLA simulatoru: porast broja prekršaja pri vožnji na cesti s isprekidanom razdjelnom linijom. U ovom testnom slučaju, još je izraženija razlika gdje je model pri vožnji na cesti s isprekidanom razdjelnom linijom zahtijevao 3.2 ljudske intervencije po kilometru više u odnosu vožnje na cesti s punom razdjelnom linijom. Trenirani model je pokazivao određenu nesigurnost pri vožnji uz isprekidanu liniju, očitovano prilaženjem središtu kolnika u prazninama između isprekidanih linija. Točnije, kada je sljedeća linija udaljena od kamere vozila, model prepoznaje značajke koje predstavljaju rubove kolnika, te se pozicionira bliže središtu kolnika, a time i udaljava od središta svoje prometne trake.

Treći testni skup scenarija je zamišljen kao provjera mogućnosti autonomne vožnje algoritma u urbanom naselju, odnosno u gradskoj vožnji. Kako je već navedeno, algoritam nije u stanju odlučiti o skretanju na raskrižju bez drugih naprednih sustava vožnje. Zato je korištena mapa Test3 bez raskrižja i semafora, s velikim brojem uskih zavoja od 90 stupnjeva, koji simuliraju gradske ulice. Korištena je konstantna brzina od 30 km/h. Rezultati testiranja autonomne vožnje u CARLA simulatoru prikazani su u tablici 4.6., a rezultati testiranja u AirSim simulatoru u tablici 4.7.

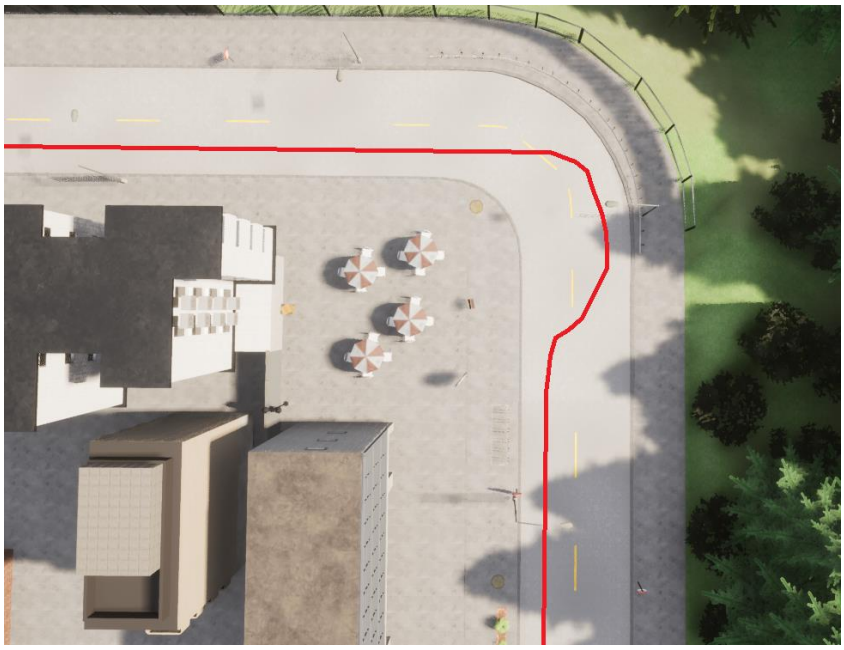
Tablica 4.6. Rezultati testiranja na mapi Test3 u CARLA simulatoru

Vrsta razdjelne linije	**Vremenski uvjeti i doba dana**	**Broj ljudskih intervencija**	**Broj ljudskih intervencija/km**	**Autonomija [%]**
Isprekidana linija	Vedro, podne	7	3,89	82,50
Isprekidana linija	Vedro, zalazak sunca	4	2,22	90,08
Isprekidana linija	Vedro, noć	9	5,00	77,50
Isprekidana linija	Jaka kiša, podne	11	6,11	72,50
Isprekidana linija	Kišno, podne	15	8,33	62,50

Tablica 4.7. Rezultati testiranja na mapi Test3 u AirSim simulatoru

Vrsta razdjelne linije	**Vremenski uvjeti i doba dana**	**Broj ljudskih intervencija**	**Broj ljudskih intervencija/km**	**Autonomija [%]**
Isprekidana linija	Vedro	5	2,78	49,33
Isprekidana linija	Kiša	5	2,78	49,33
Isprekidana linija	Snijeg	5	2,78	49,33
Isprekidana linija	Padajuće lišće	5	2,78	49,33
Isprekidana linija	Prašina	5	2,78	33,33
Isprekidana linija	Magla	5	2,78	78,67

PilotNet algoritam je u oba simulatora pokazao lošije rezultate u gradskoj vožnji, što se nedvosmisleno oslikava u velikom porastu ljudskih intervencija. Vozilo nije u stanju kontinuirano prolaziti desni zavoj pod kutom od 90 stupnjeva (Slika 4.11).



Sl. 4.11. Primjer zavoja iz mape Test3 na kojem nastupa gubitak kontrole vozila

Algoritam tek u trenutku približavanja suprotnog ruba ceste prepoznaje zavoj, dok desni rub ceste ne ulazi u kadar središnje kamere vozila. Veći broj ljudskih intervencija u CARLA simulatoru se može pripisati usporenom radu zbog premalog broju obrađenih video okvira u sekundi (engl.

frames per second, FPS). Korištena mapa Test3 sadrži velik broj vizualnih objekata te je njihovo generiranje znatno usporilo izvršavanje ovog testnog scenarija. Točnije, radi se o 15 FPS, dok su ostali testni scenariji izvršeni pri 30 FPS. Što se tiče brzine izvođenja same procjene modela, mjereno je trajanje izvođenja funkcije *model.predict()*, koja vraća procijenjeni zakret upravljača vozila. U svakom testnom scenariju zabilježeno je prosječno vrijeme procjene zakreta upravljača. Prosječna vremena procjene modela iz obaju simulatora prikazana su u tablici 4.8.

Tablica 4.8. Prosječno vrijeme procjene zakreta upravljača vozila u korištenim simulatorima

Simulator	**Prosječno vrijeme procjene zakreta upravljača vozila [s]**	**Prosječan broj okvira po sekundi u simulatoru**
CARLA	0,005898063	28
AirSim	0,005940179	32

Zbog korištenja slika male rezolucije (200x66 elemenata slike) kao ulaznih podataka u *PilotNet* mrežu, model procjenjuje zakret upravljača velikom brzinom. Može se reći da je prikladan za rad u stvarnom vremenu. Međutim, u ovoj implementaciji je ograničen brzinom izvođenja simulatora na istom sklopovlju. Isto tako, u stvarnom ugradbenom sustavu bi postojala druga ograničenja poput brzine učitavanje slike, prijenosa podataka i drugih postupaka koji nisu uključeni u prikazano mjerenje.

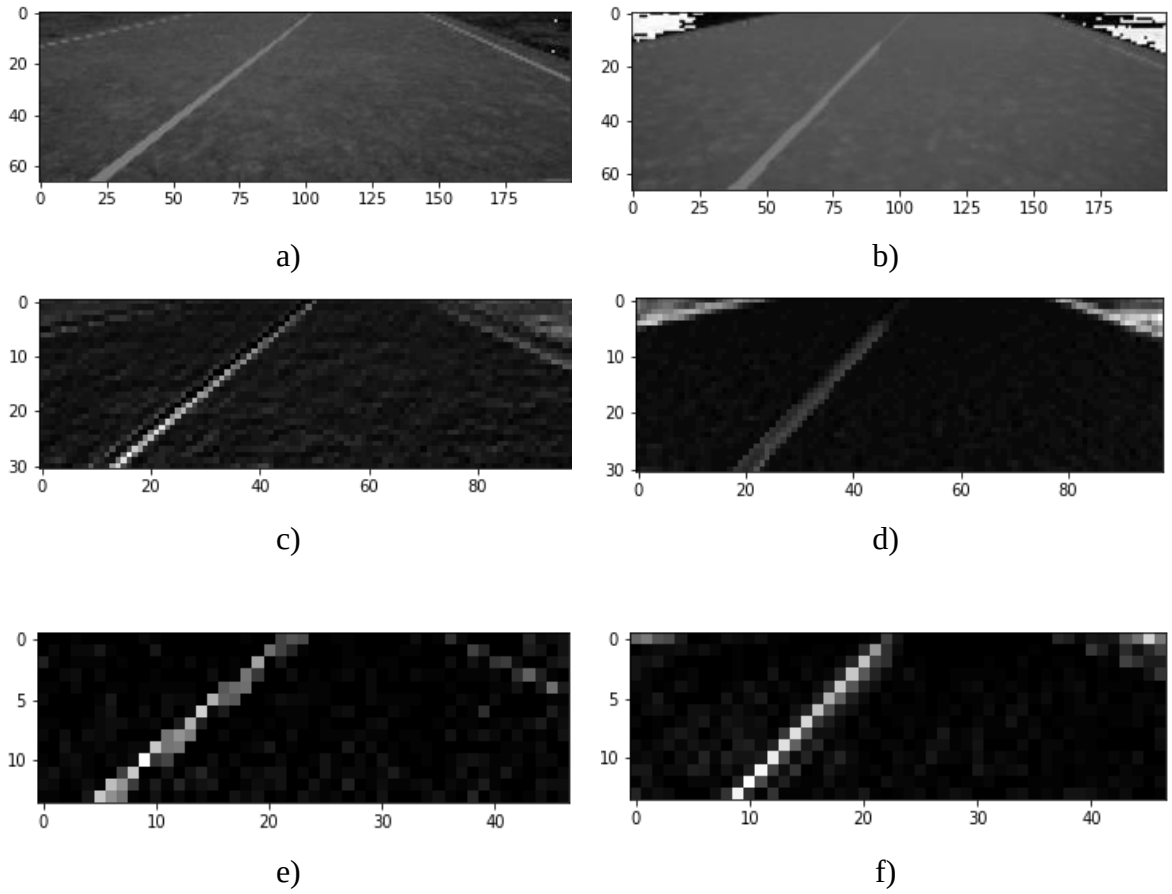
Gledajući sva tri opisana testna scenarija, ukupno je provedeno 119 km vožnje u CARLA simulatoru i 79 km vožnje u AirSim simulatoru. Na kraju je izračunat postotak autonomije za svaki simulator. Usporedba ostvarenog postotka autonomije u obama simulatorima prikazana je u tablici 4.9.

Tablica 4.9. Postotak autonomije

Simulator	**Autonomija [%]**
CARLA simulator	91,65
AirSim simulator	71,87

Predstavljeni algoritam je u izvornom radu postigao autonomiju od 98% [8]. Isti algoritam je u tri opisana testna scenarija pokazao manju autonomiju od predstavljene. Važno je napomenuti da je model treniran na drugačijem skupu podataka, uz to i testiran na drugačijim testnim sekvencama. Isti model je u AirSim simulatoru pokazao lošije rezultate u svim provedenim testovima. Iako su

korištene iste mape za testiranje, različiti grafički prikaz simulacije je utjecao na rezultate. U svrhu usporedbe kako kvaliteta ulazne slike utječe na procjenu *PilotNet* mreže, preuzet je po jedan video okvir (200x66) s kamere svakog simulatora (u što sličnijim uvjetima). Odabrani video okviri su predani kao ulazni podatak *PilotNet* modelu, a zabilježene su aktivacijske mape CNN. Skaliranjem i vizualizacijom aktivacijskih mapa u svakom konvolucijskom sloju je generirana prosječna slika na kojoj je vidljivo koje značajke sa slike imaju najveću aktivaciju, tj. koja područja ulazne slike najviše doprinose izlazu. Na slici 4.12. prikazana je usporedba vizualizacije aktivacijskih mapa iz prva dva konvolucijska sloja za slike iz CARLA simulatora i AirSim simulatora.

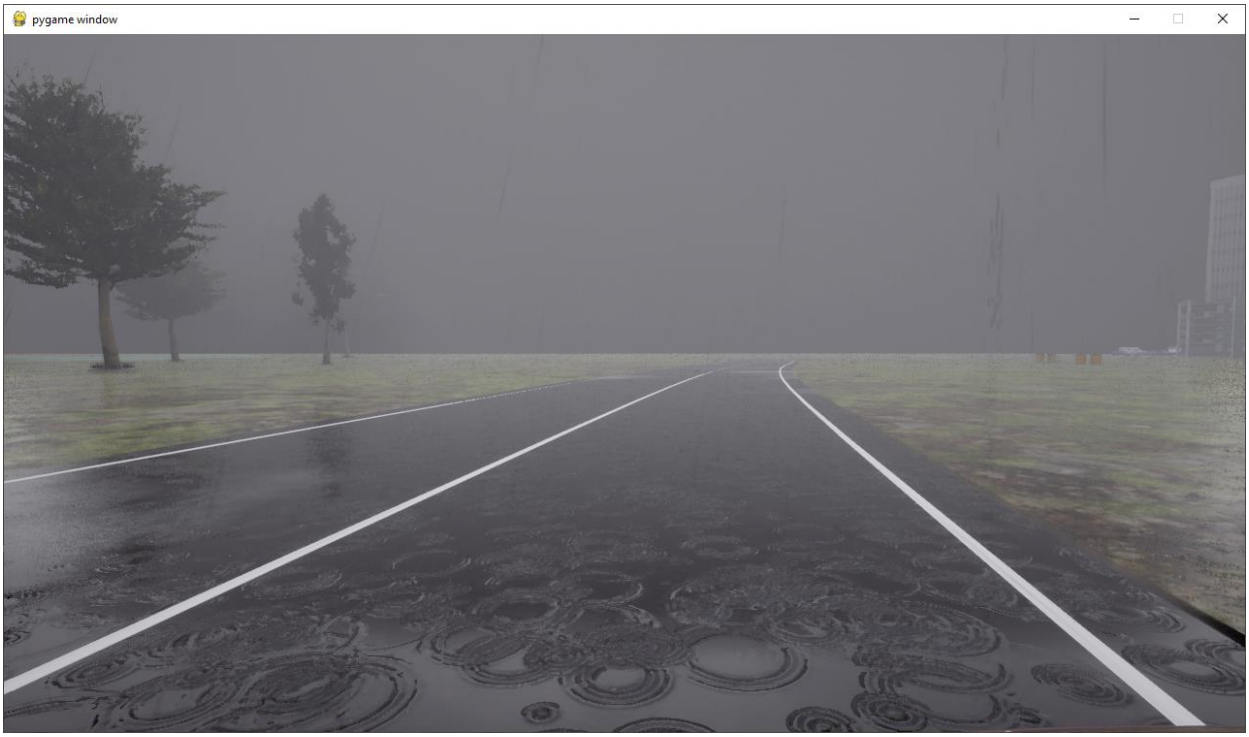


Sl. 4.12. Vizualizacija aktivacijskih mapa: (a) Normalizirani ulazni podatak iz CARLA simulatora, (b) normalizirani ulazni podatak iz AirSim simulatora, (c) prosječna slika prvog konvolucijskog sloja za ulazni podatak iz CARLA simulatora, (d) prosječna slika prvog konvolucijskog sloja za ulazni podatak iz AirSim simulatora, (e) prosječna slika drugog konvolucijskog sloja za ulazni podatak iz CARLA simulatora, (f) prosječna slika drugog konvolucijskog sloja za ulazni podatak iz AirSim simulatora

Na primjeru iz slike 4.12 vidljivo je da je već u vizualizaciji ulaznog sloja, u slučaju kada je ulazni video okvir iz AirSim simulatora (4.12 (b)), desna linija prometne trake slabije vidljiva. Nakon

prvog konvolucijskog sloja, desna linija prometne trake je vidljiva u slučaju video okvira iz CARLA simulatora (4.12.(c)), dok nije vidljiva u slučaju video okvira iz AirSim simulatora (4.12(d)). Model je u slici iz AirSim simulatora pronalazio značajke koje predstavljaju rub kolnika, dok je model u slici iz CARLA simulatora uspješno pronašao značajke koje predstavljaju oznake prometne trake. Samim time je model u većini slučajeva lošije pozicionirao vozilo u AirSim simulatoru što može objasniti lošije rezultate testa. Trenirani model je testiran u velikom broju različitih uvjeta i situacija, koje nije moguće jednostavno postići u stvarnom svijetu. Pokazao se poprilično uspješnim, no postoje nedostaci koje je moguće popraviti u budućem radu. Zbog manjka prikladnih i dostupnih skupova podataka, model se u maloj mjeri pokazao osjetljivim na promjenjive vremenske uvjete koje je teško nadomjestiti augmentacijom podataka. Također, prikazano je lošije ponašanje vozila u situacijama pri uskim i naglim zavojima. Navedeni problem bi mogao biti riješen korištenjem skupa podataka za treniranje snimljenih s većim brojem postavljenih kamera na vozilu.

Općenito, korištenje simulatora daje mogućnost ponavljanja testova koristeći različite scenarije, stvaranje prilagođenih scenarija po potrebi za testiranje sadašnjih i budućih autonomnih sustava te mogućnost trenutnog izvršavanja različitih testova. To je brza, pouzdana i učinkovita metoda testiranja koja daje točne rezultate. Oba korištena simulatora pružaju potrebna sredstva za rad algoritma koristeći dostupne senzore. Algoritam je treniran na skupu podataka iz stvarnog svijeta, a samim time su bile potrebne modifikacije slanja upravljačke naredbe: CARLA simulator se pokazao osjetljiviji i preciznije zakreće vozilo u odnosu na AirSim simulator. Oba simulatora sadrže različite vremenske uvjete. Najveći nedostatak vremenskih uvjeta u CARLA simulatoru je taj što nema implementiranu mogućnost snježnih uvjeta, dok je kod AirSima osnovni problem nemogućnost promjene doba dana u simulatoru, prvenstveno nedostaje mogućnost noćne vožnje. Iako AirSim ima implementirane snježne, kišne i ostale vremenske uvjete, pokazao je jako malu razinu realizma u odnosu na CARLA simulator. Na slici 4.13 je usporedba kiše iz CARLA i AirSim simulatora. U AirSimu se vidi padanje kapi kiše na podlogu, međutim simulator ne generira lokve i ne prikazuje refleksiju svjetlosti od mokru podlogu, a uz to ne mijenja se osvjetljenje potpunog scenarija.



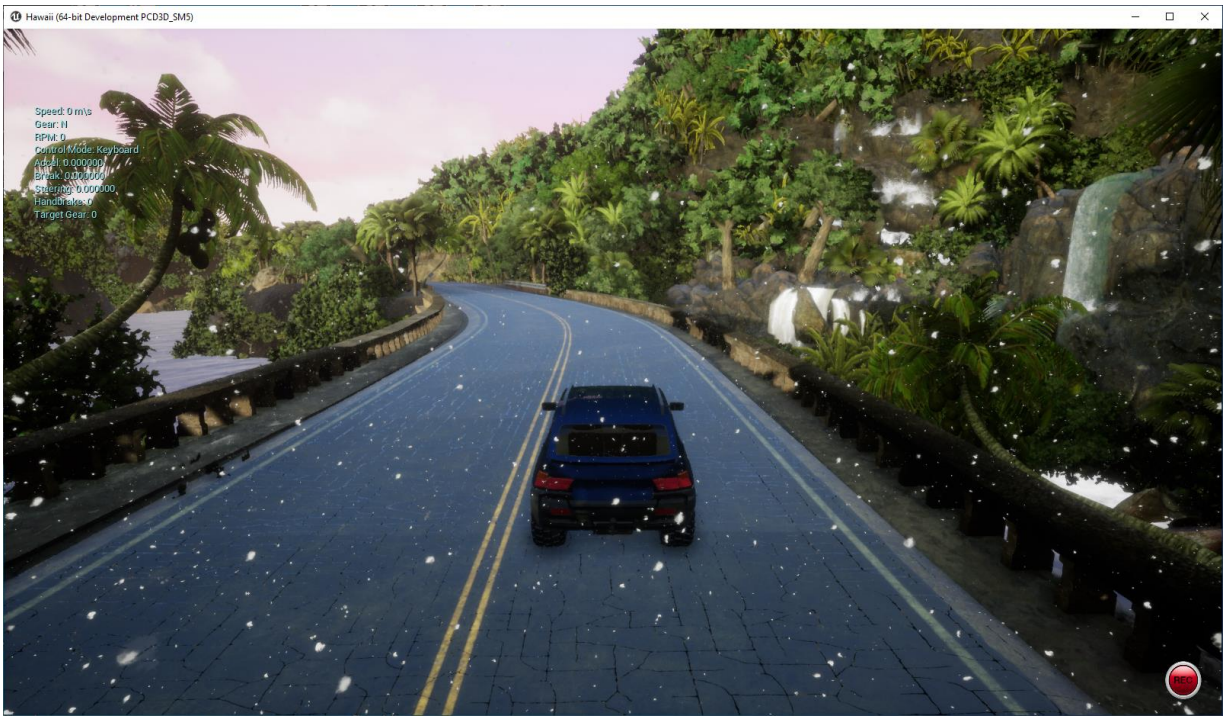
a)

b)

Sl. 4.13. Primjer kišnih uvjeta: (a) CARLA simulator, (b) AirSim simulator

Prednost AirSima u odnosu na CARLA simulator je prikaz snijega u vožnji. CARLA ima različite otežane vremenske uvjete, ali niti jedan ne implementira navedene pojavu. No opet,

implementacija snijega nije na zadovoljavajućoj razini: snijeg se ne zadržava na podlozi ceste, a time ne utječe na vidljivost oznaka na kolniku. (Slika 4.14).



Sl. 4.14. Primjer snježnih vremenski uvjeta u AirSim simulatoru

5. ZAKLJUČAK

U ovom diplomskom radu istražen je i obrađen problem procjene kuta zakreta upravljača vozila. Prikazano je i objašnjeno programsko rješenje koje na temelju ulazne slike s prednje strane vozila samostalno odlučuje o zakretu upravljača i upravlja vozilom. Algoritam je zasnovan na *PilotNet* arhitekturi konvolucijske neuronske mreže, a treniranjem parametara mreže model učinkovito prepoznaje potrebne značajke iz kojih definira putanju kretanja vozila. Kako bi testirali ispravnost treniranog modela, algoritam je prilagođen za rad u CARLA i AirSim simulatorima. Vožnja je provedena na trima stazama različitih oblika i karakteristika, a pritom su vladali različiti vremenski uvjeti. Algoritam je postigao autonomiju od 91.65 % u CARLA simulatoru i 71.87 % u AirSim simulatoru. *PilotNet* model se pokazao uspješnijim u CARLA simulatoru, gdje su vladali uvjeti sličniji onima u stvarnom svijetu. AirSim simulator nije pokazao zadovoljavajuću razinu grafičkog prikaza okruženja vozila u simulaciji, a time i *PilotNet* mreža nije prepoznavala potrebne značajke za točnije upravljanje vozilom. Testiranje algoritma u različitim testnim scenarijima je upravo pokazalo da postoje situacije gdje model učestalo griješi, a ustanovljeno je i različito ponašanje vozila u različitim vremenskim uvjetima.

Najveći razlog je upravo taj što je model treniran na podacima iz stvarnoga svijeta, a testiran u simulatorima. Između ostalog, korišteni skup podataka za treniranje ne sadrži slike koje prikazuju različite vremenske uvjete. Korištenje kvalitetnijih skupova podataka bi izravno poboljšalo ponašanje autonomnog vozila, no u nedostatku istih postignut je značajan rezultat. Nameće se pitanje kakvo bi ponašanje prikazao model u slučaju da je treniran koristeći skupove podataka pružene od strane simulatora. Takva analiza biti će provedena u budućem radu. Također, algoritam je demonstrirao mogućnost rada u stvarnom vremenu, što je od velike važnosti za ugradbene sustave u automobilskoj industriji. *PilotNet* čini dobru osnovu za autonomnu vožnju, ali procjenjuje kut zakreta upravljača za svaki video kadar (okvir) posebno, što je u suprotnosti s uobičajenim shvaćanjem principa kontinuirane vožnje. U tom području postoji mjesto za budući napredak. Vožnja se, osim prostornih značajki, može promatrati i kroz vremensku domenu što bi se budućem radu moglo implementirati u samu arhitekturu mreže. Korišteni simulatori su pokazali velik broj pogodnosti za razvoj algoritama autonomne vožnje. Sadrže detaljnu dokumentaciju, podržavaju sve potrebne vrste senzora i omogućuju jednostavnu implementaciju upravljačkog algoritma. CARLA simulator sadrži realnije postavke vizualne scene, dok AirSim pokazuje nižu razinu realizma, što može biti presudno u algoritmima koji se zasnivaju na obradi slike.

LITERATURA

[1] S. Singh, „Critical reasons for crashes investigated in the national motor vehicle crash causation survey“.

[2] „ISO 26262: Road Vehicles – Functional Safety, International Organization for Standardization“,.

[3] V. Umamaheswari, S. Amarjyoti, T. Bakshi, i A. Singh, „Steering angle estimation for autonomous vehicle navigation using hough and Euclidean transform“, u *2015 IEEE International Conference on Signal Processing, Informatics, Communication and Energy Systems (SPICES)*, Kozhikode, India, velj. 2015, str. 1–5. doi: 10.1109/SPICES.2015.7091469.

[4] M. R. Rochan, K. A. Alagammai, i J. Sujatha, „Enhanced navigation using computer vision-based steering angle calculation for autonomous vehicles“, *Encycl. Semantic Comput. Robot. Intell.*, sv. 02, izd. 01, str. 1850007, lip. 2018, doi: 10.1142/S2529737618500077.

[5] *Gazebo simulator*. [Na internetu]. Dostupno na: https://gazebosim.org/docs

[6] Pawlicki, Dan-Shyang Lee, Hull, i Srihari, „Neural network models and their application to handwritten digit recognition“, u *IEEE International Conference on Neural Networks*, San Diego, CA, USA, 1988, str. 63–70 sv.2. doi: 10.1109/ICNN.1988.23913.

[7] Pomerleau, Gusciora, Touretzky, i Kung, „Neural network simulation at Warp speed: how we got 17 million connections per second“, u *IEEE International Conference on Neural Networks*, San Diego, CA, USA, 1988, str. 143–150 sv.2. doi: 10.1109/ICNN.1988.23922.

[8] A. Pomerleau, „ALVINN: An Autonomous Land Vehicle In a Neural Network“.

[9] M. Bojarski *i ostali*, „End to End Learning for Self-Driving Cars“. arXiv, 25. travanj 2016. Pristupljeno: 02. rujan 2022. [Na internetu]. Dostupno na: http://arxiv.org/abs/1604.07316

[10] U. M. Gidado, H. Chiroma, N. Aljojo, S. Abubakar, S. I. Popoola, i M. A. Al-Garadi, „A Survey on Deep Learning for Steering Angle Prediction in Autonomous Vehicles“, *IEEE Access*, sv. 8, str. 163797–163817, 2020, doi: 10.1109/ACCESS.2020.3017883.

[11] S. Sharma, G. Tewolde, i J. Kwon, „Behavioral Cloning for Lateral Motion Control of Autonomous Vehicles Using Deep Learning“, u *2018 IEEE International Conference on Electro/Information Technology (EIT)*, Rochester, MI, svi. 2018, str. 0228–0233. doi: 10.1109/EIT.2018.8500102.

[12] *TORCS - The Open Racing Car Simulator*. [Na internetu]. Dostupno na: https://sourceforge.net/projects/torcs/

[13] Z. Chen i X. Huang, „End-to-end learning for lane keeping of self-driving cars“, u *2017 IEEE Intelligent Vehicles Symposium (IV)*, Los Angeles, CA, USA, lip. 2017, str. 1856–1860. doi: 10.1109/IVS.2017.7995975.

[14] *Comma.ai dataset*. [Na internetu]. Dostupno na: https://github.com/commaai/comma2k19

[15] L. Chi i Y. Mu, „Deep Steering: Learning End-to-End Driving Model from Spatial and Temporal Visual Cues“. arXiv, 12. kolovoz 2017. Pristupljeno: 02. rujan 2022. [Na internetu]. Dostupno na: http://arxiv.org/abs/1708.03798

[16] B. Osinski *i ostali*, „Simulation-Based Reinforcement Learning for Real-World Autonomous Driving“, u *2020 IEEE International Conference on Robotics and Automation (ICRA)*, Paris, France, svi. 2020, str. 6411–6418. doi: 10.1109/ICRA40945.2020.9196730.

[17] *Carla simulator*. [Na internetu]. Dostupno na: https://carla.org/

[18] M. V. Smolyakov, A. I. Frolov, V. N. Volkov, i I. V. Stelmashchuk, „Self-Driving Car Steering Angle Prediction Based On Deep Neural Network An Example Of CarND Udacity Simulator“, u *2018 IEEE 12th International Conference on Application of Information and Communication Technologies (AICT)*, Almaty, Kazakhstan, lis. 2018, str. 1–5. doi: 10.1109/ICAICT.2018.8747006.

[19] *Udacity Simulator*. [Na internetu]. Dostupno na: https://github.com/udacity/self-driving-car-sim

[20] V. Rausch, A. Hansen, E. Solowjow, C. Liu, E. Kreuzer, i J. K. Hedrick, „Learning a deep neural net policy for end-to-end control of autonomous vehicles“, u *2017 American Control Conference (ACC)*, Seattle, WA, USA, svi. 2017, str. 4914–4919. doi: 10.23919/ACC.2017.7963716.

[21] *CarSim simulator*. [Na internetu]. Dostupno na: https://www.carsim.com/

[22] *AirSim simulator*. [Na internetu]. Dostupno na: https://microsoft.github.io/AirSim/

[23] Y. Chen *i ostali*, „LiDAR-Video Driving Dataset: Learning Driving Policies Effectively“, u *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Salt Lake City, UT, USA, lip. 2018, str. 5870–5878. doi: 10.1109/CVPR.2018.00615.

SAŽETAK

U ovom diplomskom radu opisan je problem procjene kuta zakreta upravljača vozila, što uključuje pregled postojećih rješenja za procjenu kuta zakreta upravljača vozila zasnovanih na računalnom vidu, dubokom učenju i podržanom učenju. Dan je osvrt na postojeće simulatore koji omogućuju razvoj algoritama autonomne vožnje. Uspoređeni su CARLA i AirSim simulatori koji su odabrani kao testno okruženje algoritma za autonomnu vožnju. Opisana je implementacija odabranog *PilotNet* algoritma u simulatore, s ciljem usporedbe mogućnosti koje pružaju navedeni simulatori. Odabrani algoritam procjenjuje kut zakreta upravljača vozila na temelju slike s vozila koju pružaju simulatori, a zatim se šalje procijenjeni kut zakreta upravljača vozila u obliku upravljačke naredbe vozila. Koristeći prednosti koje pružaju simulatori, algoritam je testiran u stvarnom vremenu na simuliranim cestama. Algoritam je testiran na ukupno 198 km vožnje u dvama različitim simulatorima, a pri tome je model 91.65 % vremena uspješno upravljao vozilom u CARLA simulatoru, dok je 78.87 % vremena ispravno upravljao vozilom u AirSim simulatoru. Ispitane su različite pogodnosti koje navedeni simulatori pružaju za razvoj i testiranje algoritama autonomne vožnje. CARLA simulator se pokazao pogodniji za testiranje i razvoj algoritama autonomne vožnje koji se zasnivaju na obradi slike. Sadrži realnije postavke vizualne scene i sadrži veći broj različitih vremenskih uvjeta.

Ključne riječi: konvolucija, neuronska mreža, CARLA, AirSim, simulator, autonomna vožnja

COMPARISON OF THE PERFORMANCE OF AUTONOMOUS DRIVING ALGORITHM BASED ON VEHICLE STEERING ANGLE PREDICTION IN DIFFERENT SIMULATORS FOR AUTONOMOUS DRIVING ALGORITHMS DEVELOPMENT

ABSTRACT

This graduate thesis describes the problem of vehicle steering angle prediction, which includes a review of existing solutions for vehicle steering angle prediction based on computer vision, deep learning and Reinforcement learning. An overview of existing simulators that enable the development of autonomous driving algorithms is given. CARLA and AirSim simulators, which were selected as the test environment of the autonomous driving algorithm, were compared. The implementation of the selected PilotNet algorithm in simulators is described, with the aim of comparing the possibilities provided by the mentioned simulators. The selected algorithm predicts the steering angle of the vehicle based on the image from the vehicle, provided by the simulators, and then the predicted steering angle of the vehicle is sent in the form of a vehicle control command. Taking advantage of simulators, the algorithm was tested in real time on simulated roads. The algorithm was tested on a total of 198 km of driving in two different simulators, and the model successfully controlled the vehicle 91.65% of the time in the CARLA simulator, while it controlled the vehicle correctly in the AirSim simulator 78.87% of the time. The various benefits that the mentioned simulators provide for the development and testing of autonomous driving algorithms have been examined. The CARLA simulator proved to be more suitable for testing and developing autonomous driving algorithms based on image processing. It contains more realistic visual scene settings and contains a greater number of different weather conditions.

Keywords: convolution, neural network, CARLA, AirSim, simulator, autonomous driving

ŽIVOTOPIS

David Dumančić rođen je 12. veljače 1998. godine u Požegi. 2016 godine po završetku Tehničke škole Požega upisuje preddiplomski sveučilišni studij na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku, smjer računarstvo. 2020. godine stječe akademski naziv sveučilišni prvostupnik (lat. *baccalaureus*) inženjer računarstva. Iste godine upisuje diplomski sveučilišni studij automobilsko računarstvo i komunikacije.

Potpis:

_____

PRILOZI

- P.3.1. – Konačni skup podataka (elektronički prilog)
- P.3.2. – Parametri modela algoritma za upravljanje modelom vozila zasnovanog na procjeni kuta zakreta upravljača vozila (elektronički prilog)
- P.3.3. – Virtualno okruženje CARLA simulatora (Mapa Test1) (elektronički prilog)
- P.3.4. – Virtualno okruženje CARLA simulatora (Mapa Test2) (elektronički prilog)
- P.3.5. – Virtualno okruženje CARLA simulatora (Mapa Test3) (elektronički prilog)
- P.3.6. – Virtualno okruženje AirSim simulatora (Mapa Test1) (elektronički prilog)
- P.3.7. – Virtualno okruženje AirSim simulatora (Mapa Test2) (elektronički prilog)
- P.3.8. – Virtualno okruženje AirSim simulatora (Mapa Test3) (elektronički prilog)
- P.3.9 – Skripta za upravljanje vozilom u CARLA simulatoru – *drive_model_carla.py* (elektronički prilog)
- P.3.10 – Skripta za upravljanje vozilom u AirSim simulatoru – *drive_model_airsim.py* (elektronički prilog)