Multimodalno autonomno upravljanje vozilom od kraja do kraja

Barić, Matija

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

Sveučilišni diplomski studij Automobilsko računarstvo i komunikacije

Multimodalno autonomno upravljanje vozilom od kraja do kraja

Diplomski rad

Matija Barić

Osijek, 2024. godina.

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

Obrazac D1: Obrazac za ocjenu diplomskog rada na sveučilišnom diplomskom studiju

Ocjena diplomskog rada na sveučilišnom diplomskom studiju

Ime i prezime pristupnika:	Matija Barić
Studij, smjer:	Sveučilišni diplomski studij Automobilsko računarstvo i
Mat. br. pristupnika, god.	D65 ARK, 07.10.2022.
JMBAG:	0165081298
Mentor:	prof. dr. sc. Marijan Herceg
Sumentor:	
Sumentor iz tvrtke:	Zvonimir Kaprocki
Predsjednik Povjerenstva:	izv. prof. dr. sc. Ratko Grbić
Član Povjerenstva 1:	prof. dr. sc. Marijan Herceg
Član Povjerenstva 2:	prof. dr. sc. Mario Vranješ
Naslov diplomskog rada:	Multimodalno autonomno upravljanje vozilom od kraja do kraja
Znanstvena grana diplomskog rada:	**Telekomunikacije i informatika (zn. polje elektrotehnika)**
Zadatak diplomskog rada:	Paradigma vožnje od kraja do kraja (engl. end-to-end) odnosi se na učenje holističkih modela koji mogu izravno preslikati neobrađene podatke sa različitih senzora u upravljačke signale autonomnog vozila. Modeli vožnje od kraja do kraja mogu prihvatiti navigacijske naredbe visoke razine ili biti ograničeni na specifične navigacijske pod-zadatke kao što su zadržavanje u voznoj traci i uzdužna kontrola. U okviru ovog diplomskog rada potrebno je razviti model zasnovan na dubokom učenju koji će omogućiti autonomnu vožnju vozila u CARLA simulatoru. Model je prvotno potrebno trenirati
Datum ocjene pismenog dijela diplomskog rada od strane mentora:	20.09.2024.
Ocjena pismenog dijela diplomskog rada od strane mentora:	Izvrstan (5)
Datum obrane diplomskog rada:	30.09.2024.
Ocjena usmenog dijela diplomskog rada (obrane):	Izvrstan (5)
Ukupna ocjena diplomskog rada:	Izvrstan (5)
Datum potvrde mentora o predaji konačne verzije diplomskog rada čime je pristupnik završio sveučilišni diplomski studij:	30.09.2024.

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

IZJAVA O IZVORNOSTI RADA

Osijek, 30.09.2024.

Ime i prezime Pristupnika:	Matija Barić
Studij:	Sveučilišni diplomski studij Automobilsko računarstvo i komunikacije
Mat. br. Pristupnika, godina upisa:	D65 ARK, 07.10.2022.
Turnitin podudaranje [%]:	13

Ovom izjavom izjavljujem da je rad pod nazivom**: Multimodalno autonomno upravljanje vozilom od kraja do kraja**

izrađen pod vodstvom mentora prof. dr. sc. Marijan Herceg

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 pristupnika:

Sadržaj

1. UVOD

Konstantno povećanje sudionika u prometu i razvoj tehnologije vodilo je do bržeg i jednostavnijeg transporta ljudi i robe. Nastankom prvoga automobila s unutarnjim izgaranjem, koji je bio u potpunosti mehanički, patentiran 1885. godine od strane Karla Benza, pa sve do danas, proizvođači automobila nastoje poboljšati sigurnost putnika u svojim vozilima [1]. Neki od prvih i najznačajnijih sigurnosnih elektronskih sustava u vozilu su elektronska kontrola stabilnosti (engl. *Electronic Stability Control* - ESC) te sustav protiv blokiranja kotača (engl. *Anti-lock Braking System* - ABS). Kako bi se poboljšala sigurnost putnika u prometu, konstantno se radi na razvoju novih sustava za sigurnost i pomoć vozaču (engl. *Advanced Driver Assistance Systems* - ADAS). Nacionalna uprava za sigurnost u cestovnom prometu (engl. *National Highway Traffic Safety Administration* - NHTSA) je objavila podatke o razvoju sigurnosti za putnike u vozilima koji prikazuju kako se godišnje broj spašenih života povećao sa 115 na 27621 spašenih života kroz period od 1960. godine do 2012. godine na području Sjedinjenih Američkih Država. Njihova procjena je da je razvoj sigurnosnih sustava za putnike u vozilima spasio oko 600 tisuća života u spomenutom periodu [2]. Prema istraživanjima NHTSA organizacije, 94% nesreća motornih vozila u prometu uzrokovano je ljudskom greškom [3].

Danas veliki broj proizvođača automobila, kao i drugih poduzeća u automobilskoj industriji, radi na razvoju sustava autonomne vožnje s ciljem povećanja sigurnosti putnika u prometu, te u konačnici smanjivanje broja prometnih nesreća i stradalih. Da bi se to ostvarilo, sustavi u automobilu moraju funkcionirati zajedno i komunicirati s raznim senzorima koji su ugrađeni u vozila namijenjena za tu svrhu kao što su kamere, radar, LiDAR i ostali senzori [4]. Često se koristi komunikacija s globalnim sustavom za pozicioniranje (engl. *Global Positioning System* - GPS) te komunikacija s infrastrukturom kako bi automobil dobio informacije koje su bitne za autonomnu vožnju. Sve informacije iz okoline i senzora prosljeđuju se softveru koji je uglavnom zasnovan na umjetnoj inteligenciji (engl. *artificial intelligence* - AI) i na temelju tih informacija softver donosi odluke o upravljanju automobilom. Na ideji autonomne vožnje temelji se izrada ovoga diplomskog rada, koji je izrađen kako bi funkcionirao u simulatoru vožnje, CARLA simulatoru [5]. CARLA je simulator otvorenog koda za istraživanje autonomne vožnje, razvijena kako bi podržala razvoj, obuku i validaciju sustava za autonomnu vožnju.

Glavni cilj ovog rada je izrada modela temeljenog na umjetnoj inteligenciji, koji je rezultat treniranja konvolucijske neuronske mreže (engl. *Convolutional neural network* - CNN). Model omogućuje potpunu autonomnu kontrolu vozila unutar simulacijskog okruženja. U radu je razvijen

i opisan model koji koristi multimodalni pristup, integrirajući podatke s više senzora, kao što su RGB i dubinska kamera, kako bi se postigla što preciznija i sigurnija kontrola nad vozilom. Model također komunicira sa upraviteljem prometa koji je integriran unutar simulacijskog okruženja s ciljem navigacije vozila po mapi.

U drugom poglavlju rada objašnjen je problem izrade algoritma za autonomnu vožnju u simulatoru vožnje te je dan pregled postojećih rješenja za multimodalnu autonomnu vožnju od kraja do kraja u simulatoru vožnje. U trećem poglavlju je opisan postupak izrade vlastitog rješenja, prikazana su rješenja poput programskog koda za skupljanje baze podataka, treniranje modela neuronske mreže, izgled baze podataka, te dodatni pojmovi. Četvrto poglavlje predstavlja pregled testnog okruženja, CARLA simulatora, objašnjen je postupak testiranja, predstavljeni su rezultati testiranja te su predložene smjernice unaprijeđena algoritma. Peto poglavlje sadrži zaključak rada.

2. PROBLEM MULTIMODALNOG AUTONOMNOG UPRAVLJANJA VOZILA

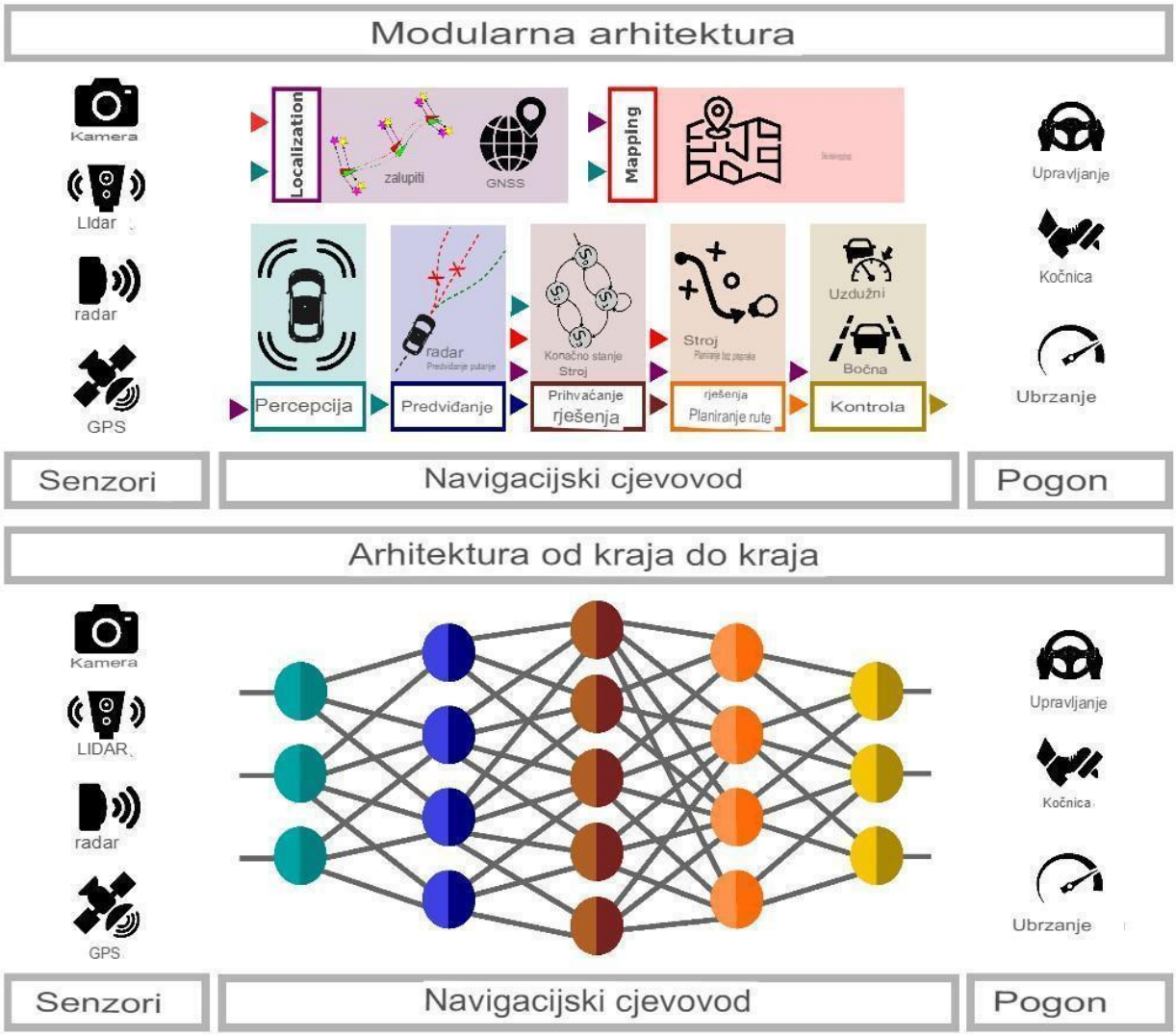
2.1. Pregled postojećih rješenja

U ovome potpoglavlju predstavljen je pregled postojećih rješenja koji su za isti problem multimodalne *end-to-end* autonomne vožnje predložili slična rješenja koristeći CARLA simulator kao testno okruženje.

Znanstveni rad pod nazivom „Multimodal End-to-End Autonomous Driving“ autora Yi Xiao, Felipe Codevilla, Akhil Gurram, Onay Urfalioglu i Antonio M. López objavljen u IEEE Transactions on Intelligent Transportation System je određen kao referentni rad za izradu ovoga diplomskoga rada [6]. U svome radu autori su istaknuli kako je umjetna inteligencija ključna komponenta u upravljanju autonomnih vozila te da postoje dva glavna pristupa razvoja umjetne inteligencije za upravljanje autonomnim vozilima. Prvi pristup je modularni pristup, drugi je *end-to-end* pristup. Modularni pristup razlaže glavni zadatak na više podzadataka. Tako se zadatak autonomne vožnje u modularnom pristupu dijeli na podzadatke kao što su percepcija okoline, planiranje manevara te sama kontrola autonomnog vozila. Ovaj pristup predstavlja podzadatke kao module gdje je svaki modul zadužen za određeni aspekt vožnje, time omogućujući veću mogućnost objašnjenja odluke koju umjetna inteligencija donosi. S druge strane, svaki pojedinačni modul zahtijeva veliku količinu označenih podataka, što povećava složenost i produžuje vrijeme potrebno za razvoj cjelokupnog sustava. *End-to-end* pristupom se želi postići izravno mapiranje svih podataka s dostupnih senzora u kontrolne vrijednosti za upravljanje automobilom bez razdvajanja na podzadatke. Ovakav pristup je donedavno bio manje poznat, ali je u zadnje vrijeme postao popularan zbog svoje učinkovitosti, posebno zato što smanjuje vrijeme potrebno za označavanje podataka. Označavanje podataka odnosi se na proces ručnog označavanja značajki u podacima (slike, videozapisi i ostalo) kako bi se te značajke prepoznale i koristile u treningu modela. Na primjer, ručno označavanje objekata kao što su pješaci, semafori i druga vozila na slikama u svrhu treniranja. Modularni pristup zahtijeva veliku količinu označenih podataka za svaki pojedinačni modul (npr. percepcija objekata, planiranje ruta, kontrola vozila). To uključuje detaljan rad ljudi koji pregledavaju svaku sliku ili videozapis i ručno označavaju elemente koji su relevantni za određeni modul. Ovaj proces je vrlo dugotrajan. *End-to-end* pristup radi na način da izravno mapira ulazne podatke iz senzora u izlazne kontrolne signale (gas, kočnica i kut zakreta volana) bez potrebe za razlaganjem zadatka na podzadatke. Model uči na temelju neoznačenih podataka

umjesto detaljno označenih podataka. *End-to-end* modeli za svoj rad vrlo često koriste CNN za potrebe obrade podataka dobivenih sa senzora te za samo donošenje odluka o upravljanju vozilom.

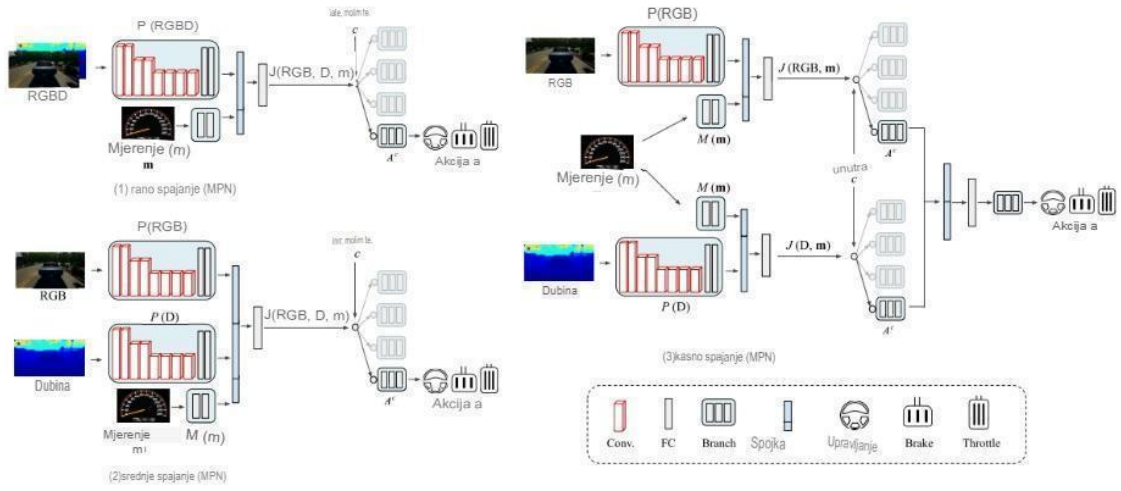
U znanstvenom radu naziva „Integrating Modular Pipelines with End-to-End Learning: A Hybrid Approach for Robust and Reliable Autonomous Driving Systems“ je slikovito i jasno prikazana razlika između ova dva pristupa [7]. Slika 2.1. prikazuje razliku između modularnog i *end-to-end* pristupa za autonomnu vožnju.



Slika 2.1: *Razlika u radu između modularne i end-to-end arhitekture neuronske mreže za upravljanje autonomnim vozilom*

U radu [6], autori su se fokusirali na multimodalni *end-to-end* pristup, te takav pristup istovremeno obrađuje podatke s više senzora kao što su RGB kamera i dubinski senzori. Dubinskim senzorima se smatraju LiDAR i dubinska kamera u CARLA simulatoru. Korištenjem toga istoga simulatora, autori su radili na poboljšanju performansi modela umjetne inteligencije

korištenjem više tipova senzora za razliku od modela koji koristi samo jedan senzor koji je uobičajeno RGB kamera. U radu su objašnjene razlike između rane, srednje i kasne fuzije podataka te su dani rezultati za svaki od modela s različitim pristupom. Njihovi rezultati pokazuju kako je rana fuzija podataka koristeći multimodalni pristup značajno poboljšala performanse modela u uvjetima simulacije naspram srednje i kasne fuzije podataka. Na slici 2.2. koja je preuzeta iz spomenutoga rada prikazana je razlika između rane, srednje i kasne fuzije podataka koristeći multimodalni pristup.



Slika 2.2: *Prikaz razlike između rane, srednje i kasne fuzije podataka koristeći multimodalni pristup za autonomnu vožnju*

Dodatno, u radu nije korišten samo multimodalni pristup za autonomnu vožnju nego su autori također testirali model koji koristi samo jedan senzor, RGB kameru, gdje se dubinski podaci procjenjuju iz RGB slika. Takav pristup smanjuje troškove i kompleksnost opreme koja je potrebna za autonomna vozila. Autori su naveli dodatne smjernice za daljnje istraživanje i mogućnosti za razvoj u ovome području.

Drugi rad koji je proučavan je rad pod nazivom „Multi-Modal End-To-End Autonomous Driving via Conditional Imitation Learning“ [8]. Autori rada su Alex Zuzow i Charles Nimo. Njihov rad istražuje utjecaj različitih senzora na performanse modela treniranoga za autonomnu vožnju također koristeći CARLA simulator. U radu se ističe kako je ključan izazov u razvoju autonomnih vozila percepcija okoline oko autonomnog vozila. Autori su istražili kako uklanjanje i dodavanje istoga senzora koristeći ranu multimodalnu fuziju utječe na performanse modela u kontekstu uvjetovanog oponašanja učenja (engl. *Conditional Imitation Learning*- CIL). U radu su korištena četiri senzora ili modaliteta a to su RGB slika, LiDAR, optički tok i brzina vozila. Rezultati su dobiveni koristeći dvije arhitekture enkodera. Prvi način je koristeći unaprijed

treniranu *EfficientNet* mrežu, a drugi način je *EfficientNet* mreža, ali se na posljednjem bloku povezuje transformer. Iz rezultata koje su dobili zaključuju kako optički tok poboljšava performanse modela, ali zbog oštrih augmentacija RGB slika, model je nestabilan tokom treniranja.

Treći, znanstveni rad je znanstveni rad naziva „Urban Driving with Conditional Imitation Learning“ autora Jeffrey Hawke, Richard Shen, Corina Gurau, Siddharth Sharma, Daniele Reda, Nikolay Nikolov, Przemyslaw Mazur, Sean Micklethwaite, Nicolas Griffiths, Amar Shah i Alex Kendall [9]. Spomenuti rad istražuje izazove koje donosi autonomna vožnja u kompliciranim urbanim okruženjima. Autori koriste metode uvjetovanog oponašanja učenja tako što se oponaša ljudska vožnja u simulatoru. Model uči na temelju stvarnih podataka prikupljenih tijekom vožnje. Taj pristup omogućava modelu da se prilagodi raznim situacijama bez potrebe za ručnim označavanjem reakcija, te tako čini sustav fleksibilnijim i robusnijim. Podaci u bazu podataka se spremaju tijekom vožnje, mapirajući senzorske ulaze kao što su RGB slika i kontrole kojima se upravlja automobil (gas, kočnica i kut zakreta volana) i istovremeno spremajući naredbe o ruti kojom automobil vozi (skretanje lijevo, desno, prolazak ravno). Model se trenirao na podacima koji su sakupljeni tijekom 30 sati vožnje u simulatoru od strane čovjeka. Model se sastoji od nekoliko ključnih komponenti: obrade podataka sa senzora, integracije informacija iz različitih senzora te modula za upravljanje. Duboka neuronska mreža koristi ulazne RGB slike za izdvajanje relevantnih značajki, dok se integracijom senzorskih podataka kombiniraju informacije sa svih dostupnih senzora. Modul za upravljanje koristi ove obrađene podatke zajedno s uputama o ruti kako bi donio odluke o kretanju vozila. Podatkovni skup uključuje slike RGB kamere koje snimaju lijevo, naprijed i desno uz podatke o brzini i upravljanju. Autori su dokazali kako je ovim pristupom moguće stvoriti model koji koristi prethodno navedene informacije te tako kontrolirati automobil. Dodaju kako u budućnosti bi se mogao poboljšati model koristeći učenje iz ljudskih intervencija u prometu te unaprjeđivanjem robusnosti modela.

3. PRIJEDLOG VLASTITOG RJEŠENJA AUTONOMNOG UPRAVLJANJA VOZILOM KORISTEĆI END-TO-END PRISTUP

3.1 Opis korištenih tehnologija

Python

Python je interpretirani, interaktivni, objektno orijentirani programski jezik koji uključuje module, iznimke, dinamičke tipove podataka vrlo visoke razine i klase. Podržava više programskih paradigmi izvan objektno orijentiranog programiranja, kao što je proceduralno i funkcionalno programiranje. *Python* kombinira nevjerojatnu snagu s vrlo jasnom sintaksom. Ima sučelja za mnoga sistemske pozive i biblioteke, kao i za razne prozorske sustave, te je proširiv u C ili C++. Također se može koristiti kao jezik proširenja za aplikacije koje trebaju programibilno sučelje. *Python* je prenosiv: radi na mnogim operacijskim sustavima kao što su *Unix*, uključujući *Linux, macOS* te *Windows* [10].

Anaconda

Anaconda je distribucija *open-source* programskih jezika *Python* i *R* za podatkovnu znanost u svrhu upravljanja paketima i njihovu implementaciju. Verzijama paketa u *Anaconda-i* upravlja sustav za upravljanje paketima, *conda*, koji analizira trenutno okruženje prije izvođenja instalacije kako bi se izbjeglo ometanje drugih okvira i paketa. *Anaconda* distribucija dolazi s više od 250 automatski instaliranih paketa. Preko 7500 dodatnih *open-source* paketa može se instalirati iz *PyPI-a,* kao i paket *conda* i upravitelj virtualnog okruženja. Također uključuje grafičko korisničko sučelje (engl. *graphical user interface* - GUI), *Anaconda Navigator*, kao grafičku alternativu sučelju naredbenog retka. *Anaconda Navigator* uključen je u distribuciju *Anaconda* i omogućuje korisnicima pokretanje aplikacija i upravljanje *conda* paketima, okruženjima i kanalima bez korištenja naredbi naredbenog retka. Navigator može tražiti pakete, instalirati ih u okruženje, pokretati pakete i ažurirati ih [11].

Tensorflow

TensorFlow je popularan okvir za strojno i duboko učenje. *TensorFlow* je besplatna i otvorena biblioteka koja je objavljena 9. studenog 2015. godine, a razvilo ju je *Google Brain Team*. Biblioteka je bazirana je na programskim jezicima *Python, C++ i Java.* TensorFlow se oslanja na napredne algoritme i matematičke modele za obradu i analizu velikih količina podataka. Kroz numeričke izračune, TensorFlow omogućava preciznu manipulaciju podacima, optimizaciju

modela i brzo izvođenje složenih matematičkih operacija koje su ključne za treniranje i izvođenje modela strojnog učenja. Osim toga, protok podataka (engl. *data flow*) u TensorFlowu omogućava paralelno procesiranje i optimizaciju resursa, što dodatno ubrzava cijeli proces treniranja modela. Ove značajke omogućavaju korisnicima da implementiraju složene algoritme strojnog učenja na učinkovit i skalabilan način, čime se cjelokupni postupak razvoja modela strojnog učenja čini bržim i jednostavnijim.. *TensorFlow* može trenirati i pokretati duboke neuronske mreže za prepoznavanje slika, klasifikaciju rukom pisanih brojeva, rekurentne neuronske mreže, ugrađivanje riječi, obradu prirodnog jezika, detekciju videozapisa i još mnogo toga. *TensorFlow* se može pokretati na više središnjih procesorskih jedinica (engl. *Central Processing Unit*- CPU) ili grafičkih procesorskih jedinica (engl. *Graphical Processin Unit* - GPU), kao i na mobilnim operativnim sustavima. [12].

OpenCV

OpenCV (engl. *Open Source Computer Vision Library*) biblioteka je softvera za računalni vid i strojno učenje otvorenog koda. *OpenCV* je napravljen kako bi osigurao zajedničku infrastrukturu za aplikacije računalnog vida i ubrzao korištenje strojne percepcije u komercijalnim proizvodima. Biblioteka ima više od 2500 optimiziranih algoritama, što uključuje opsežan skup klasičnih i najsuvremenijih algoritama računalnog vida i strojnog učenja. Ovi se algoritmi mogu koristiti za otkrivanje i prepoznavanje lica, identifikaciju objekata, klasificiranje ljudskih radnji u videozapisima, praćenje pokreta kamere, praćenje pokretnih objekata, izdvajanje 3D modela objekata, proizvodnju 3D oblaka točaka iz stereo kamera, spajanje slika kako bi se proizvela visoka rezolucija sliku cijelog prizora, pronađite slične slike iz baze podataka slika, uklonite crvene oči sa slika snimljenih bljeskalicom, pratite pokrete očiju, prepoznajte krajolik i postavite markere za prekrivanje s proširenom stvarnošću, itd. *OpenCV* ima više od 47 tisuća korisnika zajednica i procijenjeni broj preuzimanja veći od 18 milijuna. Biblioteka se intenzivno koristi u tvrtkama, istraživačkim grupama i sličnim organizacijama [13].

CUDA

CUDA je platforma za razvoj softvera koja se koristi za ubrzavanje paralelnog računalstva. To je specijalizirani programski jezik za pisanje programa koji rade na GPU CUDA, a radi s većinom operativnih sustava. CUDA tehnologija omogućuje paralelnu obradu razbijanjem zadatka na tisuće manjih "niti" koje se izvode neovisno. NVIDIA CUDA je tehnologija koja postoji od sredine 2000-ih kada se prvi put pojavila kao način za poboljšanje performansi NVIDIA GPU-a. Danas ga koristi širok raspon industrija i sektora, uključujući, ali ne ograničavajući se na računalnu

grafiku, računalne financije, rudarenje podataka, strojno učenje i znanstveno računalstvo. CUDA je softverska platforma koja omogućuje ubrzano računanje. To je specijalizirani programski jezik koji radi na GPU CUDA, a radi s većinom operativnih sustava [14].

Numpy

NumPy je temeljni paket za znanstveno računalstvo u *Pythonu*. To je biblioteka *Pythona* koja pruža višedimenzionalni objekt niza, razne izvedene objekte (kao što su maskirani nizovi i matrice) i niz rutina za brze operacije na nizovima, uključujući matematičke, logičke, manipulaciju oblikom, sortiranje, odabir, I/O , diskretne Fourierove transformacije, osnovna linearna algebra, osnovne statističke operacije, slučajna simulacija i još mnogo toga. U središtu *NumPy* paketa nalazi se objekt *ndarray*. On enkapsulira n-dimenzionalne nizove homogenih tipova podataka, s mnogim operacijama koje se izvode u kompajliranome kodu [15].

3.1.1. CARLA simulator

CARLA je razvijena od temelja kako bi podržala razvoj, obuku i validaciju sustava za autonomnu vožnju. Uz *open-source* kod i protokole, CARLA nudi otvorene digitalne resurse (urbane planove, zgrade, vozila) koji su stvoreni u tu svrhu i mogu se slobodno koristiti. Platforma za simulaciju podržava fleksibilnu specifikaciju skupova senzora, uvjete okoline, potpunu kontrolu svih statičkih i dinamičkih aktera, generiranje karata i još mnogo toga [16]. Prednosti CARLA simulatora:

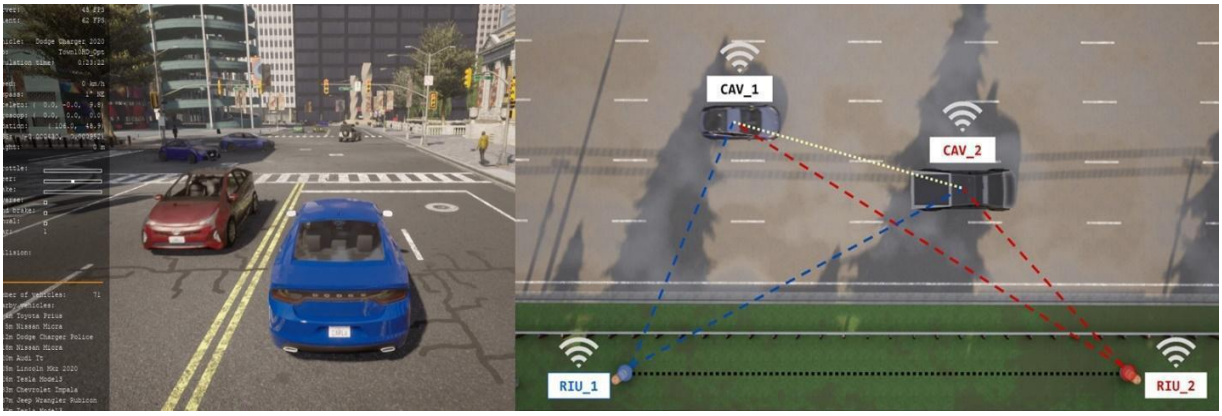
- Skalabilnost putem poslužiteljske *multi-client* arhitekture: više klijenata u istom ili u različitim čvorovima može kontrolirati različite aktere.
- Fleksibilni API: CARLA pruža snažan API koji korisnicima omogućuje kontrolu svih aspekata povezanih sa simulacijom, uključujući stvaranje prometa, ponašanje pješaka, vremenske prilike, senzore i još mnogo toga.
- Paket senzora za autonomnu vožnju: korisnici mogu konfigurirati različite pakete senzora uključujući LIDAR-e, više kamera, senzore dubine i GPS između ostalog.
- Brza simulacija za planiranje i kontrolu: ovaj način rada onemogućuje renderiranje kako bi se ponudilo brzo izvođenje simulacije prometa i ponašanja na cesti za koje grafika nije potrebna.
- Generiranje karata: korisnici mogu jednostavno kreirati vlastite karte prema standardu ASAM OpenDRIVE putem alata kao što je *RoadRunner*.

- Osnove autonomne vožnje: pružamo osnovne linije autonomne vožnje kao agente koji se mogu pokretati u CARLA-i, uključujući agenta *AutoWare* i agenta za učenje uvjetne imitacije [16].

Na slici 3.1. prikazan je izgled jedne od mapa unutar simulatora.

Slika 3.1: *Prikaz izgleda CARLA simulatora*

Na slici 3.2. prikazane su različite mogućnosti koje nudi CARLA simulator tijekom rada simulacije.



Slika 3.2: *Prikaz mogućnosti praćenja parametara kontroliranoga automobila za vrijeme izvođenje simulacije te prikaz komunikacije različitih objekata unutar simulacije*

Funkcionalnost CARLA-e opsežno je opisana u dokumentaciji. Nekoliko istaknutih detalja koji pokrivaju neke od najkorisnijih i najtraženijih značajki CARLA-e:

- Akteri: CARLA-ini akteri su entiteti koji međusobno djeluju unutar simulacije poput vozila, pješaka i prometnih signala.

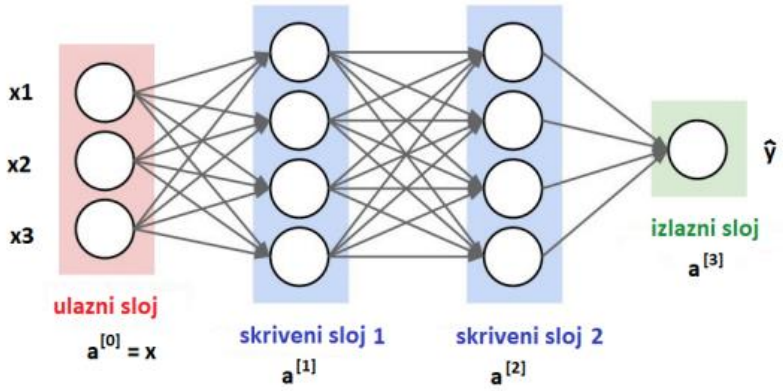
- Senzori: CARLA omogućuje korištenje niza modela senzora iz stvarnog svijeta kao što su kamere, LIDAR i RADAR.
- Upravitelj prometa: CARLA-in upravitelj prometa (engl. *Traffic Manager*) kontrolira NPC-ove (engl. *non-player character*) kako bi omogućio njihovu autonomnu vožnju.
- ROS most: CARLA-in *ROS* most omogućuje besprijekorno povezivanje s operativnim sustavom robota (engl. *Robot Operating System* - ROS).

Na slici 3.3. prikazan je rad nekoliko senzora koji su dodijeljeni promatranome automobilu. Može se vidjeti prikaz rada RGB kamere, LiDAR senzora i dubinske kamere te samih parametara automobila tijekom rada simulacije.

Slika 3.3: *Prikaz rada nekih od mogućih senzora koji prate promatrani automobil*

3.1.2. Opis konvolucijske neuronske mreže

Neuronske mreže dio su strojnog učenja (engl. *Machine Learning* - ML) i ključne su za algoritme dubokog učenja. Sastoje se od slojeva čvorova (engl. *node*), uključujući ulazni sloj, jedan ili više skrivenih slojeva te izlazni sloj. Svaki čvor je povezan s drugim čvorom i ima pridruženu težinu i prag. Ako izlaz nekog od čvora premaši određenu vrijednost praga, taj čvor se aktivira i šalje određeni signal u sljedeći sloj mreže s kojim je povezan. Ako se ne dogodi prijelaz vrijednosti praga, podaci se ne prosljeđuju dalje. Na slici 3.4. prikazana je pojednostavljena duboka neuronska mreža [17].



Slika 3.4: *Primjer duboke neuronske mreže s dva skrivena sloja i izlaznim slojem s jednim neuronom*

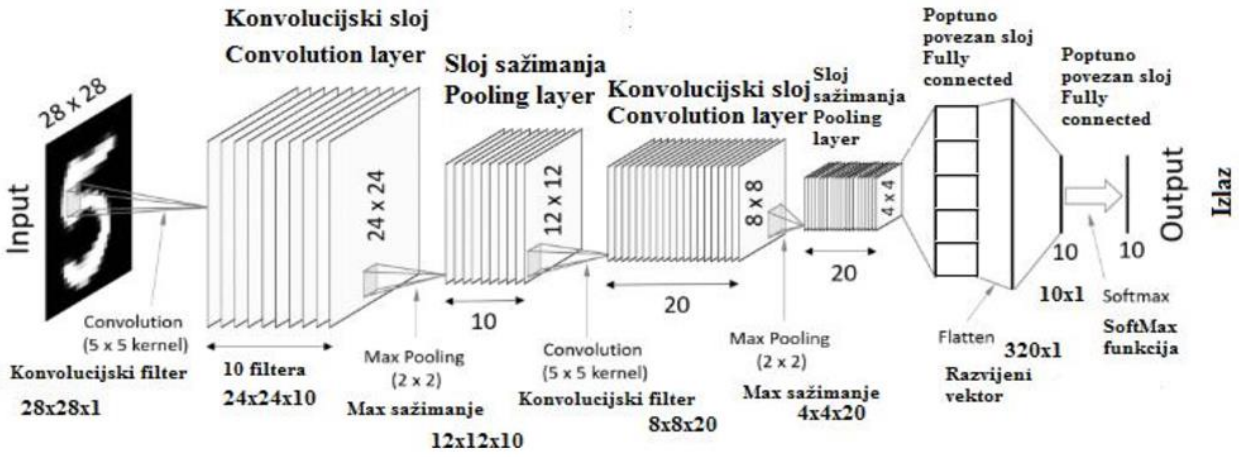
CNN često se koriste za zadatke klasifikacije objekata na slici i računalnog vida. Prije CNN, za identifikaciju objekata na slikama korištene su ručne, dugotrajne metode izdvajanja značajki. Međutim, CNN sada pružaju skalabilniji pristup klasifikaciji slika i zadacima prepoznavanja objekata, koristeći principe linearne algebre, posebno matričnog množenja za prepoznavanje uzoraka unutar slike. Takve metode mogu biti računalno zahtjevne, jer zahtijevaju grafičke procesorske za treniranje modela. CNN ima tri glavne vrste slojeva a to su: konvolucijski sloj (engl. *Convolutional layer*), sloj udruživanja (engl. *Pooling layer*) i potpuno povezani sloj (engl. *Fully connected layer*)

Konvolucijski slojevi mogu biti praćeni dodatnim konvolucijskim slojevima ili slojevima udruživanja, a potpuno povezani sloj je završni sloj. Krećući se kroz slojeve, CNN postupno prepoznaje složenije i apstraktnije značajke slike. Na početku, slojevi prepoznaju osnovne elemente poput rubova i kutova, dok kasniji slojevi prepoznaju složenije oblike i strukture unutar slike, poput dijelova objekata ili čak cijelih objekata. Raniji ili prvi slojevi fokusirani su na jednostavne značajke, kao što su boje i rubovi.

Konvolucijski sloj

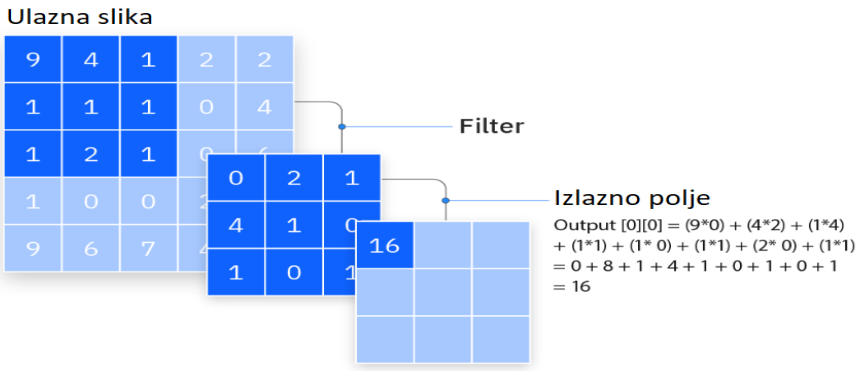
Konvolucijski sloj glavni je građevni blok CNN. Sadrži skup filtara (ili kernela), čije se parametre uči tijekom treniranja. Sloj se sastoji od ulaznih podataka, filtera i mape značajki. Ulaz u konvolucijski može biti RGB slika. Konvolucijski sloj funkcionira po principu da se detektor značajki (*kernel* ili filter) sustavno kreće preko slike, te tako analizira svaki dio slike u unaprijed određenim koracima, što se naziva *stride.* Filter je dvodimenzionalni niz težina, obično veličine 3x3. Proces kretanja filtera po slici i izračunavanja skalarnih produkata između vrijednosti piksela

i vrijednosti unutar filtera naziva se konvolucija. Konvolucija je matematička operacija koja omogućuje mreži da prepozna značajke u slikama ili drugim vrstama podataka, pri tome stvarajući izlaznu mapu značajki. Težine filtera ostaju fiksne tijekom svakog prolaza preko slike, ali se prilagođavaju tijekom procesa treniranja zbog optimizacije prepoznavanja značajki. Na slici 3.5. prikazan je izgled i način rada CNN koja ima nekoliko konvolucijskih slojeva, slojeva sažimanja i potpuno povezanih slojeva [17].



Slika 3.5: *Izgled konvolucijske neuronske mreže s ulazom, dva konvolucijska sloja, dva sloja sažimanja te dva potpuno povezana sloja i izlazom*

Tri ključna hiperparametra koja utječu na veličinu izlaza su broj filtera (utječe na dubinu izlaza), *stride* (udaljenost koju *kernel* prelazi preko ulaza), *zero-padding* (dodaje nule oko rubova ulaza kako bi se prilagodila veličina izlaza). Nakon svake konvolucijske operacije, primjenjuje se aktivacijska funkcija *ReLU* (engl. *Rectified Linear Unit*) na mapu značajki te se time uvodi nelinearnost u model [17]. Na slici 3.6. prikazan je primjer rada filtera veličine 3x3 na ulaznoj slici te izlaz koji taj filter daje.



Slika 3.6: *Prikaz rada i rezultat konvolucijskog filtra veličine 3x3 na ulaznoj slici*

Sloj udruživanja

Sloj udruživanja smanjuje dimenzionalnost ulaznih podataka, odnosno smanjuje se broj parametara koje mreža mora obrađivati. Sloj udruživanja sažima informacije iz većih područja slike u manji broj značajki, pri tome zadržavajući najvažnije informacije. Slično konvolucijskom sloju, operacija udruživanja pomiče filter preko cijelog ulaza, ali ovaj filter nema težinu. Umjesto toga, *kernel* primjenjuje agregacijsku funkciju na vrijednosti unutar prijemnog polja, popunjavajući izlazni niz. Postoje dvije glavne vrste udruživanja:

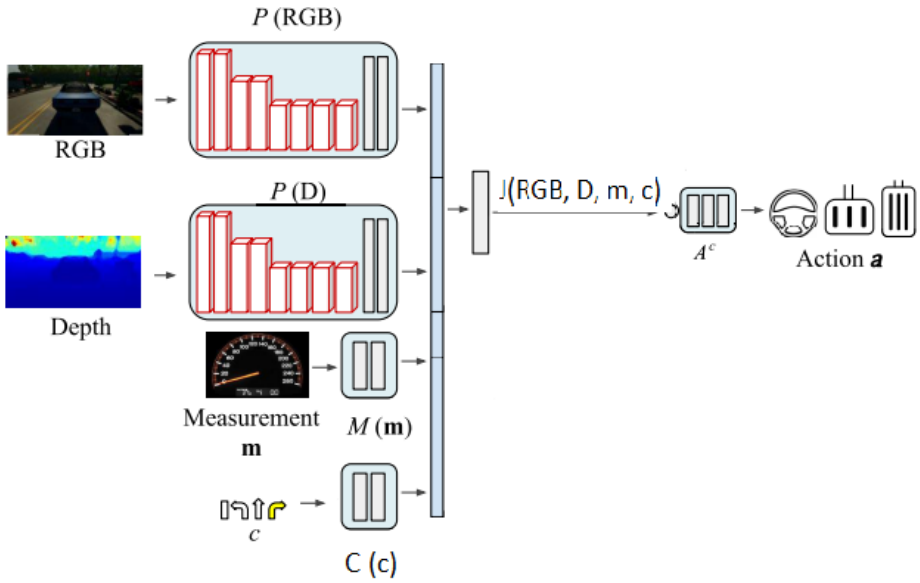
- Sažimanje maksimalnom vrijednošću (engl. *Max pooling*): dok se filter pomiče preko ulaza, bira piksel s najvećom vrijednošću za slanje u izlazni niz. Ovaj pristup se češće koristi u odnosu na *average pooling*.
- Kombiniranje više klasifikatora (engl. *Average pooling*): dok se filter pomiče preko ulaza, računa prosječnu vrijednost unutar prijemnog polja za slanje izlazni niz.

Potpuno povezani sloj

U potpuno povezanom sloju, svaki čvor u izlaznom sloju izravno je povezan s čvorom u prethodnom sloju. Ovaj sloj obavlja zadatak klasifikacije na temelju značajki ekstrahiranih kroz prethodne slojeve i njihove različite filtere. Dok konvolucijski i sloj udruživanja obično koriste *ReLU* funkcije, potpuno povezani slojevi obično koriste *softmax* aktivacijsku funkciju za pravilno klasificiranje ulaza, proizvodeći vjerojatnost od 0 do 1 [17].

3.2. Opis vlastitog rješenja problema

U ovom potpoglavlju bit će predstavljena generalna shema rješenja za razvoj modela autonomne vožnje, razvijenog u sklopu ovog diplomskog rada. Cilj rada je razviti model sposoban za autonomnu vožnju u simulacijskom okruženju, koristeći CARLA simulator i metodologiju uvjetovanog imitacijskog učenja. Model koji je razvijen temelji se na kombinaciji dva različita senzora: RGB kamere i dubinske kamere, kako bi se poboljšale performanse u odnosu na modele koji koriste samo jedan senzor, najčešće RGB kameru. Na slici 3.7. prikazan je cjelokupni sustav autonomnog upravljanja vozilom koji se koristi u ovom radu.



Slika 3.7. *Primjer cjelokupnog sustava autonomnog upravljanja vozilom koji se koristi u ovom radu*

Sustav se sastoji od nekoliko ključnih komponenti koje omogućavaju autonomnu vožnju. Ulazni podaci, kao što su RGB slika, dubinska slika, brzina vozila i naredba za skretanje se obrađuju prije predaje modulu neuronske mreže. Na temelju prethodno obrađenih ulaznih podataka, kontrolni modul donosi odluke o upravljanju vozilom pomoću predavanja podataka modelu neuronske mreže. Model neuronske mreže kao izlaz predaje vrijednosti potrebne za upravljanje automobilom (vrijednosti gasa, kočnice i zakreta volana) kontrolnome modulu.

Struktura sustava obuhvaća sljedeće ključne komponente:

- Obrada ulaznih podataka: RGB i dubinska slika, te brzina vozila kao i naredbe za skretanje predstavljaju ulazne podatke.
- Konvolucijska neuronska mreža: CNN mreža koristi ulazne podatke, generira značajke iz RGB i dubinskih slika te kombinira ove informacije s dodatnim podacima pomoću sloja za fuziju kako bi se dobile kontrole za upravljanje vozilom.
- Kontrolni modul: Na temelju izlaza modela neuronske mreže ovaj modul donosi odluke o upravljanju vozilom

Podaci korišteni za treniranje modela prikupljeni su u CARLA simulatoru koristeći dostupni autopilot. Treniranje modela provodi se na prikupljenim podacima, pri čemu se naglasak stavlja na augmentaciju RGB i dubinskih slika kako bi se postigla veća robusnost modela. U sljedećim

potpoglavljima biti će detaljno objašnjeni svaki od ovih dijelova, uključujući postupak prikupljanja podataka i postupak treniranja modela.

3.3. Obrada ulaznih signala

Ulazni podaci koje model neuronske mreže koristi za predviđanje kontrola za automobil su RGB slika, dubinska slika, naredba za skretanje i podatak o brzini vozila. Prije predavanja RGB i dubinske slike modelu neuronske mreže, te slike se moraju obraditi kako bi odgovarale zadanim parametrima koji su unaprijed određeni. Objašnjenje zašto se slike obrađuju na sljedeći način će biti u nastavku rada gdje će se detaljnije usporediti izvorna slika RGB i dubinske kamere sa slikama koje se predaju modelu neuronske mreže. Prije predavanja RGB slike modelu, odbacuje se 30% gornjeg dijela i 20% donjeg dijela slike, odnosno određuje se regija interesa (engl. *Region of interest*- ROI). Nakon toga, mijenja se rezolucija slike u rezoluciju 400x240 piksela, te se radi normalizacija koja je potrebna za normalno funkcioniranje modela neuronske mreže. Dubinska slika se obrađuje na sličan način, traži se *ROI,* odbacuje se 35% gornjeg dijela, 10% donjeg dijela te 15% lijeve i desne strane slike. Nakon toga mijenja se rezolucija u 244x144 piksela i radi se normalizacija slike. Na slici 3.8. prikazan je primjer programskog koda za predobradu RGB i dubinske slike unutar korištene *python* skripte za evaluaciju modela.

```
def rgb_callback(image, sensor_data):
    global predict_image
    img = np.reshape(np.copy(image.raw_data), (image.height, image.width, 4))
    img = img[:, :, :3]
    img = img[180:480, :]
    img = cv2.resize(img, (400, 240))
    sensor_data['rgb_image'] = img
    predict_image = img / 255.0

def depth_callback(image, sensor_data):
        global predict_depth
        image.convert(carla.ColorConverter.LogarithmicDepth)
        depth_image = np.reshape(np.copy(image.raw_data), (image.height,
image.width, 4))
        # Cropping parameters
        top_crop_percent = 0.35
        bottom_crop_percent = 0.1
        left_crop_percent = 0.15
        right_crop_percent = 0.15
        # Original dimensions
        original_height = depth_image.shape[0]  # e.g., 480
        original_width = depth_image.shape[1]  # e.g., 640
        # Calculate crop sizes
```

```
        top_crop = int(original_height * top_crop_percent)
        bottom_crop = original_height - int(original_height *
bottom_crop_percent)
        left_crop = int(original_width * left_crop_percent)
        right_crop = original_width - int(original_width * right_crop_percent)
        cropped_array = depth_image[top_crop:bottom_crop, left_crop:right_crop,
:]
        depth_img = cv2.resize(cropped_array, (244, 144))
        predict_depth = depth_img / 255.0
```

Slika 3.8: Primjer programskog koda za predobradu RGB i dubinske slike

Sljedeći ulazni podatak je brzina vozila koja se dobiva računskim putem tijekom izvođenja *python* skripte. Na slici 3.9. prikazan je programski kod koji izračunava trenutnu brzinu vozila prije predavanja modelu neuronske mreže.

```
velocity_car = my_vehicle.get_velocity()
    speed_m_per_s = math.sqrt(velocity_car.x**2 + velocity_car.y**2 +
velocity_car.z**2)
    speed_km_per_h = speed_m_per_s * 3.6
    speed = np.array([speed_km_per_h])
```

Slika 3.9: Primjer programskog koda za izračunavanje trenutne brzine vozila

Posljednji ulazni podatak koji je potreban za predviđanje kontrole vozila je naredba o skretanju koja se dobiva pomoću upravitelja prometa. Na slici 3.10. prikazan je primjer programskog koda za dohvaćanje trenutne naredbe o skretanju.

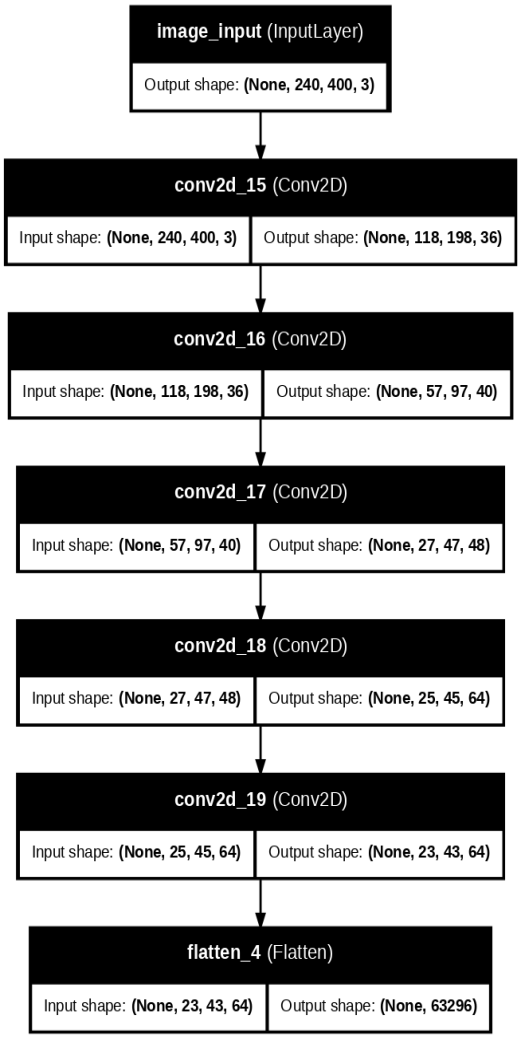
```
next_action = traffic_manager.get_next_action(my_vehicle)
# Access the road option
road_option = next_action[0]
```

Slika 3.10: Primjer programskog koda za dohvaćanje trenutne naredbe o skretanju

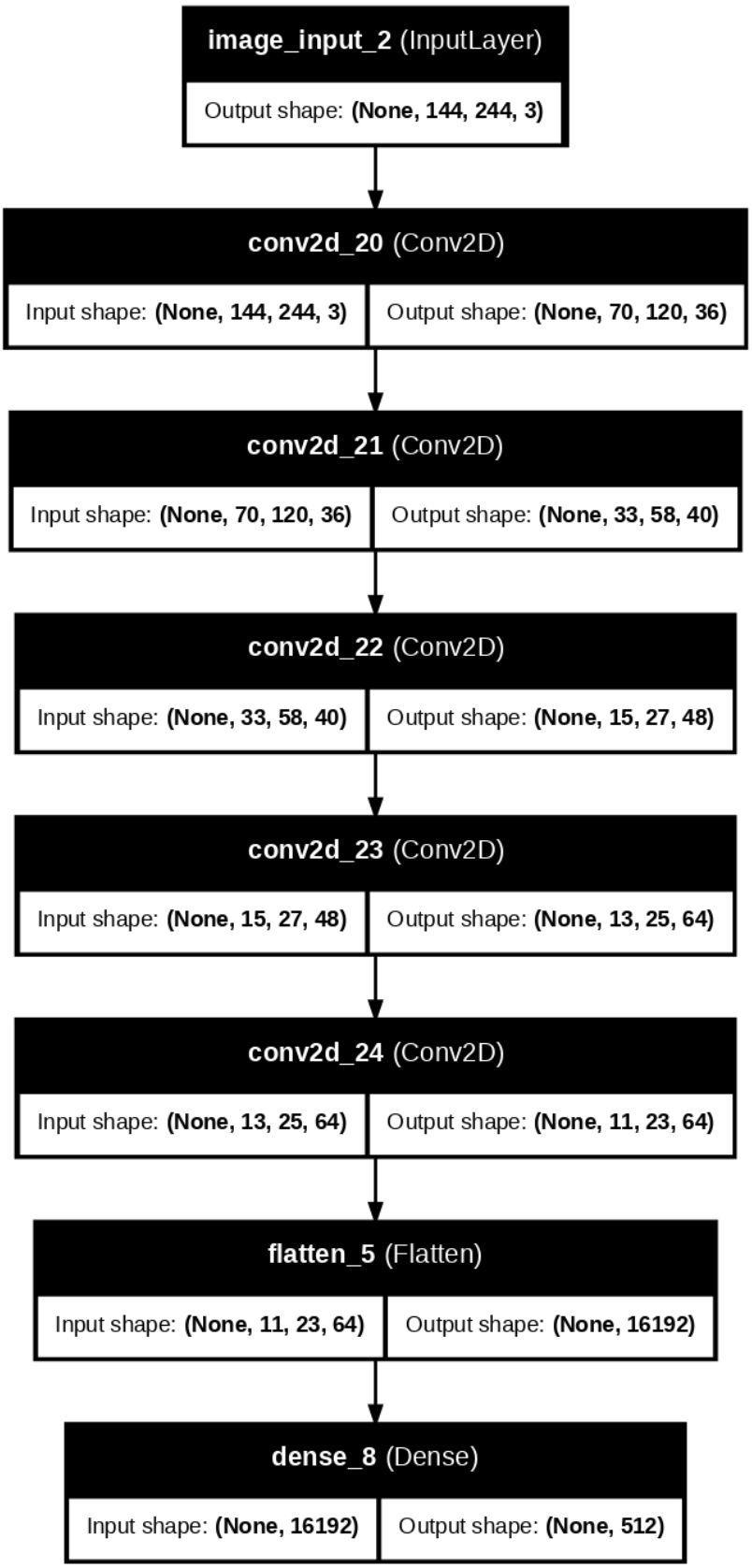
3.4. Konvolucijska neuronska mreža

Arhitektura neuronske mreže korištene za izradu ovoga rada i evaluaciju modela sastoji se od dva ulaza za slike, ulaza za brzinu vozila i ulaza za opciju skretanja. Mreža koristi konvolucijske slojeve za obradu ulaznih slika i potpuno povezane (engl. *fully connected*) slojeve za kombiniranje svih ulaza te kao rezultat generira izlaz u obliku naredbi za upravljanjem vozila. Obrada prvog ulaza u model se odnosi na obradu slike RGB kamere, a za obradu se koriste konvolucijski slojevi s različitim brojem filtera i veličinama *kernela* za ekstrakciju značajki iz slike. Nakon konvolucijskog sloja koristi se *BatchNormalization* za normalizaciju podataka. Na kraju se provodi funkcija *Flatten()* kako bi se taj ulaz mogao spojiti s ostalim ulazima. Postupak obrade

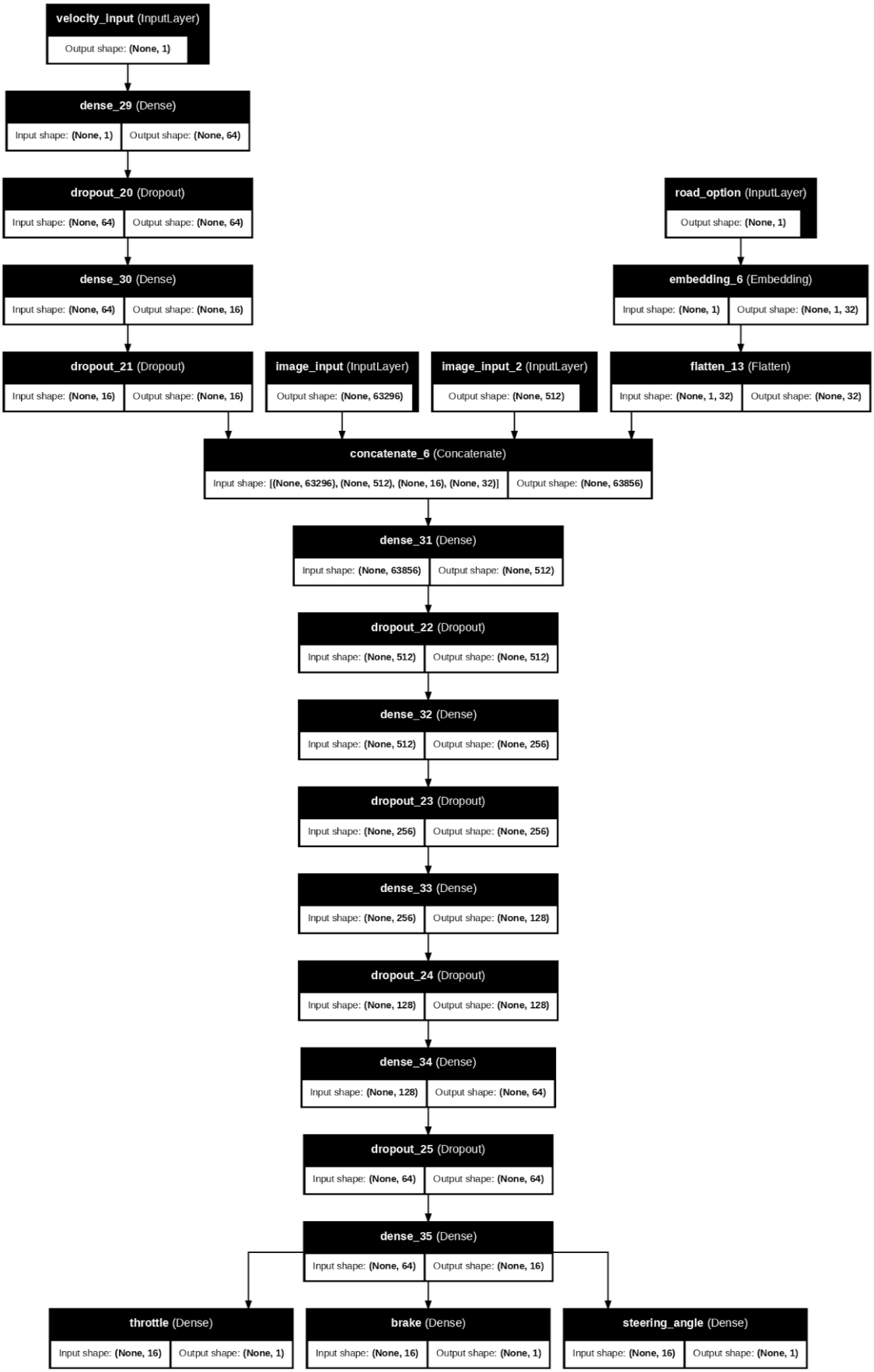
drugog ulaza se odnosi na obradu slike dubinske kamere čiji je postupak sličan kao i za prvi ulaz, ali se razlikuje u veličinama slika i broju filtera. Za obradu ulaza brzine vozila se prije spajanja s ostalim ulazima koristi potpuno povezani sloj (engl. *dense*) s *relu* aktivacijskom funkcijom i s *dropout* slojem za regularizaciju. Obrada ulaza opcije skretanja koristi *Embedding* sloj koji služi za pretvaranje cijelih brojeva u vektore. Nakon obrade svakog ulaza pojedinačno, oni se spajaju koristeći *concatenate* funkciju. Poslije spajanja, koristi se nekoliko potpuno povezanih slojeva s *ReLU* aktivacijom i d*ropout* slojevima za regularizaciju. Model ima tri izlaza kao što je prije tokom rada navedeno, a to su gas, kočnica i kut zakreta volana. U prilogu P.3.1. dan je primjer koda za definiranje arhitekture modela koji se koristio za izradu ovoga rada. Na slikama 3.11., 3.12. i 3.13. prikazana je vizualizacija arhitekture modela koja se koristila za izradu rada. Slika 3.11. predstavlja obradu ulazne RGB slike, slika 3.12. predstavlja obradu ulazne dubinske slike dok slika 3.13. predstavlja spajanje svih ulaza i dobivanje izlaza modela.



Slika 3.11: Primjer vizualizacije arhitekture modela za obradu ulazne RGB slike



Slika 3.12: Primjer vizualizacije arhitekture modela za obradu ulazne dubinske slike



Slika 3.13: Primjer vizualizacije arhitekture modela za spajanje ulaza i dobivanje izlaza modela

3.5. Kontrolni modul

Kontrolni modul brine o pravovremenom upravljanju automobilom te o odlučivanju je li potrebno ručno upravljati automobilom ukoliko dođe do pogrešnog predviđanja kontrole od strane modela. Ako postoji potreba za ručnim upravljanjem vozila, modul omogućava korisniku tijekom evaluacije ručno kontroliranje vozila kako bi se ispravila pogreška nastala u vožnji. Funkcija *manual_predict()* kao argumente prima podatke o vozilu kojim model upravlja i trenutnu opciju skretanja. Podatak o trenutnom stanju vozila se koristi za određivanje trenutne brzine vozila. Slike RGB i dubinske kamere se dohvaćaju pomoću funkcija *get_rgb_image()* i *get_depth_image()* koje kao rezultat vraćaju slike u trenutnome vremenu i odgovarajućem formatu (odnosi se na predobradu opisanu u potpoglavlju 3.3.) koji je potreban za predaju modelu. Funkcija *model.predict()* brine o pravovremenom uključenju ručnog kontroliranja vozila te o pretvaranju vrijednosti izlaza modela neuronske mreže u odgovarajući oblik podataka koji se predaje vozilu. Kontrole se naposljetku predaju automobilu. Na slici 3.14. prikazan je primjer programskog koda *manual_predict()* funkcije za upravljanje vozila.

```
def manual_predict2(my_vehicle, road_option, manual_override_active,
manual_throttle_override, manual_brake_override):

    predict_image = get_rgb_image()
    predict_depth = get_depth_image()
    # Prikupljanje trenutnih podataka o brzini i slikama
    velocity_car = my_vehicle.get_velocity()
    speed_m_per_s = math.sqrt(velocity_car.x**2 + velocity_car.y**2 +
velocity_car.z**2)
    speed_km_per_h = speed_m_per_s * 3.6
    speed = np.array([speed_km_per_h])
    input_data = [np.array([predict_image]), np.array([predict_depth]), speed,
np.array([road_option])]

    # Predikcija pomoću modela
    pred = model.predict(input_data)
    throttle_pred = pred[0][0][0]
    brake_pred = pred[1][0][0]
    steering_pred = pred[2][0][0]

    # Ako je manualni način rada aktivan, prepisuju se predikcije
    if manual_override_active:
        throttle_pred = manual_throttle_override
        brake_pred = manual_brake_override

    # Postavljanje kontrolnih komandi vozilu
    vehicle_control = my_vehicle.get_control()
```

```
    vehicle_control.throttle = float(throttle_pred)
    vehicle_control.brake = float(brake_pred)
    vehicle_control.steer = float(steering_pred)

    # Primjena kontrolnih komandi na vozilo
    my_vehicle.apply_control(vehicle_control)
```

Slika 3.14: *Primjer programskog koda funkcije za predviđanje kontrola za upravljanjem vozilom*

3.6. Postupak izrade baze podataka korištene za treniranje modela

U ovome potpoglavlju opisan je postupak izrade baze podataka potrebne za proces treniranja modela. Prikupljanje kvalitetnih i raznovrsnih podataka jedan je od ključnih koraka u razvoju pouzdanog modela za autonomno upravljanje vozilom. S obzirom na to da model prilikom rada koristi kombinaciju podataka iz različitih senzora, poput RGB i dubinskih kamera, te dodatnih informacija kao što su brzina vozila i naredbe za skretanje, važno je osigurati dosljednost i točnost ovih podataka. Osim tih podataka, za potrebe treniranja prikupljaju se dodatno podaci o vrijednostima gasa, kočnice i kuta zakreta volana zajedno s već navedenim podacima te su ta tri podatka ujedno podaci koje model predviđa. Podaci za kreiranje baze podataka prikupljeni su u CARLA simulatoru, koji omogućuje stvaranje različitih uvjeta za vožnju unutar simulatora, mijenjajući vremenske uvjete te same postavke prometa kao što su gustoća prometa, brzina kretanja vozila u prometu i ostalo. Tim mogućnostima CARLA simulator omogućuje kreiranje raznovrsne baze podataka. Kako bi se osigurala konzistentnost i točnost tijekom skupljanja podataka, korišten je autopilot dostupan unutar CARLA simulatora. Autopilot osigurava nesmetano prikupljanje podataka i eliminira mogućnost pogreške koja je moguća prilikom upravljanja vozilom od strane čovjeka. Kako bi se povećala raznolikost prikupljenih podataka, tijekom vožnje autopilota unesene su smetnje u kontrolu promatranog automobila u obliku naglih promjena smjera kretanja automobila. Autopilot prepoznaje smetnje i reagira na njih. Reakcije autopilota prikupljaju se za stvaranje baze podataka. Podaci koji se odnose na reakciju autopilota omogućuju bolju generalizaciju naučenoga modela.

Koraci potrebni za izradu baze podataka:

- Izrada *Python* skripte za spajanje s CARLA simulatorom: Kreiranje skripte koja se povezuje s CARLA simulatorom i inicijalizira sve potrebne senzore (poput RGB i dubinske kamere) te objekata unutar simulatora. Skripta osigurava da su svi senzori ispravno konfigurirani za prikupljanje podataka.

- Praćenje procesa prikupljanja podataka.
- Obrada prikupljenih podataka: korištenjem *Python* skripte primjenjuju se operacije za obradu slike u svrhu dobivanja regije od interesa (engl. *Region of interes* - ROI) i promjenu rezolucije slike. Ovaj korak ključan je kako bi se dobio format podataka prikladan za treniranje modela.

U nastavku rada, svaki korak biti će dodatno objašnjen.

Izrada Python skripte za spajanje s CARLA simulatorom i inicijalizacija potrebnih senzora

Kao što je spomenuto, skripta koja će biti opisana spaja se sa CARLA simulatorom, definira različite senzore i osigurava njihovu ispravnu konfiguraciju. Prvi korak pri izradi skripte je uvoz svih potrebnih biblioteka, kao što su *carla*, *numpy*, *cv2*, *pygame* i ostale biblioteke ključne za rad s CARLA simulatorom, slikama i senzorima. Na slici 3.15. prikazan je primjer koda za uvoz potrebnih biblioteka.

```
Import glob
import os
import sys
import time
import math
import cv2
import csv
import numpy as np
import carla
import pygame
from pygame.locals import K_ESCAPE, K_q
from carla import VehicleLightState as vls
from agents.navigation.global_route_planner import GlobalRoutePlanner
import argparse
import logging
import random
```

Slika 3.15: *Primjer koda za uvoz potrebnih biblioteka tijekom izrade skripte*

Sljedeći korak je priprema za spajanje sa simulatorom. Definira se IP adresa i *port* na koji se spaja sa simulatorom. Na slici 3.16. prikazan je primjer koda za spajanje sa simulatorom.

```
client = carla.Client('127.0.0.1', 2000)
client.set_timeout(10.0)
```

Slika 3.16: *Primjer koda za postavljanje IP adrese i porta potrebnih za spajanje sa simulatorom*

Za postavljanje inicijalne konfiguracije simulatora, potrebno je učitati željenu mapu u CARLA simulator i postaviti po potrebi vremenske uvijete. Na slici 3.17. prikazan je primjer koda za učitavanje mape i postavljanje željenih vremenskih uvjeta unutar simulatora.

```
world = client.get_world()
world = client.load_world('Town03')

weather = carla.WeatherParameters(
    cloudiness=0.0,
    precipitation=0.0,
    precipitation_deposits=0.0,
    wind_intensity=0.0,
    sun_azimuth_angle=40,
    sun_altitude_angle=36,
    fog_density=0.0,
    fog_distance=0.0,
    fog_falloff=0.0,
    scattering_intensity=0.0,
    mie_scattering_scale=0.0,
    rayleigh_scattering_scale=0.03310000151395798,
    dust_storm=0.0
)
world.set_weather(weather)
```

Slika 3.17: *Primjer koda za postavljanje željene mape i vremenskih uvjeta unutar simulatora*

Kako bi se mogla pratiti kontrola autopilota nad automobilom, automobil se mora stvoriti unutar simulacije i upravljanje mu mora biti zadano pomoću autopilota. Na slici 3.18. prikazan je primjer programskog koda za stvaranje automobila i postavljanje upravljanja pomoću autopilota.

```
my_vehicle = world.try_spawn_actor(random.choice(blueprints),
start_spawn_point)
my_vehicle.set_autopilot(True)
```

Slika 3.18: *Primjer programskog koda za stvaranje automobila i postavljanje upravljanja pomoću autopilota*

Nadalje, potrebno je inicijalizirati senzore kao što su RGB i dubinska kamera. Za inicijalizaciju senzora, svaki je senzor potrebno zasebno konfigurirati. Tijekom prikupljanja baze podataka korištene su četiri RGB kamere s različitim kutom vidnog polja (engl. *Field of View* - FOV). Vrijednosti koje su bile postavljene za četiri različita kuta vidnog polja su 105, 110, 115 i 120 stupnjeva. Vidno polje dubinske kamere je postavljeno na 110 stupnjeva i ono se nije mijenjalo tokom prikupljanja baze podataka. Rezolucija sve četiri RGB kamere je 800x600 što su zadane postavke unutar simulatora, dok dubinska kamera ima rezoluciju 640x480. Kamere su konfigurirane tako da su unutar simulacije postavljene u visini retrovizora. Korištenjem četiri kamere istovremeno se spremaju četiri različite slike s različitim vidnim poljem (105, 110, 115 i 120) u bazu podataka. Parametri koji se prate na vozilu, kao što su gas, kočnica, kut zakreta volana, trenutna brzina i opcija skretanja, u istom trenutku se spremaju zajedno sa sve četiri različite slike. Za različita vidna polja, kontrole automobila koje su spremljene su iste, te je osigurana jednakost podataka za kontrolu automobila prilikom kreiranja baze podataka bez obzira na razliku u stupnjevima vidnog polja. Razlog ove konfiguracije je omogućavanje treniranja modela na različitim vidnim poljima RGB kamere (a da su pri tome kontrole s kojima se upravljalo automobil jednake kao i za kamere s drugim vidnim polje) kako bi se na kraju treniranja usporedili rezultati i donio zaključak koji model ima najbolje performanse. Na slici 3.19. prikazan je primjer koda za inicijalizaciju RGB kamere (s vidnim poljem od 105 stupnjeva) i dubinske kamere.

```
    library = world.get_blueprint_library()
    camera_bp_105 = library().find('sensor.camera.rgb')
    camera_bp_105.set_attribute('fov', '105')
    camera_init_trans = carla.Transform(carla.Location(x=0.4, z=1.6))
    camera_105 = world.spawn_actor(camera_bp_105, camera_init_trans,
attach_to=my_vehicle)

    depth_cam_bp = world.get_blueprint_library().find('sensor.camera.depth')
    depth_camera = world.spawn_actor(depth_cam_bp, camera_init_trans,
attach_to=my_vehicle)

    depth_cam_bp.set_attribute('image_size_x', '640')  # Image width
    depth_cam_bp.set_attribute('image_size_y', '480')  # Image height
    depth_cam_bp.set_attribute('fov', '110')  # Field of View
```

Slika 3.19: *Prikaz koda za inicijalizaciju RGB kamere s vidnim poljem od 105 stupnjeva i dubinske kamere koje su konfigurirane tako da prate kretanje automobila*

Na slikama 3.20. i 3.21. prikazani su primjeri slika sa RGB i dubinske kamere. Slika RGB kamere je snimljena na kameri koja ima vrijednost vidnog polja 105 stupnjeva. Kao što je prethodno navedeno, dubinska kamera snima slike koje su manje rezolucije, 640x480. Sama kamera unutar simulatora drugačije je konfigurirana, što je razlog zašto se neki objekti na slici dubinske kamere ne vide, dok se na slici RGB kamere vide. Slike 3.20. i 3.21. dio su baze podataka korištenje za treniranje modela.

Slika 3.20: *Primjer slike RGB kamere unutar CARLA simulatora s vidnim poljem od 105 stupnjeva*

Slika 3.21:*Primjer slike dubinske kamere unutar CARLA simulatora*

Kako bi se promijenila vrijednost vidnog polja RGB kamere, potrebno je inicijalizirati novu kameru. Na slici 3.22. prikazan je programski kod potreban za inicijalizaciju RGB kamere s vidnim poljem kamere od 110 stupnjeva.

```
camera_bp_110 = world.get_blueprint_library().find('sensor.camera.rgb')
camera_bp_110.set_attribute('fov', '110')
camera_init_trans = carla.Transform(carla.Location(x=0.4, z=1.6))
```

```
   camera_110 = world.spawn_actor(camera_bp_110, camera_init_trans,
attach_to=my_vehicle)
```

Slika 3.22: *Prikaz koda za inicijalizaciju RGB kamere s vidnim poljem od 110 stupnjeva koja je konfigurirana tako da prati kretanje automobila*

Na slici 3.23. prikazan je primjer slike RGB kamere s vidnim poljem od 110 stupnjeva unutar CARLA simulatora.

Slika 3.23: *Primjer slike RGB kamere unutar CARLA simulatora s vidnim poljem od 110 stupnjeva*

Na slici 3.24. prikazan je programski kod potreban za inicijalizaciju RGB kamere s vidnim poljem kamere od 115 stupnjeva.

```
   camera_bp_115 = world.get_blueprint_library().find('sensor.camera.rgb')
   camera_bp_115.set_attribute('fov', '115')
   camera_init_trans = carla.Transform(carla.Location(x=0.4, z=1.6))
   camera_115 = world.spawn_actor(camera_bp_115, camera_init_trans,
attach_to=my_vehicle)
```

Slika 3.24: *Prikaz koda za inicijalizaciju RGB kamere s vidnim poljem od 115 stupnjeva koja je konfigurirana tako da prati kretanje automobila*

Na slici 3.25. prikazan je primjer slike RGB kamere s vidnim poljem od 115 stupnjeva unutar CARLA simulatora.

Slika 3.25: *Primjer slike RGB kamere unutar CARLA simulatora s vidnim poljem od 115 stupnjeva*

Na slici 3.26. prikazan je programski kod potreban za inicijalizaciju RGB kamere s vidnim poljem kamere od 120 stupnjeva.

```
    camera_bp_120 = world.get_blueprint_library().find('sensor.camera.rgb')
    camera_bp_120.set_attribute('fov', '120')
    camera_init_trans = carla.Transform(carla.Location(x=0.4, z=1.6))
    camera_120 = world.spawn_actor(camera_bp_120, camera_init_trans,
attach_to=my_vehicle)
```

Slika 3.26: *Prikaz koda za inicijalizaciju RGB kamere s vidnim poljem od 115 stupnjeva koja je konfigurirana tako da prati kretanje automobila*

Na slici 3.27. prikazan je primjer slike RGB kamere s vidnim poljem od 120 stupnjeva unutar CARLA simulatora.

Slika 3.27: *Primjer slike RGB kamere unutar CARLA simulatora s vidnim poljem od 120 stupnjeva*

Ako se usporede slike s različitim vrijednostima vidnog polja RGB kamere, može se primijetiti razlika među njima. Razliku je lako uočiti na lijevom rubu slike, posebno na bijelom automobilu. Povećanjem vidnog polja kamere, povećava se vidljivi dio automobila. Međutim, povećanjem vidnog polja RGB kamere dolazi do distorzije slike, stoga treba paziti na postavljanje odgovarajućeg vidnog polja. Slijedeći korak je postaviti povratne funkcije (engl. *callback*) za svaku kameru. Povratna funkcija je odgovorna za proces predobrade slike prije spremanja u bazu podataka, ako postoji potreba za tim procesom. Funkcija kao argumente prima sliku i rječnik (engl. *dictionary*). Rječnik *sensor_data* služi kao spremište za slike dobivene s različitih kamera u simulaciji. Ključevi u rječniku (npr. '*rgb_image_105*') predstavljaju naziv senzora ili specifičan kut kamere, dok su vrijednosti slike (*numpy* matrice) dobivene sa tih senzora. Rječnik omogućuje jednostavan i brz pristup ovim slikama tijekom simulacije. Kada kamera snimi određenu sliku (engl. *frame*), aktivira se funkcija slušatelja za određenu kameru i funkcija radi obradu slike. Kako bi slika odgovarala traženim zahtjevima za spremanjem u bazu podataka, provodile su se sljedeće operacije nad slikom:

- Kopiranje sirovih podataka (engl. *raw data*) u *NumPy* polje i preoblikovanje u 4D polje
- Postavljanje alfa kanala (svojstvo prozirnosti) na vrijednost 255 (puna neprozirnost)
- Spremanje slike u rječnik pod odgovarajućim ključem (engl. *key*)

Kako bi se slika mogla spremiti, potrebno je inicijalizirati rječnik s odgovarajućim ključevima, odnosno četiri ključa za svaku od četiri RGB kamere i jedan ključ za dubinsku kameru. Nakon potrebne inicijalizacije postavlja se funkcija slušatelja (engl. *listen*) za svaku od kamera. Na slici 3.28. prikazan je primjer programskog koda za definiranje funkcije slušatelja za kameru (vidno polje od 105 stupnjeva), inicijalizaciju novoga ključa u rječniku i postavljanje funkcije slušatelja.

```
def rgb_callback_105(image, data_dict):
    img = np.reshape(np.copy(image.raw_data), (image.height, image.width, 4))
    img[:,:,3] = 255 #Setting the alpha to 255
    data_dict['rgb_image_105'] = img

sensor_data = {'rgb_image_105': np.zeros((600, 800, 3))}
camera_105.listen(lambda image: rgb_callback_105(image, sensor_data))
```

Slika 3.28: *Prikaz koda za definiranje funkcije slušatelja, inicijalizacija novog ključa u rječniku i postavljanje funkcije slušatelja*

Pokretanje i nadzor procesa prikupljanja podataka

Završetkom postavljanja simulacijskog okruženja i inicijalizacijom svih senzora, sve je spremno za početak prikupljanja baze podataka. Prikupljanje se odvija u beskonačnoj petlji (engl. *while loop*), prateći stanje automobila. Podaci o automobilu se dohvaćaju, spremaju u polje (engl. *array*) i zajedno sa slikama kamere u tome istom trenutku se šalju funkciji koja se brine o spremanju svih tih podataka u bazu podataka. Na slici 3.29. prikazan je pojednostavljeni primjer programskog koda beskonačne petlje u kojoj se prati stanje automobila i poziva se funkcija za spremanje podataka. Primjer koda je pojednostavljen, iz razloga što u skripti koja je korištena za prikupljanje baze podataka, programski kod se sastoji od velikog broj linija programskog koda te bi u ovom obliku prikaza bio vrlo nepregledan. Takav programski kod sadrži nekolicinu deklaracija privremenih varijabli, provjeru simulacijskog okruženja kako bi se prilagodila brzina spremanja podataka i ostalo.

```
    while True:
        world.tick()
        if(is__it_safe_to_insert_noise_for_autopilot(my_vehicle) is True):
            insert_noise(my_vehicle)
        control = my_vehicle.get_control()
        throttle = control.throttle
        brake = control.brake
        steer = traffic_manager.get_next_action(my_vehicle)
        velocity = my_vehicle.get_velocity()
        speed = math.sqrt(velocity.x**2 + velocity.y**2 + velocity.z**2) * 3.6

        data = [throttle, brake, steer, speed]
        save_data(sensor_data['rgb_image_105'], sensor_data['rgb_image_110'],
sensor_data['rgb_image_115'], sensor_data['rgb_image_120'],
sensor_data['depth_image'], data)
```

Slika 3.29: *Primjer pojednostavljenog programskog koda beskonačne petlje u kojoj se odvija praćenje stanja automobila i slanje podataka kako bi se spremili u bazu podataka*

Podaci o automobilu se dohvaćaju postojećim funkcijama za dohvaćanje stanja vozila. Podatak o trenutnoj opciji skretanja se dohvaća pomoću modula za upravljanje prometom. Pomoću ovog modula, u svakom trenutku za svaki automobil može se dohvatiti njegova trenutna opcija skretanja. Modulu se predaje objekt automobila, a kao rezultat modul vraća opciju skretanja u obliku teksta. Modul za upravljanje prometom u stvarnome vremenu može dohvatiti opciju

skretanja za predani objekt automobila prema odgovarajućoj ruti automobila kojom se on kreće. Vrijednost praćenja linije (engl. *LaneFollow*) pretvara se u broj 0, vrijednost skretanja lijevo (engl. *Left*) pretvara se u broj 1, vrijednost prolaska ravno u raskrižju (engl. *Straight*) pretvara se u broj 2, a vrijednost skretanja desno (engl. *Right*) pretvara se u broj 3. Vrijednosti se pretvaraju u brojeve zbog treniranja modela, odnosno u neuronskoj mreži koriste se brojevi za oznake skretanja umjesto tekstualnog oblika. Podaci o trenutnoj vrijednosti gasa, kočnice i kuta zakreta volana dohvaćaju se pozivanjem funkcije *get_control()* nad objektom automobila. Točnost tih podataka je bitna za treniranje modela. Kontinuiranim praćenjem stanja vozila i dohvaćanjem bitnih podataka, istovremeno se poziva funkcija *save_data()* koja se brine o spremanju podataka u bazu podataka. Funkcija prima argumente kao što su slike s RGB kamera s vidnim poljem od 105, 110, 115 i 120 stupnjeva (odnosno četiri različite slike), sliku dubinske kamere i podatke o stanju automobila (gas, kočnica, kut zakreta volana, brzina, opcija skretanja). Na slici 3.30. prikazan je primjer koda funkcije koja sprema podatke u bazu podataka, ali ne radi nikakvu obradu primljenih slika ili podataka.

```
    frame_id = 0
    def save_data(rgb_image_105, rgb_image_110, rgb_image_115, rgb_image_120,
depth_image, data):
        cv2.imwrite(f'path/to/save/rgb_105/frame_{frame_id}.jpg', rgb_image_105)
        cv2.imwrite(f'path/to/save/rgb_110/frame_{frame_id}.jpg', rgb_image_110)
        cv2.imwrite(f'path/to/save/rgb_115/frame_{frame_id}.jpg', rgb_image_115)
        cv2.imwrite(f'path/to/save/rgb_120/frame_{frame_id}.jpg', rgb_image_120)
        cv2.imwrite(f'path/to/save/depth/frame_{frame_id}.jpg', depth_image)
        data.append(path/to/save/rgb_105/frame_{frame_id}.jpg)
        data.append(path/to/save/rgb_110/frame_{frame_id}.jpg)
        data.append(path/to/save/rgb_115/frame_{frame_id}.jpg)
        data.append(path/to/save/rgb_120/frame_{frame_id}.jpg)
        data.append(path/to/save/depth/frame_{frame_id}.jpg)
        with open('path/to/save/data.csv', 'a') as file:
            writer = csv.writer(file)
        frame_id += 1
```

Slika 3.30: *Primjer programskog koda funkcije za spremanje podataka u bazu podataka*

U primjeru programskog koda za beskonačnu petlju, na početku petlje postoji uvjet koji provjerava je li sigurno unijeti smetnje u kontrolu autopilota. Ova provjera se vrši pozivanjem funkcije kojoj se predaje objekt promatranog automobila. Uvjet da se unos smetnji smatra sigurnim se odnosi na trenutnu lokaciju automobila te se provjerava je li automobil u neposrednoj blizini

nekog drugog automobila ili sudionika u prometu kako bi se mogle unijeti smetnje. Ako bi se smetnja unijela na nesiguran način, vrlo vjerojatno da bi nakon nekoliko pokušaja promatrani automobil se zabio u drugog sudionika u prometu ili u neki objekt na mapi. Provjerom se osigurava da će autopilot biti u mogućnosti vratiti automobil na ispravan put nakon smetnje. Funkcija za provjeru sigurnosti unosa smetnji je složena i obrađuje podatke u stvarnome vremenu uzimajući u obzir okolinu automobila kako bi se procijenilo je li sigurno unijeti smetnju u kontrolu autopilota. Ako je rezultat funkcije pozitivan, poziva se funkcija za unos smetnji u kontrolu autopilota. Provjera kuta zakreta volana odlučuje koliko jake smetnje se mogu unijeti u kontrolu autopilota, a da autopilot može ispraviti pogreške koje su nastale unosom smetnji. Provjera se vrši tako da prilikom većeg kuta zakreta volana unosi se manje zahtjevna smetnja jer automobil prolazi kroz nekakav zavoj, te bi unosom više zahtjevne smetnje automobil bio izbačen s ceste i autopilot ne bi imao mogućnost vratiti se na ispravan put. Dok se automobil nalazi na ravnoj cesti, mogu se ubaciti više zahtjevne smetnje (naglo skretanje prema pločniku ili prema sredini ceste) kako bi se snimila reakcija autopilota te je takva zahtjevnost smetnje prihvatljiva jer autopilot ima mogućnost ispravka takvih smetnje na ravnoj cesti. U prilogu P.3.2. dan je primjer programskog koda za unos smetnji u kontrolu autopilota.

Praćenje procesa prikupljanja podataka

U prethodnom odlomku objašnjen je proces izrade *Python* skripte koja omogućuje prikupljanje baze podataka. U ovom odlomku ukratko će biti opisane izmjene u skripti i na što je sve obraćena pažnja tijekom prikupljanja baze podataka kako bi se postigla raznovrsnost podataka. CARLA simulator u zadanoj verziji nudi korisnicima šest gradova za korištenje. Prilikom izrade baze podataka, podaci su prikupljeni u četiri grada, s naglaskom na postizanje raznovrsnosti, dok će preostala dva grada služiti kao testni gradovi. Ukupno baza podataka sadrži oko milijun različitih zapisa, što znači da su podaci o vožnji prikupljeni milijun puta tijekom vožnje. Spremljeno je četiri milijuna slika RGB kamera (zbog četiri različita kuta vidnog polja) i milijun slika dubinske kamere. Postotak podataka snimljenih u trenucima kada autopilot ispravlja unesene smetnje varira između 15 i 25 posto, ovisno o gradu. Ako je većina snimljenih podataka na nekoj mapi snimljena po ravnoj cesti, postotak snimljenih smetnji će biti veći zbog naravno veće mogućnosti za unosom smetnji, dok prilikom mape s puno skretanja i oštrih zavoja neće biti toliko mogućnosti za unosom smetnji te će postotak za tu mapu biti manji. Razlog zbog kojega se pridodaje značaju snimanja ovakvih ispravaka smetnji je što se poboljšava sposobnost generalizacije modela. Model uči kako ispraviti greške u vožnji po uzoru na podatke iz baze koji su stvoreni ispravkom smetnji autopilota. Podaci su snimani u uvjetima bez sjena okolnih objekata, što je postignuto promjenom vremenskih

postavki. Razlog tome je što su tijekom testiranja modela, koji su trenirani na podacima snimljenim sa sjenama, modeli ponekad imali poteškoća s prepoznavanjem prometne trake ili bi prelazili linije koje odvajaju prometne trake. Kroz promjenu četiri grada unutar simulatora, većinski dio podataka za svaki grad je snimljen po sunčanom vremenu, a manji dio po kišovitom vremenu. Tijekom snimanja po inicijalnom postavljenom vremenu, mijenjana je gustoća prometa kako bi se postigla raznovrsnost podataka.

Obrada prikupljenih podataka

Nakon što je baza podataka prikupljena, spremljene slike trebaju se obraditi kako bi odgovarale zahtjevima treniranja modela. Sve RGB slike su izmijenjene na isti način. Izrezano je 30% gornjeg dijela i 20% donjeg dijela slike. Taj proces naziva se određivanjem regije od interesa (engl. *Region of Interest* - ROI). U gornjim dijelovima većine slika vidljivi su samo oblaci ili zgrade, ovisno o mapi, dok se u donjem dijelu slike vidi poklopac motora automobila zbog pozicije kamere. Nakon određivanja regije od interesa, rezolucija slika promijenjena je na 400x240 piksela. Na slici 3.31. prikazan je primjer programskog koda za obradu RGB slika nakon izrade baze podataka.

```
def process_images_from_csv(csv_file_path, output_folder):
    df = pd.read_csv(csv_file_path, header=None)
    for image_path in df[7]:
        process_and_save_image(image_path, output_folder)

def process_and_save_image(image_path, output_folder):
    try:
        img = Image.open(image_path)
        width, height = img.size
        top = height * 0.30
        bottom = height * 0.80
        img_cropped = img.crop((0, top, width, bottom))
        img_resized = img_cropped.resize((400, 240))

        file_name = os.path.basename(image_path)

        save_path = os.path.join(output_folder, file_name)
        os.makedirs(output_folder, exist_ok=True)
        img_resized.save(save_path)
```

```
            print(f"Image saved to {save_path}")

        except Exception as e:
            print(f"Failed to process {image_path}: {str(e)}")
csv_file_path = 'my_dataset.csv'
process_images_from_csv(csv_file_path, output_folder)
```

Slika 3.31: *Primjer programskog koda za obradu RGB slika nakon završenog procesa prikupljanja baze podataka.*

Na slici 3.32. prikazan je primjer slike RGB kamere rezolucije 800x600, prije obrade slike za proces treniranja modela.

Slika 3.32: *Primjer slike RGB kamere s rezolucijom 800x600*

Na slici 3.33. prikazan je primjer slike RGB kamere rezolucije 400x240, nakon obrade slike za proces treniranja modela.

Slika 3.33: *Primjer slike RGB kamere koja je pripremljena za proces treniranja modela, rezolucije 400x240*

Na prikazanim slikama 3.32. i 3.33, jasno se može vidjeti razlika u vidu kvalitete slike. Regija od interesa je smanjena kako bi se moglo lakše pronaći bitnije značajke te smanjiti kompleksnost neuronske mreže. Slike dubinske kamere nije potrebno izrezivati zbog zadane konfiguracije dubinske kamere unutar simulatora. Slikama dubinske kamere je bilo potrebno samo promijeniti rezoluciju. Taj proces je omogućen s par izmjena u programskom kodu za obradu slika RGB kamere koji je prethodno prikazan. Proces obrade slike funkcionira isto, jedina razlika je što se ne traži regija od interesa. Na slici 3.34. prikazan je primjer slike dubinske kamere s zadanom konfiguracijom u simulatoru prije obrade slike za proces treniranja modela.

Slika 3.34: *Primjer slike dubinske kamere bez obrade slike za potrebe treniranja modela*

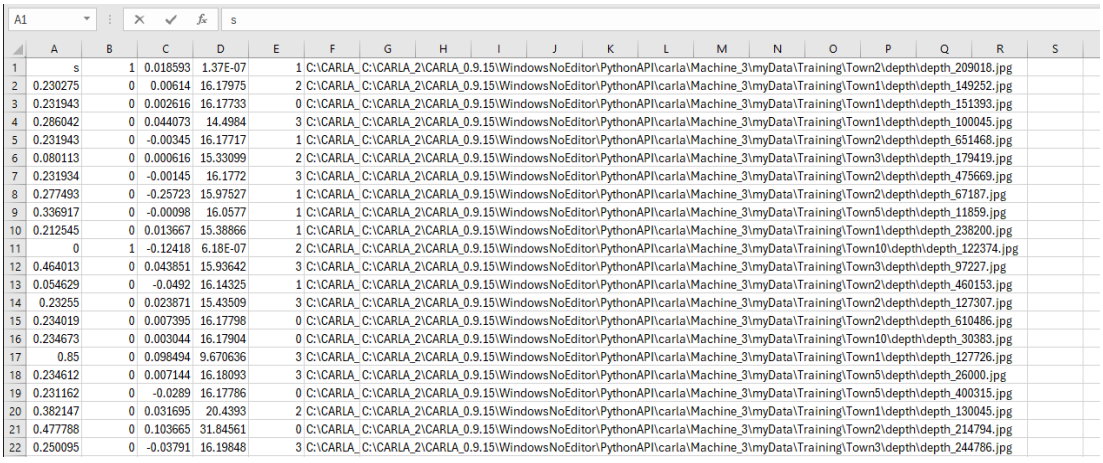
Na slici 3.35. prikazan je primjer slike dubinske kamere nakon obrade slika, prema zahtjevima potrebnim za proces treniranja modela.

Slika 3.35: *Primjer slike dubinske kamere nakon obrade slike prema zahtjevima za proces treniranja modela*

Razliku u slikama dubinske kamere možemo vidjeti na primjeru istih slika, odnosno na slici 3.34. prije obrade i na slici 3.35. nakon obrade prema zahtjevima za proces treniranja modela. Razlika je jasno vidljiva u kvaliteti slike, slično kao u primjeru obrade RGB slike. Nakon obrade slika, baza podataka je spremna za proces treniranja modela. Baza podataka je *excel* tablica, a redoslijed, odnosno struktura podataka unutar tablice je sljedeća:

- Prvi stupac: podatak o vrijednosti gasa
- Drugi stupac: podatak o vrijednosti kočnice
- Treći stupac: podatak o vrijednosti kuta zakreta volana
- Četvrti stupac: podatak o brzini vozila
- Peti stupac: podatak o trenutnoj opciji skretanja vozila (0 – vozilo prati voznu traku, 1 – vozilo skreće lijevo, 2 – vozilo na raskrižju ide ravno, 3 – vozilo skreće desno)
- Šesti stupac: putanja do slike RGB kamere s određenim kutom vidnog polja
- Sedmi stupac: putanja do slike dubinske kamere

Izgled same baze podataka biti će prikazan na slici 3.36.



	A	B	C	D	E	F
1	s	1	0.018593	1.37E-07	1	C:\CARLA_C:\CARLA_2\CARLA_0.9.15\WindowsNoEditor\PythonAPI\carla\Machine_3\myData\Training\Town2\depth\depth_209018.jpg
2	0.230275	0	0.00614	16.17975	2	C:\CARLA_C:\CARLA_2\CARLA_0.9.15\WindowsNoEditor\PythonAPI\carla\Machine_3\myData\Training\Town1\depth\depth_149252.jpg
3	0.231943	0	0.002616	16.17733	0	C:\CARLA_C:\CARLA_2\CARLA_0.9.15\WindowsNoEditor\PythonAPI\carla\Machine_3\myData\Training\Town1\depth\depth_151393.jpg
4	0.286042	0	0.044073	14.4984	3	C:\CARLA_C:\CARLA_2\CARLA_0.9.15\WindowsNoEditor\PythonAPI\carla\Machine_3\myData\Training\Town1\depth\depth_100045.jpg
5	0.231943	0	-0.00345	16.17717	1	C:\CARLA_C:\CARLA_2\CARLA_0.9.15\WindowsNoEditor\PythonAPI\carla\Machine_3\myData\Training\Town2\depth\depth_651468.jpg
6	0.080113	0	0.000616	15.33099	2	C:\CARLA_C:\CARLA_2\CARLA_0.9.15\WindowsNoEditor\PythonAPI\carla\Machine_3\myData\Training\Town3\depth\depth_179419.jpg
7	0.231934	0	-0.00145	16.1772	3	C:\CARLA_C:\CARLA_2\CARLA_0.9.15\WindowsNoEditor\PythonAPI\carla\Machine_3\myData\Training\Town2\depth\depth_475669.jpg
8	0.277493	0	-0.25723	15.97527	1	C:\CARLA_C:\CARLA_2\CARLA_0.9.15\WindowsNoEditor\PythonAPI\carla\Machine_3\myData\Training\Town2\depth\depth_67187.jpg
9	0.336917	0	-0.00098	16.0577	1	C:\CARLA_C:\CARLA_2\CARLA_0.9.15\WindowsNoEditor\PythonAPI\carla\Machine_3\myData\Training\Town5\depth\depth_11859.jpg
10	0.212545	0	0.013667	15.38866	1	C:\CARLA_C:\CARLA_2\CARLA_0.9.15\WindowsNoEditor\PythonAPI\carla\Machine_3\myData\Training\Town1\depth\depth_238200.jpg
11	0	1	-0.12418	6.18E-07	2	C:\CARLA_C:\CARLA_2\CARLA_0.9.15\WindowsNoEditor\PythonAPI\carla\Machine_3\myData\Training\Town10\depth\depth_122374.jpg
12	0.464013	0	0.043851	15.93642	3	C:\CARLA_C:\CARLA_2\CARLA_0.9.15\WindowsNoEditor\PythonAPI\carla\Machine_3\myData\Training\Town3\depth\depth_97227.jpg
13	0.054629	0	-0.0492	16.14325	1	C:\CARLA_C:\CARLA_2\CARLA_0.9.15\WindowsNoEditor\PythonAPI\carla\Machine_3\myData\Training\Town2\depth\depth_460153.jpg
14	0.23255	0	0.023871	15.43509	3	C:\CARLA_C:\CARLA_2\CARLA_0.9.15\WindowsNoEditor\PythonAPI\carla\Machine_3\myData\Training\Town2\depth\depth_127307.jpg
15	0.234019	0	0.007395	16.17798	0	C:\CARLA_C:\CARLA_2\CARLA_0.9.15\WindowsNoEditor\PythonAPI\carla\Machine_3\myData\Training\Town2\depth\depth_610486.jpg
16	0.234673	0	0.003044	16.17904	0	C:\CARLA_C:\CARLA_2\CARLA_0.9.15\WindowsNoEditor\PythonAPI\carla\Machine_3\myData\Training\Town10\depth\depth_30383.jpg
17	0.85	0	0.098494	9.670636	3	C:\CARLA_C:\CARLA_2\CARLA_0.9.15\WindowsNoEditor\PythonAPI\carla\Machine_3\myData\Training\Town1\depth\depth_127726.jpg
18	0.234612	0	0.007144	16.18093	3	C:\CARLA_C:\CARLA_2\CARLA_0.9.15\WindowsNoEditor\PythonAPI\carla\Machine_3\myData\Training\Town5\depth\depth_26000.jpg
19	0.231162	0	-0.0289	16.17786	0	C:\CARLA_C:\CARLA_2\CARLA_0.9.15\WindowsNoEditor\PythonAPI\carla\Machine_3\myData\Training\Town5\depth\depth_400315.jpg
20	0.382147	0	0.031695	20.4393	2	C:\CARLA_C:\CARLA_2\CARLA_0.9.15\WindowsNoEditor\PythonAPI\carla\Machine_3\myData\Training\Town1\depth\depth_130045.jpg
21	0.477788	0	0.103665	31.84561	0	C:\CARLA_C:\CARLA_2\CARLA_0.9.15\WindowsNoEditor\PythonAPI\carla\Machine_3\myData\Training\Town2\depth\depth_214794.jpg
22	0.250095	0	-0.03791	16.19848	3	C:\CARLA_C:\CARLA_2\CARLA_0.9.15\WindowsNoEditor\PythonAPI\carla\Machine_3\myData\Training\Town3\depth\depth_244786.jpg

Slika 3.36: *Primjer slike koja prikazuje bazu podataka pripremljenu za proces treniranja modela*

3.7. Postupak treniranja modela neuronske mreže

U ovom potpoglavlju detaljno će biti objašnjeni koraci i procedure korištene za treniranje modela neuronske mreže. Biti će prikazana priprema podataka, izgradnja i trening modela te funkcije i tehnike korištene za optimizaciju i evaluaciju modela. Prvi korak prema izradi skripte za treniranje modela neuronske mreže je inicijalizacija okruženja, odnosno uključivanje svih potrebnih biblioteka. U prilogu 3.3. prikazan je programski kod za uključivanje potrebnih biblioteka i provjeravanje dostupnosti grafičke procesorske jedinice.

Prije postupka kreiranja modela, baza podataka se dijeli na trening i validacijski skup podataka. Validacijski skup podataka služi procjeni performansi modela na neviđenim podacima, koji su dio inicijalne baze podataka. Novonastali skupovi podataka spremaju u zasebne CSV datoteke. Podaci se dijele u omjeru 80:20. To znači da će 80% podataka inicijalne baze podataka biti izdvojeno za trening skup podataka, dok će 20% pripasti validacijskom skupu. Varijabla *steps_per_epoch* određuje koliko će se koraka treniranja izvoditi u jednoj epohi, a izračunava se dijeljenjem broja uzoraka u trening skupu s vrijednosti *batch_size* varijable koja se predaje kao argument funkciji *split_and_save_data*. *Batch_size* je jedan od ključnih hiperparametara u procesu treniranja modela neuronske mreže. Parametar predstavlja broj uzoraka podataka koji se koriste u jednoj iteraciji tijekom treniranja modela. Dijeljenjem podataka u manje skupove (engl. *batch*), omogućena je efikasnija i brža obrada podataka. To dolazi do izražaja pri korištenju velikih skupova podataka. Manja vrijednost *batch_size* varijable u pravilu poboljšava generalizaciju modela i smanjuje memorijske zahtjeve, dok veća vrijednost može ubrzati treniranje korištenjem prednosti paralelne obrade na GPU-ovima. Na slici 3.37. prikazan je primjer koda za podjelu podataka na trening i validacijski skup podataka.

```
   def split_and_save_data(input_csv_path, train_csv_path, val_csv_path,
batch_size, test_size=0.2, random_state=33):
       data = pd.read_csv(input_csv_path)
       train_data, val_data = train_test_split(data, test_size=test_size,
random_state=random_state)
       train_data.to_csv(train_csv_path, index=False)
       val_data.to_csv(val_csv_path, index=False)
       print(f"Training data saved to {train_csv_path}")
       print(f"Validation data saved to {val_csv_path}")
       steps_per_epoch = len(train_data) // batch_size
       validation_steps = len(val_data) // batch_size
       return steps_per_epoch, validation_steps
   input_csv_path = 'multi_dataset.csv'
   train_csv_path = 'multi_train.csv'
   val_csv_path = 'multi_val.csv'
   batch_size = 100
   steps_per_epoch, validation_steps = split_and_save_data(input_csv_path,
train_csv_path, val_csv_path, batch_size)
```

Slika 3.37: *Primjer koda za postupak podjele baze podataka na trening i validacijski skup podataka*

Nakon kreiranog trening i validacijskog skupa podataka, potrebno je kreirati model koji će se koristiti za treniranje. Taj postupak podrazumijeva definiranje arhitekture modela, odnosno određivanje slojeva, broja neurona u svakom sloju, funkcija aktivacija i ostalih parametara te je ovaj dio prethodno opisan u potpoglavlju 3.4. Konvolucijska neuronska mreža. Model se kompilira koristeći Adam optimizator i srednju kvadratnu pogrešku kao funkciju gubitka za svaki izlaz. Postavljene su težine gubitka (engl. *loss_weights*) s naglaskom na kut zakreta volana. Težine gubitka se koriste kako bi se različitim izlazima modela pridala različita važnost tijekom treniranja. Vrijednosti u *loss_weights* određuju koliko je važan svaki od tih izlaza prilikom izračuna ukupnog gubitka modela. Prema tome, tijekom treniranja, model se više fokusira na dobivanje što boljih rezultata za izlaz kuta zakreta volana. . Ukupan zbroj loss_weights težina je 3.7, a prema raspodijeli vrijednosti svakom od tri izlaza određuje se važnost prilikom treniranja.

Tijekom učenja modela neuronske mreže koriste se optimizator koji prilagođava težine modela tijekom vremena s ciljem smanjivanja funkcije gubitka. Ključan parametar je brzina učenja (engl. *learning rate*). *ExponentialDecay* je tehnika koja omogućuje postupno smanjivanje brzine učenja tijekom treniranja. Nakon određenog broja koraka (engl. *decay_steps*) parametar brzine učenja će se smanjiti. Smanjuje se tako da trenutna vrijednost parametra učenja se množi sa stopom smanjenja brzine učenja (engl. *decay_rate*). *ExponentialDecay* omogućava dinamičko prilagođavanje brzine učenja tijekom treniranja što pomaže pri bržem konvergiranju modela na početku učenja, a stabilnijem završetku učenja. Na slici 3.38. dan je primjer programskog koda za inicijalizaciju *ExponentialDecay-a*.

```
lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate=0.00035,
    decay_steps=7929,
    decay_rate=0.94,
    staircase=True)
```

Slika 3.38: *Primjer programskog koda za inicijalizaciju ExponentialDecay-a*

Prije početka i tijekom treniranja modela, radi se priprema trening skup podataka koji je prethodno nastao podjelom baze podataka na trening i validacijski skup podataka. Podaci se tokom pripreme preuzimaju iz datoteke, odvija se predprocesiranje i organiziranje u format koji je odgovarajući za unos u model. Prilikom ovog postupka koristi se *TensorFlow*. Definiraju se zadani tipovi podataka i imena stupaca za svaki stupac u CSV datoteci. Koristeći funkciju *tf.data.experimental.make_csv_dataset* učitava se CSV datoteka i pretvara u *TensorFlow* bazu podataka. Funkcija *map(preprocess_row, num_parallel_calls=tf.data.AUTOTUNE)* služi za

paralelno primjenjivanje funkcije *preprocess*_row na svaki element u bazi podataka. Funkcija *load_and_preprocess_image(data['image_path'])* učitava i radi pretprocesiranje slike s predanih putanja. Učitavaju se slike iz datoteke, dekodiraju i normaliziraju. Poziva se *augment_image_batch(images)* funkcija koja nasumično radi augmentaciju predanih slika, odnosno radi se promjena svjetline, kontrasta i zasićenosti u svrhu povećanja raznolikosti skupa podataka i bolje generalizacije modela. Isti postupak pripreme podataka se radi tokom validacijskog postupka, odnosno na kraju svake epohe treniranja, ali je razlika što se na validacijski skup podataka ne primjenjuje postupak augmentacije. U prilogu P.3.4. dan je primjer programskog koda za pripremu trening i validacijskog skupa podataka.

Tijekom treniranja koriste se *Callback* funkcije kao što su:

- *tf.keras.callbacks.EarlyStopping*: funkcija zaustavlja treniranje ako se validacijski gubitak (engl. *vall_loss*) ne poboljšava nakon određenog broja epoha.
- *tf.keras.callbacks.ModelCheckpoint*: funkcija koja sprema najbolju verziju modela, model s najmanjom vrijednosti funkcije gubitka
- *ShuffleTrainingDataCallback*: prilagođena povratna funkcija koja miješa podatke za treniranje na početku svake epohe

Pozivajući *model.fit()* funckiju počinje treniranje modela. Funkcija prima slijedeće parametre:

- *train_dataset*: Skup podataka za treniranje.
- *steps_per_epoch*: Broj *batch*-eva po epohi.
- *epochs*: Broj epoha za treniranje.
- *validation_data*: Skup podataka za validaciju.
- *validation_steps*: Broj *batch-eva* za validaciju.
- *callbacks*: Lista *callback* funkcija

Po završetku treniranja, model se sprema u obliku .h5 datoteke te se grafički prikazuju rezultati treniranja. Na slici 3.39. prikazan je primjer programskog koda za pokretanje postupka treniranja modela i spremanja rezultata treniranja.

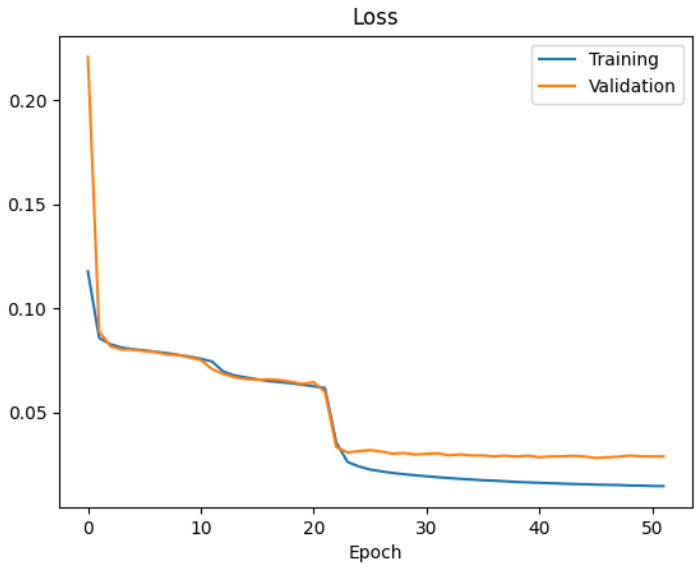
```
   def train_model(model, train_dataset, val_dataset, epochs, steps_per_epoch,
validation_steps):
       callbacks = [
           tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=6,
verbose=1, restore_best_weights=True),
```

```
            tf.keras.callbacks.ModelCheckpoint('best_multi_4.h5',
save_best_only=True, monitor='val_loss', mode='auto', verbose=1),
            PrintLREveryNBatch(),
            ShuffleTrainingDataCallback(train_csv_path='multi_train.csv')
        ]
        history = model.fit(
            train_dataset,
            steps_per_epoch=steps_per_epoch,
            epochs=epochs,
            validation_data=val_dataset,
            validation_steps=validation_steps,
            callbacks=callbacks,
            verbose=1
        )
        return history
    input_csv_path = 'multi_dataset.csv'
    train_csv_path = 'multi_train.csv'
    val_csv_path = 'multi_val.csv'
    batch_size = 100
    steps_per_epoch, validation_steps = split_and_save_data(input_csv_path,
train_csv_path, val_csv_path, batch_size)
    epochs = 64
    history = train_model(model, train_dataset, val_dataset, epochs,
steps_per_epoch, validation_steps)
    model.save('multi_4.h5')
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.legend(['Training', 'Validation'])
    plt.title('Loss')
    plt.xlabel('Epoch')
    plt.savefig('multi_4.png')
```

Slika 3.39: *Primjer programskog koda za pokretanje postupka treniranja modela, spremanja rezultata treniranja u obliku .h5 datoteke i grafički prikaz tijeka treniranja*

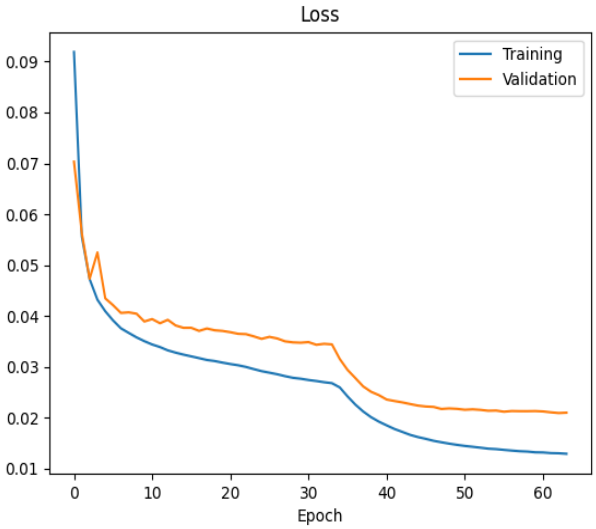
Funkcija gubitka za treniranje mjeri koliko dobro model predviđa izlazne vrijednosti za skup podataka za treniranje. Tijekom svake epohe, vrijednost funkcije gubitka izračunava se na temelju predikcija modela i stvarnih izlaznih vrijednosti (npr. predviđena vrijednost upravljača,

kočnice i gasa u odnosu na stvarne vrijednosti). Funkcija gubitka za validaciju mjeri koliko dobro model predviđa izlazne vrijednosti za skup podataka za validaciju. Ovo je važno kako bi se provjerila generalizacija modela, odnosno kako će model raditi na novim, dosad neviđenim podacima. Kako bi se odredio najbolji model na kojem će se provesti evaluacija, postupak treniranja ponovljen je nekoliko puta s različitim verzijama baza podataka. Razlika je bila u snimljenim slikama RGB kamere, odnosno kutu vidnog polja kamere. Kao što je prethodno navedeno, slike RGB kamere snimljene su pri promjeni kuta vidnog polja između četiri vrijednosti: 105, 110, 115 i 120 stupnjeva. Model s najboljim rezultatima bio je onaj koji je koristio bazu podataka s kutom vidnog polja od 115 stupnjeva, te je kao takav odabran za evaluaciju u CARLA simulatoru. Ponavljanjem postupka treniranja, svaki pokušaj razlikovao se u vrijednostima parametara kao što su *batch_size*, *learning_rate*, broj epoha i ostali. Model s najboljim rezultatima imao je vrijednosti tih i ostalih bitnih parametara prema programskoj skripti koja je prethodno prikazana kao primjer, a značenja svih parametara su objašnjena. Na slici 3.40. prikazane su vrijednosti funkcija gubitka za treniranje i validaciju tijekom procesa treniranja odabranog modela.



Slika 3.40: *Prikaz rezultata funkcija gubitka tijekom treniranja i validacije za model koji kombinira RGB i dubinsku kameru kao ulazne senzore*

Na slici 3.41. prikazane su vrijednosti funkcija gubitka treniranja i validacije tijekom procesa treniranja modela za najbolji model koji koristi samo RGB kameru kao ulazni senzor.



Slika 3.41: *Prikaz rezultata funkcija gubitka tijekom treniranja i validacije za model koji koristi samo RGB kameru kao ulazni senzor*

Prilikom treniranja koristi se srednje kvadratna pogreška (engl. *Mean Squared Error* - MSE) kao funkciju gubitka za sve tri izlazne vrijednosti gas, kočnica i kut zakreta volana (engl. *throttle, brake, steering_angle*). Funkcija gubitka mjeri koliko su predikcije modela različite od stvarnih vrijednosti, s ciljem minimiziranja razlike tijekom treniranja. MSE funkcija izračunava kvadrat razlike između predikcija modela i stvarnih vrijednosti te radi prosjek razlike. MSE je odabrana jer daje veću težinu većim pogreškama, što pomaže u postizanju preciznijih predikcija. Uz funkciju gubitka, performanse modela evaluirane su pomoću srednje apsolutne pogreške (engl. Mean Absolute Error - MAE) za svaki izlaz (gas, kočnica , kut zakreta volana). MAE mjeri prosječno apsolutno odstupanje predikcija modela od stvarnih vrijednosti, te je odabrana kao metrika koja pruža dodatni uvid u preciznost modela. Na primjer, za jedan *batch* podataka, MAE mjeri srednju apsolutnu grešku na temelju predikcije kontrola za upravljanjem automobilom koje je model dao kao izlaz te ih uspoređuje s kontrolama iz baze podataka za iste okvire slika iz *batcha*. Rezultati MAE metrike po završetku treniranja prikazani su u tablici 3.1.

throttle MAE	0.0530
brake MAE	0.0538
steering angle MAE	0.0277

Tablica 3.1: *Rezultati MAE metrike po završetku treniranja modela koji koristi RBG i dubinsku kameru*

Po završetku treniranja spremaju se dva modela kao rezultati treniranja. Prvi model je model koji se sprema na kraju treniranja s određenom vrijednosti *loss* funkcije, dok drugi model prati

najbolju, odnosno najmanju vrijednost *loss* funkcije. Kako se vrijednost *loss* funkcije smanjuje, najbolji model se ažurira. Odabrani model za evaluaciju čiji će rezultati biti sljedeće prikazani je završio treniranje nakon 56. epohe zbog prestanka smanjenja *loss* funkcije. To se postiglo aktiviranjem *EarlyStopping* povratne funkcije (engl. *Callback*) koja je dio *tensorflow.keras.callbacks* biblioteke i brine se o ranijem zaustavljanju treniranja modela. Parametri koji su korišteni tijekom treniranja su prikazani u tablici 3.2.

batch_size	100
epochs	64
learning_rate	0.00035 (ExponentialDecay)
loss_weights: throttle brake steering_angle	 0.7 1.0 2.0
optimizer	Adam

Tablica 3.2: *Prikaz hyperparametara korištenih za treniranje modela*

4. EVALUACIJA PERFORMANSI MODELA ZA AUTONOMNO UPRAVLJANJE VOZILOM KORISTEĆI END-TO-END PRISTUP

Za evaluaciju performansi modela unutar CARLA simulatora koristit će se dvije mape koje nisu bile dio baze podataka i jedna mapa koja je bila dio baze podataka. Za svaku mapu bit će kreirane tri testne rute na kojima će se evaluirati performanse modela. Proces evaluacije provodit će se deset puta za svaku rutu kako bi se dobio bolji uvid u statistiku performansi na temelju više ponavljanja. Evaluacije će se provoditi u istim vremenskim uvjetima koji su bili zadani tijekom prikupljanja baze podataka, ali će se za deset različitih ponavljanja mijenjati gustoća prometa. Gustoća prometa mjerit će se brojem vozila stvorenih na mapi, što će utjecati na vrijeme trajanja vožnje i mogući broj pogrešaka prilikom upravljanja automobilom. Nakon završetka evaluacije u standardnim vremenskim uvjetima, provest će se dodatna evaluacija u promjenjivim vremenskim uvjetima, kao što su kiša, magla ili noć. Time će se dobiti uvid u sposobnost generalizacije treniranog modela u uvjetima koji nisu bili dio baze podataka ili čine vrlo mali dio podataka u bazi podataka tijekom treniranja. Model za proces evaluacije koristi RGB i dubinsku kameru te će sljedeći rezultati evaluacije prikazivati rezultate toga modela. Naknadno, nakon evaluacije modela koji koristi RGB i dubinsku kameru, provesti će se kratka evaluacija modela na nepoznatoj mapi koji koristi samo RGB kameru kako bi se mogli ukratko usporediti rezultati ta dva modela. Tijekom evaluacije performansi pratit će se moguće pogreške koje model može napraviti za vrijeme upravljanja automobilom. Moguće pogreške koje model može napraviti tijekom upravljanja automobilom i koje su unaprijed definirane za praćenje su:

- Prolazak kroz crveno svijetlo na raskrižju
- Prolazak kroz crveno svjetlo i sudar unutar raskrižja
- Zeleno svjetlo nije prepoznato na raskrižju
- Nepravilno skretanje unutar raskrižja
- Nepravilno skretanje unutar raskrižja i sudar unutar raskrižja
- Sudar s automobilom ispred
- Zaustavljanje na neprikladnom mjestu
- Zaustavljanje na raskrižju bez semafora i neprepoznavanje prometne situacije
- Prelazak u drugu prometnu traku ili preko linija na kolniku

Za potrebe evaluacije, potrebno je napisati skriptu sličnu onoj za prikupljanje podataka. Radi se inicijalizacija testnog okruženja, spajanje sa simulatorom, stvaranje vozila i ostale radnje. Kako bi se generirala ruta kojom će vozilo voziti, koristi se *GlobalRoutePlanner*. Potrebno je predati

početnu i krajnju točku, a *GlobalRoutePlanner* kao rezultat vraća upute za vozilo koje je potrebno slijediti kako bi se stiglo od početne do krajnje točke. Na slici 4.1. prikazan je primjer programskog koda pomoću kojega se mogu dobiti upute za vožnju od početne do krajnje točke.

```
    def create_route_planner(world, start, destination):
        grp = GlobalRoutePlanner(world.get_map(), 2.0)
        return grp.trace_route(start.location, destination.location)

    route = create_route_planner(world, start_spawn_point,
destination_spawn_point)
    intersection_options = get_intersection_road_options(route)
```

Slika 4.1: *Primjer koda za dohvaćanje uputa o vožnji od početne do krajnje točke*

Pozivom funkcije *get_intersection_road_options*, u varijablu se spremaju samo upute o skretanjima koja je potrebno realizirati na pojedinim raskrižjima. Izvršavanjem beskonačne *while* petlje, model započinje s autonomnom vožnjom po zadanoj ruti. Na slici 4.2. prikazan je programski kod beskonačne *while* petlje u kojoj se neprestano poziva funkcija *manual_predict()* te joj se predaju parametri potrebni za predviđanje kontrole nad automobilom.

```
    try:
        while True:
            # Prikaži RGB sliku
            cv2.imshow("RGB_Image", sensor_data['rgb_image'])
            if cv2.waitKey(1) == ord('q'):
                break
            waypoint_index = get_closest_waypoint(my_vehicle, route)
            lookahead_index = min(waypoint_index + 10, len(route) - 1)
            lookahead_road_option = route[lookahead_index][1]
            # Ažuriraj trenutnu opciju skretanja ukoliko je skretanje blizu
            if lookahead_road_option in [LEFT, STRAIGHT, RIGHT]:
                current_road_option = lookahead_road_option
                target_waypoint_index = lookahead_index
            # Vrati se na praćenje trake nakon prolaska ciljanog pokazatelja
            if target_waypoint_index != -1 and waypoint_index >=
target_waypoint_index:
                current_road_option = LANEFOLLOW
                target_waypoint_index = -1
            road_option_int = map_road_option_to_int(current_road_option)
```

```
            manual_predict(my_vehicle, road_option_int)
            world.tick()
```

Slika 4.2: *Primjer programskog koda beskonačne petlje potrebne za evaluaciju modela*

Funkcija *manual_predict()* je prethodno objašnjena u potpoglavlju 3.5. Kontrolni modul, te je dan detaljan opis rada funkcije. Kao što je prethodno navedeno, za svaku mapu kreirati će se 3 rute. Kako bi se osigurali različiti testni uvjeti na istoj mapi, kreirane su rute sa što manjim postotkom preklapanja između ruta, a da su pri tome dovoljno zahtjevne. Na slikama 4.4., 4.5. i 4.6. prikazane su iscrtane rute u CARLA simulatoru na kojima će se provesti testiranje performansi modela. Rute se odnose na grad *Town01*.

Slika 4.4: *Prikaz prve rute na prvoj mapi*

Slika 4.5: *Prikaz druge rute na prvoj mapi*

Slika 4.6: *Prikaz treće rute na prvoj mapi*

Za svaku rutu biti će kreirana tablica s podacima o performansama modela tijekom evaluacije i podacima o samoj ruti na kojoj je provedena evaluacija. Prva mapa na kojoj će model biti testiran nije dio ukupne baze podataka, što znači da modelu okruženje nije poznato. Mjera autonomije predstavlja uspješnost modela u performansama autonomne vožnje. Formula 4.1. predstavlja postupak računanja autonomije:

$$autonomija = \left(1 - \frac{(broj\ intervencija)\ x\ 6\ sekundi}{ukupno\ vrijeme\ vožnje\ [s]}\right) x\ 100 \qquad (4\text{-}1)$$

Mjera autonomije dobivena je brojanjem ljudskih intervencija tijekom testiranja modela. Intervencija je potrebna kada se automobil ne ponaša u skladu s prometnim propisima (npr. prelazak linije, prolazak kroz crveno svjetlo i slično) ili kada model tijekom upravljanja automobilom pogrešno predviđa kontrole, što može rezultirati pogrešnim skretanjem ili zaustavljanjem na nepredviđenom mjestu. Šest sekundi određeno je kao prosječno vrijeme potrebno za ljudsku intervenciju, odnosno za preuzimanje kontrole nad vozilom. Vremenski uvjeti za testiranje na prvoj ruti identični su onima korištenim tijekom prikupljanja podataka na drugim mapama, tj. uvjeti su bili dobri. Duljina prve rute iznosi 815 metara, a broj raskrižja koje model mora proći je pet. Rezultati evaluacije modela prilikom vožnje po prvoj su prikazani u tablici 4.1.

Rb.	Gustoća prometa (broj automobila na mapi)	Ukupno vrijeme vožnje u sekundama	Broj intervencija	Autonomija u postotcima	Pogreške tijekom vožnje koje je model napravio
1.	10	293	4	91.8	Prolazak kroz crveno: 1 Prelazak preko linije: 1 Nepravilno skretanje unutar raskrižja : 3
2.	20	296	4	91.89	Prolazak kroz crveno: 1 Prelazak preko linije: 2 Nepravilno skretanje unutar raskrižja : 3
3.	30	292	3	93.9	Prelazak preko linije: 1 Nepravilno skretanje unutar raskrižja : 2
4.	40	298	3	93.9	Prolazak kroz crveno: 2 Prelazak preko linije što je uzrokovalo sudarom : 1 Nepravilno skretanje unutar raskrižja : 1
5.	50	345	4	93.1	Prolazak kroz crveno: 1 Sudar unutar raskrižja: 1 Nepravilno skretanje unutar raskrižja : 3
6.	65	349	5	91.4	Prolazak kroz crveno: 1 Prolazak kroz crveno svijetlo i sudar unutar raskrižja : 1 Prelazak preko linije: 1 Nepravilno skretanje unutar raskrižja : 3
7.	80	429	4	94.4	Nepravilno skretanje unutar raskrižja: 3 Nepravilno skretanje i sudar unutar raskrižja: 1
8.	100	416	4	94.2	Nepravilno skretanje unutar raskrižja: 3 Nepravilno skretanje i sudar unutar raskrižja: 1
9.	120	443	5	93.1	Nepravilno skretanja unutar raskrižja: 3 Nepravilno skretanje i sudar unutar raskrižja: 1 Prelazak preko linije: 2 Zaustavljanje na neprikladnom mjestu: 1
10.	145	538	6	93.3	Nepravilno skretanja unutar raskrižja: 4

					Nepravilno skretanje i sudar unutar raskrižja: 1 Prelazak preko linije: 1 Zaustavljanje na neprikladnom mjestu: 2

Tablica 4.1: *Rezultati evaluacije modela koji koristi RGB i dubinsku kameru tijekom vožnje po prvoj ruti unutar prve mape*

Evaluacijom modela tijekom vožnje na prvoj ruti dobiveni su rezultati koji prikazuju očekivano ponašanje. Za rutu iste dužine, povećanjem prometa povećavalo se i vrijeme potrebno da model završi vožnju. To podrazumijeva duže čekanje na semaforima i sporiju vožnju. Broj intervencija varirao je između tri i šest, ovisno o situacijama tijekom vožnje. Primjerice, intervencije su bile češće ako su na svim raskrižjima bila vozila, ili rjeđe ako su se vozila pojavljivala samo na nekoliko njih. Broj pogrešaka zabilježenih tijekom određene vožnje u tablici nije nužno povezan s brojem intervencija tijekom te vožnje. Na primjer, model ponekad uspije sam ispraviti pogreške, poput prelaska preko linije, bez potrebe za intervencijom. Također, prolazak kroz crveno svjetlo ne zahtijeva intervenciju ako nije došlo do sudara. Automobil može proći kroz crveno svjetlo i napraviti pogrešku, ali intervencija nije potrebna ako se vožnja može uspješno nastaviti. Većina intervencija tijekom vožnje odnosi se na ispravljanje vožnje prilikom skretanja na raskrižjima, gdje model ponekad pokuša skrenuti lijevo ili desno, ali se sudari s vozilom u suprotnoj traci ili pređe na nogostup, što zahtijeva intervenciju. Prosječan postotak autonomije za ovu rutu iznosi 93,1%. Vremenski uvjeti tijekom testiranja na drugoj ruti identični su onima na prvoj ruti. Duljina druge rute iznosi 664 metra, a broj raskrižja kroz koja model mora proći je šest. Rezultati evaluacije modela prilikom vožnje po drugoj ruti bit će prikazani u tablici 4.2.

Rb.	Gustoća prometa (broj automobila na mapi)	Ukupno vrijeme vožnje u sekundama	Broj intervencija	Autonomija u postotcima	Greške tijekom vožnje koje je model napravio
1.	10	244	5	87.7	Prelazak preko linije: 2 Nepravilno skretanje unutar raskrižja: 3
2.	20	251	4	90.4	Prelazak preko linije: 2 Nepravilno skretanje unutar raskrižja: 3
3.	30	276	5	89.1	Prelazak preko linije: 2 Prolazak kroz crveno: 1 Nepravilno skretanje unutar raskrižja: 3

4.	40	254	4	90.5	Prelazak preko linije: 2 Nepravilno skretanje unutar raskrižja: 3 Nepravilno skretanje i sudar unutar raskrižja: 1
5.	50	276	4	91.3	Prelazak preko linije: 1 Nepravilno skretanja unutar raskrižja: 2 Nepravilno skretanje i sudar unutar raskrižja: 2
6.	65	297	5	89.8	Prelazak preko linije: 2 Nepravilno skretanja unutar raskrižja: 3 Nepravilno skretanje i sudar unutar raskrižja: 1
7.	80	292	4	91.7	Prelazak preko linije: 2 Nepravilno skretanje unutar raskrižja: 3 Nepravilno skretanje i sudar unutar raskrižja: 2
8.	100	316	6	88.6	Nepravilno skretanja unutar raskrižja: 3 Nepravilno skretanje i sudar unutar raskrižja: 2 Prelazak preko linije: 2 Zaustavljanje na neprikladnom mjestu: 2
9.	120	327	6	88.9	Nepravilno skretanja unutar raskrižja: 4 Nepravilno skretanje i sudar unutar raskrižja: 1 Prelazak preko linije: 2 Zaustavljanje na neprikladnom mjestu: 1
10.	140	344	5	91.2	Nepravilno skretanja unutar raskrižja: 3 Nepravilno skretanje i sudar unutar raskrižja: 2 Prelazak preko linije: 2

Tablica 4.2: *Rezultati evaluacije modela koji koristi RGB i dubinsku kameru tijekom vožnje po drugoj ruti unutar prve mape*

Evaluacijom modela tijekom vožnje na drugoj ruti dobiveni su rezultati koji prikazuju slično ponašanje kao i tijekom testiranja na prvoj ruti. Za rutu iste dužine, povećanjem prometa povećavalo se i vrijeme potrebno da model završi vožnju. Broj intervencija varirao je između četiri i šest, ovisno o situacijama tijekom vožnje, slično kao i na prvoj ruti. Broj pogrešaka zabilježenih tijekom određene vožnje nije nužno povezan s brojem intervencija za tu vožnju. Tipovi pogrešaka

su vrlo slični, ali za razliku od prve rute, na ovoj ruti model je pravio veći broj pogrešaka, poput prelaska preko linije i nepravilnog skretanja, što je uzrokovalo sudare u raskrižjima. Prosječan postotak autonomije za ovu rutu iznosi 89,9%. Vremenski uvjeti tijekom testiranja na trećoj ruti identični su onima na prvoj i drugoj ruti. Duljina treće rute iznosi 726 metara, a broj raskrižja kroz koja model mora proći je šest. Rezultati evaluacije modela prilikom vožnje po trećoj ruti bit će prikazani u tablici 4.3.

Rb.	Gustoća prometa (broj automobila na mapi)	Ukupno vrijeme vožnje u sekundama	Broj intervencija	Autonomija u postotcima	Greške tijekom vožnje koje je model napravio
1.	10	308	5	92.2	Prelazak preko linije: 2 Nepravilno skretanje unutar raskrižja: 4 Zeleno svijetlo nije prepoznato na raskrižju: 1
2.	20	305	5	90.2	Prelazak preko linije: 1 Nepravilno skretanje unutar raskrižja: 4 Zeleno svijetlo nije prepoznato na raskrižju: 1
3.	30	302	5	90.0	Prelazak preko linije: 2 Prolazak kroz crveno: 1 Nepravilno skretanje unutar raskrižja: 3 Zeleno svijetlo nije prepoznato na raskrižju: 1
4.	40	310	6	88.4	Prelazak preko linije: 2 Nepravilno skretanje unutar raskrižja: 3 Zeleno svijetlo nije prepoznato na raskrižju: 1 Zaustavljanje na neprikladnom mjestu: 1
5.	50	307	6	88.3	Prelazak preko linije: 2 Nepravilno skretanje unutar raskrižja: 2 Prolazak kroz crveno: 1 Zeleno svijetlo nije prepoznato na raskrižju: 1
6.	65	325	5	90.7	Prelazak preko linije: 2 Nepravilno skretanje unutar raskrižja: 3 Sudar s autom ispred: 1 Zeleno svijetlo nije prepoznato na raskrižju: 1

7.	80	382	5	92.1	Prelazak preko linije: 2 Nepravilno skretanja unutar raskrižja: 4 Prolazak kroz crveno: 1
8.	100	356	6	89.9	Nepravilno skretanja unutar raskrižja: 3 Nepravilno skretanje i sudar unutar raskrižja: 1 Prelazak preko linije: 2 Zaustavljanje na neprikladnom mjestu: 2 Zeleno svijetlo nije prepoznato na raskrižju: 1
9.	120	413	7	89.8	Nepravilno skretanja unutar raskrižja: 4 Nepravilno skretanje i sudar unutar raskrižja: 1 Prelazak preko linije: 2 Zaustavljanje na neprikladnom mjestu: 1 Zeleno svijetlo nije prepoznato na raskrižju: 1
10.	140	421	7	90.0	Nepravilno skretanja unutar raskrižja: 4 Zaustavljanje na neprikladnom mjestu: 2 Sudar s autom ispred: 1 Nepravilno skretanje i sudar unutar raskrižja: 1 Prelazak preko linije: 2

Tablica 4.3: *Rezultati evaluacije modela koji koristi RGB i dubinsku kameru tijekom vožnje po trećoj ruti unutar prve mape*

Evaluacijom modela tijekom vožnje na trećoj ruti dobiveni su rezultati koji prikazuju slično ponašanje kao i prilikom testiranja na prvoj i drugoj ruti. Za rutu iste dužine, povećanjem prometa povećavalo se i vrijeme potrebno da model završi vožnju. Broj intervencija varirao je između pet i sedam, ovisno o situacijama tijekom vožnje. Broj pogrešaka zabilježenih tijekom određene vožnje nije nužno povezan s brojem intervencija za tu vožnju. Tipovi pogrešaka su vrlo slični kao kod prve i druge rute, dok se na trećoj ruti izdvaja neprepoznavanje zelenog svjetla na raskrižju. Ta pogreška može biti uzrokovana osvjetljenjem, odnosno situacijom kada je RGB kamera okrenuta prema suncu. Dakle, broj intervencija ne znači nužno jednak broj pogrešaka. Pogreška prilikom skretanja može imati različite ishode, na primjer, automobil može skrenuti s nepravilnom putanjom i sudariti se s drugim objektom ili početi voziti po pločniku. Također, ista pogreška može značiti da je automobil krivudao po raskrižju, prešao u drugu prometnu traku, ali se vratio u

ispravnu traku i nastavio vožnju. Prosječan postotak autonomije za ovu rutu iznosi 90,16%. Prosječan postotak autonomije za sve tri rute na kojima je model evaluiran iznosi 91,05%.

Sljedeća mapa je *Town04* koja kao i mapa *Town01* nije dio ukupne baze podataka. Odabir ruta je proveden na isti način kao i za prvu mapu, odnosno pokušavajući odrediti rute s najmanjim poklapanjem. Kreirane su ukupno tri rute za mapu *Town04*. Na slikama 4.7., 4.8. i 4.9. prikazane su iscrtane rute u CARLA simulatoru na kojima će se provesti testiranje performansi modela.

Slika 4.7: *Prikaz prve rute na drugoj mapi*

Slika 4.8: *Prikaz druge rute na drugoj mapi*

Slika 4.9: *Prikaz treće rute na drugoj mapi*

Duljina prve rute iznosi 588 metara, a broj raskrižja koje model mora proći je šest. Broj automobila na ovoj mapi će biti odmah veći u startu zbog veličine mape i broja mogućih mjesta na kojima se automobili mogu stvoriti u CARLA simulatoru. Rezultati evaluacije modela prilikom vožnje po prvoj su prikazani u tablici 4.4.

Rb.	Gustoća prometa (broj automobila na mapi)	Ukupno vrijeme vožnje u sekundama	Broj intervencija	Autonomija u postotcima	Pogreške tijekom vožnje koje je model napravio
1.	50	246	4	90.2	Prolazak kroz crveno: 2 Prelazak preko linije: 1 Nepravilno skretanje unutar raskrižja : 2 Zaustavljanje na neprikladnom mjestu: 1 Zeleno svijetlo nije prepoznato na raskrižju: 1
2.	65	252	4	90.4	Prolazak kroz crveno: 1 Prelazak preko linije: 2 Nepravilno skretanje unutar raskrižja : 2 Zaustavljanje na neprikladnom mjestu: 1 Zeleno svijetlo nije prepoznato na raskrižju: 2
3.	70	261	5	88.5	Prolazak kroz crveno: 2 Prelazak preko linije: 1

					Nepravilno skretanje unutar raskrižja : 2 Zeleno svijetlo nije prepoznato na raskrižju: 1
4.	85	266	5	88.7	Prolazak kroz crveno: 1 Prelazak preko linije: 2 Nepravilno skretanje unutar raskrižja : 1 Zeleno svijetlo nije prepoznato na raskrižju: 1 Prolazak kroz crveno svijetlo i sudar unutar raskrižja : 1
5.	100	277	5	89.1	Prolazak kroz crveno: 1 Prelazak preko linije: 1 Nepravilno skretanje unutar raskrižja : 2 Zeleno svijetlo nije prepoznato na raskrižju: 2
6.	120	283	4	91.5	Prolazak kroz crveno: 2 Prelazak preko linije: 1 Nepravilno skretanje unutar raskrižja : 2 Zeleno svijetlo nije prepoznato na raskrižju: 1
7.	140	298	5	89.9	Prolazak kroz crveno: 1 Prelazak preko linije: 2 Nepravilno skretanje unutar raskrižja : 1 Nepravilno skretanje i sudar unutar raskrižja : 1 Zeleno svijetlo nije prepoznato na raskrižju: 1
8.	160	313	6	88.4	Prolazak kroz crveno: 2 Prelazak preko linije: 2 Nepravilno skretanje unutar raskrižja : 1 Nepravilno skretanje i sudar unutar raskrižja : 1 Zaustavljanje na neprikladnom mjestu: 1 Zeleno svijetlo nije prepoznato na raskrižju: 1
9.	180	317	5	90.5	Prolazak kroz crveno: 1 Nepravilno skretanja unutar raskrižja: 1 Nepravilno skretanje i sudar unutar raskrižja: 1 Prelazak preko linije: 2

					Zaustavljanje na neprikladnom mjestu: 1 Zaustavljanje na neprikladnom mjestu: 1
10.	200	329	7	87.2	Prolazak kroz crveno: 2 Prelazak preko linije: 2 Nepravilno skretanje unutar raskrižja : 2 Zeleno svijetlo nije prepoznato na raskrižju: 1 Zaustavljanje na neprikladnom mjestu: 1 Prolazak kroz crveno svijetlo i sudar unutar raskrižja : 1

Tablica 4.4: *Rezultati evaluacije modela koji koristi RGB i dubinsku kameru tijekom vožnje po prvoj ruti unutar druge mape*

Kao i tijekom evaluacije na prvoj mapi, broj intervencija nije direktno povezan s brojem ukupnih grešaka koje je vozilo napravilo. Razlog toga je prethodno objašnjen. Tijekom evaluacije na ovoj mapi povećanjem ukupnog broja vozila povećavalo se i potrebno vrijeme za prolazak rute. Model je uspješno prepoznao sva potrebna skretanja, ali neka nije uspio realizirati, što je rezultat bio sudar s drugim vozilom ili zaustavljanje. Tijekom svih 10 vožnji, model je imao problema s prepoznavanjem zelenog svijetla na semaforu. Taj problem može biti uzrokovan slabom rezolucijom slike koja se predaje modelu za predviđanje kontrola. Slika 4.10. prikazuje primjer raskrižja na kojemu model ne može prepoznati zeleno svijetlo te je potrebna intervencija kako bi se automobil pokrenuo.

Slika 4.10: *Primjer raskrižja gdje automobil nije uspio prepoznati paljenje zelenoga svijetla*

Broj intervencija varirao je između pet i sedam, ovisno o situacijama tijekom vožnje. Prosječan postotak autonomije za ovu rutu iznosi 89,44%. Duljina druge rute iznosi 490 metara, a

broj raskrižja kroz koja model mora proći je šest. Rezultati evaluacije modela prilikom vožnje po drugoj ruti bit će prikazani u tablici 4.5.

Rb.	Gustoća prometa (broj automobila na mapi)	Ukupno vrijeme vožnje u sekundama	Broj intervencija	Autonomija u postotcima	Pogreške tijekom vožnje koje je model napravio
1.	50	311	5	90.3	Prolazak kroz crveno: 3 Prelazak preko linije: 1 Nepravilno skretanje unutar raskrižja : 2 Zeleno svijetlo nije prepoznato na raskrižju: 1
2.	65	316	5	90.5	Prolazak kroz crveno: 3 Prelazak preko linije: 2 Nepravilno skretanje unutar raskrižja : 2 Zeleno svijetlo nije prepoznato na raskrižju: 2
3.	70	314	5	90.4	Prolazak kroz crveno: 2 Prelazak preko linije: 1 Nepravilno skretanje i sudar unutar raskrižja : 2 Zeleno svijetlo nije prepoznato na raskrižju: 1 Prolazak kroz crveno svijetlo i sudar unutar raskrižja : 1
4.	85	327	5	90.8	Prolazak kroz crveno: 2 Prelazak preko linije: 2 Nepravilno skretanje unutar raskrižja : 1 Zeleno svijetlo nije prepoznato na raskrižju: 2 Prolazak kroz crveno svijetlo i sudar unutar raskrižja : 1
5.	100	340	6	89.4	Prolazak kroz crveno: 2 Prelazak preko linije: 2 Nepravilno skretanje unutar raskrižja : 1 Nepravilno skretanje i sudar unutar raskrižja : 1 Zeleno svijetlo nije prepoznato na raskrižju: 2 Prolazak kroz crveno svijetlo i sudar unutar raskrižja : 1

6.	120	343	6	89.5	Prolazak kroz crveno: 2 Prelazak preko linije: 1 Nepravilno skretanje unutar raskrižja : 1 Nepravilno skretanje i sudar unutar raskrižja : 1 Zeleno svijetlo nije prepoznato na raskrižju: 1 Prolazak kroz crveno svijetlo i sudar unutar raskrižja : 2
7.	140	359	6	89.9	Prolazak kroz crveno: 2 Prelazak preko linije: 1 Nepravilno skretanje unutar raskrižja : 2 Nepravilno skretanje i sudar unutar raskrižja : 1 Zeleno svijetlo nije prepoznato na raskrižju: 2 Prolazak kroz crveno svijetlo i sudar unutar raskrižja : 1
8.	160	365	7	88.4	Prolazak kroz crveno: 2 Prelazak preko linije: 1 Nepravilno skretanje unutar raskrižja : 2 Nepravilno skretanje i sudar unutar raskrižja : 1 Zaustavljanje na neprikladnom mjestu: 1 Zeleno svijetlo nije prepoznato na raskrižju: 2 Prolazak kroz crveno svijetlo i sudar unutar raskrižja : 1
9.	180	385	6	90.6	Prolazak kroz crveno: 2 Nepravilno skretanja unutar raskrižja: 2 Nepravilno skretanje i sudar unutar raskrižja: 1 Prelazak preko linije: 2 Zeleno svijetlo nije prepoznato na raskrižju: 2
10.	200	388	7	89.1	Prolazak kroz crveno: 3 Prelazak preko linije: 2 Nepravilno skretanje unutar raskrižja : 2 Zeleno svijetlo nije prepoznato na raskrižju: 1

					Prolazak kroz crveno svijetlo i sudar unutar raskrižja : 1

Tablica 4.5: *Rezultati evaluacije modela koji koristi RGB i dubinsku kameru tijekom vožnje po drugoj ruti unutar druge mape*

Tijekom evaluacije na drugoj ruti, povećanjem ukupnog broja vozila povećavalo se i potrebno vrijeme za prolazak rute, slično kao i tijekom evaluacije prve rute. Model je uspješno prepoznao sva zadana skretanja, ali neka nije uspio realizirati, što je rezultat bio sudar s drugim vozilom ili zaustavljanje. Za razliku od prve rute, povećao se broj prolazaka kroz crveno, a neki prolasci su rezultirali sudarom. Prema tome povećao se broj intervencija. Tijekom svih 10 vožnji, model je imao problema s prepoznavanjem zelenog svijetla na semaforu. Taj problem može biti uzrokovan slabom rezolucijom slike koja se predaje modelu za predviđanje kontrola kao i tijekom evaluacije prve rute. Broj intervencija varirao je između pet i sedam, ovisno o situacijama tijekom vožnje. Prosječan postotak autonomije za ovu rutu iznosi 89,89%. Duljina treće rute iznosi 588 metara, a broj raskrižja kroz koja model mora proći je tri. Rezultati evaluacije modela prilikom vožnje po drugoj ruti bit će prikazani u tablici 4.6.

Rb.	Gustoća prometa (broj automobila na mapi)	Ukupno vrijeme vožnje u sekundama	Broj intervencija	Autonomija u postotcima	Pogreške tijekom vožnje koje je model napravio
1.	50	365	3	95.0	Prolazak kroz crveno: 1 Prelazak preko linije: 2 Nepravilno skretanje unutar raskrižja : 2 Zeleno svijetlo nije prepoznato na raskrižju: 1
2.	65	367	4	93.4	Prolazak kroz crveno: 1 Prelazak preko linije: 2 Nepravilno skretanje unutar raskrižja : 1 Zeleno svijetlo nije prepoznato na raskrižju: 1
3.	70	371	4	93.5	Prolazak kroz crveno: 1 Prelazak preko linije: 2 Nepravilno skretanje i sudar unutar raskrižja : 1 Zeleno svijetlo nije prepoznato na raskrižju: 1
4.	85	388	4	93.8	Prolazak kroz crveno: 1 Prelazak preko linije: 2

					Nepravilno skretanje unutar raskrižja : 2 Zeleno svijetlo nije prepoznato na raskrižju: 1
5.	100	392	5	92.3	Prelazak preko linije: 2 Nepravilno skretanje unutar raskrižja : 2 Prolazak kroz crveno svijetlo i sudar unutar raskrižja : 1
6.	120	393	5	92.3	Prolazak kroz crveno: 1 Prelazak preko linije: 2 Nepravilno skretanje unutar raskrižja : 2 Nepravilno skretanje i sudar unutar raskrižja : 1 Zeleno svijetlo nije prepoznato na raskrižju: 1
7.	140	412	5	92.7	Prelazak preko linije: 1 Nepravilno skretanje unutar raskrižja : 2 Zeleno svijetlo nije prepoznato na raskrižju: 1 Prolazak kroz crveno svijetlo i sudar unutar raskrižja : 1 Zaustavljanje na neprikladnom mjestu: 1
8.	160	411	4	94.1	Prolazak kroz crveno: 1 Prelazak preko linije: 2 Nepravilno skretanje unutar raskrižja : 1 Zaustavljanje na neprikladnom mjestu: 1 Zeleno svijetlo nije prepoznato na raskrižju: 1
9.	180	433	5	93.1	Prolazak kroz crveno: 1 Nepravilno skretanja unutar raskrižja: 2 Nepravilno skretanje i sudar unutar raskrižja: 1 Prelazak preko linije: 2 Zeleno svijetlo nije prepoznato na raskrižju: 1
10.	200	437	5	93.1	Prolazak kroz crveno: 1 Prelazak preko linije: 2 Nepravilno skretanje unutar raskrižja : 2

					Zeleno svijetlo nije prepoznato na raskrižju: 1 Zaustavljanje na neprikladnom mjestu: 1

Tablica 4.6: *Rezultati evaluacije modela koji koristi RGB i dubinsku kameru tijekom vožnje po trećoj ruti unutar druge mape*

Tijekom evaluacije na trećoj ruti, povećanjem ukupnog broja vozila povećavalo se i potrebno vrijeme za prolazak rute, slično kao tijekom evaluacije prve i druge rute. Model je uspješno prepoznao sva zadana skretanja, ali neka nije uspio realizirati, što je rezultat bio sudar s drugim vozilom ili zaustavljanje. Za razliku od prve i druge rute, broj prolazaka kroz crveno nije velik jer postoji samo jedan semafor na ruti. Većina intervencija odnosila se na samu vožnju po cesti, za razliku od prethodne dvije rute gdje su intervencije uglavnom bile potrebne u raskrižjima. Veliki dio rute se odnosi na otvorenu cestu s više traka te je to predstavljalo problem pri paralelnoj vožnji s nekim drugim autom ili pri promjeni vozne trake. Broj intervencija varirao je između tri i pet, ovisno o situacijama tijekom vožnje. Prosječan postotak autonomije za ovu rutu iznosi 93,33%.

Sljedeća mapa je *Town02* koja je dio ukupne baze podataka. Odabir ruta je proveden na isti način kao za prvu i drugu mapu, odnosno pokušavajući odrediti rute s najmanjim poklapanjem. Kreirane su ukupno tri rute za mapu *Town02*. Na slikama 4.10., 4.11. i 4.12. prikazane su iscrtane rute u CARLA simulatoru na kojima će se provesti testiranje performansi modela.

Slika 4.10: *Prikaz prve rute na trećoj mapi*

Slika 4.11: *Prikaz druge rute na trećoj mapi*

Slika 4.12: *Prikaz treće rute na trećoj mapi*

Duljina prve rute iznosi 354 metra, a broj raskrižja koje model mora proći je tri. Rezultati evaluacije modela prilikom vožnje po prvoj su prikazani u tablici 4.7.

Rb.	Gustoća prometa (broj automobila na mapi)	Ukupno vrijeme vožnje u sekundama	Broj intervencija	Autonomija u postotcima	Pogreške tijekom vožnje koje je model napravio
1.	10	189	3	90.4	Prelazak preko linije: 1 Nepravilno skretanje unutar raskrižja : 2
2.	20	185	3	90.2	Prelazak preko linije: 2 Nepravilno skretanje unutar raskrižja : 1
3.	30	188	3	90.4	Prelazak preko linije: 1 Nepravilno skretanje unutar raskrižja : 1 Zeleno svijetlo nije prepoznato na raskrižju: 1
4.	40	206	3	91.2	Prelazak preko linije: 1 Nepravilno skretanje unutar raskrižja : 1 Zeleno svijetlo nije prepoznato na raskrižju: 1
5.	50	207	3	91.2	Prelazak preko linije: 2 Nepravilno skretanje unutar raskrižja : 1
6.	65	225	4	92	Prelazak preko linije: 1 Nepravilno skretanje unutar raskrižja : 2 Zeleno svijetlo nije prepoznato na raskrižju: 1
7.	80	223	4	92.1	Prelazak preko linije: 2 Nepravilno skretanje unutar raskrižja : 2
8.	95	243	4	92.5	Prelazak preko linije: 2 Nepravilno skretanje unutar raskrižja : 2 Zeleno svijetlo nije prepoznato na raskrižju: 1
9.	110	246	4	92.6	Prolazak kroz crveno: 1 Nepravilno skretanja unutar raskrižja: 2 Nepravilno skretanje i sudar unutar raskrižja: 1
10.	125	261	4	93.1	Prolazak kroz crveno: 1 Prelazak preko linije: 1 Nepravilno skretanje unutar raskrižja : 2

Tablica 4.7: *Rezultati evaluacije modela koji koristi RGB i dubinsku kameru tijekom vožnje po prvoj ruti unutar treće mape*

Tijekom evaluacije na prvoj ruti, povećanjem ukupnog broja vozila povećavalo se i potrebno vrijeme za prolazak rute, slično kao tijekom evaluacije prve i druge mape. Model je uspješno prepoznao sva zadana skretanja, ali mali broj njih nije uspio realizirati. Većina grešaka se odnosila na prelazak linije te pogrešno skretanje. Neka raskrižja su veća te iz toga razloga automobil započne skretanje, ali cesta kojom bi trebao voziti ne dođe u vidno polje kamere te je potrebna intervencija. Broj intervencija se mijenjao između tri i četiri, ovisno o situacijama tijekom vožnje. Prosječan postotak autonomije za ovu rutu iznosi 91,57%. Može se vidjeti da se intervencije odnose samo na skretanje što je očekivano, ali model se bolje ponaša na poznatoj mapi. Greške kao što su prolazak kroz crveno i neprepoznavanje zelenog svijetla su minimalne. Postotak autonomije je sličan kao na mapama koje nisu dio baze za treniranje, ali razlog tomu je što su rute na tim mapama duže i vremenski više treba da se završe. Ako bi se uzele u obzir samo greške, vidjelo bi se da automobil pravi znatno manje grešaka na poznatoj mapi u odnosu na nepoznatu.

Duljina druge rute iznosi 282 metra, a broj raskrižja koje model mora proći je pet. Rezultati evaluacije modela prilikom vožnje po prvoj su prikazani u tablici 4.8.

Rb.	Gustoća prometa (broj automobila na mapi)	Ukupno vrijeme vožnje u sekundama	Broj intervencija	Autonomija u postotcima	Pogreške tijekom vožnje koje je model napravio
1.	10	174	3	89.6	Prelazak preko linije: 1 Nepravilno skretanje unutar raskrižja : 3
2.	20	175	3	89.7	Prelazak preko linije: 2 Nepravilno skretanje unutar raskrižja : 2
3.	30	178	3	89.8	Prelazak preko linije: 1 Nepravilno skretanje unutar raskrižja : 2 Zeleno svijetlo nije prepoznato na raskrižju: 1
4.	40	201	3	91	Prelazak preko linije: 1 Nepravilno skretanje unutar raskrižja : 3
5.	50	205	3	91.2	Prolazak kroz crveno: 1 Prelazak preko linije: 1 Nepravilno skretanje unutar raskrižja : 3
6.	65	219	4	91.7	Prelazak preko linije: 1 Nepravilno skretanje unutar raskrižja : 3 Zeleno svijetlo nije prepoznato na raskrižju: 1

7.	80	218	4	91.7	Prelazak preko linije: 2 Nepravilno skretanje unutar raskrižja : 2 Zeleno svijetlo nije prepoznato na raskrižju: 1
8.	95	236	4	92.3	Prelazak preko linije: 2 Nepravilno skretanje unutar raskrižja : 2 Zeleno svijetlo nije prepoznato na raskrižju: 1
9.	110	238	4	92.4	Prolazak kroz crveno: 1 Nepravilno skretanja unutar raskrižja: 3 Nepravilno skretanje i sudar unutar raskrižja: 1
10.	125	252	4	92.8	Prolazak kroz crveno: 1 Prelazak preko linije: 2 Nepravilno skretanje unutar raskrižja : 2

Tablica 4.8: *Rezultati evaluacije modela koji koristi RGB i dubinsku kameru tijekom vožnje po drugoj ruti unutar treće mape*

Tijekom evaluacije na drugoj ruti, povećanjem ukupnog broja vozila povećavalo se i potrebno vrijeme za prolazak rute, slično kao tijekom evaluacije prve rute. Broj intervencija se mijenjao između tri i četiri, ovisno o situacijama tijekom vožnje. Prosječan postotak autonomije za ovu rutu iznosi 91,22%. Model se slično ponašao kao i tijekom evaluacije prve rute što je očekivano te je tip grešaka isti.

Duljina treće rute iznosi 415 metara, a broj raskrižja koje model mora proći je tri. Rezultati evaluacije modela prilikom vožnje po trećoj ruti su prikazani u tablici 4.9.

Rb.	Gustoća prometa (broj automobila na mapi)	Ukupno vrijeme vožnje u sekundama	Broj intervencija	Autonomija u postotcima	Pogreške tijekom vožnje koje je model napravio
1.	10	155	0	100	
2.	20	157	0	100	
3.	30	158	1	96.2	Prelazak preko linije: 1
4.	40	160	0	100	
5.	50	185	1	96.7	Prolazak kroz crveno: 1
6.	65	181	0	100	
7.	80	175	0	100	Prelazak preko linije: 1
8.	95	195	1	96.9	Prelazak preko linije: 1

9.	110	185	0	100	
10.	125	192	0	100	

Tablica 4.9: *Rezultati evaluacije modela koji koristi RGB i dubinsku kameru tijekom vožnje po trećoj ruti unutar treće mape*

Tijekom evaluacije na trećoj ruti, povećanjem ukupnog broja vozila povećalo se i potrebno vrijeme za prolazak rute, što je isti slučaj kao tijekom evaluacije prve i druge rute. Prosječan postotak autonomije za ovu rutu iznosi 98,98%. Razlika ove rute u odnosu na prve dvije je što je ova ruta nije zahtjevna, nema puno skretanja, a prolazak kroz raskrižja je uglavnom ravno. Ovime se pokazalo kako model na poznatoj mapi i vožnjom po laganoj ruti ima skoro savršene rezultate. Greške su bile nekoliko prelazaka preko linije tijekom skretanja u zavoju ili na raskrižju te je automobil jednom prilikom prošao kroz crveno, prateći automobil ispred sebe.

Kako bi se provela evaluacija modela u različitim vremenskim uvjetima u odnosu na one u kojima je snimana baza podataka izabrana je prva ruta sa 50 stvorenih automobila unutar treće mape koja je dio ukupne baze podataka. Cilj ove evaluacije je prikazati sposobnost modela prilikom upravljanja vozilom pri lošim vremenskim uvjetima. Vremenski uvjeti koji će se koristiti su oblačno vrijeme, slaba kiša, jaka kiša, mrak te vlažno vrijeme bez kiše. Rezultati evaluacije prikazani su u tablici 4.10.

Rb.	Gustoća prometa (broj automobila na mapi)	Ukupno vrijeme vožnje u sekundama	Broj intervencija	Autonomija u postotcima	Pogreške tijekom vožnje koje je model napravio
1.	50	208	5	85.5	**Oblačno** Prolazak kroz crveno: 1 Prelazak preko linije: 2 Nepravilno skretanje unutar raskrižja : 2 Zeleno svijetlo nije prepoznato na raskrižju: 1
2.	50	209	6	82.7	**Slaba kiša** Prolazak kroz crveno: 2 Prelazak preko linije: 2 Nepravilno skretanje unutar raskrižja : 2 Zeleno svijetlo nije prepoznato na raskrižju: 2
3.	50	236	7	82.2	**Jaka kiša** Prolazak kroz crveno: 1 Prelazak preko linije: 2

					Nepravilno skretanje i sudar unutar raskrižja : 1 Nepravilno skretanje unutar raskrižja : 2 Zeleno svijetlo nije prepoznato na raskrižju: 2 Zaustavljanje na neprikladnom mjestu: 1
4.	50	224	6	83.9	**Mrak** Prolazak kroz crveno: 1 Prelazak preko linije: 3 Nepravilno skretanje unutar raskrižja : 3 Zeleno svijetlo nije prepoznato na raskrižju: 1
5.	50	212	6	83.1	**Vlažno** Prelazak preko linije: 2 Nepravilno skretanje unutar raskrižja : 2 Prolazak kroz crveno svijetlo i sudar unutar raskrižja : 1 Zaustavljanje na neprikladnom mjestu : 1 Zeleno svijetlo nije prepoznato na raskrižju: 1

Tablica 4.10: *Rezultati evaluacije modela koji koristi RGB i dubinsku kameru u različitim lošim vremenskim uvjetima vožnje po drugoj ruti unutar treće mape*

Rezultati evaluacije pokazuju kako promjene u vremenskim uvjetima unutar simulacijskog okruženja utječu na rezultate vožnje. Model je napravio više grešaka u odnosu na istu rutu po normalnim vremenskim uvjetima. Najviše grešaka se dogodilo tijekom vožnje po jakoj kiši, odnosno za vrijeme vožnje model je teže prepoznavao raskrižja, svijetlo na semaforu i ostale sudionike u prometu te je stoga napravio više grešaka i bilo je potrebe za većim brojem intervencija. Prosječni postotak autonomije je 83.48%.

Kako bi se dobila usporedba u performansama između multimodalnog modela koji koristi RGB i dubinsku kameru s modelom koji koristi samo RGB kameru izabrana je ista ruta koja je korištena za evaluaciju modela u različitim vremenskim uvjetima. Rezultati evaluacije modela koji koristi samo RGB kameru biti će prikazani u tablici 4.11.

Rb.	Gustoća prometa (broj automobila na mapi)	Ukupno vrijeme vožnje u sekundama	Broj intervencija	Autonomija u postotcima	Pogreške tijekom vožnje koje je model napravio
1.	10	191	4	87.3	Prolazak kroz crveno: 1 Prelazak preko linije: 1 Nepravilno skretanje unutar raskrižja : 2
2.	20	188	4	87.2	Prelazak preko linije: 2 Nepravilno skretanje unutar raskrižja : 2 Zeleno svijetlo nije prepoznato na raskrižju: 1
3.	30	190	5	84.3	Prolazak kroz crveno: 1 Prelazak preko linije: 2 Nepravilno skretanje i sudar unutar raskrižja : 2 Zeleno svijetlo nije prepoznato na raskrižju: 1
4.	40	211	4	88.6	Prelazak preko linije: 2 Nepravilno skretanje unutar raskrižja : 1 Zeleno svijetlo nije prepoznato na raskrižju: 1 Zaustavljanje na neprikladnom mjestu: 1
5.	50	209	5	85.6	Prolazak kroz crveno: 1 Prelazak preko linije: 2 Nepravilno skretanje unutar raskrižja : 1 Nepravilno skretanje i sudar unutar raskrižja : 1 Zeleno svijetlo nije prepoznato na raskrižju: 1 Prolazak kroz crveno svijetlo i sudar unutar raskrižja : 1
6.	65	231	6	84.4	Prolazak kroz crveno: 2 Prelazak preko linije: 2 Nepravilno skretanje unutar raskrižja : 1 Zeleno svijetlo nije prepoznato na raskrižju: 1 Zaustavljanje na neprikladnom mjestu: 1
7.	80	230	5	86.9	Prolazak kroz crveno: 2 Prelazak preko linije: 2 Nepravilno skretanje unutar raskrižja : 2

					Sudar s automobilom ispred: 1
8.	95	255	6	85.8	Prolazak kroz crveno: 1 Prelazak preko linije: 2 Nepravilno skretanje unutar raskrižja : 2 Nepravilno skretanje i sudar unutar raskrižja : 1 Zaustavljanje na neprikladnom mjestu: 1 Sudar s automobilom ispred: 1
9.	110	254	6	85.8	Prolazak kroz crveno: 2 Nepravilno skretanja unutar raskrižja: 1 Prelazak preko linije: 2 Zeleno svijetlo nije prepoznato na raskrižju: 1 Zaustavljanje na neprikladnom mjestu: 1
10.	125	274	6	86.8	Prolazak kroz crveno: 1 Prelazak preko linije: 2 Nepravilno skretanje unutar raskrižja : 2 Zeleno svijetlo nije prepoznato na raskrižju: 1 Zaustavljanje na neprikladnom mjestu: 1 Sudar s automobilom ispred: 1

Tablica 4.11: *Rezultati evaluacije modela koji koristi samo RGB kameru tijekom vožnje po prvoj ruti unutar treće mape*

Rezultati evaluacije pokazuju znatno lošije rezultate evaluacije modela koji koristi samo RGB kameru u odnosu na model koji koristi RGB i dubinsku kameru. Model je radio više grešaka koje se odnose na prepoznavanje daljine, odnosno više puta je prošao kroz crveno ili se nije na vrijeme zaustavio te se više puta nepravilno zaustavio na cesti ili sudario s drugim automobilom. Nepravilno zaustavljanje bilo je rezultat kasnog prepoznavanja vozila ispred automobila s kojim upravlja model te naglo kočenje dok je sudar s automobilom ispred rezultat toga da model nije prepoznao drugi automobil ispred sebe i nije zakočio. Sukladno tome, bilo je potrebno više vremena za prolazak rute. Prosječan postotak autonomije je 86.27% što je manje u odnosu na postotak od 91.57% koji je ostvario model korištenjem RGB i dubinske kamere. Razlika između uspješnosti ta dva modela bi bila izraženija kada bi se modeli evaluirali na više dužih ili težih ruta te bi se dobio bolji uvid u performanse tih dvaju modela.

ZAKLJUČAK

Cilj ovog diplomskog rada bio je razviti sustav za autonomnu vožnju koristeći konvolucijsku neuronsku mrežu unutar simulacijskog okruženja, CARLA simulatora. Sustav se temelji na kombinaciji nekoliko ulaznih podataka, slike RGB i dubinske kamere, podatak o brzini i trenutnoj naredbi skretanja. Arhitektura mreže obuhvaća nekoliko konvolucijskih slojeva koji su zaduženi za procesiranje slike te potpuno povezani slojevi koji integriraju brzinu vozila i opcije skretanja. Izlaz mreže su naredbe za upravljanje vozilom, gas, kočnica i kut zakreta volana. Model je treniran koristeći standardne tehnike strojnog učenja, pri čemu su se optimizirali parametri kao što su *batch_size,* broj epoha i *learning_rate.* Evaluacije modela neuronske mreže se izvršila unutar tri različite mape simulatora te na 3 različite rute na pojedinačnoj mapi. Dvije mape nisu bile dio ukupne baze podataka koja je služila za treniranje modela, dok je jedna mapa dio ukupne baze podataka. Prilikom evaluacije na nepoznatim mapama, model je pokazao zadovoljavajuće sposobnosti autonomne vožnje s prosjekom autonomnosti od 91.11%. Model je uspješno prolazio manja raskrižja te blage zavoje, dok su veća raskrižja bila problem zbog nepreglednosti. Na rutama s manjom gustoćom prometa dogodilo se manje grešaka, odnosno što je bilo više auta na mapi više je mogućnosti za greške. Na rutama gdje se pojavljuju velika raskrižja, javljao se problem ne prepoznavanja svijetla na semaforu zbog velike udaljenosti samoga semafora. Automobil nekada nije stao na crveno, a nekada nije krenuo na zeleno svijetlo. Primjer takvog slučaja dan je tijekom evaluacije. Suprotno od nepoznatih mapa, model je na jednoj poznatoj mapi pokazao vrlo dobre rezultate. Tijekom evaluacije na poznatoj mapi, model je imao manji broj grešaka u odnosu na nepoznate mape, što je i očekivano. Greške su se odnosile na prelazak preko linije ili na nepravilno skretanje unutar raskrižja dok problema s prolaskom kroz crveno svijetlo ili nepravilnim zaustavljanjem nije bilo. S obzirom na to da je velik dio baze podataka sniman u istim vremenskim uvjetima, napravljena je evaluacije na poznatoj mapi u promjenjivim vremenskim uvjetima kao što su kiša i oblačno vrijeme. Rezultati toga testa su više grešaka prilikom vožnje u odnosu na vožnju u vremenskim uvjetima jednakim prilikom snimanja baze podataka. Zaključak na temelju rezultata evaluacije je da model može uspješno predvidjeti ponašanje automobila u većini prometnih situacija. Model nije u mogućnosti uspješno predvidjeti ponašanje tijekom vožnje unutar velikih raskrižja i tijekom različitih vremenskim uvjeta u odnosu na one u kojima je snimana baza podataka za treniranje. Dodatno, evaluacijom modela koji koristi samo RGB kameru može se zaključiti kako korištenjem dodatnoga senzora, odnosno dubinske kamere postotak autonomije se povećava te model pravi manje grešaka tijekom vožnje. Kako bi se ove greške smanjile, potrebno je snimiti veću bazu podataka koja će sadržavati vožnju po različitim vremenskim

uvjetima te prilikom treniranja modela neuronske mreže dodatno izmijeniti strukturu mreže i vrijednosti hiperparametara uzimajući u obzir veću bazu podataka i potrebu za upravljanjem automobila u težim uvjetima vožnje.

LITERATURA

[1] “Benz Patent Motor Car: The first automobile (1885–1886) | Mercedes-Benz Group > Company > Tradition > Company History.”, S interneta: https://group.mercedes-benz.com/company/tradition/company-history/1885-1886.html [30.7.2024.]

[2] “How Vehicle Safety Has Improved Over the Decades | NHTSA., S interneta: https://www.nhtsa.gov/how-vehicle-safety-has-improved-over-decades. [30.7.2024.]

[3] “Car Accidents and Human Error | LawInfo.”, S interneta: https://www.lawinfo.com/resources/car-accident/how-many-car-accidents-are-caused-by-human-er.html. [30.7.2024]

[4] “Što su autonomna vozila? 6 mitova i stvarnosti - DIR.hr.”, S interneta: https://dir.hr/sto-su-autonomna-vozila/. [30.7.2024]

[5] “CARLA Simulator.”, S interneta: https://carla.org//. [30.7.2024]

[6] “[1906.03199v2] Multimodal End-to-End Autonomous Driving.”, S interneta: https://arxiv.org/abs/1906.03199v2. [30.7.2024]

[7] L. A. Rosero, I. P. Gomes, J. A. R. da Silva, C. A. Przewodowski, D. F. Wolf, and F. S. Osório, “Integrating Modular Pipelines with End-to-End Learning: A Hybrid Approach for Robust and Reliable Autonomous Driving Systems,” *Sensors*, vol. 24, no. 7, p. 2097, Jan. 2024, doi: 10.3390/s24072097. S interneta: https://www.mdpi.com/1424-8220/24/7/2097. [25.7.2024]

[8] “Multimodal End to End Autonomous Driving via Conditional Imitation Learning | Charles Nimo.”, S interneta: https://charlesnimo.me/project/robot-learning/. [30,7,2024]

[9] “[1912.00177] Urban Driving with Conditional Imitation Learning.”, S interneta: https://arxiv.org/abs/1912.00177. [30.7.2024]

[10] “General Python FAQ — Python 3.12.4 documentation.”, S interneta: https://docs.python.org/3/faq/general.html. [30.7.2024]

[11] “What is Anaconda? | Domino Data Science Dictionary.”, S interneta: https://domino.ai/data-science-dictionary/anaconda. [30.7.2024]

[12] “What is Tensorflow | TensorFlow Introduction - Javatpoint”, S interneta: https://www.javatpoint.com/tensorflow-introduction. [30.7.2024]

[13] “About - OpenCV.”, S interneta: https://opencv.org/about/. [30.7.2024]

[14] “Understanding NVIDIA CUDA: The Basics of GPU Parallel Computing.”, S interneta: https://www.turing.com/kb/understanding-nvidia-cuda. [30.7.2024]

[15] “What is NumPy? — NumPy v2.0 Manual.”, S interneta: https://numpy.org/doc/stable/user/whatisnumpy.html. [30.7.2024]

[16] “Introduction - CARLA Simulator.”, S interneta: https://carla.readthedocs.io/en/latest/start_introduction/. [30.7.2024]

[17] “What are Convolutional Neural Networks? | IBM.”, S interneta: https://www.ibm.com/topics/convolutional-neural-networks. [30.7.2024]

SAŽETAK

U ovom radu dan je pregled postojećih rješenja za autonomnu vožnju unutar CARLA simulatora koja su zasnovana na *end-to-end* načinu rada. Izrađen je novi sustav koji se također temelji na *end-to-end* načinu rada koristeći CARLA simulator kao simulacijsko okruženje. Izrađeni sustav koristi kombinaciju više ulaza kako bi uspješno predvidio upravljanje automobilom, odnosno radi se predviđanje vrijednosti gasa, kočnice i kuta zakreta volana. Ulazi su slike RGB i dubinske kamere te informacije o trenutnoj brzini vozila i naredbi za skretanje. Cilj rada je dizajnirati, trenirati i evaluirati model s ciljem autonomne vožnje. Model je treniran na podacima koji su prikupljeni u simulacijskom okruženju CARLA simulatoru, a evaluacija je provedena na tri mape. Dvije od tri evaluacijske mape nisu dio ukupne baze podataka, dok je jedna mapa dio baze podataka koja se koristila za treniranje. Neuronska mreža sastoji se od više konvolucijskih slojeva koji obrađuju vizualne podatke RGB i dubinske kamere dok se brzina vozila i naredbe za skretanje unose u potpuno povezane slojeve. Rezultati evaluacije pokazali su kako se model dobro snalazi u normalnim uvjetima vožnje, odnosno u jednostavnijim prometnim situacijama, dok prilikom složenijih prometnih situacija moguće su greške prilikom upravljanja automobilom. Također, broj grešaka koje je model napravio tijekom vožnje na poznatoj mapi je znatno manji od broja grešaka na nepoznatim mapama što ukazuje na potrebu proširenja baze podataka s različitim uvjetima vožnje. Ovaj rad prikazuje mogućnosti primjene konvolucijskih neuronskih mreža u području autonomne vožnje koristeći kombinaciju senzorskih podataka. Model koji koristi kombinaciju senzorskih podatak je pokazao bolje rezultate tijekom evaluacije u odnosu na model koji koristi samo jedan senzor, RGB kameru.

Ključne riječi: *end-to-end,* konvolucija, neuronska mreža, CARLA simulator, autonomna vožnja, baza podataka

End-to-end multimodal autonomous vehicle control

ABSTRACT

This paper provides an overview of existing solutions for autonomous driving within the CARLA simulator, which are based on end-to-end mode of operation. A new system was developed which is also based on the end-to-end mode of operation using the CARLA simulator as the simulation environment. The developed system uses a combination of several inputs to successfully predict the car's steering, i.e. the gas, brake and steering angle values are predicted. Inputs are RGB and depth camera images and information about current vehicle speed and turn commands. The aim of the work is to design, train and evaluate a model with the goal of autonomous driving. The model was trained on data collected in the simulation environment CARLA simulator, and the evaluation was performed on three maps. Two of the three evaluation folders are not part of the overall database, while one folder is part of the database used for training. The neural network consists of multiple convolutional layers that process RGB and depth camera visual data while vehicle speed and turn commands are fed into fully connected layers. The evaluation results showed that the model copes well in normal driving conditions, i.e. in simpler traffic situations, while in more complex traffic situations mistakes are possible when driving the car. Also, the number of errors made by the model while driving on a known map is significantly lower than the number of errors on unknown maps, which indicates the need to expand the database with different driving conditions. This paper shows the possibilities of applying convolutional neural networks in the field of autonomous driving using a combination of sensor data. The model that uses a combination of sensor data showed better results during the evaluation compared to the model that uses only one sensor, the RGB camera.

Keywords: end-to-end, convolution, neural network, CARLA simulator, autonomous driving, database

ŽIVOTOPIS

Matija Barić rođen je 17.10.200. u Slavonskome Brodu. Završio je srednjoškolsko obrazovanje u Tehničkoj školi Slavonski Brod. Nakon srednje škole upisuje preddiplomski sveučilišni studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku. Akademski naziv sveučilišni prvostupnik (baccalaureus) inženjer računarstva stječe 2022. godine. Iste godine upisao je diplomski sveučilišni studij računarstva, smjer automobilsko računarstvo i komunikacije te nakon položenog testiranja postaje stipendist TTTech Auto CEE u Osijeku.

PRILOZI

Prilog P.3.1: Primjer programskog koda za definiranje arhitekture modela neuronske mreže

```
    def create_model():
        # Definiranje ulaza za slike
        image_input = Input(shape=(240, 400, 3), name='image_input')
        image_input_2 = Input(shape=(144, 244, 3), name='image_input_2')
        velocity_input = Input(shape=(1,), name='velocity_input')
        road_option_input = Input(shape=(1,), dtype='int32', name='road_option')

        # Obrada prvog ulaza za slike (RGB kamera)
        x = Conv2D(36, (5, 5), activation='relu', strides=(2, 2),
padding='valid')(image_input)
        x = BatchNormalization()(x)
        x = Conv2D(40, (5, 5), activation='relu', strides=(2, 2),
padding='valid')(x)
        x = BatchNormalization()(x)
        x = Conv2D(48, (5, 5), activation='relu', strides=(2, 2),
padding='valid')(x)
        x = BatchNormalization()(x)
        x = Conv2D(64, (3, 3), activation='relu', strides=(1, 1),
padding='valid')(x)
        x = BatchNormalization()(x)
        x = Conv2D(64, (3, 3), activation='relu', strides=(1, 1),
padding='valid')(x)
        x = BatchNormalization()(x)
        print(f"Shape of x before flatten: {x.shape}")
        x = Flatten()(x)

        # Obrada drugog ulaza za slike (dubinska kamera)
        y = Conv2D(36, (5, 5), activation='relu', strides=(2, 2),
padding='valid')(image_input_2)
        y = BatchNormalization()(y)
        y = Conv2D(40, (5, 5), activation='relu', strides=(2, 2),
padding='valid')(y)
        y = BatchNormalization()(y)
```

```
        y = Conv2D(48, (5, 5), activation='relu', strides=(2, 2),
padding='valid')(y)
        y = BatchNormalization()(y)
        y = Conv2D(64, (3, 3), activation='relu', strides=(1, 1),
padding='valid')(y)
        y = BatchNormalization()(y)
        y = Conv2D(64, (3, 3), activation='relu', strides=(1, 1),
padding='valid')(y)
        y = BatchNormalization()(y)
        print(f"Shape of y before flatten: {y.shape}")
        y = Flatten()(y)
        y = Dense(512, activation='relu')(y)

        # Obrada ulaza za brzinu vozila
        velocity_input2 = Dense(64, activation='relu')(velocity_input)
        velocity_input2 = Dropout(0.2)(velocity_input2)
        velocity_input2 = Dense(16, activation='relu')(velocity_input2)
        velocity_input2 = Dropout(0.2)(velocity_input2)

        # Obrada ulaza za opciju skretanja
        road_option_embedding = Embedding(input_dim=4,
output_dim=32)(road_option_input)
        road_option_embedding = Flatten()(road_option_embedding)

        # Spajanje svih ulaza
        concatenated = concatenate([x, y, velocity_input2,
road_option_embedding])
        concatenated = Dense(512, activation='relu')(concatenated)
        concatenated = Dropout(0.3)(concatenated)
        concatenated = Dense(256, activation='relu')(concatenated)
        concatenated = Dropout(0.3)(concatenated)
        concatenated = Dense(128, activation='relu')(concatenated)
        concatenated = Dropout(0.25)(concatenated)
        concatenated = Dense(64, activation='relu')(concatenated)
        concatenated = Dropout(0.2)(concatenated)
        concatenated = Dense(16, activation='relu')(concatenated)

        # Definiranje izlaza modela
```

```
        throttle = Dense(1, name='throttle')(concatenated)
        brake = Dense(1, name='brake')(concatenated)
        steering_angle = Dense(1, name='steering_angle')(concatenated)

        # Kreiranje modela
        model = Model(inputs=[image_input, image_input_2, velocity_input,
road_option_input],
                      outputs=[throttle, brake, steering_angle])

        # Kompilacija modela
        adam_optimizer = tf.keras.optimizers.Adam(learning_rate=lr_schedule)
        model.compile(
            optimizer=adam_optimizer,
            loss={'throttle': 'mse', 'brake': 'mse', 'steering_angle': 'mse'},
            loss_weights={'throttle': 0.7, 'brake': 1.0, 'steering_angle': 2.0},
            metrics={
                'throttle': [throttle_mae, 'mae'],
                'brake': [brake_mae, 'mae'],
                'steering_angle': [steering_angle_mae, 'mae']
            }
        )
        model.summary()
        return model

    model = create_model()
```

Prilog P.3.2: Primjer programskog koda za unos smetnji u kontrolu autopilota.

```
    def add_noise_teleport(vehicle):
        transform = vehicle.get_transform()
        steer_check = np.absolute(vehicle.get_control().steer)
        if steer_check < 0.12 and np.absolute(transform.rotation.pitch) < 1:
            chosen_range = random.choice([(-60, -30), (30, 60)])
            angle = random.randint(*chosen_range)
            distance = random.uniform(0.7, 1.3)
            location = carla.Location(transform.location.x + np.sin(angle) *
distance, transform.location.y + np.cos(angle) * distance, transform.location.z)
            rotation = carla.Rotation(transform.rotation.pitch,
transform.rotation.yaw + (np.random.random() * 2 - 1) * 5,
transform.rotation.roll)
            vehicle.set_transform(carla.Transform(location, rotation))

        if steer_check >= 0.12 and steer_check < 0.25 and
np.absolute(transform.rotation.pitch) < 1:
            chosen_range = random.choice([(-45, -25), (25, 45)])
            angle = random.randint(*chosen_range)
            distance = random.uniform(0.5, 0.95)
            location = carla.Location(transform.location.x + np.sin(angle) *
distance, transform.location.y + np.cos(angle) * distance, transform.location.z)
            rotation = carla.Rotation(transform.rotation.pitch,
transform.rotation.yaw + (np.random.random() * 2 - 1) * 5,
transform.rotation.roll)

            vehicle.set_transform(carla.Transform(location, rotation))

        if steer_check >= 0.25 and steer_check < 0.45 and
np.absolute(transform.rotation.pitch) < 1:
            chosen_range = random.choice([(-35, -20), (20, 35)])
            angle = random.randint(*chosen_range)
            distance = random.uniform(0.3, 0.6)
            location = carla.Location(transform.location.x + np.sin(angle) *
distance, transform.location.y + np.cos(angle) * distance, transform.location.z)
```

```
            rotation = carla.Rotation(transform.rotation.pitch,
transform.rotation.yaw + (np.random.random() * 2 - 1) * 5,
transform.rotation.roll)

            vehicle.set_transform(carla.Transform(location, rotation))

        if steer_check > 0.45 and np.absolute(transform.rotation.pitch) < 1:
            chosen_range = random.choice([(-17, -7), (7, 17)])
            angle = random.randint(*chosen_range)
            distance = random.uniform(0.1, 0.2)
            location = carla.Location(transform.location.x + np.sin(angle) *
distance, transform.location.y + np.cos(angle) * distance, transform.location.z)
            rotation = carla.Rotation(transform.rotation.pitch,
transform.rotation.yaw + (np.random.random() * 2 - 1) * 5,
transform.rotation.roll)

            vehicle.set_transform(carla.Transform(location, rotation))
```

Prilog P.3.3: Primjer koda za inicijalizaciju okruženja potrebnog za treniranje neuronske mreže

```
import os
import pandas as pd
import numpy as np
import cv2
import random
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, Conv2D, Flatten
from tensorflow.keras.layers import Dropout, BatchNormalization, Embedding,
Reshape, concatenate
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint,
Callback
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from tensorflow.keras.backend import clear_session

# Clear any session leftovers
clear_session()

# Set memory growth on GPUs
gpus = tf.config.experimental.list_physical_devices('GPU')
if gpus:
    try:
        for gpu in gpus:
            tf.config.experimental.set_memory_growth(gpu, True)
    except RuntimeError as e:
        print(e)
```

Prilog P.3.4: Primjer programskog koda za pripremu trening i validacijskog skupa podataka

```
    def load_and_preprocess_image(path):
        def load_image(img_path):
            img = tf.io.read_file(img_path)
            img = tf.image.decode_jpeg(img, channels=3)
            img = tf.cast(img, tf.float32) / 255.0
            return img
        return tf.map_fn(load_image, path, fn_output_signature=tf.float32)

    def load_and_preprocess_image_2(path):
        def load_image(img_path):
            img = tf.io.read_file(img_path)
            img = tf.image.decode_jpeg(img, channels=3)
            img = tf.cast(img, tf.float32) / 255.0
            return img
        return tf.map_fn(load_image, path, fn_output_signature=tf.float32)

    def prepare_dataset(train_csv_path, val_csv_path, batch_size, augment=True):
        column_defaults = [tf.float32, tf.float32, tf.float32, tf.float32,
tf.int32, tf.string, tf.string]
        column_names = ['throttle', 'brake', 'steering_angle', 'speed',
'road_option', 'image_path', 'image_path_2']

        train_dataset = tf.data.experimental.make_csv_dataset(
            train_csv_path,
            batch_size=batch_size,
            column_names=column_names,
            num_epochs=1,
            shuffle=True,
            num_parallel_reads=tf.data.AUTOTUNE,
            prefetch_buffer_size=tf.data.AUTOTUNE
        ).map(preprocess_row, num_parallel_calls=tf.data.AUTOTUNE).repeat()

        val_dataset = tf.data.experimental.make_csv_dataset(
            val_csv_path,
            batch_size=batch_size,
            column_names=column_names,
```

```
            column_defaults=column_defaults,
            num_epochs=1,
            shuffle=False,
            prefetch_buffer_size=tf.data.AUTOTUNE
        ).map(lambda x: preprocess_row(x, augment=False),
num_parallel_calls=tf.data.AUTOTUNE).repeat()

        return train_dataset, val_dataset

    def preprocess_row(data, augment=True):
        images = load_and_preprocess_image(data['image_path'])
        images_2 = load_and_preprocess_image_2(data['image_path_2'])

        if augment:
            images = augment_image_batch(images)
            images_2 = augment_image_batch(images_2)

        features = {
            'image_input': images,
            'image_input_2': images_2,
            'velocity_input': data['speed'],
            'road_option': data['road_option']
        }

        labels = {
            'throttle': data['throttle'],
            'brake': data['brake'],
            'steering_angle': data['steering_angle']
        }

        return (features, labels)

    train_dataset, val_dataset = prepare_dataset(train_csv_path, val_csv_path,
batch_size, augment=True)
```