УНИВЕРЗИТЕТ У НИШУ
ЕЛЕКТРОНСКИ ФАКУЛТЕТ
Катедра за рачунарство

УПРАВЉАЊЕ АУТОНОМНИМ ВОЗИЛИМА ПРИМЕНОМ КОНВОЛУЦИОНИХ НЕУРОНСКИХ МРЕЖА

- ДИПЛОМСКИ РАД -

Задатак:

Проучити методе за предикцију угла скретања у циљу управљања аутономним возилима коришћењем конволуционих неуронских мрежа. Упознати се са симулаторима који омогућавају обучавање ових мрежа. У практичном делу имплементирати прототип система који на основу података са камере на возилу одређује угао скретања.

Ментор: проф. др Александар Милосављевић **Кандидат:** Дамјан Ћупић 16959

Комисија:

1. _____ Датум пријаве: _____

2. _____ Датум предаје: _____

3. _____ Датум одбране: _____

Ниш, 2023.

Садржај

1. Увод ... 3
2. Аутономна возила ... 4
2.1. Историјски развој ... 4
2.2. SAE класификација ... 6
2.3. Сензори ... 7
2.4. Архитектура система ... 9
2.4.1. Модуларни систем ... 10
2.4.2. Систем од-краја-до-краја ... 11
3. Технике дубоког учења за управљање аутономним возилима ... 13
3.1. Конволуционе неуронске мреже ... 13
3.2. Имитационо учење ... 14
3.3. Учење поткрепљивањем ... 18
3.4. Симулатори аутономне вожње ... 20
3.4.1. CARLA ... 21
3.4.2. LGSVL ... 23
3.4.3. AirSim ... 24
3.4.4. Udacity Self-Driving Car Simulator ... 25
4. Имплементација прототипа система за предикцију угла скретања возила 28
5. Закључак ... 42
Литература ... 43

1. Увод

Са развојем технологије и аутомобилске индустрије развијала се и идеја о реализацији аутономних система, који ће моћи самостално да управљају возилима (без потребе за возачем). Употреба оваквих возила требало би да допринесе побољшању ефикасности и економичности вожње, па многе велике компаније раде на усавршавању својих модела како бисмо у будућности могли да их видимо на улицама.

Поред механичких унапређења, код савремених возила све већи значај има софтвер: аутоматизација је већ неко време заступљена у аутомобилима и у великој мери помаже возачима при управљању, а очекује се да ће ови системи моћи да надгледају окружење, да доносе одлуке и у потпуности замене возача. У процесу аутономног управљања возилом најзначајнију улогу имају сензори (попут камере, радара и лидара), заједно са алгоритмима вештачке интелигенције.

Код аутономних возила разликују се два архитектурна приступа: модуларни и од-краја-до-краја (енг. end-to-end), при чему је модуларна архитектура заступљенија у пракси. Код приступа од-краја-до-краја циљ је генерисати управљачке команде (угао скретања, притисак на педале гаса и кочнице) директно на основу опажања са сензора. За ту намену се могу искористити конволуционе неуронске мреже, које су погодне за обраду слика, односно података са камера. Једну од најпознатијих мрежа за генерисање угла скретања возила употребом камере представила је компанија NVIDIA. Њена мрежа, названа PilotNet, клонирала је понашање возача у процесу који се назива: имитационо учење. Поред овог начина обучавања, издваја се учење поткрепљивањем, које се заснива на интеракцији агента са окружењем.

Приликом тренирања аутономних система један од најзначајнијих корака јесте симулација, с обзиром на то да је потребно генерисати велику количину тренинг-података и тестирати много различитих ситуација које се могу догодити у саобраћају. У ове сврхе су развијени напредни симулаторски програми, који нуде широк спектар могућности. Неки од најпознатијих комерцијалних симулатора су CarCraft и Webviz, а међу симулаторима отвореног кода велику примену имају CARLA, LGSVL и AirSim.

Овај рад је организован у пет поглавља. После уводног дела, у другом поглављу су дати историјски преглед развоја аутономних возила, њихова класификација и осврт на најважније сензоре који се користе у овим возилима, а затим су размотрени архитектурни приступи према којима се реализују овакви системи. У трећем поглављу су обрађене технике дубоког учења, као што су имитационо учење и учење поткрепљивањем – које се могу користити за реализацију управљања аутономним возилима. Највише пажње је посвећено PilotNet конволуционој неуронској мрежи, која успешно генерише угао скретања директно на основу слика са камера закачених за возило. Након тога су истражене могућности симулатора отвореног кода за тренирање и тестирање аутономних система, међу којима се издваја CARLA симулатор (нуди најреалистичнију 3D графику). У четвртом поглављу, које обухвата имплементацију прототипа система за предикцију угла скретања возила, најпре је извршено тренирање неколико варијација PilotNet мреже над скупом података који садржи слике стварних путева. Затим је у другом делу пројекта поступак поновљен коришћењем Udacity Self-Driving Car симулатора, који је једноставан, али погодан за упознавање са разматраним проблемом. У првом случају је вршено упоређивање предиктованих са тачним вредностима за угао скретања, а у другом је мерено растојање виртуелног аутомобила од средине стазе којом се креће – при чему су резултати упоређивани коришћењем средње квадратне грешке. Последње поглавље садржи преглед ове области и анализу добијених резултата.

2. Аутономна возила

Аутономна, или самовозећа возила (енг. self-driving vehicle) су она која се могу безбедно кретати без потребе за интервенцијом од стране возача. Ова возила спознају своје окружење користећи сензоре, а добијене информације потом обрађује софтвер, како би одабрао најбољу путању до одредишта и управљао возилом (поступајући по саобраћајним правилима). Када је реч о терминологији, треба разликовати „аутоматизовано“ возило, које нуди помоћ возачу, и „потпуно аутономно“ возило, које се може кретати у саобраћају самостално. [1]

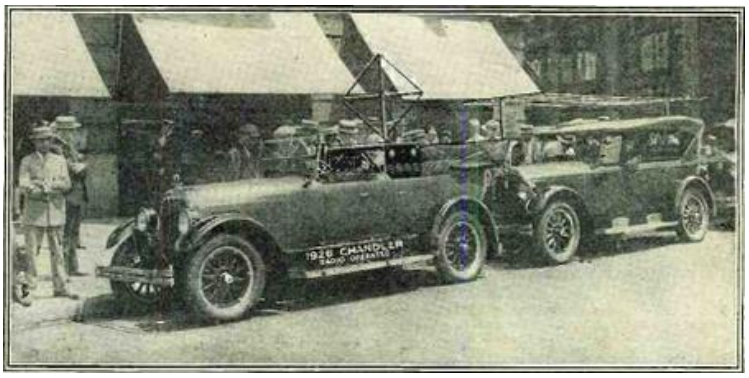
Један од главних разлога за увођење аутономних возила јесте смањивање броја саобраћајних несрећа, с обзиром на то да су оне у 94% случајева проузроковане људском грешком (вожња под дејством алкохола и наркотика, непажња, умор). [2] Поред тога, ова возила могу бити доступна и особама које из различитих разлога нису у могућности да управљају стандардним возилима, и могу побољшати функционисање јавног превоза и допринети смањењу гужви. Такође, њихова примена би утицала на смањење загађења околине, а као резултат би имала и уштеду новца (трошкови услед судара, транспорт људи и робе, гориво) и времена (које се уместо за управљање возилом може искористити за неке друге активности).

Ова технологија се суочава и са многобројним изазовима, а на првом месту је безбедност учесника у саобраћају: при томе је један од најзахтевнијих задатака дефинисање правила по којима ће возило поступати у ризичним ситуацијама. [1] Системи у аутономним возилима имају потешкоће са функционисањем у хаотичном градском окружењу (где је тешко предвидети намере осталих учесника саобраћаја – поготово пешака), а с обзиром на то да се ослањају на информације са сензора, на доношење одлука могу утицати и неповољни временски услови (киша, снег, магла, мрак). Такође, пошто аутономна возила користе услуге рачунарства у облаку и сарађују са другим возилима (и елементима путне инфраструктуре), може доћи до „сајбер напада“ и компромитовања комуникационих канала, као и до злоупотребе велике количине података генерисане приликом вожње. Поред технолошких изазова, ширу употребу ових возила успорава и потреба за доношењем и усаглашавањем законских норми којима се дефинише њихово учешће у саобраћају.

2.1. Историјски развој

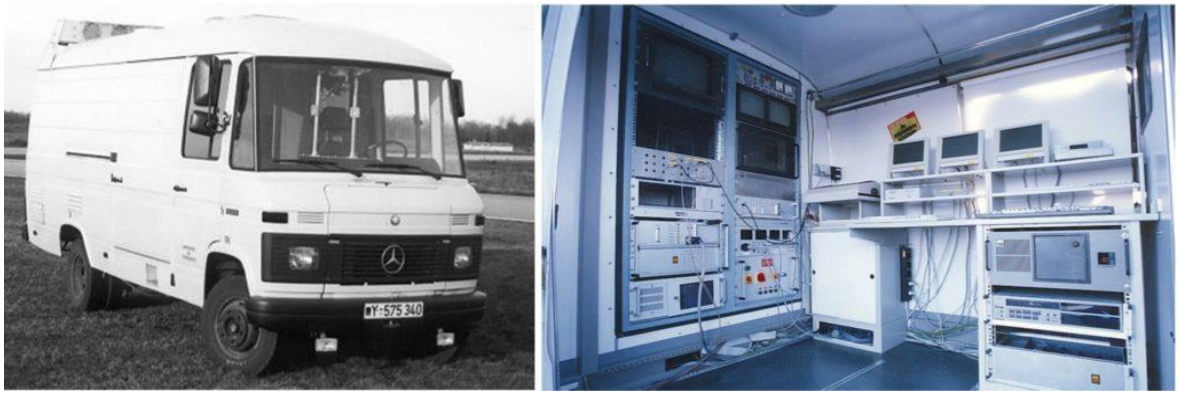
Истраживања везана за развој аутономних возила су привукла пажњу јавности када је 1925. године у Њујорку компанија Houdina Radio Control представила један од првих аутоматизованих система за вожњу. [3] Аутомобилом названим „Америчко Чудо“ (енг. „American Wonder“) управљано је радио сигналима из аутомобила који се кретао иза њега (приказано на слици бр. 1).

Следећи значајан период за развој ове технологије био је током педесетих година XX века, када су за управљање возилом коришћени електрични импулси (електрична кола уграђена у пут). Аутомобил компаније General Motors, назван „Firebird II“, користио је овакав систем за кретање. [4] Први полуаутоматизовани аутомобил са перцепцијом окружења направио је 1977. године тим из јапанске лабораторије Tsukuba. [5] Њихово возило је пратило беле ознаке на улици двема камерама и обрађивало податке аналогним рачунаром – а достигло је брзину од 30 km/h.



Слика 1: Аутомобил „Америчко Чудо“ [3]

Инжењер Ernst Dickmanns је 1980. године са својим тимом користио Mercedes-Benz комби (приказан на слици бр. 2) за тестирање аутономне вожње. [5] Овај комби, опремљен камерама, сензорима и рачунаром успешно је контролисао правац возила, убрзање и кочење, достижући брзину од 96 km/h (на улицама без саобраћаја). Dickmanns се сматра пиониром у развоју технологије за помоћ при вожњи, која се може срести у данашњим аутомобилима.



Слика 2: VaMoRs Mercedes комби [5]

Напретку развоја аутономних возила значајно су допринели пројекти Navlab, ALV и Prometheus, покренути 80-их година прошлог века. У оквиру ALV (Autonomous Land driven Vehicle) пројекта је реализовано прво управљање возилом коришћењем лидар система, рачунарског вида и вештачких неуронских мрежа. Значајна прекретница догодила се 1995. године, када је Navlab возило прешло 4,585 km од Питсбурга у Пенсилванији до Сан Дијега у Калифорнији, од чега 98.2% аутономно – крећући се просечном брзином од 102.7 km/h. Ово возило је заправо било полуаутономно, зато што је користило неуронске мреже за управљање воланом, а гас и кочнице били су контролисани од стране возача (из безбедносних разлога). [4]

Велико интересовање истраживачких тимова за ову област подстакла су такмичења „Grand Challenge“ из 2004. и 2005. године, као и „Urban Challenge“ из 2007. године, која је организовала DARPA (енг. Defense Advanced Research Projects Agency). У том периоду започети су бројни пројекти везани за аутономна возила, а један од најпознатијих је Гуглов пројекат из 2009. године. Нешто касније, 2015. године, компанија Tesla Motors додаје „Autopilot“ функционалност свом возилу Model S, што га сврстава међу прве доступне аутомобиле који могу самостално да се одржавају у саобраћајној траци, мењају траке и паркирају се. Исте године Гуглово возило „Firefly“

обавља прву потпуно аутономну вожњу на јавним путевима. [4] Децембра 2020. компанија Waymo (настала из Гугловог пројекта) прва комерцијализује аутономни такси сервис назван „Waymo One“, у Фениксу (Аризона, САД), а данас је ова услуга доступна и у Сан Франциску. На слици бр. 3 је приказано како изгледа Jaguar I-PACE возило опремљено „Waymo Driver“ технологијом. Данас многе велике компаније (Mercedes-Benz, General Motors, Cruise, Google, Uber, Toyota, Volvo) интензивно раде на развоју технологија које ће возилима омогућити што већи ниво аутономије.

Слика 3: Jaguar I-PACE аутомобил опремљен „Waymo Driver“ технологијом [6]

2.2. SAE класификација

Удружење инжењера аутомобилске индустрије (енг. Society of Automotive Engineers, SAE) дефинише шест нивоа аутономије у возилима, означених бројевима од нула до пет, при чему са порастом нивоа расте и ниво аутономије. [7] На слици бр. 4 илустровано је за које активности за задужен возач, а које може обављати систем на сваком нивоу.

	Управљање и убрзавање/ успоравање	Надгледање окружења	Возач преузима контролу ако систем откаже	Систем има потпуну контролу у свим условима
НИВО 0	Возач	Возач	Да	Не
НИВО 1	Возач/ Систем	Возач	Да	Не
НИВО 2	Систем	Возач	Да	Не
НИВО 3	Систем	Систем	Да	Не
НИВО 4	Систем	Систем	Не	Не
НИВО 5	Систем	Систем	Не	Да

Слика 4: Нивои аутономије у возилима

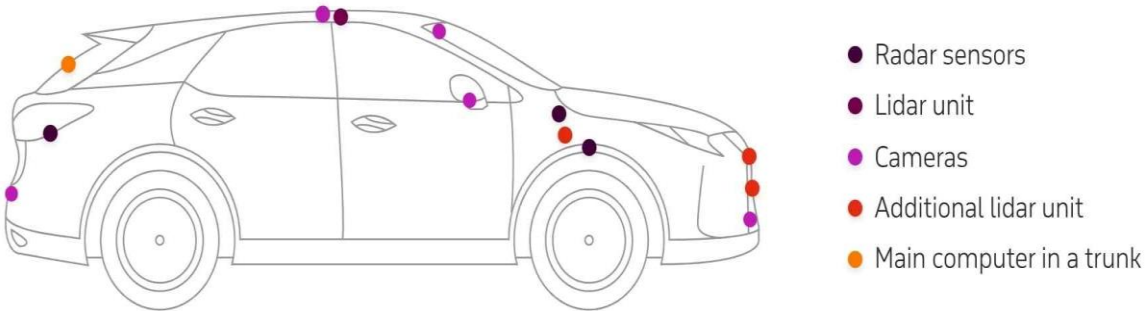
Ниво 0 се назива и ниво без аутоматизације – зато што систем може генерисати упозорења, али нема никакву контролу над возилом (већ је возач одговоран за управљање и надгледање окружења). Овом нивоу припада већина возила које можемо срести на путевима. На нивоу 1 аутоматизовани систем пружа помоћ возачу: возач и систем деле контролу над возилом (нпр. возач може контролисати управљач, а систем може да води рачуна о брзини), али возач мора бити спреман да преузме потпуно управљање у било ком тренутку. На нивоу 2 је постигнута делимична аутоматизација: систем може преузети контролу (управљач, убрзавање, кочење), али возач мора надгледати вожњу и бити спреман да реагује по потреби (контакт између волана и возачевих руку је обавезан).

Један од најзначајнијих прелаза је са нивоа 2 на ниво 3, зато што је од нивоа 3 систем задужен за надгледање окружења. Овај ниво подразумева условну аутоматизацију: возач не мора константно да обраћа пажњу на вожњу (може нпр. да гледа филм), јер ће систем водити рачуна о свим критичним ситуацијама, али мора бити спреман да у одређеном временском интервалу (који је дефинисан од стране произвођача) интервенише уколико се то од њега захтева (због чега су оваква возила најчешће ограничена на вожњу аутопутевима). Висока аутоматизација постиже се на нивоу 4, на коме возачева пажња није захтевана (он може спавати или напустити своје седиште). Међутим, оваква вожња је омогућена само на одређеним географским подручјима (за које постоје детаљне мапе и одговарајућа путна инфраструктура) и у специфичним условима. У осталим случајевима систем мора да безбедно прекине вожњу и паркира возило (уколико возач не преузме управљање). Потпуно аутономна вожња се постиже на нивоу 5, где човекова интервенција није захтевана ни у једном тренутку: возило се може кретати на свим путевима, било где у свету и при било којим временским условима (очекује се да оваква возила неће имати управљач и педале за додавање гаса и кочење).

Између мануелног управљања аутомобилима (SAE ниво 0) и потпуно аутономних аутомобила (SAE ниво 5) постоји велики број различитих модела возила који имају одређени ниво аутономије. [1] Најпознатије технологије за аутономно управљање су „Autopilot“ (ниво 2) компаније Tesla, „Drive Pilot“ (ниво 3) компаније Mercedes и „Waymo Driver“ (ниво 4) компаније Waymo. Без обзира на велики напредак технологије последњих година, и данас се комерцијално доступна возила са аутономним карактеристикама сврставају максимално у ниво 2, међутим, у другој половини 2023. године очекује се да се на тржишту појаве први Mercedes модели нивоа 3. Аутомобили нивоа 4 још увек нису доступни за ширу употребу, али се могу срести на улицама појединих места (такси услуге, возила за чишћење улица и доставу робе).

2.3. Сензори

Највећи значај међу хардверским елементима у аутономним возилима имају сензори, јер они омогућавају возилу да „види“ шта се дешава на путу. Радар, лидар и камера су главни сензори када је у питању перцепција окружења, а њихова најчешћа позиција је приказана на слици бр. 5. Поред ових сензора, користе се и додатни, као што су нпр. инерцијални сензори и GPS. Рачунар који управља возилом је повезан са свим сензорима и често се налази у гепеку, при чему се може сместити тако да остави довољно простора за коришћење путницима.



Слика 5: Најчешћа позиција сензора на аутомобилу [8]

Камере пружају возилу увид у своје окружење – користе се за детекцију и препознавање објеката (као што су друга возила, пешаци, бициклисти, саобраћајни знакови, светла на семафорима, ознаке на коловозу, светла на возилима хитних служби). [9] Неке од најзначајнијих предности камера у односу на друге сензоре јесу осетљивост на боје и могућност примене напредних алгоритама рачунарског вида. Поред тога, реч је о пасивним сензорима, који не емитују сигнале, тако да не долази до проблема мешања са другим сигналима. Међутим, подаци са камера некада могу бити непоуздани – при лошим временским условима и у ситуацијама када су боје објеката врло сличне боји позадине. Неке врсте камера које налазе примену у аутономним возилима су: камере са осветљењем, термалне камере, омнидирекционе камере (за поглед у 360°) и камере за детекцију догађаја (реагују на промене у осветљењу).

Радар (енг. RAdio Detection And Ranging, RADAR) сензори (слика бр. 6 лево) користе радио таласе: могу детектовати објекте независно од временских услова и омогућавају одређивање њихове удаљености и брзине. [9] Функционишу тако што предајник емитује радио таласе у одређеном смеру, а пријемник прикупља рефлектоване таласе и анализира их. Ови сензори су постављени око возила и имају улогу да обезбеде надгледање „мртвих“ углова, да помогну при одржању возила на средини коловозне траке, кочењу и паркирању, као и да контролишу удаљеност од других објеката. У поређењу са камером, радар мање прецизно моделује облик објеката, што као резултат може имати погрешно препознавање (нпр. идентификовање бициклисте као моториста). Са друге стране, обрада података са радара захтева мање рачунарске снаге у односу на камеру и лидар.

Слика 6: Радар (лево) и лидар (десно) сензори [10]

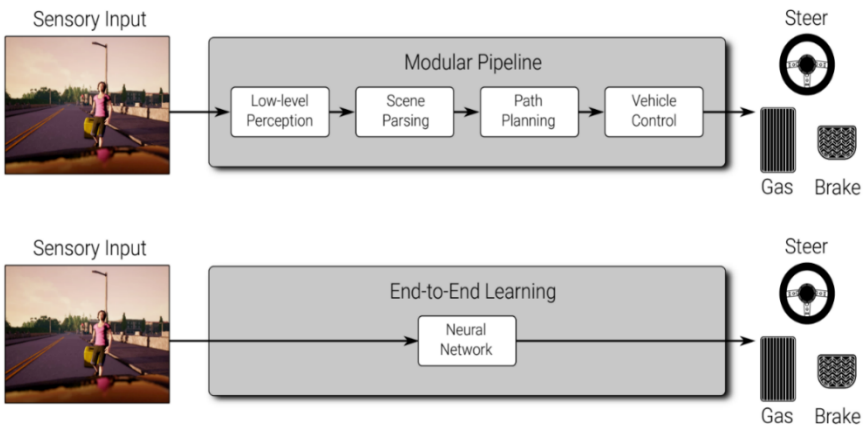
Лидар (енг. Light Detection And Ranging, LiDAR) сензори (слика бр. 6 десно) функционишу слично као радар, уз разлику да уместо радио таласа користе ласере. [9] Поред веома прецизне детекције објеката и одређивања растојања до њих (мерењем времена које је потребно рефлектованој светлости да падне на фотодетектор), лидар омогућава и креирање 3D мапе окружења – емитује велики број инфрацрвених зрака у окружење и након њиховог рефлектовања креира тачке у 3D простору. Недостацима ових сензора могу се сматрати висока цена (знатно су скупљи од других сензора) и то што могу бити непоуздани при неповољним временским условима. Неки произвођачи аутомобила (попут Tesle) покушавају да заобиђу коришћење лидара и уместо тога се ослањају на информације са камера (и радара).

Инерцијални сензори (енг. Inertial Measurement Unit, IMU), као што су акцелерометар и жироскоп, задужени су за кретање аутомобила, а када је реч о навигацији, значајану улогу има глобални позициони систем (енг. Global Positioning System, GPS) – који GPS пријемницима даје информације о геолокацији и времену. Аутономна возила користе GPS за одређивање својих координата у простору – често у комбинацији са дигиталним мапама (како би се повећала прецизност добијених података). [9]

2.4. Архитектура система

Софтверски систем аутономних возила може бити дизајниран као централизован или дистрибуиран. Код централизованих система целокупна обрада потребна за постизање аутономне вожње се обавља у возилу, које може, али не мора комуницирати са другим возилима и инфраструктурним елементима – ово је најзаступљенији приступ међу постојећим решењима, односно представља стандард у индустрији. Са друге стране, код дистрибуираних система задаци везани за аутономну вожњу могу бити подељени – уз коришћење VANET (енг. Vehicular Ad hoc NETwork) мреже, при чему се комуникација возила са било чим (од уређаја пешака до семафора) назива V2X („vehicle to everything“) и омогућава дељење информација о саобраћајној мрежи. Најзначајнији изазови код овог приступа су сама имплементација VANET мреже, као и филтрирање података, тј. издвајање корисних информација. Повезани аутономни системи возила се могу описати термином „интернет возила“ (енг. „Internet of Vehicles“, IoV). [11]

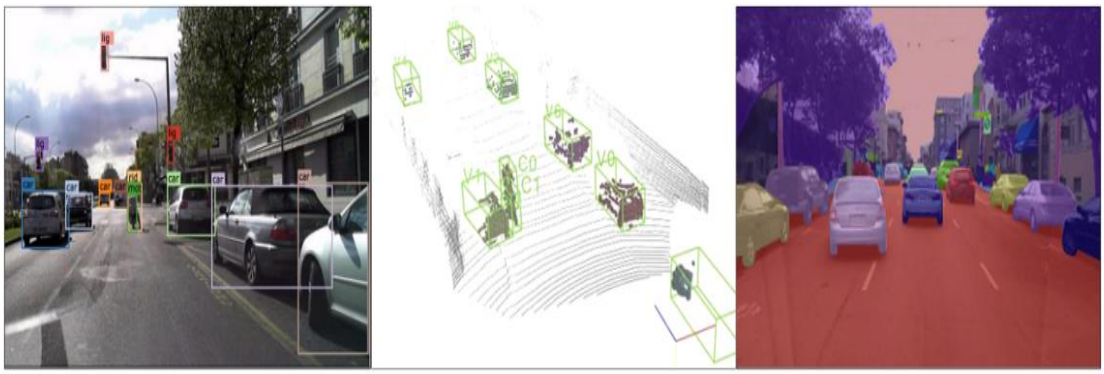
У реализацији аутономних система могу се издвојити два архитектурна приступа: модуларни и од-краја-до-краја. У оба случаја се са сензора добијају подаци о стању окружења, које систем користи да би управљао актуаторима (волан, педала кочнице и педала гаса). Разлика између њих је у реализацији обраде добијених података: модуларна архитектура подразумева декомпозицију комплексног проблема на више једноставнијих компоненти – које се могу засебно развијати, тренирати и тестирати, док систем од-краја-до-краја директно генерише управљачке команде на основу података са сензора. [12] Најважнији задаци који се обављају у оквиру модула у првој архитектури су: перцепција окружења, локализација и мапирање, доношење одлука и планирање путање, као и управљање возилом, при чему се користе класични алгоритми и алгоритми вештачке интелигенције (који постају све доминантнији). Друга архитектура се најчешће у потпуности ослања на дубоко учење, односно неуронске мреже: овакви системи могу реализовати аутономну вожњу до одређене мере, и још увек не могу заменити модуларне системе. На слици бр. 7 приказана је илустрација ова два архитектурна приступа.



Слика 7: Модуларни систем (изнад) и систем од-краја-до-краја (испод) [12]

2.4.1. Модуларни систем

У модуларном систему модул за перцепцију окружења задужен је за обраду података са више различитих сензора, односно може укључити рачунарски вид (енг. computer vision), који обрађује податке са камера, и фузију података са осталих сензора (енг. sensor fusion). [13] На слици бр. 8 у левом делу је приказана примена рачунарског вида за 2D детекцију и препознавање објеката, у средини је илустрована 3D детекција објеката на основу података добијених са лидар уређаја, а у десном делу је пример семантичке сегментације слике. Технике рачунарског вида омогућавају детекцију и праћење других возила, пешака и осталих објеката, док се методом комбиновања података са радара и лидара могу прецизније добити позиција и брзина детектованих објеката. Коришћење података са више сензора обезбеђује бољу перцепцију окружења него када би се користили појединачни сензори (удруживањем лидара и радара користи се предност лидара да прецизно одређује растојања и представља окружење у три димензије, а радара да одређује брзину објеката). Фузија се може вршити над било којим сензорима, а један од најпознатијих алгоритама који се примењује у ту сврху јесте Калманов филтер. [14]



Слика 8: 2D детекција објеката (лево), 3D детекција објеката на основу лидар података (средина) и семантичка сегментација (десно) [15]

Локализација је корак процесирања у оквиру кога се одређује тачна локација аутомобила на мапи. Пошто прецизност добијених GPS координата варира, оне се не користе директно, већ је потребно применити додатне технике обраде да би се побољшала њихова тачност. Неке од ових техника су: одометрија (користи се прорачун промене положаја возила током времена), Калманов филтер (може одредити стање возила), филтери честица (креирају се честице око области где су запажања са сензора слична мапи окружења) и SLAM (Simultaneous Localization And Mapping – истовремено се креира мапа окружења и одређује позиција возила на њој). [16]

Планирање путање је корак у оквиру кога се одређује путања до одредишта – то је линија која пролази кроз тачке пута (одређене у простору у времену) и усмерава возило када да се позиционира на коју локацију. [17] Пре тога потребно је предвидети шта ће сваки елемент окружења урадити, при чему се могу за сваку ситуацију моделовати сви могући поступци, или се може користити машинско учење, односно ослањање на већ виђене сценарије. Након предвиђања понашања окружења следи процес доношења одлука. За одлучивање у локалном окружењу возила се могу користити Френеове координате, које имају две осе: s-оса указује на напредовање у односу на стазу, а d-оса на удаљеност од центра стазе. За генерисање путање на глобалном нивоу (одабир улица) најчешће се користе алгоритми одмеравања (тражење по ширини, Дијкстра, A*, RRT) – који су веома ефикасни по питању времена извршавања, и алгоритми учења поткрепљивањем – који представљају све популарнији приступ.

Модул за управљање аутомобилом је задужен за кретање возила, како би оно пратило испланирану путању. Овај модул користи алгоритме који се називају регулатори (контролери) и служе за управљање актуаторима. Најчешће коришћени регулатори су PID (енг. Proportional Integral Derivative) и MPC (енг. Model Predictive Control). PID је најједноставнији алгоритам за контролу који има пропорционално, интегрално и диференцијално дејство – за рачунање вредности (нпр. угла скретања или убрзања) користи прорачун грешке, која представља несклад између путање којом би возило требало да се креће и путање којом се оно заправо креће. Овај алгоритам се брзо имплементира и примењив је само у једноставнијим ситуацијама. MPC је пример напреднијег алгоритма, који узима у обзир ограничења и ефикаснији је, али и захтевнији за имплементацију: заснива се на променама угла скретања и притиска на педалу гаса или кочнице. [18]

Главне предности модуларног приступа су то што коришћење више модула омогућава поделу проблема на једноставније потпроблеме за које већ постоје решења која се могу применити (из домена роботике, рачунарског вида, динамике возила), и то што се ови модули могу развијати паралелно. Са друге стране, недостаци модуларне архитектуре јесу склоност ка пропагацији грешака и велика сложеност система.

2.4.2. Систем од-краја-до-краја

Вожња од-краја-до-краја (енг. end-to-end driving) се назива и: директна перцепција, зато што се возилом управља директно на основу улаза са сензора. Један од првих система који је користио учење од-краја-до-краја за управљање возилом представио је Dean Pomerleau 1989. године у раду „ALVINN: Autonomous Land Vehicle In a Neural Network“ [19]. У овом раду је показано да потпуно повезана неуронска мрежа може управљати возилом у одређеним условима. ALVINN мрежа је била

трослојна и користила је слике са камере и ласерски даљиномер као улазе, а као излаз генерисала је угао скретања.

Код овакве вожње разликују се три главна приступа: директно надгледано дубоко учење, дубоко учење поткрепљивањем и неуроеволуција. Приликом примене надгледаног дубоког учења највише се користе дубоке конволуционе неуронске мреже, при чему се захтевају тренинг-подаци добијени директним посматрањем и мерењем, који се најчешће генеришу тако што се снима вожња обучених возача, након чега се мрежа тренира да опонаша начин њихове вожње (имитационо учење). Са друге стране, методе засноване на учењу поткрепљењем не имитирају понашање возача, већ уче како да постигну оптималну вожњу на основу искуства. Неуроеволуција се односи на примену еволутивних алгоритама за тренирање вештачких неуронских мрежа – ова метода није толико популарна као претходне две, и системи засновани на њој нису имплементирани у реалном свету, већ само у симулатору. Предност неуроеволуције у односу на остале методе је то што не захтева пропагацију уназад (енг. back propagation). [11]

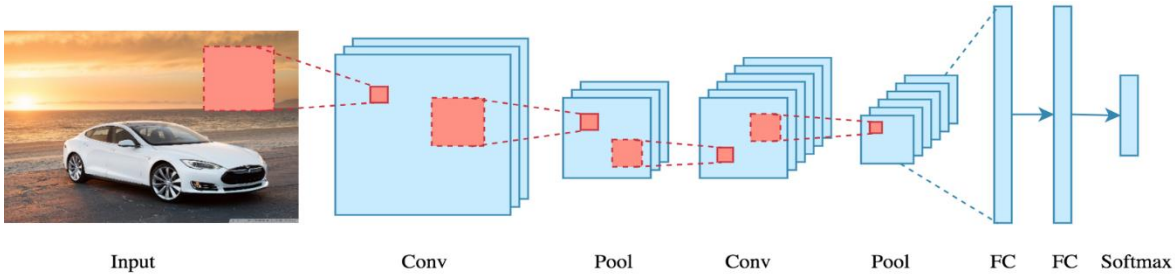
Најзначајнија предност „end-to-end“ система у односу на модуларне је то што оптимизују све кораке процесирања симултано, тако да би таква архитектура могла довести до бољих перформанси и мањих система. Међутим, ова архитектура још увек није примењена у реалном градском окружењу – реализоване су само демонстративне вожње. [12] Недостак оваквог начина тренирања система јесте слаба генерализација за догађаје за које је доступно мало тренинг-података, као и то што је често тешко открити зашто долази до грешке.

3. Технике дубоког учења за управљање аутономним возилима

Дубоко учење је подобласт вештачке интелигенције и машинског учења, у оквиру које се користе вишеслојне (дубоке) неуронске мреже – међу којима су најпознатије конволуционе. Ове мреже могу да науче значајне особине из података, а у аутономним возилима налазе примену због могућности ефикасног издвајања корисних информација из примљених сигнала са сензора.

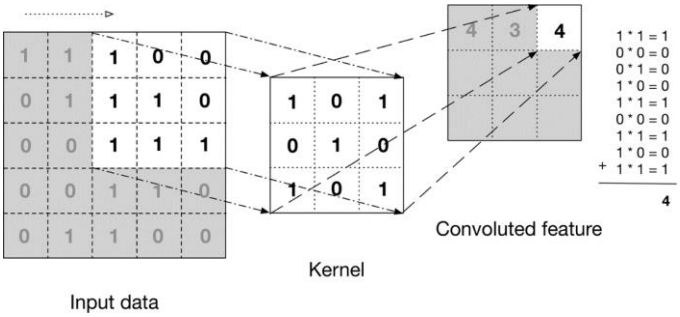
3.1. Конволуционе неуронске мреже

Конволуционе неуронске мреже (енг. Convolutional Neural Networks, CNN) користе конволуцију, математичку операцију којом се редукује слика како би била лакша за процесирање, при чему се задржавају све важне карактеристике садржане у њој. Ове мреже (као и остале неуронске мреже) сачињене су од слојева вештачких неурона – који обављају функцију тежинске суме улаза и генеришу излазну вредност. На слици бр. 9 је приказан пример архитектуре конволуционе мреже, која поред улазног слоја садржи више конволуционих слојева, слојеве сажимања и потпуно повезане слојеве, између којих се налазе слојеви активационе функције. Први конволуциони слој издваја једноставније, а наредни слојеви све сложеније карактеристике. Ове мреже се обучавају коришћењем алгоритма пропагације уназад, а параметри који се дефинишу за сваки слој су: димензија филтера, допуна нулама, померај филтера и дубина (број филтера).



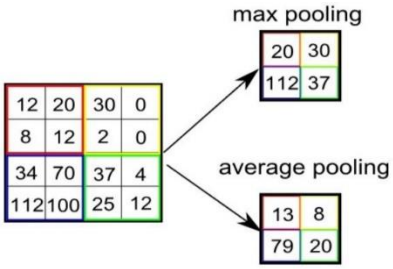
Слика 9: Илустрација архитектуре конволуционе мреже [20]

На слици бр. 10 је ради поједностављења приказана операција конволуције над сивом сликом уз примену филтера (кернела) представљеног матрицом димензија 3×3. Сива слика има једну раван и приказана је као матрица вредности пиксела, а пошто се RGB слика састоји од три равни, овај процес би се реализовао за све три матрице.



Слика 10: Пример примене конволуције [20]

Слој сажимања има улогу да смањи димензију издвојених мапа карактеристика, што за резултат има смањење комплексности израчунавања и спречавање преобучавања мреже (енг. overfitting) – када модел не генерализује добро над новим подацима. Користе се два типа овог слоја: сажимање максимумом и сажимање усредњавањем (слика бр. 11). Сажимање максимумом подразумева тражење максималне вредности у делу матрице који је прекривен кернелом, што даје боље резултате него усредњавање тих вредности.



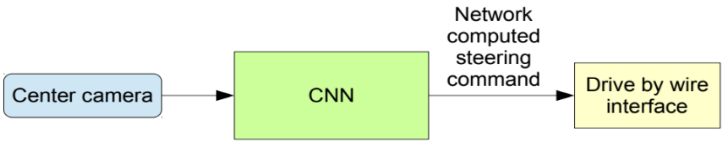
Слика 11: Илустрација операције сажимања [20]

Слој активационе функције се најчешће јавља након сваког слоја: активациона функција се користи за побољшавање перформанси мреже, а једна од најкоришћенијих јесте ReLU, која се представља једначином: $f(x) = max(0, x)$. На крају мреже се најчешће налази један или више потпуно повезаних слојева (код којих су неурони повезани са свим неуронима из претходног слоја). Када се ради о проблему класификације на излаз мреже се доводе вероватноће припадања слике различитим класама – функција која се најчешће користи у том процесу је „softmax“, а када је реч о регресији излаз може бити и једна вредност (нпр. угао скретања).

Конволуционе неуронске мреже се примењују за детекцију и препознавање објеката на сликама, означавање видео-снимака, анализу текста, препознавање говора, процесирање природних језика, а у роботима и аутономним возилима се могу користити за детекцију препрека, категоризацију сцене, детекцију саобраћајних трака, препознавање пешака итд. Неке од најпознатијих архитектура конволуционих мрежа су: LeNet, AlexNet, VGGNet, GoogLeNet и ResNet.

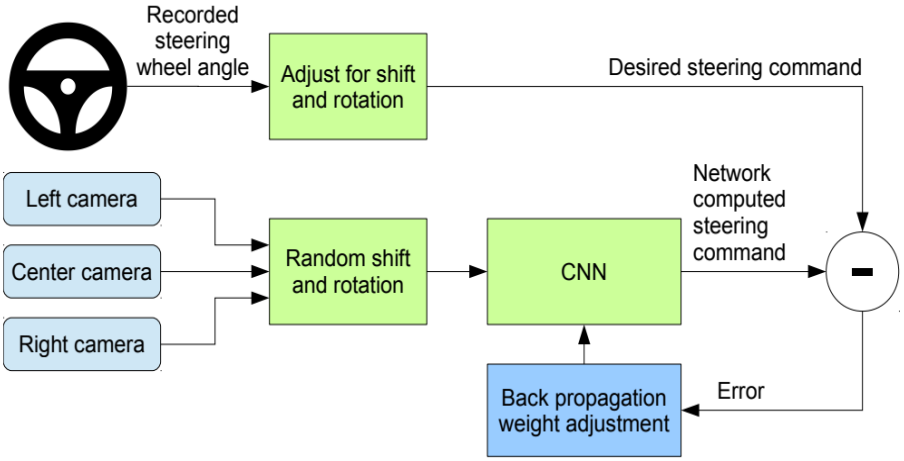
3.2. Имитационо учење

Имитационо учење код аутономних возила, односно клонирање понашања (енг. behavior cloning) јесте приступ према коме неуронска мрежа учи да мапира опажања са сензора у управљачке команде. Компанија NVIDIA је 2016. године у раду „End to End Learning for Self-Driving Cars“ [21] (инспирисаним DAVE пројектом [22]) представила дубоку конволуциону неуронску мрежу (названу „PilotNet“) за предикцију угла скретања аутомобила, која је омогућила вожњу на локалним путевима са или без означених линија, аутопутевима, паркинзима и неасфалтираним путевима. Ова мрежа је мапирала пикселе са централне предње камере директно у угао скретања (слика бр. 12) да би обезбедила одржавање возила у саобраћаној траци.



Слика 12: Генерисање угла скретања [21]

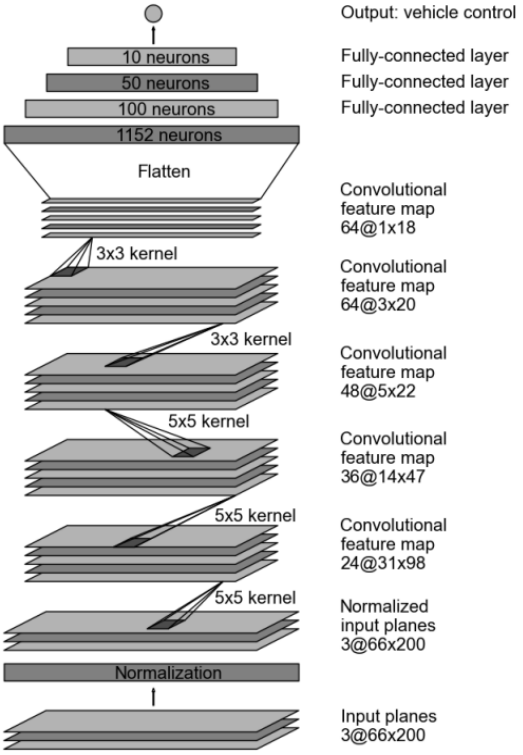
Тренинг-подаци (око 72 сата вожње) су прикупљени (при различитим светлосним и временским условима) са три камере монтиране на „drive-by-wire“ аутомобил: видео са тих камера је сниман заједно са углом скретања, добијеним са CAN (Controller Area Network) магистрале возила. При тренирању су коришћени само случајеви када се возило одржава у траци, а одбачени су остали (промена траке, скретање). Да би систем био независан од геометрије аутомобила, угао скретања је представљен као 1/r, где је r радијус скретања возила у метрима – ова вредност има благи прелаз кроз нулу од левих скретања (негативне вредности) ка десним (позитивне вредности). Тренинг-скуп садржи слике издвојене из видео-снимака, упарене са одговарајућим углом скретања. Да би мрежа научила да се опоравља од грешака, овај скуп је проширен додатним случајевима када је аутомобил удаљен од центра траке и заротиран у односу на пут. Два специфична померања у односу на централни положај могу се добити са леве и десне камере, а додатна померања и ротације су симулиране транформацијом погледа (енг. viewpoint transformation) слике са најближе камере, при чему је вредност за угао скретања подешена тако да омогућава враћање возила на централну локацију и оријентацију. На слици бр. 13 илустрован је процес тренирања: слике са камера се прослеђују мрежи која генерише угао скретања, при чему се тежине мреже прерачунавају коришћењем алгоритма пропагације уназад. Мрежа је тренирана тако да минимизује средњу квадратну грешку.



Слика 13: Тренирање неуронске мреже [21]

На слици бр. 14 је приказана PilotNet архитектура – улазна слика се пребацује у YUV простор боја и прослеђује мрежи, која се састоји из девет слојева: нормализациони, пет конволуционих и три потпуно повезана слоја. Нормализациони слој извршава нормализацију слике, конволуциони слојеви издвајају значајне карактеристике (одабрани су емпиријски), а потпуно повезани слојеви генеришу излазну вредност којом се контролише управљач. У прва три конволуциона слоја је коришћен кернел димензија 5×5 и померај 2×2, а у наредна два 3×3 кернел, и померај 1×1. Добијена мрежа има око 250 000 параметара. На слици бр. 15 је зеленом бојом обележено које карактеристике слика ова мрежа користи при учењу.

Перформансе мреже су прво евалуиране у симулатору, а затим на реалним путевима. Проценат времена током кога је мрежа управљала возилом у симулатору је рачунат тако што је бројано колико пута је човек морао да преузме контролу над возилом (случајеви када је возило било удаљено од средине пута за више од једног метра), при чему је сматрано да је за једну интервенцију потребно шест секунди. Приликом тестирања на реалним путевима вожња је била аутономна 98% времена.

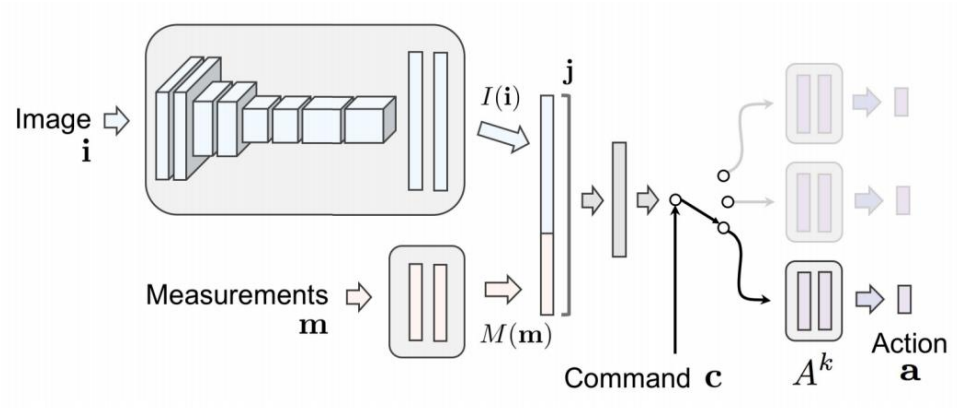


Слика 14: Архитектура PilotNet мреже [23]

Слика 15: Значајне карактеристике које издваја PilotNet [23]

Генерисање само угла скретања није довољно да би се управљало аутомобилом на раскрсницама и омогућило возилу да прати одређену путању. Из тог разлога наредне, побољшане верзије PilotNet мреже (засноване на модификованој ResNet архитектури), као улаз имају пикселе, а као излаз генеришу границе трака и жељену путању у 3D координатном систему. Ову путању затим користи засебан контролни систем за управљање возилом. Додатна предност оваквог излаза мреже је то што се може комбиновати са информацијама из других извора, као што су мапе и системи за детекцију препрека. [24]

У раду „End-to-end Driving via Conditional Imitation Learning“ [25] Felipe Codevilla и сарадници су предложили условно имитационо учење као решење за елиминацију потребе за интервенцијом возача на раскрсницама и при промени саобраћајне траке. Овакав приступ подразумева да се возилом управља на основу слика са камера и навигационих команди високог нивоа, које представљају намеру возача. Ове команде може задати путник током вожње или се могу добити из модула за планирање путање – при прикупљању тренинг-података, возач је при приближавању раскрсницама користио дугме и тиме обележавао команду коју предузима. На слици бр. 16 је илустровано како функционише овај систем: на улазу се налазе слика *i* и вектор *m*, који представља мере (нпр. тренутна брзина аутомобила), а излаз из мреже је акција *a*. Акцију представљају угао скретања и убрзање, које може бити позитивно (што одговара убрзавању), и негативно (што одговара кочењу или вожњи уназад). Модул *I(i)* је имплементиран као конволуциона мрежа и задужен је за обраду слика, а модул *M(m)* као потпуно повезана мрежа која обрађује мере. Резултати ових модула се спајају и прослеђују „прекидачу“ – који представља дискретан скуп команди, при чему команда c_i селектује грану A_i. Гране уче стратегије које одговарају различитим командама (праћење траке, скретање лево, скретање десно, вожња право).



Слика 16: Систем за условно имитационо учење [25]

Dian Chen и сарадници су у раду „Learning by cheating“ [26] показали да се имитационо учење може декомпоновати на две фазе: прва је тренирање агента који „вара“ имајући приступ привилегованим информацијама, а у другој фази тај агент постаје „учитељ“ резултујућег агента, који има приступ само визуелним подацима (са предње камере) и „не вара“. Привилеговане информације подразумевају тачан изглед окружења и позиције свих учесника у саобраћају из „птичије перспективе“ – представљају се мапом која садржи бинарне индикаторе за објекте, карактеристике пута и семафоре. Оба агента су представљена конволуционим мрежама заснованим на ResNet архитектури, које на основу тренинг путања генеришу тачке пута по којима возило треба да се креће. Привилеговани агент, који учи да опонаша возача на основу тренинг путања и привилегованих података, може се посматрати као „бела кутија“, која агенту који се ослања на сензоре обезбеђује боље надгледање него оригиналне тренинг путање. При обучавању се користе и додатни улази попут брзине возила и команди високог нивоа (као код условног имитационог учења), а за конвертовање добијених тачака у управљачке команде (угао скретања, убрзање, кочење) користи се PID контролер. Примена оваквог система је демонстрирана у CARLA симулатору, али уз употребу метода за трансфер из симулатора у реални свет његово коришћење не мора бити ограничено на симулацију.

3.3. Учење поткрепљивањем

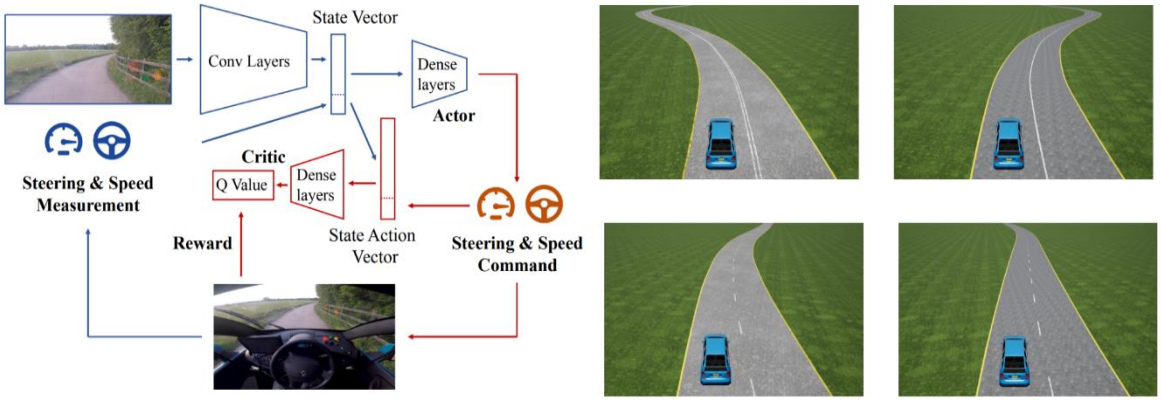
Перцепција окружења реализована коришћењем техника надгледаног учења захтева тренирање модела свим могућим сценаријима који се могу десити у реалном свету – што је веома захтеван процес. Други приступ који налази примену у аутономним возилима јесте учење поткрепљивањем (енг. Reinforcement Learning, RL) – које представља грану машинског учења засновану на Марковљевом процесу одлучивања (енг. Markov Decision Process, MDP). [27] MDP се дефинише кроз скуп стања, скуп акција, функцију прелаза између стања, функцију награђивања и фактор умањења. Циљ је наћи стратегију која ће максимизовати функцију награђивања од почетног до крајњег стања, при чему се агент обучава кроз интеракцију са окружењем и учи на основу искуства. При дефинисању функције награђивања у аутономним возилима могу се посматрати различити аспекти, као што су нпр. потрошња енергије, безбедност учесника саобраћаја и најбржа путања до одредишта.

Један од најчешће коришћених алгоритама за решавање MDP проблема је Q-учење, где се акције a∈A одређују за свако стање s∈S на основу функције $Q: S \times A \rightarrow R$. У зависности од домена проблема, скуп могућих акција може бити дискретан или континуалан. Када је скуп акција дискретан Q-функција се може апроксимирати коришћењем дубоке конволуционе неуронске мреже, што се примењује у алгоритму названом DQN (Deep Q-Networks). Када је скуп акција континуалан, користе се DDAC (Deep Deterministic Actor Critic) алгоритми, код којих се уводе две функције: *actor* (која мапира стања у акције) и *critic* (која евалуира вредност предузете акције у датом стању) – ове функције такође могу бити представљене коришћењем неуронских мрежа. Од значаја су и рекурентне неуронске мреже (енг. Recurrent Neural Networks, RNN), које су погодне за обраду временских серија података. Оне налазе примену код POMDP (Partially Observable MDP) проблема, који се јавља када се реално стање окружења не може добити на основу једног посматрања (јер се константно мења), већ се треба посматрати током дужег временског периода. LSTM (Long Short-Term Memory) је посебан тип рекурентних неуронских мрежа, који праћењем улаза, излаза и стања меморије одређује временске зависности у секвенцама података. Ове мреже се могу користити да контролишу које информације треба задржати из претходног стања, а које одбацити. Када се LSTM интегришу у DQN формирају се дубоке рекурентне Q мреже (енг. Deep Recurrent Q Networks, DRQN). [27]

У раду „Deep Reinforcement Learning framework for Autonomous Driving“ [27] Ahmad El Sallab и сарадници су предложили примену дубоког учења поткрепљивањем за реализацију система аутономне вожње. Њихов систем укључује рекурентне неуронске мреже за интеграцију података са сензора, као и моделе пажње (енг. attention models) који су задужени за издвајање само релевантних информација. Учење поткрепљивањем се заснива на моделу који треба да генерише најбољу стратегију коју агент може пратити (најбољу акцију коју ће преузети у тренутном стању), тако да укупна акумулирана награда буде максимизована. Због избегавања ризика при тренирању RL система на реалним путевима, овај систем је имплементиран у TORCS 3D симулатору. Добијени резултати су демонстрирали аутономно управљање у комплексним кривинама на путу, при једноставним интеракцијама са другим возилима. Предложени систем користио је као улазе стања окружења и њихову агрегацију током времена, а као излаз генерисао акције вожње (угао скретања, брзину, вредности за убрзање и кочење), при чему су очитани подаци са сензора груписани и прослеђени конволуционој неуронској мрежи како би закључила шта је међу њима значајно и обавила фузију тих података.

Alex Kendall и сарадници су у раду „Learning to Drive in a Day“ [28] приказали прве обећавајуће резултате у примени дубоког учења поткрепљивањем за аутономну вожњу реалних аутомобила. Њихов систем је успешно омогућио возилу да прати саобраћајну траку користећи слике са предње камере (и информације о брзини возила и углу скретања). Циљ је био креирати роботски систем који ће приступити вожњи као и човек: у одсуству мапа и експлицитних правила, ослањајући се на разумевање непосредног окружења и праћење једноставних комади високог нивоа (команде усмеравања). Коришћен је DDPG (Deep Deterministic Policy Gradients) алгоритам за тренирање агента (који се примењује за континуалне акције), а награда је дефинисана као растојање пређено без потребе да безбедносни возач преузме контролу над возилом. Оваква функција награђивања је општа, али има бројна ограничења: не разматра условљавање у зависности од навигационог циља, а како агент напредује, интервенције од стране возача ће бити мање фреквентне – што ће као резултат имати слабији тренинг сигнал.

RL алгоритми се могу поделити на две категорије: засновани на моделу и алгоритми без модела. У првом случају се уче експлицитни модели за функцију промене стања и функцију награђивања, који се онда користе за проналажење стратегије која максимузује кумулативну награду, а у другом се директно одређује вредност *Q(s, a)* за предузимање акције *a* у стању *s*, а затим се прати стратегија која селектује акцију са највећом вредношћу у сваком стању. Приступ без модела је општији и може се применити за било који задатак, а приступ заснован на моделу је онолико добар колико је добар модел. У овом систему је коришћен алгоритам без модела, а за реализацију одржавања возила у траци коришћена је мала конволуциона мрежа (и у симулатору и у реалном свету) са четири конволуциона слоја, које деле *Actor* и *Critic* модели (слика бр. 17 лево). Затим је стање представљено као вектор, на који се надовезују акције. У обе мреже се након тога примењују потпуно повезани слојеви, пре регресије излаза. За потребе тестирања развијен је 3D симулатор (слика бр. 17 десно), а затим је тестирање обављено и на стварним путевима. [28]



Слика 17: Илустрација алгоритма за обучавање агента (лево) и вожња у симулатору (десно) [28]

Највећи недостатак учења поткрепљивањем без модела је то што често не постоји довољно података и захтева се велики број интеракција са окружењем. Са друге стране, методе засноване на моделу уче модел динамике окружења на основу прикупљених података и користе га за тренирање стратегије вожње. Ове методе захтевају знатно мање интеракција са окружењем, али је потребно да окружење обухвата велики број различитих ситуација, што није практично у реалном свету (с обзиром на цену и безбедност). [12]

3.4. Симулатори аутономне вожње

Тестирање возила са одређеним нивоом аутономије може бити обављено у затвореном окружењу, на јавним путевима (уз одговарајућу дозволу и присуство особе која ће преузети управљање по потреби), или у симулираном окружењу. [29] Како је тестирање на јавним путевима небезбедно, скупо, и многе ситуације се не понављају често, рачунарска симулација је веома важан део тренирања ових возила.

Симулатор аутономне вожње треба да што боље представи ситуације које се могу десити у реалном свету и да омогући симулацију свих сегмената вожње: од перцепције до реализације управљачких команди. Основне функционалности које добар симулатор треба имати су: реалистични модели сензора или подршка за пријем података са стварних сензора, локализација возила у окружењу, могућност креирања мапа или интерфејс за учитавање спољашњих мапа, као и интерфејс за имплементацију потребних алгоритама. Један од најзначајнијих сегмената тестирања у симулатору је постојање реалистичног виртуелног окружења (слика бр. 18), које садржи саобраћајну сигнализацију, статичке (зграде, дрвеће) и динамичке објекте (возила, пешаци, животиње, бициклисти) – чије понашање треба одговарати њиховом природном понашању. За постизање реалистичног окружења могу се користити енџини за игре (енг. game engines) или HD мапе реалног окружења, које ће се приказивати у симулацији. Такође, виртуелно окружење мора подржати различите терене и временске услове (симулатори који не нуде висок ниво детаља се не користе за тестирање перцепције), као и генерисање тренинг-података за моделе машинског учења. [29]

Слика 18: Пример вожње у реалном свету (лево) и SurfelGAN симулатору (десно) [30]

При упоређивању симулатора треба узети у обзир реалистичност 3D окружења, прецизност динамике модела аутомобила, флексибилност, портабилност, скалабилност, али и време потребно корисницима за упознавање са производом и начином креирања симулација. Постоји велики број комерцијалних симулатора аутономне вожње, као што су нпр. CarCraft и SurfelGAN (које користи Waymo), Webviz и The Matrix (које користи Cruise) и DataViz (који користи Uber). [29] Поред њих, доступни су и симулатори отвореног кода, а неки од најпознатијих су CARLA, LGSVL и AirSim. У наставку рада пажња је усмерена на симулаторе отвореног кода.

3.4.1. CARLA

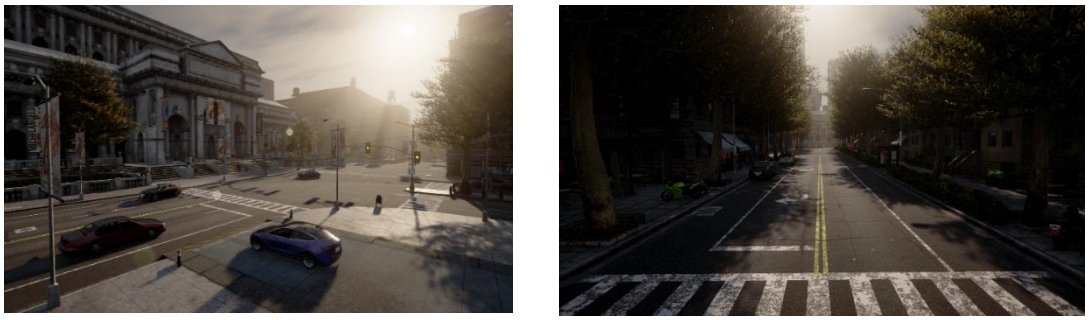
CARLA (енг. Car Learning to Act) је симулатор отвореног кода, који је направљен као подршка развоју, тренирању и валидацији система аутономне вожње у урбаним условима. [31] Најважније особине овог симулатора су: скалабилност (вишеклијентска архитектура), флексибилан API (корисници могу контролисати генерисање саобраћаја, понашање пешака, временске услове, сензоре), флексибилан скуп сензора (лидар, радар, различите камере, сензори за одређивање дубине, GPS), брзо извршавање планирања и контроле, генерисање мапа (у складу са OpenDrive стандардом), симулација различитих сценарија и интеграција роботског оперативног система (енг. Robot Operating System, ROS) – који представља колекцију алата, библиотека и конвенција за писање роботског софтвера. Поред тога, CARLA омогућава бесплатно коришћење постојећих дигиталних садржаја (возила, зграде, урбани планови). Овај симулатор се може користити за проучавање перформанси сва три приступа аутономној вожњи (модуларни, од-краја-до-краја применом имитационог учења и од-краја-до-краја применом учења поткрепљивањем). На слици бр. 19 је приказан изглед једног града у овом симулатору при различитим временским условима (ведро, дневна киша, убрзо након кише, залазак сунца). [32]

Слика 19: Изглед града при различитим временским условима у CARLA симулатору [32]

CARLA симулатор је направљен тако да нуди флексибилност и реалистичност у рендеровању и симулацији физике – имплементиран је као слој изнад Unreal Engine 4 (UE4), а дизајниран као клијент-сервер систем, где сервер извршава симулацију и рендерује сцену, а клијентски API је задужен за интеракцију између аутономног агента и сервера (коришћењем сокета). Клијент шаље команде и мета-команде серверу и прима податке за сензора као повратну информацију: команде контролишу возило и обухватају: управљање, убрзавање и кочење, а мета-команде контролишу понашање сервера и задужене су за ресетовање симулације, промену својстава окружења (временски услови, осветљење, густина саобраћаја) и одабир сензора. [32]

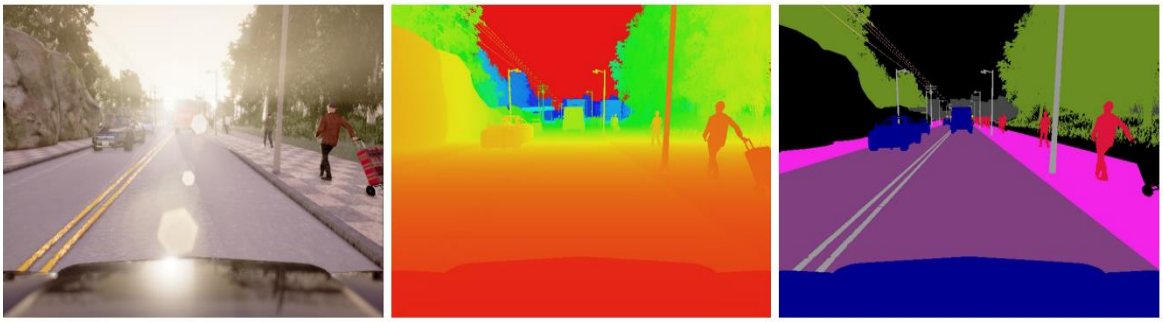
Окружење се састоји од 3D модела статичких објеката (зграде, вегетација, саобраћајни знакови, инфрастуктура), као и динамичких објеката (возила, пешаци). Возила, пешаци и саобраћајна сигнализација интерагују у оквиру симулације и називају се актерима. Сви модели су дизајнирани тако да задовоље квалитет визуелизације и брзину рендеровања (користе се „лаки“ геометријски модели и текстуре, али задржава реалистичност и ниво детаља). Сви 3D модели имају

заједничко скалирање и њихова величина одговара величини реалних објеката, а возила и пешаци (који нису у фокусу) могу детектовати и избегавати једни друге. Пешаци се крећу улицама пратећи навигационе мапе, тако да се фаворизује кретање по тротоару и означеним пешачким прелазима, али им се дозвољава и да пређу улицу у било којој тачки. Ако ауто удари пешака, он се брише и креира се нови након одређеног времена на некој другој локацији. Пешаци су насумично обучени и опционо опремљени неким додатним предметом (телефоном, торбом, кофером, кишобраном), а возила насумично обојена. Имплементирани су различити атмосферски услови (који се разликују по позицији и боји сунца, и интензитету и боји дифузионог зрачења неба), као и магла, облачност и падавине. У новијим верзијама симулатора уведена су побољшања у свим сегментима, а на слици бр. 20 је приказан напредак у домену реалистичности окружења. [32]



Слика 20: Реалистичност окружења у новијим верзијама [32]

CARLA омогућава флексибилну конфигурацију сензора и укључивање додатних модела сензора кроз API. На слици бр. 21 приказани су примери подржаних сензора, као што су RGB камере и псеудо-сензори за дубину и семантичку сегментацију. Конфигурационе параметре, као што су број камера, њихова позиција, оријентација, тип и видно поље може специфицирати корисник. [32]

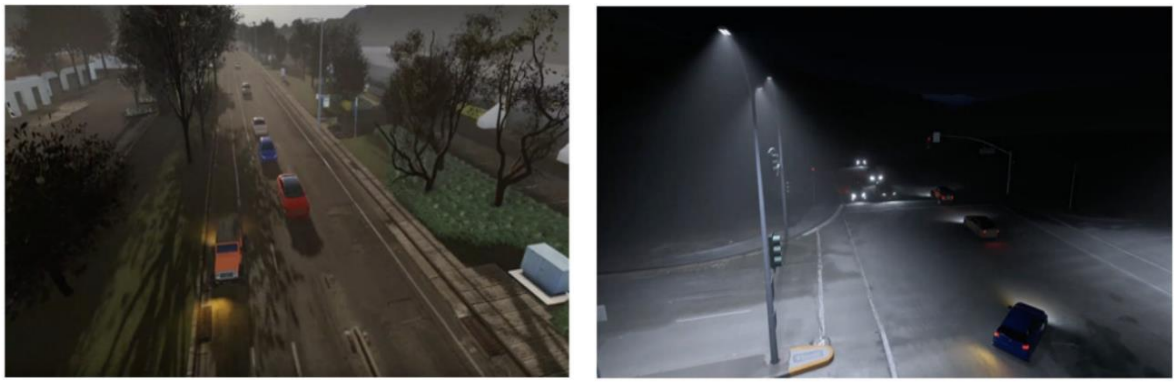


Слика 21: RGB камера (лево), псеудо-сензор за дубину (средина) и семантичку сегментацију (десно) [32]

Такође, CARLA нуди низ мера везаних за стање агента и окружења и поштовање саобраћајних правила. Мере везане за стање агента укључују локацију возила и оријентацију у односу на светски координатни систем, брзину, вектор убрзања и акумулирани утицај судара. Мере везане за правила саобраћаја укључују проценат отиска возила који прелази у погрешну траку или на тротоар, стање семафора, као и ограничење брзине на тренутној локацији возила. Поред тога, омогућен је приступ тачним локацијама и граничним правоугаоницима свих динамичких објеката у окружењу – ове информације имају кључну улогу у тренирању и евалуирању стратегија вожње. [32]

3.4.2. LGSVL

SVL симулатор [33], који је креирала компанија LG Electronics, објављен је 2018. године као пројекат отвореног кода, са циљем да омогући развој аутономних возила и сличних робота кроз високоперформансну симулацију. Овај симулатор нуди симулацију у безбедном, детерминистичком и реалистичном 3D окружењу. LGSVL је заснован на Unity енџину, интегрисан је са Autoware и Apollo платформама (које су најпознатији стекови за аутономну вожњу), подржава роботски оперативни систем (ROS, ROS2) и може генерисати HD мапе. Функције симулационог енџина се могу поделити на симулацију окружења, симулацију сензора и симулацију динамике управљања возилом, а на слици бр. 22 приказани су примери сцена креираних у овом симулатору. Параметри окружења (мапа, временски услови, доба дана, саобраћај) се могу мењати и симулација се може контролисати кроз Python API. Симулатор омогућава не само креирање 3D окружења, већ и репликацију и симулацију стварних локација креирањем њиховог „дигиталног близанца“ на основу логованих података (слике, облак тачака добијен са лидара). Поред тога, LGSVL се може користити за тестирање комуникације V2X система.

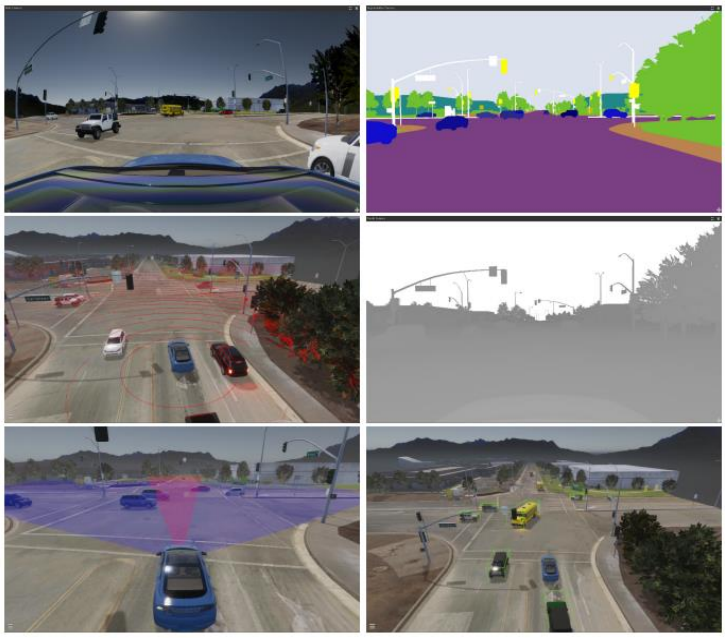


Слика 22: Примери сцена у LGSVL симулатору [33]

LGSVL симулатор омогућава генерисање синтетичких података за тренирање алгоритама машинског учења, с обзиром на то да су дубоке неуронске мреже (које захтевају много означених података) заступљене у већини модула аутономних возила. Код учења поткрепљивањем агенти поступају у окружењу на основу неке стратегије (често имплементиране кроз дубоку неуронску мрежу) и примају повратне информације од окружења – које се користе за преиспитивање (ревизију) те стратегије. Овај процес је потребно поновити велики број пута пре него што се добије оптимално решење, а LGSVL симулатор омогућава његову реализацију.

Подаци са сензора (камера, лидар, радар, GPS, IMU), могу се обрађивати симултано у реалном времену, при чему корисници подешавају њихове унутрашње и спољашње параметре конфигурацијама у JSON формату, а омогућено им је и да креирају сопствене моделе сензора. На слици бр. 23 су приказани неки од доступних сензора: лева колона приказује неколико физичких (*fish-eye* камера, лидар, радар), а десна неколико виртуелних (псеудо) сензора (сензор за сегментацију, дубину и 3D детекцију објеката). У оквиру сензора за сегментацију се комбинују семантичка сегментација и сегментација инстанци, при чему корисници бирају над којим семантичким елементима ће бити извршена сегментација инстанци: ако корисник

означи да само аутомобили и пешаци имају сегментацију инстанци, све зграде ће бити означене једном бојом, сви путеви неком другом бојом, а свако возило и пешак ће имати индивидуалну сегментациону боју различиту од осталих, при чему ће све боје за аутомобиле бити нпр. „плавкасте“, а за пешаке „црвенкасте“. [33]



Слика 23: Различити типови физичких (лево) и виртуелних (десно) сензора у LGSVL симулатору [33]

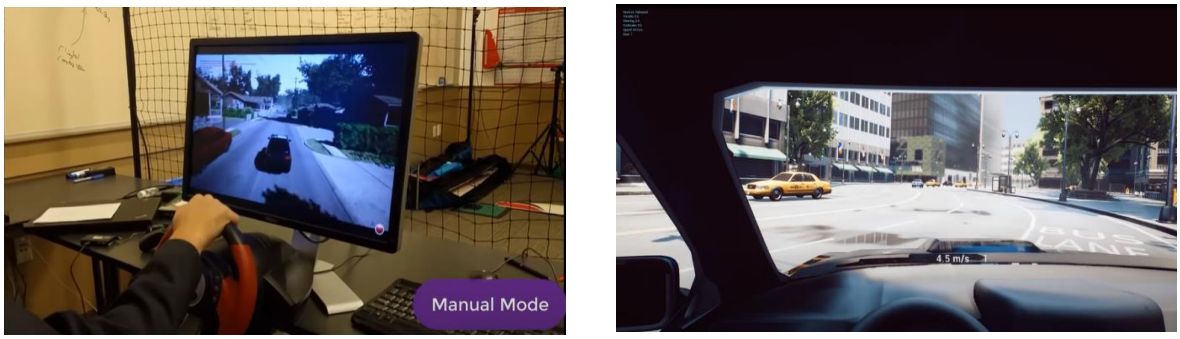
3.4.3. AirSim

AirSim симулатор [34] креирала је 2017. године компанија Microsoft, као платформу за истраживање у домену вештачке интелигенције. Реч је о симулатору који нуди могућност за безбедан развој и тестирање аутономије дронова, аутомобила и сличних возила у симулацији. AirSim је симулатор отвореног кода, заснован на Unreal Engine-у, а постоји и верзија заснована на Unity-ју. Ова платформа омогућава реализацију експеримената везаних за дубоко учење, рачунарски вид и учење поткрепљивањем. У наредним годинама развој овог симулатора је настављен кроз пројекат назван „Project AirSim“.

AirSim обезбеђује мануелно управљање возилом преко даљинског управљача, дугмића на тастатури или волана. На слици бр. 24 је у левом делу приказан мануелни режим у коме корисник коришћењем волана управља возилом, а у десном поглед из аутомобила у симулацији (где се могу подешавати временски услови). Доступан је и API за програмску интеракцију са возилом у симулатору (како би се преузеле слике и стање возила и контролисало возило) из различитих програмских језика (C++, Python, C#, Java). Омогућено је писање и тестирање кода у симулатору, и касније извршавање тог кода на стварним возилима. Када је реч о сензорима, AirSim подржава камере, барометре, IMU, GPS, магнетометар, лидар и сензор за растојања.

За генерисање тренинг-података за дубоко учење, AirSim нуди две могућности: притисак на дугме за снимање (који ће започети чување слика за сваки фрејм, као и бележење података о положају), и преко API-ја (уз могућност контроле како и где ће

се чувати ти подаци). Још једна функционалност коју нуди овај симулатор је коришћење режима названог „рачунарски вид“: у овом режиму нема возила и физике, већ се корисник употребом тастатуре (или API-ја) може кретати кроз сцену и прикупљати податке везане за дубину, нормале површина или сегментацију објеката.

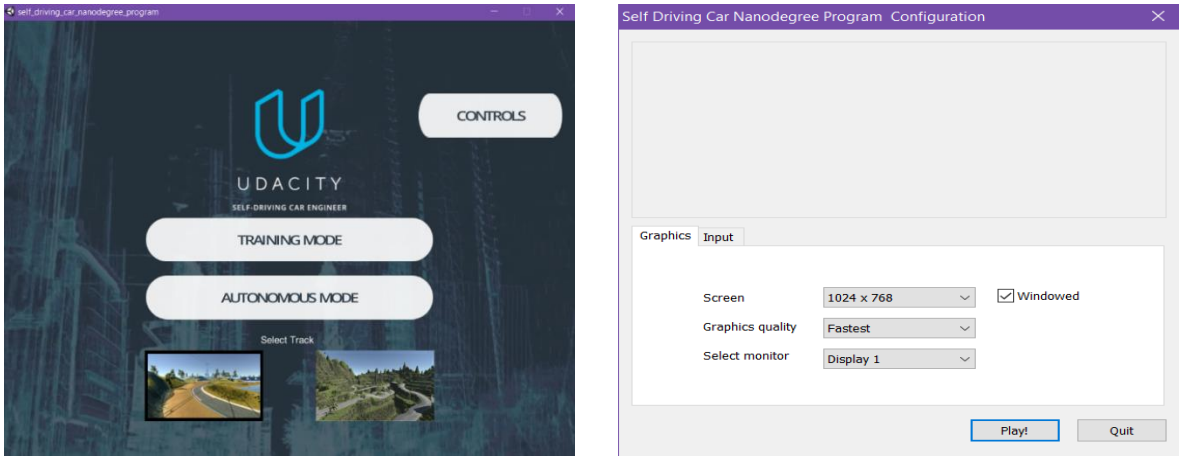


Слика 24: Мануелни режим (лево) и симулација (десно) у AirSim симулатору [34]

3.4.4. Udacity Self-Driving Car Simulator

Udacity је организација која се активно бави истраживањем у домену аутономних возила, а Udacity Self-Driving Car Simulator [35] је пројекат отвореног кода објављен у сврхе учења (за потребе курса „Self-Driving Car Nanodegree“). Овај симулатор је знатно једноставнији од претходно описаних, али може се користити за тестирање алгоритама за управљање возилом и за планирање путање (објављене су одвојене верзије за ове две намене). Симулатор је имплементиран у развојном окружењу Unity и може се модификовати кроз ово окружење, а могуће је и преузети извршне верзије које се могу одмах користити.

Верзија симулатора за управљање возилом омогућава тренирање аутомобила да се самостално креће на путу коришћењем дубоког учења. При покретању извршне апликације може се одабрати једна од две доступне стазе и један од два режима рада: тренинг и аутономни (слика бр. 25 лево). Корисник пре коришћења апликације може одабрати један од понуђених улазних уређаја за управљање аутомобилом (тастатура, миш или џојстик), као и резолуцију прозора и квалитет графике (слика бр. 25 десно).



Слика 25: Почетни прозор (лево) и конфигурациони прозор (десно) симулатора

Стаза по којој се креће аутомобил подсећа на тркачку стазу, и на њој нема стандардног саобраћаја (слика бр. 26). У тренинг режиму корисник може ручно управљати возилом, при чему му се приказују вредности за брзину и угао окретања управљача. Притиском на дугме „RECORD“ покреће се снимање, односно аутоматско чување слика са камера прикачених на аутомобил, као и података о брзини, углу скретања и вредностима које описују притисак на гас и кочницу у тим временским тренуцима. Ови подаци се могу користити за тренирање модела дубоког учења, који затим у аутономном режиму служи за управљање возилом како би се одржало у коловозној траци, тј. како би извозило комплетну стазу. Управљање се може реализовати из Python апликације, коју је могуће повезати са симулатором када је он у аутономном режиму.

Слика 26: Тренинг режим у Udacity симулатору

Друга верзија Udacity симулатора се може користити за планирање путање на виртуелном аутопуту дужине око 7 km, на коме су присутна и друга возила. Кориснику је и у овој верзији омогућена вожња у мануелном и аутономном режиму, при чему окружење изгледа као на слици бр. 27. Циљ је да возило узме у обзир кретање осталих возила и да их успешно заобилази, односно претиче уколико се крећу спорије у односу на њега – при чему и та возила могу прелазити из једне у другу траку током вожње. Аутомобил треба да избегава сударе са околним возилима, као и кретање ван своје саобраћајне траке (осим у случају престројавања). Симулатор ће генерисати упозорења у случају да се нека од ових ситуација догоди, као и уколико дође до прекорачења ограничења брзине.

Слика 27: Верзија Udacity симулатора за планирање путање

За потребе реализације планирања путање треба детаљније размотрити кретање возила, односно разложити његову брзину и убрзање на компоненте, и обрадити информације о његовој позицији и оријентацији. Симулатор и апликација (која се може назвати „планер“) комуницијају путем веб-сокета, при чему симулатор шаље податке попут позиције и брзине аутомобила, као и обједињене податке са свих сензора које се односе на друге аутомобиле. Са друге стране, симулатор очекује да му апликација достави (у задатом временском интервалу) листу тачака која представља трајекторију аутомобила којим се управља.

4. Имплементација прототипа система за предикцију угла скретања возила

У имплементационом делу овог рада је, коришћењем програмског језика Python, реализован прототип система за управљање возилом генерисањем угла скретања. Пројекат је подељен у два дела: реализација над скупом који садржи податке снимљене приликом вожње на стварним путевима, и реализација у Udacity Self-Driving Car симулатору – у оба случаја коришћена је PilotNet конволуциона неуронска мрежа, а проблем је посматран као надгледана регресија између слика са предњих камера и углова скретања. За извођење операција дубоког учења употребљена је Keras библиотека са Tensorflow *backend*-ом.

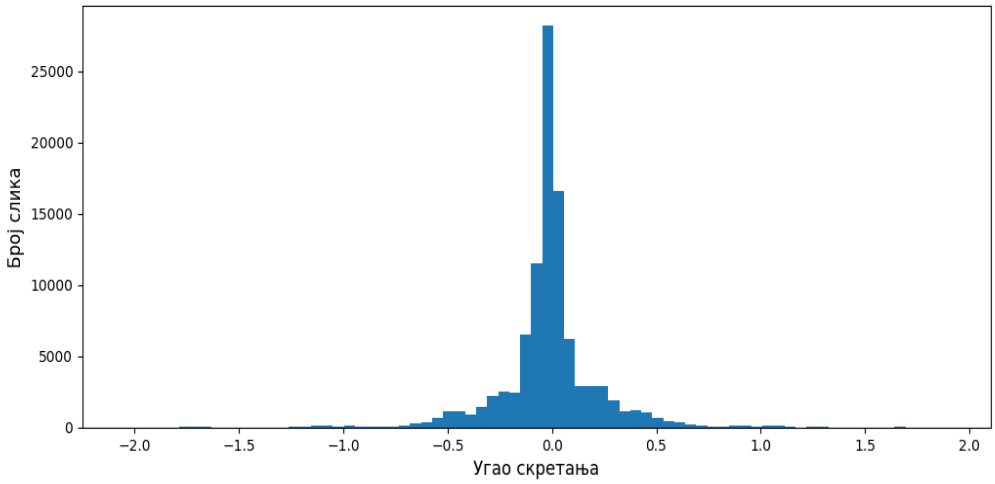
За тестирање модела над сликама стварних путева искоришћен је (јавно доступан) Udacity скуп података [36], који садржи слике са три камере и податке са осталих уређаја и сензора у возилу за око 30 минута вожње (при различитим светлосним условима, на путевима са једном или више саобраћајних трака). Ови подаци су подељени у више фајлова са „bag“ екстензијом, генерисаних у оквиру роботског оперативног система – ROS нуди могућност снимања свих размењених порука између компоненти система у „rosbag“ фајлове, из којих се може рекреирати сцена (при чему неколико секунди снимања може генерисати фајл величине више гигабајта). Да би се ови фајлови распаковали, прво је било потребно инсталирати ROS на оперативном систему Ubuntu, након чега је искоришћен „udacity-driving-reader“ [37] за издвајање потребних података. Добијено је укупно 101 396 слика, распоређених у фолдере са називима „center“, „left“, и „right“, заједно са пратећим подацима смештеним у више CSV фајлова – од којих је најзначајнији „interpolated.csv“. Овај фајл садржи 12 колона, при чему су коришћене само колоне „filename“, „angle“ и „speed“, које повезују слику са углом скретања и брзином возила. На слици бр. 28 је приказан део кода, који (коришћењем Pandas библиотеке) учитава све податке у DataFrame објекат, а на слици бр. 29 се може видети како изгледа функција за учитавање података везаних за слику са одређеним индексом у „interpolated.csv“ фајлу. Хистограм расподеле углова скретања у тренинг-скупу података (слика бр. 30) приказује да су вредности распоређене око нуле, тј. да нема много података који се односе на већа скретања.

```
def load_csv_data(csv_path):
    data = pd.read_csv(csv_path, header=None, names=["index", "timestamp", "width", "height",
                                                     "frame_id", "filename", "angle", "torque",
                                                     "speed", "lat", "long", "alt"])
    return data
```

Слика 28: Функција за учитавање података из „interpolated.csv“ фајла

```
def load_image(data, images_path, index):
    file_name = data.iloc[index]["filename"]
    image = plt.imread(os.path.join(images_path, file_name))
    angle = data.iloc[index]['angle']
    speed = data.iloc[index]['speed']

    return file_name, image, angle, speed
```

Слика 29: Функција за учитавање података везаних за једну слику



Слика 30: Расподела углова скретања у тренинг-скупу података

Мрежа која је коришћена се ослања на PilotNet архитектуру [21], односно садржи нормализациони, конволуционе и потпуно повезане слојеве. У циљу добијања што бољег модела, трениране су три варијанте ове мреже: 1) основном моделу додата ELU активациона функција након сваког слоја; 2) моделу са активационом функцијом додат један слој одбацивања (енг. dropout layer) 50% неурона након конволуционих слојева; и 3) моделу са активационом функцијом додат слој одбацивања 20% неурона након сваког конволуционог и потпуно повезаног слоја. Сви модели су тренирани сликама у RGB и YUV формату, при чему је трећи дао најбоље резултате над скупом слика за тестирање, тако да је његова имплементација дата на слици бр. 31.

```
def create_model_3():
    model = Sequential()

    model.add(Lambda(lambda x: x / 127.5 - 1., input_shape=(66, 200, 3)))
    model.add(Conv2D(24, (5, 5), activation='elu', strides=(2, 2)))
    model.add(Dropout(0.2))
    model.add(Conv2D(36, (5, 5), activation='elu', strides=(2, 2)))
    model.add(Dropout(0.2))
    model.add(Conv2D(48, (5, 5), activation='elu', strides=(2, 2)))
    model.add(Dropout(0.2))
    model.add(Conv2D(64, (3, 3), activation='elu'))
    model.add(Dropout(0.2))
    model.add(Conv2D(64, (3, 3), activation='elu'))
    model.add(Dropout(0.2))

    model.add(Flatten())

    model.add(Dense(100, activation='elu'))
    model.add(Dropout(0.2))
    model.add(Dense(50, activation='elu'))
    model.add(Dropout(0.2))
    model.add(Dense(10, activation='elu'))
    model.add(Dense(1))

    model.summary()

    return model
```

Слика 31: Имплементација PilotNet мреже

Тренирање мреже врши се у оквиру функције *train_model* (слика бр. 32). Најпре се креира и компајлира модел, при чему се користи Adam оптимизација, која је једна од најефикаснијих када је у питању тренирање дубоких неуронских мрежа. Затим се учитава скуп тренинг-података, где се један податак састоји од пара: (слика, угао). Коришћењем функције *train_test_split* из Scikit-learn библиотеке овај скуп је подељен на тренинг и валидациони део, у размери 80:20. Приликом тренирања примењени су и механизми које нуди Keras библиотека, као што су ModelCheckpoint (за чување модела добијеног након сваке епохе) и EarlyStopping (омогућава раније заустављање тренирања уколико одређени број епоха заредом нема побољшања модела). Константама LEARNING_RATE, EPOCHS, TRAIN_BATCH_SIZE и VALID_BATCH_SIZE је иницијализовано тренирање, а њихове вредности су постављене редом на 1e-4, 30, 64 и 64. У оквиру функције *generate_next_batch* (чија је имплементација приказана на слици бр. 33) генерише се један по један пакет података за тренирање, док функција *fit_generator* извршава тренирање креираног модела, који се затим чува у „model.h5“ фајлу. Историја тренирања се у функцији *save_history* чува у фајл, на основу кога се касније могу креирати различити графици.

```
def train_model(csv_path, images_path):
    model = create_model_3()
    model.compile(optimizer=Adam(LEARNING_RATE), loss="mse")

    data = load_csv_data(csv_path)
    train_data, valid_data = train_test_split(data, test_size=0.2)

    checkpoint = ModelCheckpoint('model-{epoch:02d}.h5', monitor='val_loss', verbose=1, save_best_only=False,
                                 mode='auto')

    stop = EarlyStopping(monitor="val_loss", patience=3, verbose=1)

    steps_per_epoch = int(len(train_data) / TRAIN_BATCH_SIZE)
    validation_steps = int(len(valid_data) / VALID_BATCH_SIZE)

    train_gen = generate_next_batch(train_data, images_path, TRAIN_BATCH_SIZE)
    validation_gen = generate_next_batch(valid_data, images_path, VALID_BATCH_SIZE)

    history = model.fit_generator(train_gen,
                                  steps_per_epoch,
                                  epochs=EPOCHS,
                                  validation_data=validation_gen,
                                  validation_steps=validation_steps,
                                  callbacks=[checkpoint, stop],
                                  verbose=1)

    model.save("model.h5", save_format="h5")
    save_history(history)
```

Слика 32: Функција за тренирање модела

```
def generate_next_batch(data, images_path, batch_size):
    while True:
        x_batch, y_batch = [], []
        rnd_indices = list(range(0, len(data)))
        shuffle(rnd_indices)

        for index in rnd_indices:
            file_name, img, angle, speed = load_image(data, images_path, index)

            new_image = preprocess_img(img)
            angle = camera_adjust(angle, speed, file_name)

            x_batch.append(new_image)
            y_batch.append(angle)

            if len(x_batch) == batch_size:
                yield np.array(x_batch), np.array(y_batch)
                x_batch, y_batch = [], []
```

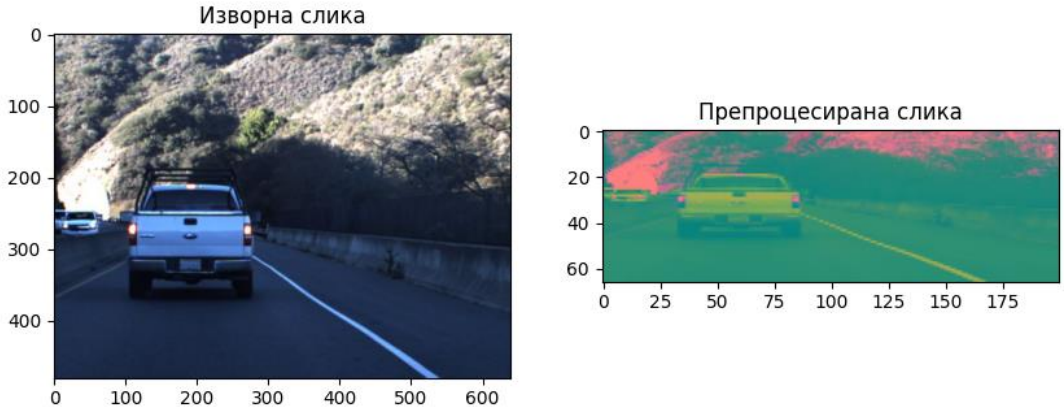
Слика 33: Функција за генерисање пакета података

Лева и десна камера су померене у односу на средишњу, тако да би требало модификовати углове скретања који су везани за њих. Како оне симулирају два специфична одступања возила од средишта саобраћајне траке, ова корекција је (у оквиру функције *camera_adjust*) рачуната као угао потребан да се возило врати на централну позицију, уз познавање његове брзине и претпоставку да су му потребне две секунде да то обави, као и да су лева и десна камера померене пола метра улево, односно удесно, у односу на средишњу камеру (дуж *x*-осе).

Све слике су пре прослеђивања моделу препроцесиране у оквиру функције *preprocess_img*, чија је имплементација дата на слици бр. 34. Овај корак обухвата одсецање горњих 100 пиксела и промену димензија слике на 66×200 пиксела, након чега се опционо врши конвертовање у YUV простор боја, при чему се одабир између RGB и YUV слика се врши кроз *clr* параметар. Пример препроцесирања једне од слика из тренинг-скупа приказан је на слици бр. 35.

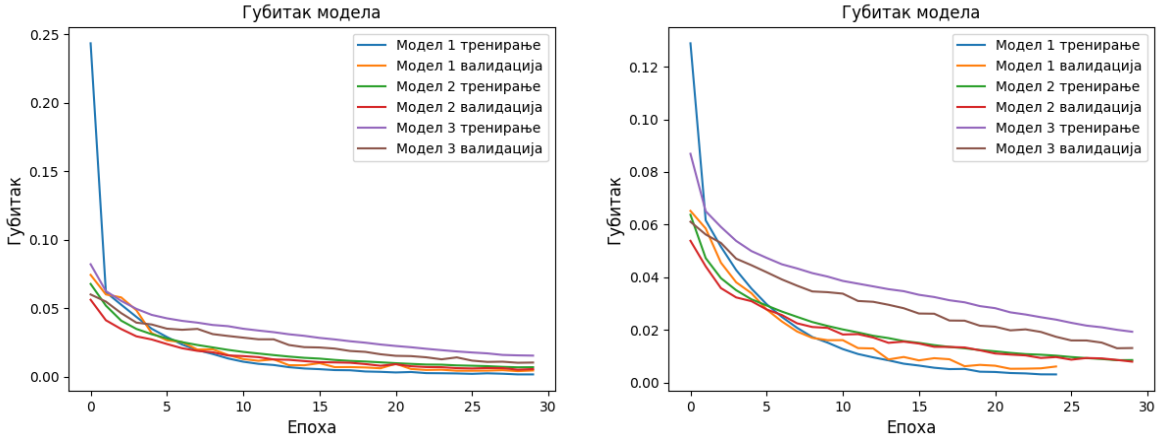
```
def preprocess_img(image, clr="RGB"):
    image = image[100:image.shape[0], 0:image.shape[1]]
    image = cv2.resize(image, (200, 66), cv2.INTER_AREA)

    if clr == "YUV":
        image = cv2.cvtColor(image, cv2.COLOR_RGB2YUV)

    return image
```

Слика 34: Функција за препроцесирање слика



Слика 35: Пример препроцесирања слике

На слици бр. 36 је приказан губитак при тренирању свих модела, за оба простора боја. Као функција губитка коришћена је MSE (средња квадратна грешка), која највише одговара регресији. Губитак при тренирању је најчешће већи него при валидацији, зато што се регуларизациони механизми искључују током валидације, а поред тога, губитак при тренирању је просек губитака над „пакетима података“ (модел се мења кроз време, при чему је губитак у почетним фазама епохе знатно већи него у последњим), док се валидациони губитак рачуна за модел какав је на крају епохе. Може се приметити да се на RGB графику све линије приближавају нули на исти начин, као и да се код YUV графика за први модел активирао механизам ранијег заустављања пре 25. епохе. Коначне вредности на крају тренирања су у оба случаја врло сличне.

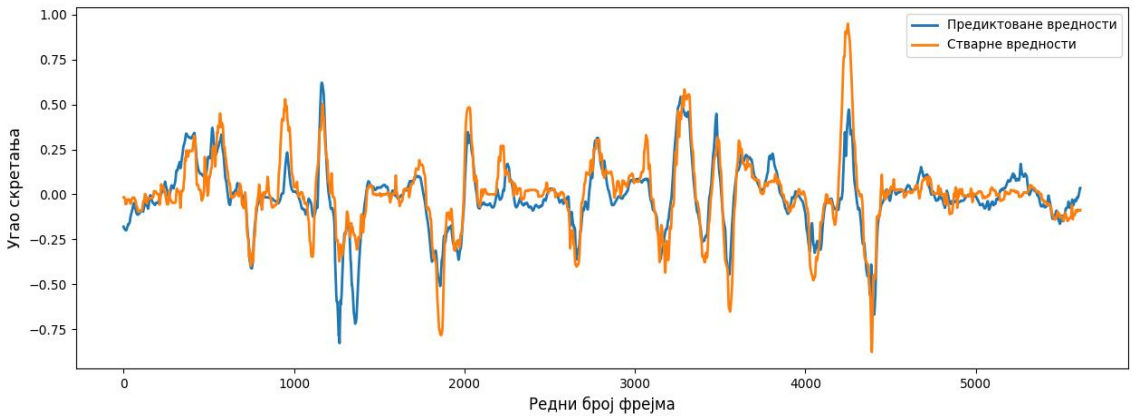


Слика 36: Губитак код тренирања модела за RGB (лево) и YUV слике (десно)

У табели бр. 1 се могу видети вредности за MSE (изражене у rad^2) над тест-скупом од 5 614 слика са централне камере за сваку од верзија модела, за оба простора боја. Може се приметити да је RGB варијанта давала мању грешку (осим за модел 2), и да је модел 3 генерисао најбоље резултате (око 0.0141 за RGB и 0.0154 за YUV). Ова вредност је рачуната као: MSE $= \Sigma \frac{(y_{pred} - y_{ref})^2}{N}$, где је N укупан број разматраних углова, y_{pred} предиктовани, а y_{ref} референтни угао. На слици бр. 37 је приказана разлика између стварних и предиктованих вредности за модел 3 над RGB сликама. Може се приметити да у неколико ситуција на графику при већим угловима скретања постоје велика одступања (која би у реалности највероватније довела до скретања аутомобила са пута).

***Табела 1**: Вредности за MSE над тест-подацима*

	RGB	YUV
Модел 1	0.0175	0.0179
Модел 2	0.0172	0.0168
Модел 3	0.0141	0.0154



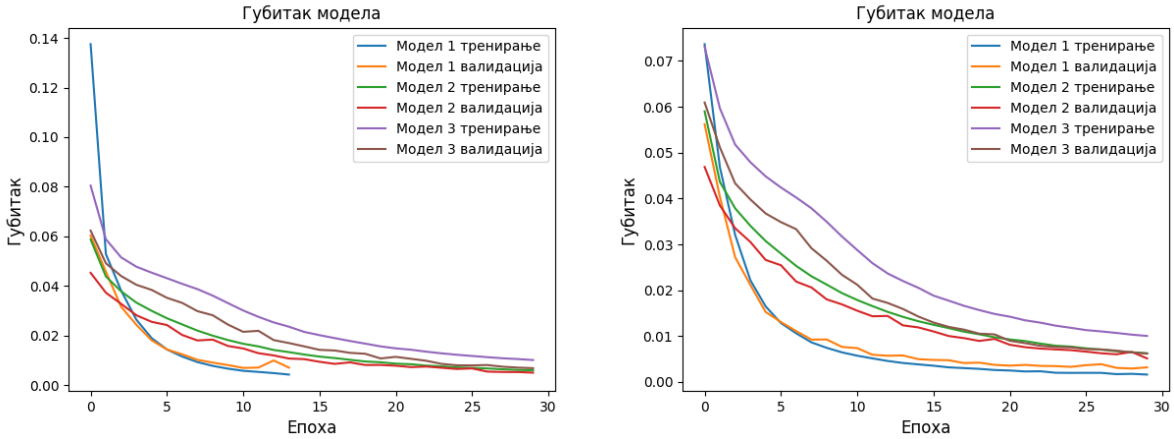
Слика 37: Разлика између стварних и предиктованих вредности за тест-скуп

Како би се добио модел који боље генерализује, односно даје мању MSE за тест-скуп, креиран је проширени (три пута већи) скуп од 304 188 слика. Нове слике су добијене применом додавања насумичног осветљења (употребом гама корекције [38]), мале ротације и пресликавања у односу на *y*-осу над сликама из постојећег скупа – што је дало расподелу углова са већом симетричношћу. На слици бр. 38 је приказана функција за генерисање нове слике на основу слике *image* (која јој се прослеђује као параметар).

```
def augment_image(image, steering_angle):
    image, steering_angle = random_flip(image, steering_angle)
    image = random_brightness(image)
    image = random_rotate(image)

    return image, steering_angle
```

Слика 38: Функција за генерисање нове слике

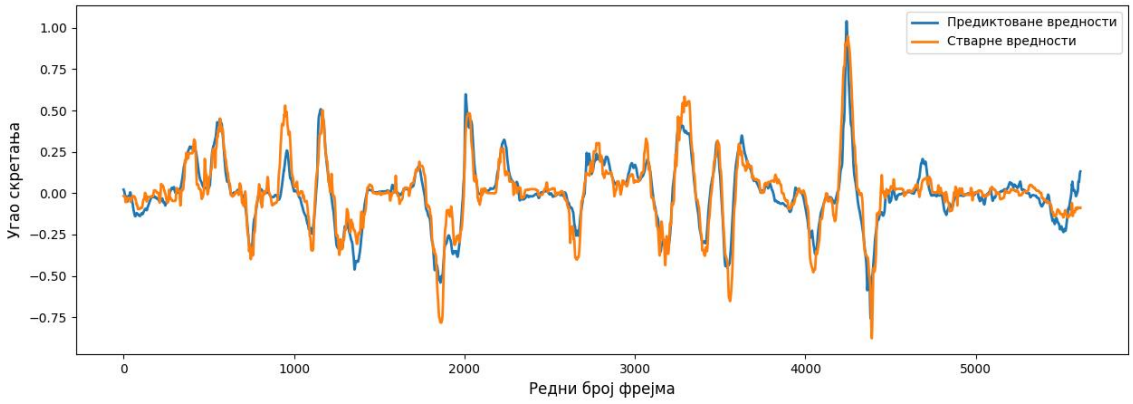
Губици при тренирању проширеног скупа података су мањи него при тренирању основног (слика бр. 39), а приметна је и разлика у вредностима за MSE над тест-скупом. Добијени резултати приказани су у табели бр. 2, где се може уочити да су у свим случајевима вредности за MSE мање него у табели 1, док се на графику (за најбољи модел) на слици бр. 40 може уочити да постоји веће поклапање са референтним угловима него у претходном случају. Велика одступања која су се јављала при већим угловима скретања сада више нису толико изражена. Вредност која је добијена за MSE за трећи модел, када су коришћене RGB слике, износи 0.0071, а за YUV слике 0.0074.



Слика 39: Губитак код тренирања модела за RGB (лево) и YUV слике (десно) при коришћењу проширеног тренинг-скупа

***Табела 2**: Вредности за MSE над тест-подацима при коришћењу проширеног тренинг-скупа*

	RGB	YUV
Модел 1	0.0129	0.0141
Модел 2	0.0086	0.0093
Модел 3	0.0071	0.0074



Слика 49: Разлика између стварних и предиктованих вредности за тест-скуп при коришћењу проширеног тренинг-скупа

Графици за упоређивање стварних и предиктованих вредности генерисани су на основу података из фајла названог „result.csv“, који је креиран коришћењем *data_frame* објекта у оквиру функције *test* (слика бр. 41). Овај објекат садржи две колоне: прва чува тачне вредности углова („ref_angle“), а друга предиктоване („pred_angle“). Предиктовани углови се добијају тако што се препроцесирана слика из тест-скупа прослеђује функцији *predict*, која се позива над претходно учитаним моделом. Приликом цртања графика је употребљена Matplotlib библиотека, при чему нису коришћене директне вредности које генерише модел, већ су углови „заглађени“ – како не би постојало нагло заокретање виртуелног волана. Угао је рачунат на следећи начин:

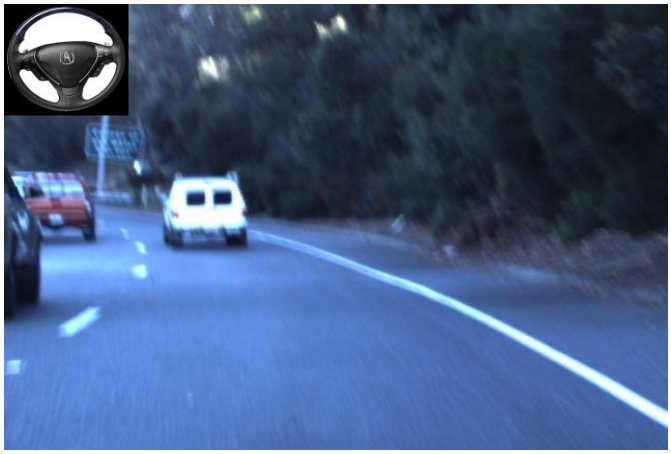
$$ugao = \alpha * ugao + (1 - \alpha) * prethodni_ugao,$$

при чему је вредност 0.1 за коефицијент α дала најбоље резултате. Овај коефицијент дефинише удео тренутно генерисаног угла у коначној вредности на коју се поставља угао скретања.

```
def test(model, csv_path, images_path):
    raw_data = pd.read_csv(csv_path, names=["frame_id", "angle", "public"])
    data_frame = []
    for i in range(len(raw_data)):
        image = plt.imread(images_path + raw_data.iloc[i]["frame_id"] + ".jpg")
        image = preprocess_img(image)

        ref_angle = raw_data.iloc[i]["angle"]
        pred_angle = float(model.predict(image[None, :, :, :], batch_size=1))

        data_frame.append([ref_angle, pred_angle])

    data = pd.DataFrame(data_frame, columns=["ref_angle", "pred_angle"])
    data.to_csv("result.csv", mode='a', index=False, header=False)
```

Слика 41: Функција за генерисање „result.csv“ фајла

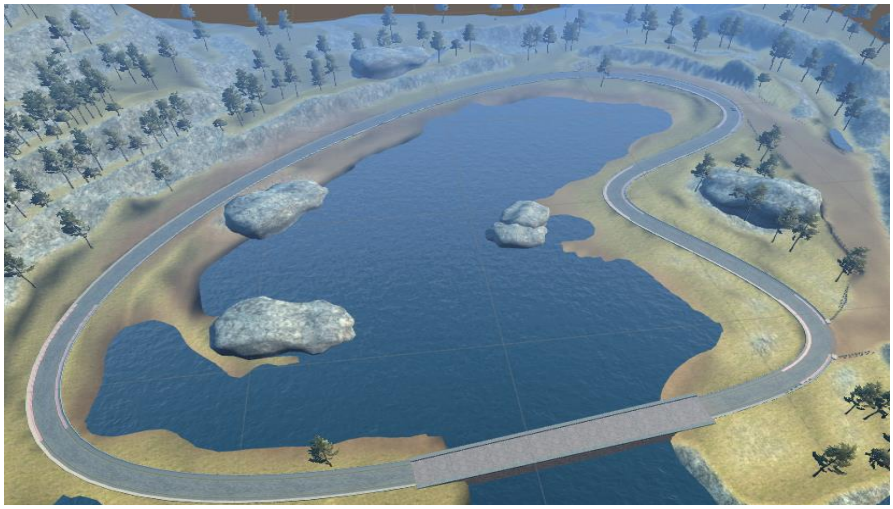
Да би се визуелизовало како би изгледала вожња када би модел генерисао угао скретања, од слика из тест-скупа направљен је видео-снимак, при чему је у горњем левом углу додата мала слика управљача, која је заротирана за онолико колика је вредност прерачунатог угла. Пример једног овако креираног фрејма може се видети на слици бр. 42.



Слика 42: Визуелизација окретања управљача

Оваква визуелизација не одговара реалном стању, зато што се морају узети у обзир динамика аутомобила и услови на путу. У пракси се за симулирање управљања аутомобилом користе симулатори аутономне вожње, а с обзиром на то да напреднији симулатори нуде високу реалистичност – они захтевају „јачи“ рачунар и подразумевају постојање GPU подршке. У оквиру овог пројекта је из тог разлога употребљен Udacity симулатор, који нема велике хардверске захтеве, и може се извршавати без графичке картице.

Симулатор нуди две подразумеване стазе, а у оквиру Unity окружења могу се креирати и нове. Прва стаза („Lake Track“) је направљена око језера (слика бр. 43), а садржи кривине, излазе и улазе на пут, мост (који има различиту подлогу) и делове где фале линије, или има додатних ознака. Друга стаза („Jungle Track“), приказана на слици бр. 44, има знатно оштрије кривине у односу на прву, као и велике узбрдице и низбрдице. У овом пројекту је симулација извршавана само на првој стази.



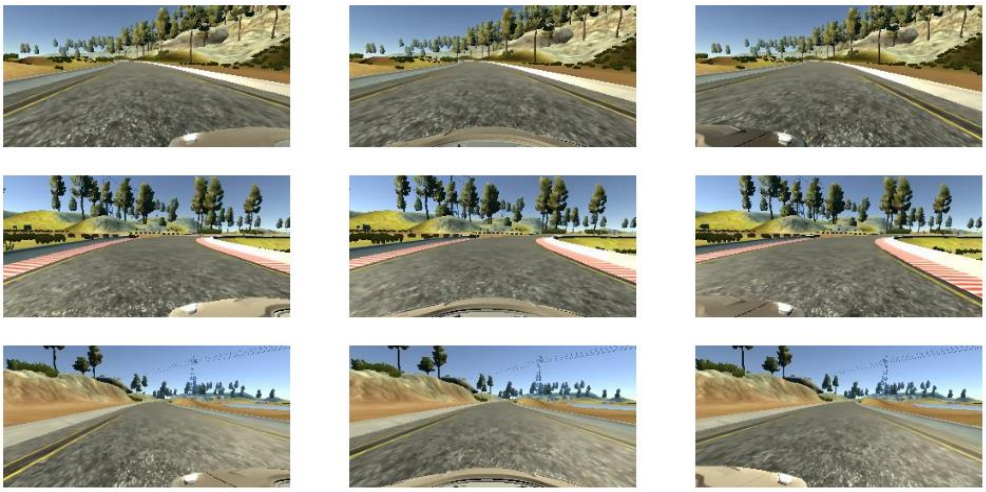
Слика 43: Поглед одозго на „Lake Track“ стазу

Слика 44: Поглед одозго на „Jungle Track“ стазу

Udacity нуди тренинг-скуп података који се састоји од 24 108 слика (са три различите камере), добијених у 8 036 фрејмова, при чему се оне налазе у фолдеру „IMG“, а пратеће информације у фајлу „driving_log.csv“. Овај скуп података је добијен вожњом неколико кругова у оба смера, када се возило креће по средини стазе, као и по ивицама стазе (након чега се возило „опоравља од грешке“ и враћа на средину). Забележени пратећи подаци укључују путање до слика са камера, брзину, угао скретања, и вредност притиска гаса и кочнице (функција за њихово учитавање је приказана на слици бр. 45). На слици бр. 46 је дато неколико насумично одабраних примера слика из овог скупа података са све три камере, док се потребни подаци везани за једну слику учитавају у оквиру функције *load_image* (слика бр. 47). Код слика са леве и десне камере угао је при тренирању коригован за 0,25. [39]

```
def load_csv_data(csv_path):
    data = pd.read_csv(csv_path, header=None,
                       names=["center", "left", "right", "steering", "throttle", "brake", "speed"])

    return data
```

Слика 45: Функција за учитавање података из „driving_log.csv“ фајла

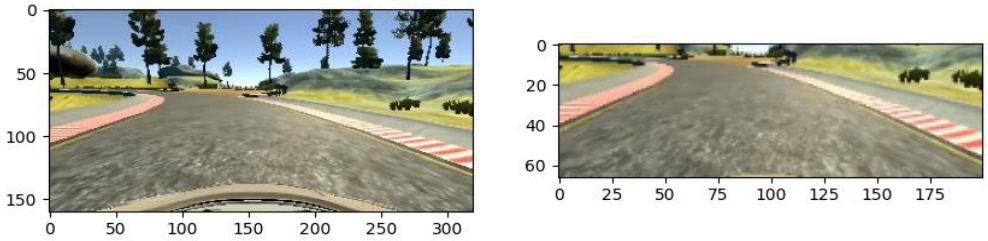


Слика 46: Насумично одабрани примери слика са леве, средишње и десне камере, респективно

```python
def load_image(data, img_path, index, position):
    file_name = data.iloc[index][position].split('/')[-1]
    img = plt.imread(os.path.join(img_path, file_name))
    angle = data.iloc[index]['steering']

    return img, angle, file_name
```

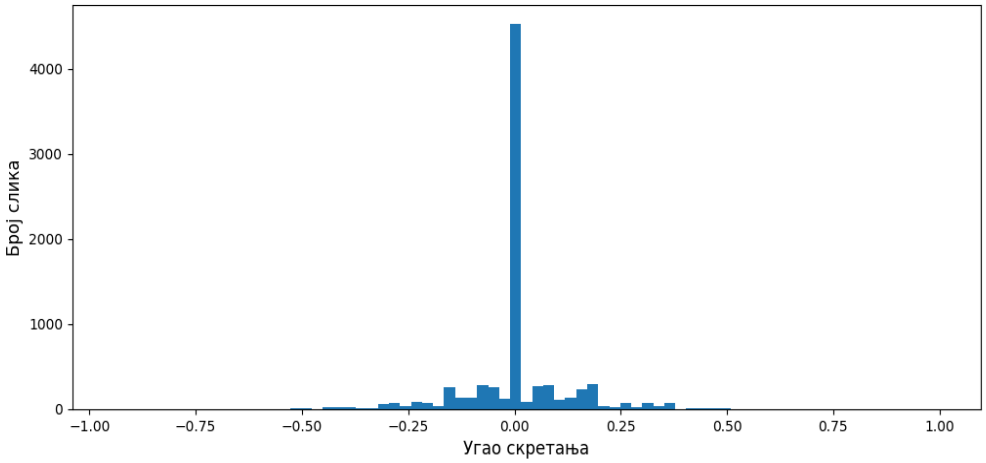
Слика 47: Функција за учитавање једне слике

Слика (димензија 160×320 пиксела) се пре прослеђивања моделу препроцесира (пример на слици бр. 48) тако да се одсече део хаубе који се види на дну слике, као и део неба који је у горњем делу слике (горњих 55 и доњих 20 пиксела). Затим се њене димензије мењају на оне које очекује модел (ширина од 200 пиксела и висина од 66 пиксела), и поред тога се опционо трансформише у YUV простор боја.

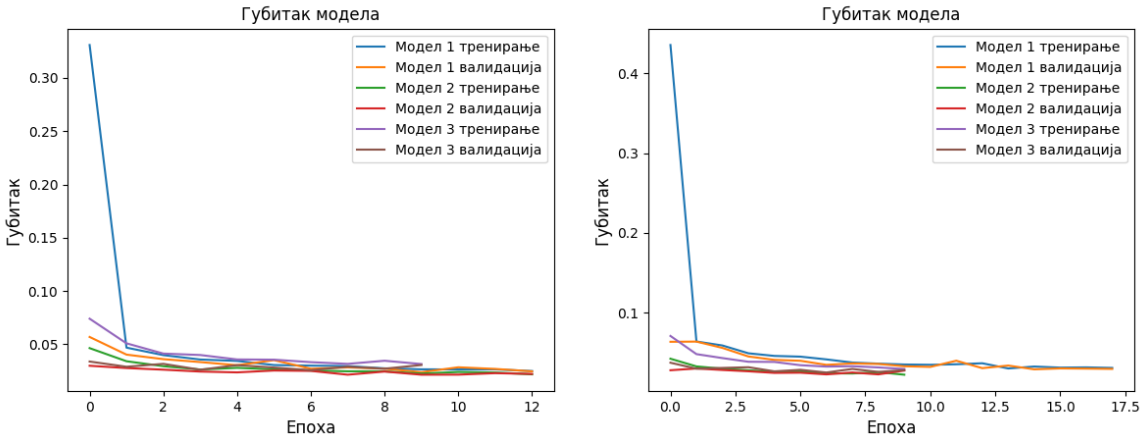


Слика 48: Оригинална (лево) и препроцесирана слика (десно)

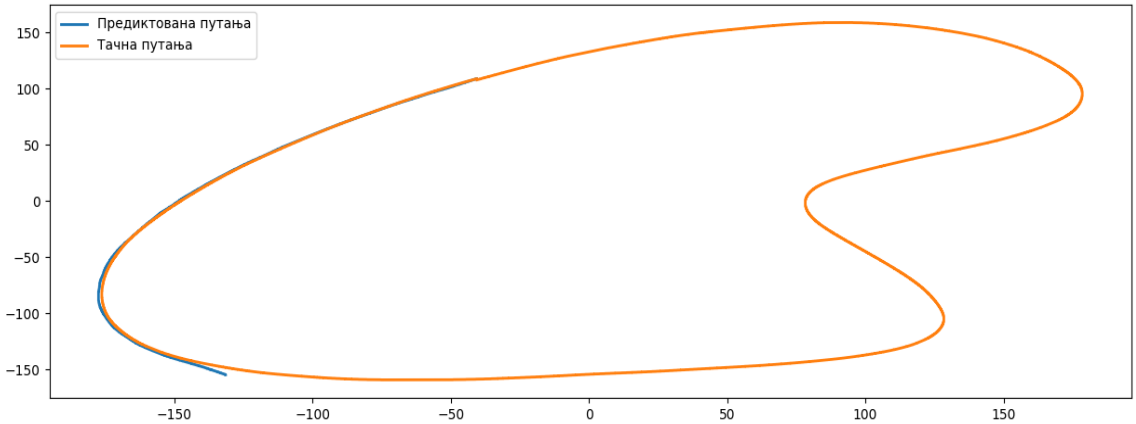
Симулатор користи углове у опсегу од -25 до +25 степени, који се складиште скалирани на опсег од -1 до 1. На графику на слици бр. 49 приказано је како изгледа расподела углова скретања, који су снимљени при константној брзини. Може се приметити да су подаци небалансирани и да постоји велики пик око нуле. Слика бр. 50 приказује губитак приликом тренирања модела, при чему ниједан од њих није дао задовољавајуће резултате (аутомобил је наилазио на проблем у кривинама и није успео да до краја извози стазу). На слици бр. 51 наранџастом бојом је приказано како би изгледала комплетна путања да се возило кретало средином траке, а плавом бојом како изгледа путања којом се оно кретало када је њиме управљао један од модела (за све моделе путања је била слична), док је на слици бр. 52 приказано како изгледа крај те путање у симулатору, односно скретање возила са пута.



Слика 49: Расподела углова скретања у тренинг-скупу података за симулатор



Слика 50: Губитак код тренирања модела за RGB (лево) и YUV слике (десно) у симулатору



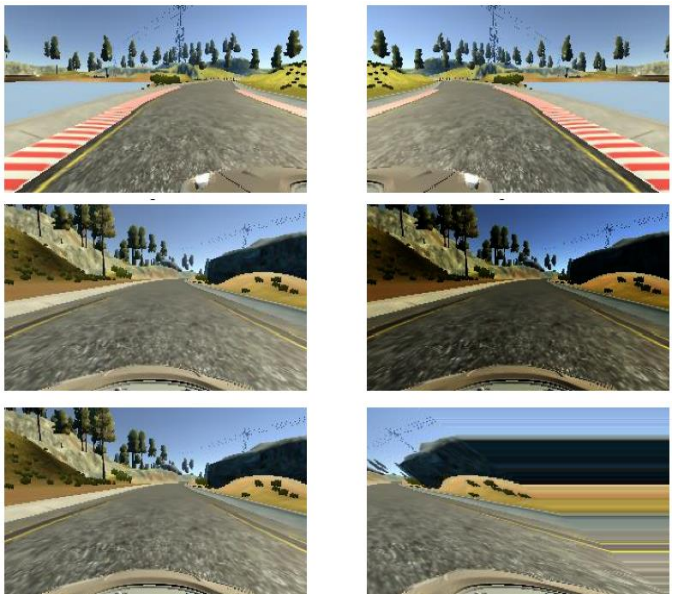
Слика 51: Предиктована (плаво) и тачна путања (наранџасто)

Слика 52: Скретање возила са пута

```
def augment_image(image, steering_angle):
    image, steering_angle = random_shear(image, steering_angle)
    image, steering_angle = random_flip(image, steering_angle)
    image = random_brightness(image)

    return image, steering_angle
```

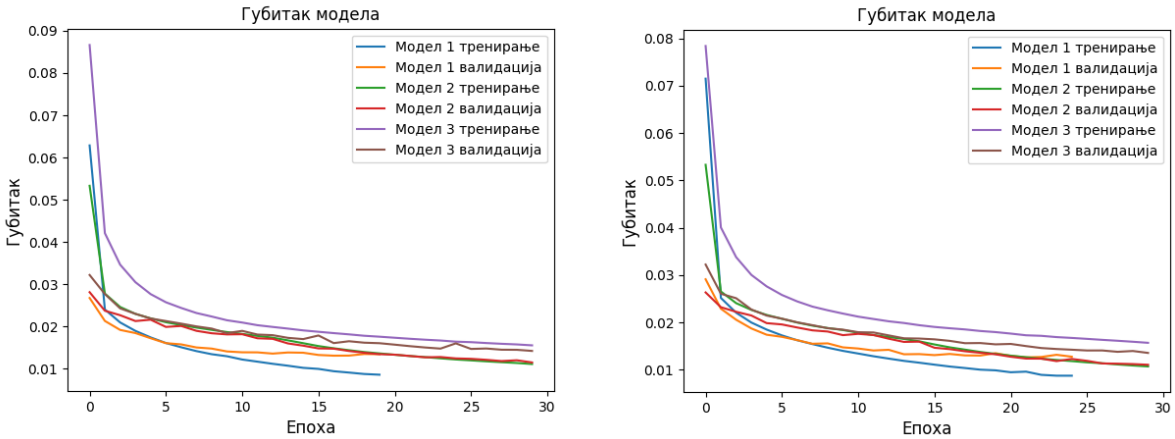
Слика 53: Функција за генерисање нове слике

Како би се добио функционалан модел генерисан је нови скуп података, применом техника за проширивање (функција *augment_image* са слике бр. 53). Нове слике су генерисане на основу постојећег скупа, уз насумичну промену осветљености, смицање (којим се додаје ефекат кривина на путу [40]) и пресликавање у односу на y-осу, при чему је вршена и корекција угла скретања. Добијени скуп садржи 320 000 слика у „IMG“ фолдеру и пратећи „driving_log.csv“ фајл. Већи број слика и технике проширивања дале су боље моделе (који су успешно извозили целу стазу), а на слици бр. 54 дати су примери ових операција.



Слика 54: Пример пресликавања слике у односу на y-осу (изнад), промене осветљења (средина) и смицања (испод)

На слици бр. 55 се могу видети губици добијени за све моделе када су они тренирани над проширеним скупом података. На графицима се може приметити да су почетне вредности за губитке мање него када су модели тренирани над полазним подацима, као и да је тренирање трајало више епоха (механизам ранијег заустављања се активирао само у случају модела 1).



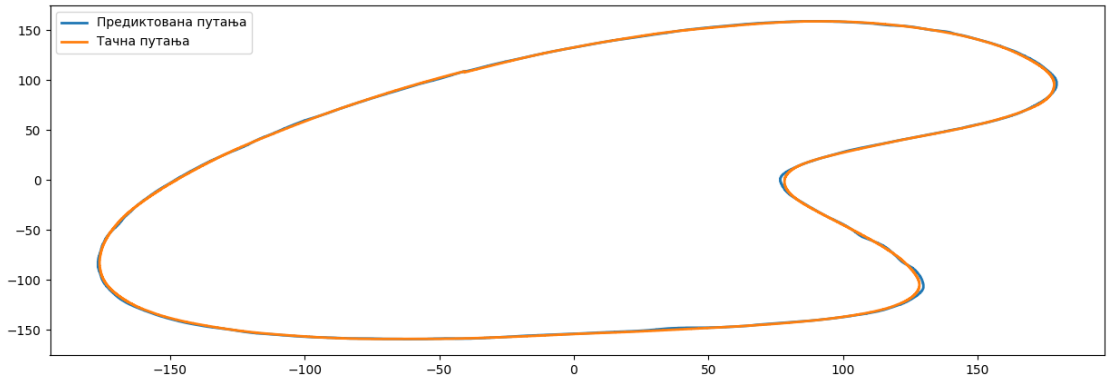
Слика 55: Губитак код тренирања модела за RGB (лево) и YUV слике (десно) при коришћењу проширеног тренинг-скупа у симулатору

Неке од мера перформанси које се могу користити приликом валидације аутономних возила су: број интервенција од стране возача или просечно пређено растојање између интервенција, прецизност, укупно време потребно за достизање циља, удобност вожње итд. У овом пројекту је посматрана прецизност, тј. удаљеност возила од средине траке, при чему је рачуната вредност за MSE (изражена у m^2) између предиктоване путање и путање која се поклапа са средиштем стазе. У табели 3 су приказане добијене вредности за сваки од модела, где се може приметити да је и у овом случају RGB варијанта модела 3 дала најбоље резултате (0.2389).

***Табела 3**: Вредности за MSE над тест-подацима при коришћењу проширеног тренинг-скупа у симулатору*

	RGB	YUV
Модел 1	0.4536	0.6242
Модел 2	0.6762	0.4628
Модел 3	0.2389	0.3581

Сви посматрани модели су успешно прешли целу стазу, а на слици бр. 56 дат је график на коме се виде предиктована и референтна путања за модел који је дао најмању грешку. Може се приметити да се ове две путање скоро у потпуности поклапају, уз одређена одступања у кривинама.



Слика 56: Предиктована (плаво) и тачна путања (наранџасто) при коришћењу проширеног тренинг-скупа

Део кода који служи за интеракцију са симулатором своди се на покретање Flask апликације на порту 4567. На слици бр. 57 приказано је које библиотеке је потребно укључити да би се обезбедила ова комуникација, и како се креирају сервер и апликација. При покретању програма (слика бр. 58) је потребно учитати мрежу, а затим и обезбедити постојање фолдера у коме ће се чувати фрејмови са централне камере, које шаље симулатор (како би касније могао да се направи видео-снимак од њих и погледа како је изгледала вожња).

```
import socketio
import eventlet.wsgi
from flask import Flask

sio = socketio.Server()
app = Flask(__name__)
```

Слика 57: Креирање сервера и апликације

```
if __name__ == '__main__':
    model = load_model(MODEL_PATH)

    if IMAGE_FOLDER != '':
        if not os.path.exists(IMAGE_FOLDER):
            os.makedirs(IMAGE_FOLDER)
        else:
            shutil.rmtree(IMAGE_FOLDER)
            os.makedirs(IMAGE_FOLDER)
        print("RECORDING THIS RUN ...")
    else:
        print("NOT RECORDING THIS RUN ...")

    # wrap Flask application with engineio's middleware
    app = socketio.Middleware(sio, app)

    # deploy as an eventlet WSGI server
    eventlet.wsgi.server(eventlet.listen(('', 4567)), app)
```

Слика 58: Покретање комуникације са симулатором

Када се покрене и симулатор у аутономном моду, у том тренутку се аутоматски позива *connect* функција, која позива функцију за емитовање иницијалних управљачких команди возилу. Вредности *steering_angle* и *throttle* су упаковане у одговарајући објекат, а симулатор их директно примењује на возило (слика бр. 59).

```
@sio.on('connect')
def connect(sid, environ):
    print("connect ", sid)
    send_control(0, 0)


def send_control(steering_angle, throttle):
    sio.emit(
        "steer", data={
            'steering_angle': steering_angle.__str__(),
            'throttle': throttle.__str__()
        }, skip_sid=True)
```

Слика 59: Функције за повезивање са симулатором и слање команди симулатору

Главна функција за обраду слика из симулатора приказана је на слици бр. 60. Информације које стижу из симулатора налазе се у објекту *data*, од којих су од значаја за апликацију биле брзина возила, као и слика са средишње камере. Примљена слика се преводи у одговарајући запис и прослеђује моделу, који генерише наредни угао скретања, док је вредност за *throttle* рачуната преко PI (Proportional Integrator) контролера, који узима у обзир тренутну брзину и тежи да одржи брзину кретања аутомобила у симулатору константном (жељена вредност брзине се поставља при иницијализацији контролера). Свака од слика се снима у фолдер, чија је путања задата кроз константу IMAGE_FOLDER, а касније се оне сортирају по временском тренутку у коме су настале, како би биле исправно уређене у видео-снимку. Овај систем је генерисао само угао скретања, али може се проширити тако да предвиђа и вредност *throttle*.

```
@sio.on('telemetry')
def telemetry(sid, data):
    if data:
        speed = data["speed"]
        img_string = data["image"]

        image = Image.open(BytesIO(base64.b64decode(img_string)))
        image_array = np.asarray(image)
        image_array = preprocess_img(image_array)

        steering_angle = float(model.predict(image_array[None, :, :, :], batch_size=1))
        throttle = controller.update(float(speed))

        send_control(steering_angle, throttle)

        if IMAGE_FOLDER != '':
            timestamp = datetime.utcnow().strftime('%Y_%m_%d_%H_%M_%S_%f')[:-3]
            image_filename = os.path.join(IMAGE_FOLDER, timestamp)
            image.save('{}.jpg'.format(image_filename))

    else:
        sio.emit('manual', data={}, skip_sid=True)
```

Слика 60: Функција за обраду података из симулатора

5. Закључак

Данас већина модернијих аутомобила има одређени степен аутоматизације, а процењује се да ће аутономна возила бити део свакодневнице до 2030. године, као и да имају потенцијала да у потпуности замене традиционална возила до 2050. године. Да би се реализовао потпуно аутономни систем за вожњу неопходно је стално унапређивати хардвер – како би се добили што прецизнији подаци, као и софтвер – како би вожња била што ефикаснија и безбеднија.

Први (и најзначајнији) корак у постизању аутономне вожње јесте перцепција окружења, која се ослања на податке са сензора (камера, лидар, радар). Сваки од сензора доприноси стварању 3D визуелизације самог возила, и стања објеката око возила. Перцепција окружења је процес који код аутономних система подразумевају оба архитектурна приступа (модуларни и од-краја-до-краја). У модуларном систему иза овог следи још неколико додатних корака, како би се добиле крајње управљачке команде, а код система од-краја-до-краја се ове команде генеришу директно на основу улазних информација.

Дубоко учење има све значајнију улогу у софтверским системима, па налази примену и у аутономним возилима. У овим системима се може искористити у оквиру различитих компоненти, а неки од задатака у којима дубоке конволуционе мреже имају кључну улогу јесу детекција и препознавање објеката на путу. Поред тога, показало се да се ове мреже (уз одређена ограничења) могу применити у системима који уче од-краја-до-краја. Једну од најпознатијих мрежа за ову намену (PilotNet) представила је компанија NVIDIA.

У овом раду фокус је био на примени конволуционих неуронских мрежа за генерисање угла скретања аутомобила. Коришћена је прва верзија PilotNet мреже, која је ову вредност генерисала на основу података са три камере (средишње, леве и десне). Како би се спречило преобучавање мреже, у две варијанте овог модела су додати слојеви одбацивања. Модели су тренирани над јавно доступним скупом података, који садржи податке добијене при вожњи на стварним путевима, као и над подацима генерисаним у Udacity симулатору, при чему је тестирано како се они понашају над тест-скупом, односно како управљају аутомобилом у аутономном режиму симулатора. Такође, модели су тренирани над основним скупом, као и над већим скупом, добијеним применом техника за генерисање додатних слика. Праћена је средња квадратна грешка (MSE), а добијени резултати у оба случаја показали су да слојеви одбацивања доприносе бољој генерализацији модела, али да већи значај имају величина тренинг скупа и технике за генерисање додатних слика.

Овакав систем се може проширити тако да мрежа не генерише само угао скретања, већ и остале управљачке команде (убрзање и кочење), чиме би се аутомобилом у потпуности управљало на основу креираног модела. Такође, треба узети у обзир да је коришћена најједноставнија варијанта мреже и да није узета у обзир путања којом се возило треба кретати до одредишта, као и да није укључен процес доношења одлука (нпр. бирање између скретања лево/десно и настављања право на раскрсницама).

Кроз овај рад је дат осврт на могућности које нуде конволуционе мреже у управљању аутономним возилима. У пракси се користе знатно комплекснији системи, који су пројектовани као модуларни, јер у стварном окружењу постоји много задатака које „возач“ треба да реализује. То је актуелна тема, а у наредним годинама се очекује значајан напредак у развоју ове технологије, док се модели аутомобила који су тренутно доступни на тржишту још увек сврставају максимално у SAE ниво 2.

Литература

[1] „Self-driving car“, https://en.wikipedia.org/wiki/Self-driving_car, [последњи приступ: 10. мај 2023]

[2] „Self-Driving Cars: The Pros & Cons of Automated Vehicles“, https://valientemott.com/auto-collisions/self-driving-cars-pros-and-cons/, [последњи приступ: 11. мај 2023]

[3] „Houdina Radio Control“, https://en.wikipedia.org/wiki/Houdina_Radio_Control, [последњи приступ: 11. мај 2023]

[4] „History of self-driving cars“, https://en.wikipedia.org/wiki/History_of_self-driving_cars, [последњи приступ: 12. мај 2023]

[5] M. Weber, „Where to? A history of autonomous vehicles“, https://computerhistory.org/blog/where-to-a-history-of-autonomous-vehicles/?key=where-to-a-history-of-autonomous-vehicles, [последњи приступ: 11. мај 2023]

[6] „Media resources“, https://waymo.com/press/, [последњи приступ: 11. мај 2023]

[7] „SAE Levels of Driving Automation™ Refined for Clarity and International Audience“, https://www.sae.org/blog/sae-j3016-update, [последњи приступ: 13. мај 2023]

[8] S. Khvoynitskaya, „3 types of autonomus vehicle sensors in self-driving cars“, https://www.itransition.com/blog/autonomous-vehicle-sensors, [последњи приступ: 20. мај 2023]

[9] „How Self-driving Cars Work: Sensor Systems“, https://www.udacity.com/blog/2021/03/how-self-driving-cars-work-sensor-systems.html, [последњи приступ: 20. мај 2023]

[10] „Introducing Waymo’s suite of custom-built, self-driving hardware“, https://medium.com/waymo/introducing-waymos-suite-of-custom-built-self-driving-hardware-c47d1714563, [последњи приступ: 17. мај 2023]

[11] E. Yurtsevr, J. Lambert, A. Carballo, K. Takeda, „A Survey of Autonomous Driving: Common Practices and Emerging Technologies“, arXiv:1906.05113v3 [cs.RO], 2. април 2020.

[12] J. Janai, F. Güney, A. Behl, A. Geiger, „Computer Vision for Autonomous Vehicles: Problems, Datasets and State of the Art“, arXiv:1704.05519v3 [cs.CV], 17. март 2021.

[13] J. Cohen, „AI… And the vehicle went autonomous“, https://thinkautonomous.medium.com/ai-and-the-vehicle-went-autonomous-e176c73239c6, [последњи приступ: 13. мај 2023]

[14] J. Cohen, „Sensor Fusion“, https://thinkautonomous.medium.com/sensor-fusion-90135614fde6, [последњи приступ: 22. мај 2023]

[15] S. Grigorescu, B. Trasnea, T. Cocias, G. Macesanu, „A Survey of Deep Learning Techniques for Autonomous Driving“, arXiv:1910.07738v2 [cs.LG], 24. март 2020.

[16] J. Cohen, „Self-Driving Cars & Localization“, https://thinkautonomous.medium.com/self-driving-car-localization-f800d4d8da49, [последњи приступ: 25. мај 2023]

[17] J. Cohen, „Path Planning for self-driving cars“, https://thinkautonomous.medium.com/can-self-driving-car-think-6c9e8d939d60, [последњи приступ: 30. мај 2023]

[18] J. Cohen, „Control Command in Self-Driving Cars“, https://thinkautonomous.medium.com/the-final-step-control-783467095138, [последњи приступ: 31. мај 2023]
[19] D. Pomerleau, „ALVINN: An Autonomous Land Vehicle In a Neural Network“, Carnegie Mellon University, 1989, https://proceedings.neurips.cc/paper/1988/file/812b4ba287f5ee0bc9d43bbf5bbe87fb-Paper.pdf, [последњи приступ: 29. мај 2023]
[20] M. Mandal, „Introduction to Convolutional Neural Networks (CNN)“, https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/, [последњи приступ: 01. јун 2023]
[21] M. Bojarski, D. Del Testa, D. Dworakowski и сар, „End to End Learning for Self-Driving Cars“, arXiv:1604.07316v1 [cs.CV], 25. април 2016.
[22] Y. LeCun, U. Muller, J. Ben, E. Cosatto, B. Flepp, „Off-Road Obstacle Avoidance through End-to-End Learning“, Advances in Neural Information Processing Systems 18 [Neural Information Processing Systems, NIPS 2005, December 5-8, 2005, Vancouver, British Columbia, Canada], 2005, https://papers.nips.cc/paper_files/paper/2005/file/fdf1bc5669e8ff5ba45d02fded729feb-Paper.pdf [последњи приступ: 13. јун 2023]
[23] M. Bojarski, P. Yeres, A. Choromanaska, K. Choromanski, B. Firner, L. Jackel , U. Muller, „Explaining How a Deep Neural Network Trained with End-to-End Learning Steers a Car“, arXiv:1704.07911v1 [cs.CV], 25. април 2017.
[24] M. Bojarski, C. Chen, J. Daw, A. Degirmenci, J. Deri, B. Firner, B. Flepp, S. Gogri, J. Hong, L. Jackel, Z. Jia, B. Lee, B. Liu, F. Liu, U. Muller, S. Payne, N. Kota, N. Prasad, A. Provodin, J. Roach, T. Rvachov, N. Tadimeti, J. Engelen, H. Wen, E. Yang, Z. Yang, „The NVIDIA PilotNet Experiments“, arXiv:2010.08776v1 [cs.CV], 17. октобар 2020.
[25] F. Codevilla, M. Müller, A. Lopez, V. Koltun, A. Dosovitskiy, „End-to-end Driving via Conditional Imitation Learning“, arXiv:1710.02410v2 [cs.RO], 2. март 2018.
[26] D. Chen, B. Zhou, V. Koltun, P. Krähenbühul, „Learning by cheating“, arXiv:1912.12294v1 [cs.RO], 27. децембар 2019.
[27] A. E. Sallab, M. Abdou, E. Perot, S. Yogamani, „Deep Reinforcement Learning framework for Autonomous Driving“, arXiv:1704.02532v1 [stat.ML], 8. април 2017.
[28] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J. Allen, V. Lam, A. Bewley, A. Shah, „Learning to Drive in a Day“, arXiv:1807.00412v2 [cs.LG], 11. септембар 2018.
[29] P. Kaur, S. Taghavi, Z. Tian, W. Shi, „A Survey on Simulators for Testing Self-Driving Cars“, arXiv:2101.05337v1 [cs.RO], 13. јануар 2021.
[30] „Simulation City: Introducing Waymo's most advanced simulation system yet for autonomous driving“, https://waymo.com/blog/2021/06/SimulationCity.html, [последњи приступ: 16. јун 2023]
[31] „CARLA Open-source simulator for autonomous driving research“, https://carla.org/, [последњи приступ: 13. јун 2023]
[32] A. Dosovitskiy , G. Ros, F. Codevilla, A. Lopez , V. Koltun, „CARLA: An Open Urban Driving Simulator“, arXiv:1711.03938v1 [cs.LG], 10. новембар 2017.
[33] G. Rong , B. H. Shin , H. Tabatabaee , Q. Lu , S. Lemke, M. Možeiko, E. Boise, G. Uhm, M. Gerow, S. Mehta, E. Agafonov, T. H. Kim, E. Sterner, K. Ushiroda, M. Reyes, D. Zelenkovsky, S. Kim, „LGSVL Simulator: A High Fidelity Simulator for Autonomous Driving“, arXiv:2005.03778v3 [cs.RO], 22. јун 2020.

[34] „Welcome to AirSim“, https://microsoft.github.io/AirSim/, [последњи приступ: 08. јун 2023]
[35] „Welcome to Udacity's Self-Driving Car Simulator“, https://github.com/udacity/self-driving-car-sim/tree/term2_collection, [последњи приступ: 01. јун 2023]
[36] „CH2_002“, https://github.com/udacity/self-driving-car/tree/master/datasets/CH2, [последњи приступ: 20. мај 2023]
[37] R. Wightman, „udacity-driving-reader“, https://github.com/rwightman/udacity-driving-reader, [последњи приступ: 20. мај 2023]
[38] A. Rosebrock, „OpenCV Gamma Correction“, https://pyimagesearch.com/2015/10/05/opencv-gamma-correction/, [последњи приступ: 20. јул 2023]
[39] V. Yadav, „An augmentation based deep neural network approach to learn human driving behavior“, https://chatbotslife.com/using-augmentation-to-mimic-human-driving-496b569760a9#.yget03t5g, [последњи приступ: 22. јун 2023]
[40] K. Sakmann, „Behavioral Cloning – make a car drive like yourself“, https://medium.com/@ksakmann/behavioral-cloning-make-a-car-drive-like-yourself-dc6021152713#.7k8vfppvk, [последњи приступ: 21. јун 2023]