# Izgradnja algoritama za upravljanje vozilom u simulatoru primjenom podržanog učenja

**Činčurak, Dorian**





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

# IZJAVA O ORIGINALNOSTI RADA

**Osijek, 26.09.2023.**

| | |
|---|---|
| **Ime i prezime studenta:** | Dorian Činčurak |
| **Studij**: | Diplomski sveučilišni studij Automobilsko računarstvo i komunikacije |
| **Mat. br. studenta, godina upisa:** | D-59ARK, 06.10.2021. |
| **Turnitin podudaranje [%]:** | 4 |

Ovom izjavom izjavljujem da je rad pod nazivom**: Izgradnja algoritama za upravljanje vozilom u simulatoru primjenom podržanog učenja**

izrađen pod vodstvom mentora izv. prof. dr. sc. Ratko Grbić

i sumentora ,

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

Potpis studenta:

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

**Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit**

**Osijek, 19.09.2023.**

**Odboru za završne i diplomske ispite**

# Imenovanje Povjerenstva za diplomski ispit

| | |
|---|---|
| **Ime i prezime Pristupnika:** | Dorian Činčurak |
| **Studij, smjer:** | Diplomski sveučilišni studij Automobilsko računarstvo i komunikacije |
| **Mat. br. Pristupnika, godina upisa:** | D-59ARK, 06.10.2021. |
| **OIB studenta:** | 64567054786 |
| **Mentor:** | izv. prof. dr. sc. Ratko Grbić |
| **Sumentor:** | , |
| **Sumentor iz tvrtke:** | Zvonimir Kaprocki |
| **Predsjednik Povjerenstva:** | prof. dr. sc. Marijan Herceg |
| **Član Povjerenstva 1:** | izv. prof. dr. sc. Ratko Grbić |
| **Član Povjerenstva 2:** | prof. dr. sc. Mario Vranješ |
| **Naslov diplomskog rada:** | Izgradnja algoritama za upravljanje vozilom u simulatoru primjenom podržanog učenja |
| **Znanstvena grana diplomskog rada:** | **Umjetna inteligencija (zn. polje računarstvo)** |
| **Zadatak diplomskog rada:** | Podržano učenje je tip strojnog učenja koji se intenzivno istražuje posljednjih godina, a svodi se na učenje optimalnog ponašanja agenta u određenoj okolini. Uobičajeno se ovdje radi o softverskim agentima koji poduzimaju određene akcije iz popisa svih mogućih akcija kako bi došli u novo stanje (primjerice u igri zmije moguće se kretati lijevo, desno ili naprijed na svakom polju unutar ograničenog prostora). Pri tome agent može dobiti nagradu - to je informacija kojom se mjeri uspješnost ponašanja agenta (primjerice u igri zmije kada zmija pojede jabuku dobiva nagradu). Npr. jedan od popularnih algoritama podržanog učenja je duboko Q učenje. U okviru diplomskog rada potrebno je izraditi algoritam za upravljanje autonomnim vozilom |
| **Prijedlog ocjene pismenog dijela ispita (diplomskog rada):** | Izvrstan (5) |
| **Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:** | Primjena znanja stečenih na fakultetu: 3 bod/boda<br>Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda<br>Jasnoća pismenog izražavanja: 2 bod/boda<br>Razina samostalnosti: 3 razina |
| **Datum prijedloga ocjene od strane mentora:** | 19.09.2023. |

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

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

**Sveučilišni studij**

# IZGRADNJA ALGORITAMA ZA UPRAVLJANJE VOZILOM U SIMULATORU PRIMJENOM PODRŽANOG UČENJA

**Diplomski rad**

**Dorian Činčurak**

**Osijek, 2023.**

# SADRŽAJ

# 1. UVOD

Zadnjih se nekoliko godina u automobilskoj industriji sve više istražuju i razvijaju autonomna vozila. Unapređenjem tehnologije i inovacija u polju umjetne inteligencije, omogućen je razvoj automobila s visokim stupnjem automatizacije. Takva su vozila sve više prisutna na cestama, a njihova razina autonomne vožnje s vremenom raste. Prema društvu automotiv inženjera (engl. *Society of Automotive Engineers* – SAE) definirano je šest razina autonomne vožnje. Razine se kreću od razine nula do razine pet [1]. Na razini nula, na sve elemente vožnje djeluje izravno samo vozač. Razina pet predstavlja potpunu automatizaciju koja omogućava samostalno kretanje vozila bez potrebe za vozačem. Ovakvo vozilo praktički nema upravljač ni papučice, a vozač ni ne treba biti u vozilu. Cilj automobilske industrije je izraditi vozilo s razinom autonomije pet. Takva bi vozila uvelike pridonijela smanjenju prometnih gužvi i povećanju generalne sigurnosti vožnje [2]. Trenutno su na tržištu tek počela izlaziti vozila s razinom autonomije tri. U svibnju 2022. godine, Mercedes-Benz je postao prvi svjetski proizvođač koji je dobio odobrenje njemačkih prometnih vlasti za legalnu upotrebu vozila s razinom autonomije tri na svojim javnim prometnicama [3]. Vozila s razinom autonomije tri senzorskim sustavima percipiraju okoliš te mogu samostalno upravljati u strogo definiranim uvjetima (npr. vožnja autocestom s jasno označenim rubovima). Za vrijeme samostalne vožnje, vozaču je dopušteno skrenuti pogled s ceste, međutim, dužan je u svakom trenutku biti spreman preuzeti kontrolu nad vozilom. Problemi na koje nailaze znanstvenici i inženjeri prilikom razvoja autonomnih vozila su fuzija velike količine podataka s raznih senzora, izvođenje algoritama autonomne vožnje u stvarnom vremenu te prijenos komandi na željene aktuatore. Danas se većina takvih problema rješava tehnikama razvijenim u području umjetne inteligencije (engl. *Artificial Inteligence* - AI).

AI je grana računalne znanosti koja se bavi razvojem strojeva koji su u mogućnosti poduzimati radnje koje oponašaju čovjekove kognitivne vještine. Središnja načela umjetne inteligencije uključuju planiranje, učenje, komunikaciju, percepciju te sposobnost pomicanja i manipuliranja objektima [4]. Za izradu ovakvog sustava nije praktično koristiti tehnike klasičnog programiranja temeljenog na pravilima, već se za tu svrhu koriste alati i tehnike razvijene u području strojnog učenja (engl. *Machine Learning* – ML). ML je jedan od osnovnih pristupa AI-a koji omogućuje računalima da uče iz podataka. Umjesto eksplicitnog programiranja svakog koraka, ML algoritmi omogućuju računalima da uče obrasce iz podataka koji su im pruženi i poboljšavaju svoje performanse iz iskustva. Ova sposobnost učenja i poboljšanja iz iskustva slična je načinu na koji ljudi uče i prilagođavaju se tijekom vremena. ML obuhvaća nekoliko pristupa koji se općenito

mogu kategorizirati u tri glavne vrste: nadzirano učenje, nenadzirano učenje i podržano učenje [5]. Podržano učenje (engl. *Reinforcement Learning* - RL) ima poseban pristup usredotočujući se na interakciju između agenta i okoline. Agent uči poduzimati radnje u okolini kako bi maksimizirao kumulativne nagrade. Ova paradigma učenja slična je načinu na koji ljudi i životinje uče putem pokušaja i pogrešaka. Podržano učenje je pronašlo primjene u robotici, računalnim igrama, autonomnim sustavima i problemima optimizacije gdje se optimalan slijed radnji treba naučiti tijekom vremena.

Za potrebe rješavanja problema upravljanja kod autonomne vožnje, podržano je učenje prikladna tehnika ML-a. Za razliku od nadziranog učenja, gdje unaprijed definirani odgovori usmjeravaju treniranje i nenadziranog učenja, koje otkriva obrasce u podacima, podržano učenje je najbolje prilagođeno za scenarije koji zahtijevaju trenutnu i prilagodljivu akciju. Oponaša ljudsko učenje, dopuštajući autonomnim vozilima da uče iz svoje okoline putem pokušaja i pogrešaka što je dobro usklađeno s dinamičnom prirodom vožnje.

U okviru diplomskog rada predložena su dva algoritma za upravljanje autonomnim vozilom pomoću podržanog strojnog učenja. U svrhu razvoja koristio se CARLA (engl. *Car learning to act*) simulator, implementacija sloja otvorenog koda preko *Unreal Engine 4* razvojnog okruženja za 3D grafiku i grafiku u stvarnom vremenu. CARLA simulator pruža informacije o okolini u obliku podataka dobivenih s RGB kamere i drugih senzora. U radu su razvijeni algoritmi upravljanja autonomnim vozilom za određene okoline (primjerice, vožnja autocestom bez prepreka, vožnja autocestom uz prisustvo statičnih ili dinamičnih objekata, promjena trake i sl.). Algoritmi su zasnovani na metodama podržanog učenja, točnije, dubokom Q-učenju (engl. *Deep Q-Learning*) i dubokom determinističkom gradijentu politike (engl. *Deep Deterministic Policy Gradient* – DDPG). Algoritmi su izrađeni u *Python* programskom jeziku koristeći *TensorFlow* razvojni okvir i CARLA Python API.

Ostatak je rada strukturiran na sljedeći način. U drugom je poglavlju detaljnije objašnjeno podržano učenje i njegove primjene u autonomnoj vožnji. Opisana su dva najčešće korištena algoritma u praksi te prednosti i nedostaci pojedinog. Također, analizirana su postojeća rješenja koja se temelje na prijašnje spomenutim algoritmima. U trećem poglavlju dan je opis vlastitih algoritama upravljanja izrađenih korištenjem metoda objašnjenih u drugom poglavlju. Opisani su radni okviri i popratni programi potrebni za postavljanje radnog okruženja CARLA simulatora. Također, dan je prijedlog moguće arhitekture agenta i pojedinih arhitektura korištenih algoritama.

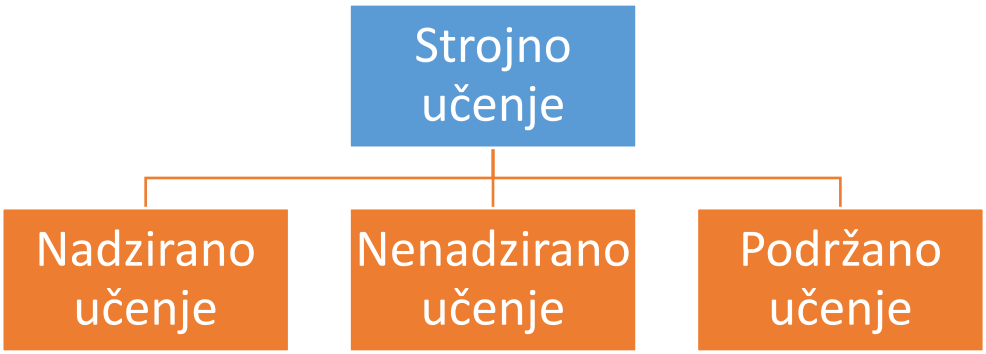
U četvrtom poglavlju predstavljena je evaluacija predloženih algoritama upravljanja vozilom u CARLA simulatoru temeljenih na podržanom učenju.

# 2. PODRŽANO UČENJE I ALGORITMI KORIŠTENI U AUTONOMNOJ VOŽNJI

Autonomno upravljanje vozilom predstavlja značajan izazov za umjetnu inteligenciju i teoriju kontinuiranog upravljanja. Sam čin vožnje najbolje je modelirati kao niz uzastopnih odluka donesenih uz povremene povratne informacije iz okoline. Jedan od načina učenja niza uzastopnih odluka u okviru AI-a može se postići podržanim učenjem. Podržano učenje jedan je od pristupa učenju gdje agent tijekom vremena poboljšava način donošenja odluka kroz iskustvo i povratne informacije iz okoline. Tehnike podržanog učenja pokazale su određeno obećanje u rješavanju složenih problema upravljanja. U idućim potpoglavljima najprije je detaljnije opisan koncept podržanog učenja te njegova usporedba s ostalim tehnikama strojnog učenja. Nakon toga su opisana dva najčešće korištena algoritma podržanog učenja i postojeća rješenja zasnovana na tim algoritmima.

## 2.1. Podržano učenje

Kao što je već spomenuto, postoje tri glavne paradigme strojnog učenja: nadzirano učenje, nenadzirano učenje i podržano učenje (slika 2.1.) [6]. Nadzirano učenje je učenje iz skupa označenih podataka za treniranje koje pruža iskusan vanjski supervizor. Cilj ove vrste učenja je da sustav ekstrapolira ili generalizira svoje odgovore tako da ispravno djeluje i u situacijama koje nisu prisutne u skupu za treniranje. Ovo je važna vrsta učenja, ali sama po sebi nije prikladna za učenje u kojoj agent treba učiti iz interakcije s okolinom. U interaktivnim problemima često je nepraktično dobiti primjere željenog ponašanja koji su ispravni i reprezentativni za sve situacije u kojima agent mora djelovati. U okolinama koje agent do sada nije vidio, gdje bi se očekivalo da je učenje najkorisnije, agent mora biti u stanju učiti iz vlastitog iskustva [7].




**Slika 2.1.** *Paradigme strojnog učenja*

S druge strane, nenadzirano se učenje obično odnosi na pronalaženje strukture skrivene u skupu neoznačenih podataka. Iako bi netko mogao razmišljati o podržanom učenju kao o vrsti učenja bez nadzora jer se ne oslanja na primjere ispravnog ponašanja, podržano učenje pokušava pronaći optimalno vladanje agenta umjesto da pokušava pronaći skrivenu strukturu. Otkrivanje strukture u agentovom iskustvu svakako može biti korisno u podržanom učenju, ali samo po sebi ne rješava problem optimalnog vladanja. Stoga, podržano se učenje smatra trećom paradigmom strojnog učenja uz nadzirano i nenadzirano učenje [8].

Podržano učenje vrsta je strojnog učenja koja agentu omogućuje učenje u interaktivnoj okolini metodom pokušaja i pogrešaka koristeći povratne informacije iz vlastitih radnji i iskustava [9]. Naime, agent treba naučiti koje akcije treba poduzimati kako bi postigao određeni cilj. Agent može biti programski kod (program), robot i sl., odnosno bilo koji sustav koji ima sposobnost donošenja odluka i izvođenja akcija na osnovu informacija iz interaktivne okoline. Interaktivna okolina se odnosi na vanjski sustav s kojim agent komunicira. To je kontekst u kojem agent djeluje, donosi odluke i prima povratne informacije. Problemi koje rješava podržano učenje uključuju učenje što učiniti, tj. kako mapirati situacije u radnje tako da agent postigne određeni cilj. Štoviše, agentu se ne govori koje radnje treba poduzeti, kao u mnogim oblicima strojnog učenja, već umjesto toga mora otkriti koje radnje najviše doprinose postizanju određenog cilja tako što će ih isprobati i dobiti povratnu informaciju iz okoline. Glavni cilj RL-a je maksimizirati numerički signal nagrade. Nagrada je numerički signal koji agent dobiva od okoline nakon poduzimanja određene akcije u danom stanju. Nagrada služi kao povratna informacija agentu, ukazujući na neposrednu korist ili poželjnost radnje koju je upravo izvršio. Nagrade se koriste za usmjeravanje procesa učenja agenta, pomažući mu da odredi koje su akcije povoljne u različitim situacijama. U najizazovnijim slučajevima, radnje mogu utjecati ne samo na trenutnu nagradu, već i na sljedeću situaciju, a time i na sve naredne nagrade [8].

Primjena podržanog učenja proteže se kroz veliki niz grana znanosti i industrija. Primjerice, kod autonomnih se vozila upotrebljava za optimizaciju putanje, planiranja kretanja, optimizacija kontrolera i sl. Također, može se koristiti u financijama za trgovanje dionicama, zdravstvenom sustavu za dijagnozu pacijenata, računalnim igricama za upravljanje različitim botovima, u preporukama vijesti i reklama na internetu, itd. [10].

Za razumijevanje RL-a potrebno je objasniti nekoliko osnovnih pojmova koji se koriste u ovoj vrsti strojnog učenja.

**Agent** – može biti programski kod (program), robot i sl., odnosno bilo koji sustav koji ima sposobnost donošenja odluka i izvođenja akcija na osnovu informacija iz interaktivne okoline.

**Okolina** – vanjski sustav s kojim RL agent komunicira, kontekst u kojem agent djeluje, donosi odluke i prima povratne informacije.

**Nagrada** – numerički signal koji agent dobiva od okoline nakon poduzimanja određene akcije u danom stanju, služi kao povratna informacija agentu, ukazujući na neposrednu korist ili poželjnost radnje koju je upravo izvršio.

**Stanje** – reprezentacija trenutačnog okruženja u kojem se agent nalazi. To su informacije koje agent koristi za donošenje odluka. Stanje može biti jednostavan vektor atributa ili složeniji format, ovisno o problemu.

**Akcija** – potez koji agent poduzima u okruženju kako bi utjecao na svoje stanje i pokušao dobiti veću nagradu. Akcije mogu biti diskretne (npr. kretanje lijevo/desno) ili kontinuirane (npr. podešavanje brzine).

**Tranzicija** – proces prelaska iz jednog stanja u drugo stanje kao rezultat agenta koji poduzima određenu radnju u interaktivnom okruženju. Tranzicija je temeljni koncept u RL-u jer ukazuje na dinamiku načina na koji akcije agenta utječu na okolinu i dovode do promjena u njezinom stanju.

Jedan od izazova koji se javlja u podržanom učenju je kompromis između istraživanja i eksploatacije (engl. *explore vs. exploit*). Da bi agent dobio mnogo nagrada, on mora preferirati radnje koje je isprobao u prošlosti i za koje se pokazalo da su učinkovite u stvaranju nagrade (eksploatacija). Ali, da bi otkrio takve akcije, on mora isprobati radnje koje nije prije odabrao (istraživanje). Agent mora iskoristiti znanja naučena iz iskustva kako bi odabrao najpoželjniju akciju koja donosi najveću nagradu, ali također mora istraživati kako bi otkrio potencijalne akcije koje u budućnosti donose veću vrijednost nagrade. Dilema je u tome što se ni istraživanje ni eksploatacija ne mogu izvoditi isključivo bez neuspjeha u izvršavanju zadataka. Agent mora isprobati niz radnji i postupno favorizirati one koje se čine najboljima. Dilemu istraživanja i eksploatacije matematičari su intenzivno proučavali desetljećima, ali još uvijek nije u potpunosti riješena [11].

Zadaci RL-a se mogu klasificirati u dvije glavne kategorije: deterministički zadaci i stohastički zadaci. Ove se kategorije odnose na prirodu interakcija između RL agenta i okruženja, posebice kako okolina reagira na agentove akcije. U determinističkim zadacima ponašanje okoline potpuno je predvidljivo i dosljedno. To znači da će s obzirom na određeno stanje i radnju okolina uvijek

prijeći u isto sljedeće stanje, a povezane nagrade bit će iste svaki put kada se ta radnja poduzme. U stohastičkim zadacima, ponašanje okoline uključuje slučajnost ili neizvjesnost. To znači da čak i ako agent poduzme istu radnju u istom stanju više puta, može naići na različite ishode u smislu sljedećih stanja i nagrada. Odgovori okoline mogu slijediti distribuciju vjerojatnosti, što dovodi do različitih ishoda za istu radnju u istom stanju. Na stohastičkom zadatku, svaka se radnja mora isprobati mnogo puta kako bi se dobila pouzdana procjena očekivane nagrade.

Jedan od osnovnih sastavnih dijelova podržanog učenja je Bellmanova jednadžba. Jednadžba nam pokazuje kakvu dugoročnu nagradu možemo očekivati, s obzirom na našu trenutnu situaciju i pod pretpostavkom da radimo najbolje što možemo u određenoj točki i svakom sljedećem koraku. Bellmanova jednadžba može se koristiti kako bismo provjerili jesmo li postigli cilj jer je glavna svrha podržanog učenja maksimizirati dugoročnu nagradu. Vrijednost sadašnjeg stanja otkriva se kada se odabere optimalni način djelovanja. Za deterministične situacije, Bellmanova jednadžba dana je izrazom (2-1) [12].

$$V(s) = max_a(R(s,a) + \gamma V(s')) \tag{2-1}$$

gdje je:

- $V(s)$ – vrijednost u danom stanju
- *max* – max funkcija koja odabire akciju koja maksimizira nagradu
- $s$ – dano stanje
- $s'$ – stanje u idućem koraku
- $a$ – akcija poduzeta u stanju $s$
- $R(s,a)$ – funkcija koja računa nagradu na temelju odabrane akcije i trenutnog stanja
- $\gamma$ – faktor smanjivanja

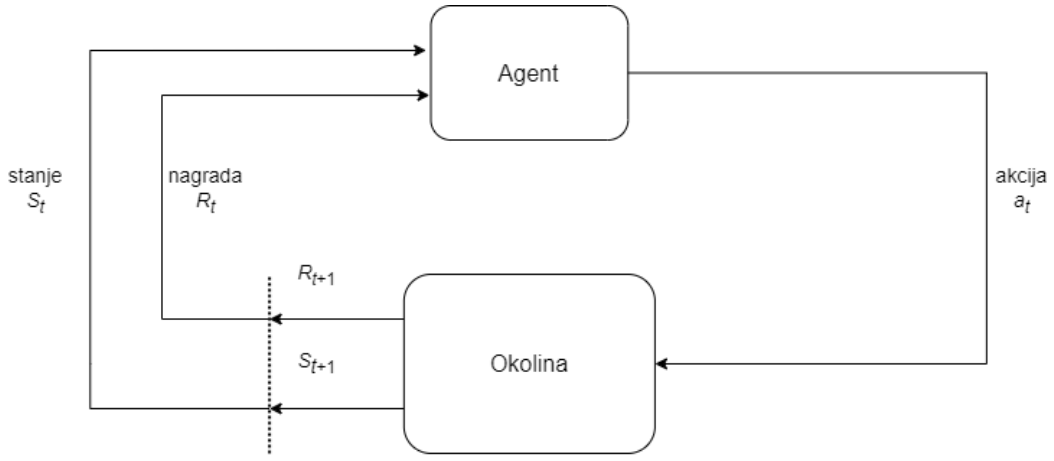
Faktor smanjivanja je hiperparametar koji se može modificirati kako bi se istaknula dugoročna nagrada ili kako bi se model usredotočio na najbolje kratkoročno rješenje.

Bellmanova jednadžba je rekurzivna funkcija budući da poziva samu sebe ($s'$ je stanje u sljedećem koraku). Može se činiti kontradiktornim da se funkcija izračunata unutar trenutnog koraka odnosi na budući korak, a ne na prethodni, tj. $V(s)$ je vrijednost u danom stanju, ali koristi $s'$ prilikom izračuna. To je zato što vrijednost radnji možemo izračunati tek nakon što smo dosegli terminalno stanje, tj. stanje u kojem agent više ne poduzima akcije niti prima nagrade. Terminalna stanja obično označavaju završetak zadatka ili epizode. Kada se agent nađe u terminalnom stanju,

proces se obrće primjenjujući faktor smanjivanja i dodajući funkciju nagrađivanja na svakom koraku dok se ne dođe do početnog koraka.

Formalno, problemi podržanog učenja se zapisuju kao matematička formulacija Markovljevog procesa donošenja odluke (eng. *Markov Decision Process* - MDP) koja zadovoljava Markovljevo svojstvo koje kaže da stohastički proces ima ovo svojstvo ako buduća raspodjela vjerojatnosti procesa, za dano trenutno stanje i sva prošla stanja, ovisi samo o trenutnome stanju i niti o jednom drugome prethodnom [13]. U MDP-u postoji entitet koji donosi odluke, nazvan agentom, koji je u interakciji s okolinom u kojoj je postavljen. Te se interakcije događaju sekvencijalno tijekom vremena. U svakom vremenskom koraku, agent će dobiti neki prikaz stanja okoline. S obzirom na ovu reprezentaciju, agent odabire akciju koju treba poduzeti. Agent se zatim prebacuje u novo stanje i dobiva nagradu kao posljedicu prethodne akcije. Kroz ovaj proces, cilj agenta je maksimizirati ukupan iznos nagrada koje prima od poduzimanja radnji u danim stanjima. To znači da agent želi maksimizirati ne samo trenutnu nagradu, već i kumulativne nagrade koje dobiva tijekom vremena [14].

U MDP-u postoji skup stanja $S$, skup akcija $A$ i skup nagrada $R$. Pretpostavit ćemo da svaki od tih skupova ima konačan broj elemenata. U svakom vremenskom koraku $t = 0, 1, 2, \ldots$ agent prima neki prikaz stanja okoline $S_t \in S$. Na temelju ovog stanja agent odabire akciju $A_t \in A$. Ovo nam daje par stanje-akcija ($S_t$, $A_t$). Vrijeme se zatim povećava na sljedeći vremenski korak $t$+1, a okolina prelazi u novo stanje $S_{t+1} \in S$. U to vrijeme agent prima numeričku nagradu $R_{t+1} \in R$ za akciju $A_t$ poduzetu u stanju $S_t$. Proces primanja nagrade možemo zamisliti kao proizvoljnu funkciju $f$ koja preslikava parove stanja-akcija u nagrade. U svakom trenutku $t$ imamo: $f(S_t, A_t) = R_{t+1}$. Putanja koja predstavlja sekvencijalni proces odabira akcije iz stanja, prijelaza u novo stanje i primanja nagrade predstavljena je slikom 2.2. [15].




**Slika 2.2.** *MDP dijagram*

Markovljev proces odlučivanja pruža matematički okvir za modeliranje donošenja odluka u situacijama u kojima su ishodi djelomično slučajni, a djelomično pod kontrolom donositelja odluka [16]. Uzmimo za primjer da je agent jedna figurica u igri „*Monopoly*“ koja se nalazi na igraćoj ploči. Agent se može kretati prema naprijed u ovisnosti o vrijednosti dobivenoj bacanjem igraće kocke s šest strana. Dakle, postoji element slučajnosti u okolini koji se mora uzeti u obzir. Gledajući jednadžbu (2-1), sada kada je prisutna slučajnost, ne zna se u kojem će $s'$ stanju agent pronaći. Umjesto toga, mora se koristiti očekivana vrijednost sljedećeg stanja. Kako bi se izračunala očekivana vrijednost sljedećeg stanja, pomnožilo bi se svih šest mogućih stanja s njihovom vjerojatnostima [17]. Pošto je vjerojatnost dobitka određene vrijednosti na kocki uniformna i iznosi $\frac{1}{6}$, očekivana vrijednost sljedećeg stanja je jednaka:

$$V(s') = \frac{1}{6}V(s_1') + \frac{1}{6}V(s_2') + \frac{1}{6}V(s_3') + \frac{1}{6}V(s_4') + \frac{1}{6}V(s_5') + \frac{1}{6}V(s_6') \tag{2-2}$$

Sada se Bellmanova jednadžba za nedeterminističke okoline može napisati u obliku:

$$V(s) = max_a(R(s,a) + \gamma \sum_{s'} P(s'|s,a)\, V(s')) \tag{2-3}$$

gdje je:

- $P(s'|s,a)$ – vjerojatnost prijelaza iz stanja $s$ u $s'$ uz odabranu akciju $a$

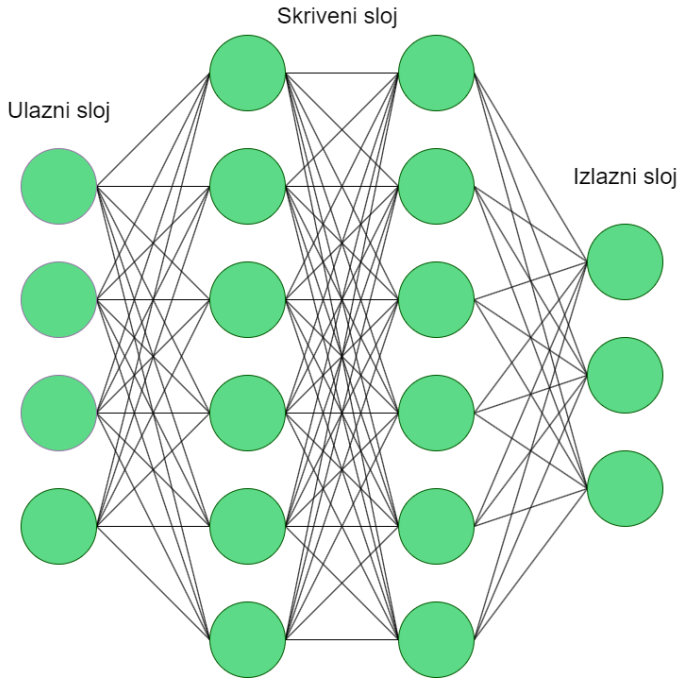
U nastavku poglavlja će se dano je više detalja o dubokom učenju i konkretnim algoritmima zasnovanima na podržanom učenju koji su korišteni za realizaciju rješenja ovog diplomskog rada te postojeća rješenja upravljanja vozilom koristeći navedene algoritme.

## 2.2. Duboko učenje

Duboko učenje je grana strojnog učenja koja se temelji na umjetnim neuronskim mrežama (engl. *Artificial Neural Networks* – ANN). Glavna značajka dubokog učenja je sposobnost učenja složenih obrazaca i odnosa unutar podataka. Duboko se učenje temelji na umjetnim neuronskim mrežama s dva ili više skrivena sloja koje su poznate i kao duboke neuronske mreže (engl. *Deep Neural Networks* – DNN). Ove neuronske mreže nadahnute su strukturom i funkcijom bioloških neurona ljudskog mozga i dizajnirane su za učenje iz velikih količina podataka. Duboko je učenje sve popularnije posljednjih godina zbog napretka u procesorskoj snazi i dostupnosti velikih skupova podataka. Treniranje DNN-a obično zahtijeva veliku količinu podataka i računalnih resursa. Međutim, dostupnost računarstva u oblaku i razvoj specijaliziranog sklopovlja, kao što su

grafičke procesorske jedinice (engl. *Graphics Processing Unit* – GPU), olakšali su treniranje dubokih neuronskih mreža [18].

ANN koristi slojeve međusobno povezanih čvorova (neurona) koji rade zajedno na obradi i učenju iz ulaznih podataka. U potpuno povezanoj dubokoj neuronskoj mreži (slika 2.3.) postoji ulazni sloj i jedan ili više skrivenih slojeva povezanih jedan za drugim [14]. Tok podataka u neuronskoj mreži uključuje sekvencijalni prolaz ulaznih podataka kroz slojeve neurona. Počevši od ulaznog sloja koji prima neobrađene podatke, informacije se progresivno transformiraju kroz skrivene slojeve koristeći težinske veze i nelinearne aktivacijske funkcije. Prilagodbe težina (engl. *weights*) i pomaka (engl. *biases*) omogućuju mreži da nauči i poboljša svoja predviđanja. Ove transformacije završavaju u izlaznom sloju, gdje se generiraju konačna predviđanja ili izlazi. Ovaj iterativni proces prosljeđivanja, popraćen ažuriranjem težina kroz učenje, omogućuje neuronskim mrežama „shvaćanje“ zamršenih obrazaca i odnosa unutar podataka i stvaranje informiranih predviđanja [19].




**Slika 2.3.** *Primjer potpuno povezane neuronske mreže s dva skrivena sloja*

Sustavi autonomne vožnje temelje svoj rad na višestrukim zadacima na razini percepcije koji su postigli visoku preciznost zahvaljujući arhitekturama dubokog učenja. Agent mora naučiti nove konfiguracije okoline, kao i predvidjeti optimalnu odluku u svakom trenutku tijekom vožnje u svojoj okolini. Ovo predstavlja visokodimenzionalni prostor stanja s obzirom na broj jedinstvenih konfiguracija pod kojima se agent i okolina promatraju. U svim stanjima se nastoji riješiti sekvencijalni problem odlučivanja, koji je formaliziran pod klasičnim pretpostavkama podržanog

učenja gdje se od agenta traži da uči i predstavlja svoje okruženje, kao i da djeluje optimalno u svakom trenutku [20]. Zbog velike kompleksnosti okruženja u kojima se autonomna vozila mogu naći, za treniranje ovakvih modela se najčešće koriste algoritmi dubokog učenja.

## 2.3. Algoritmi dubokog podržanog učenja

Duboko podržano učenje je kombinacija podržanog učenja s tehnikama dubokog učenja za rješavanje izazovnih sekvencijalnih problema donošenja odluka. Korištenje dubokog učenja najkorisnije je u problemima s visokodimenzionalnim prostorom stanja. To znači da uz duboko učenje, podržano učenje može riješiti kompliciranije zadatke s malo ili bez predznanja o svojoj okolini zbog svoje sposobnosti učenja različitih razina apstrakcija iz podataka. Međutim, da bi uspješno koristili podržano učenje u situacijama koje se približavaju složenosti stvarnog svijeta, agenti se suočavaju s teškim zadatkom: moraju kreirati učinkovite reprezentacije okoline iz visokodimenzionalnih senzorskih ulaza i koristiti ih za generalizaciju prošlih iskustava na nove situacije. To omogućuje strojevima da oponašaju neke ljudske sposobnosti rješavanja problema, čak i u visokodimenzionalnom prostoru stanja, što je prije samo nekoliko godina bilo teško zamisliti [21].

Algoritmi podržanog učenja se dijele na dva glavna tipa: algoritmi bazirani na modelu (engl. *Model-Based*) i algoritmi bez modela (engl. *Model-Free*). Algoritam temeljen na modelu koristi funkciju prijelaza i nagrađivanja za procjenu optimalne politike. U ovoj vrsti algoritma agent ima pristup modelu okruženja, tj. radnji koje je potrebno izvršiti za prelazak iz jednog stanja u drugo, priloženim vjerojatnostima i priloženim odgovarajućim nagradama. Algoritmi bazirani na modelu se koriste u scenarijima u kojima imamo potpuno znanje o okolini i kako ona reagira na različite radnje [22]. Primjer ovakvog algoritma je algoritam dinamičkog programiranja (engl. *Dynamic Programming Algorithm*).

Algoritmi bez modela pronalaze optimalnu politiku s vrlo ograničenim znanjem o dinamici okoline. Ovi algoritmi nemaju funkciju prijelaza/nagrađivanja za procjenu najbolje politike, već procjenjuju optimalnu politiku izravno iz iskustva, tj. interakcije između agenta i okoline bez ikakve naznake o funkciji nagrađivanja. Učenje bez modela je najbolje primijeniti u scenarijima koji uključuju nepotpune informacije o okolini. Na primjer, autonomna vozila imaju dinamičnu okolinu s promjenjivim prometnim uvjetima, skretanjima s rute itd. U takvim scenarijima algoritmi bez modela nadmašuju druge tehnike svojim performansama [22]. Primjer algoritma bez modela je algoritam dubokog determinističkog gradijenta politike.

### 2.3.1. Duboko Q učenje

Do sada se govorilo o vrijednosti *V*(*s*) dobivenoj prelaskom iz određenog stanja *s* u novo stanje $s'$ uz odabranu akciju $a$. Sada umjesto da se promatra vrijednost svakog stanja *V*(*s*), promatra se vrijednost svakog para stanje-akcija, koja se označava s *Q*(*s*,*a*). Promatrajući jednadžbu (2-3), izvođenjem akcije prvo što se dobiva je nagrada *R*(*s*,*a*). Sada je agent u sljedećem stanju $s'$, a budući da agent može završiti u nekoliko stanja, dodajemo vrijednost sljedećeg stanja koja je očekivana vrijednost sljedećeg stanja izražena s $P(s'|s,a)V(s')$. Jednadžba za *Q(s,a)* je sljedeća:

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a)\, V(s') \tag{2-4}$$

Ono što se može primijetiti gledajući jednadžbu (2-4) je da je vrijednost *Q*(*s*,*a*) jednaka izrazu pod max funkcijom Bellmanove jednadžbe (2-3). Intuitivno se može reći da je vrijednost *V*(*s*) u Bellmanovoj jednadžbi maksimum od svih mogućih *Q*(*s*,*a*) vrijednosti. Time se vrijednost *V*(*s*′) u jednadžbi (2-4) može napisati na sljedeći način:

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a)\, max_{a'} Q(s',a') \tag{2-5}$$

Time dobivamo rekurzivnu formulu za računanje Q-vrijednosti.

Za nedeterministička okruženja može biti vrlo teško izračunati vrijednost svakog stanja *V*(*s*′), no za ova računanja se koristi koncept vremenske razlike. Radi jednostavnosti će se ovaj koncept objasniti na determinističkoj Bellmanovoj jednadžbi:

$$Q(s,a) = R(s,a) + \gamma max_{a'} Q(s',a') \tag{2-6}$$

Dakle, zna se da prije nego što agent poduzme akciju, akcija ima Q-vrijednost *Q*(*s*,*a*). Nakon poduzimanja akcije znamo koju je nagradu $R(s,a)$ dobio agent i koja je vrijednost novog stanja $\gamma max_{a'} Q(s',a')$. Vremenska razlika *TD*(*s*,*a*) je definirana kao:

$$TD(s,a) = R(s,a) + \gamma max_{a'} Q(s',a') - Q(s,a) \tag{2-7}$$

$R(s,a) + \gamma max_{a'} Q(s',a')$ je vrijednost koja se dobiva nakon poduzimanja akcije, a vrijednost $Q_{t\text{-}1}(s,a)$ predstavlja prethodnu Q-vrijednost. Idealno bi ove dvije vrijednosti trebale biti iste jer je vrijednost $R(s,a) + \gamma max_{a'} Q(s',a')$ formula za računanje *Q*(*s*,*a*) (2-6). Međutim, te vrijednosti možda neće biti iste zbog nasumičnosti koja postoji u okruženju. Postavlja se pitanje je li došlo do razlike između ovih vrijednosti u vremenu. Vremenska se razlika definira na sljedeći način:

$$Q(s,a) = Q(s,a) + \alpha TD(s,a) \tag{2-8}$$

gdje je:

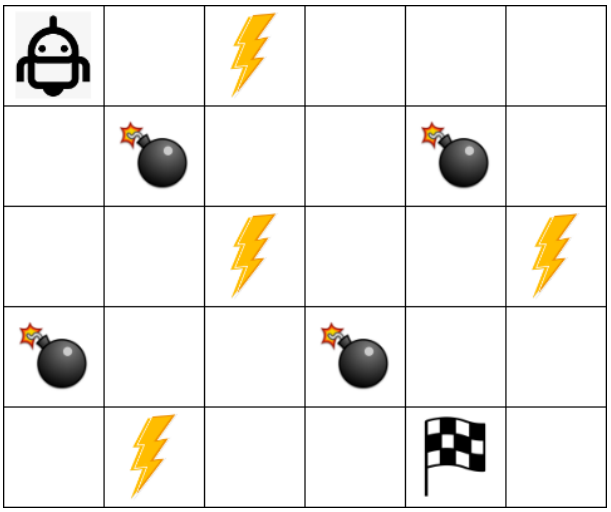
- α – stopa učenja

Uzima se prijašnja Q-vrijednost $Q(s,a)$ i dodaje se vremenska razlika pomnožena sa stopom učenja kako bi se dobila nova Q-vrijednost. Ova jednadžba pokazuje kako se Q-vrijednosti ažuriraju tijekom vremena. Jednadžba Q-učenja nakon uvrštavanja vrijednosti *TD* u jednadžbu (2-8) izgleda ovako:

$$Q(s,a) = Q(s,a) + \alpha(R(s,a) + \gamma max_{a'} Q(s',a') - Q(s,a)) \tag{2-9}$$

Radi boljeg razumijevanja, predstavit će se Q-učenje na jednostavnom primjeru. Neka je agent robot koji ima mogućnost kretanja u 2D prostoru. Robot je postavljen na početnu poziciju u okolinu strukturiranu kao polje s pet redova i šest stupaca (ukupno 30 pozicija). Na određenim su pozicijama postavljene mine, te ako se robot nađe na jednoj od tih pozicija, igra je završila. Također, na određenim pozicijama postavljena je snaga te ako se robot nađe na jednoj od tih pozicija, robot dobiva dodatnu nagradu za svoju akciju. Akcije koje robot može odraditi su kretanje naprijed, nazad, lijevo i desno. Cilj robota je pomicanjem jednu po jednu poziciju stići do određene krajnje pozicije u najkraćem mogućem vremenu. Prikaz okruženja dan je na slici 2.4. Sustav bodovanja, tj. nagrađivanja je sljedeći.

1. Robot gubi 1 bod na svakom koraku. To je učinjeno kako bi robot prošao najkraći put i stigao do cilja što je brže moguće.
2. Ako robot stane na minu, gubitak bodova je 100 i igra završava.
3. Ako robot dobije snagu ⚡, dobiva 1 bod.
4. Ako robot postigne krajnji cilj, dobiva 100 bodova.



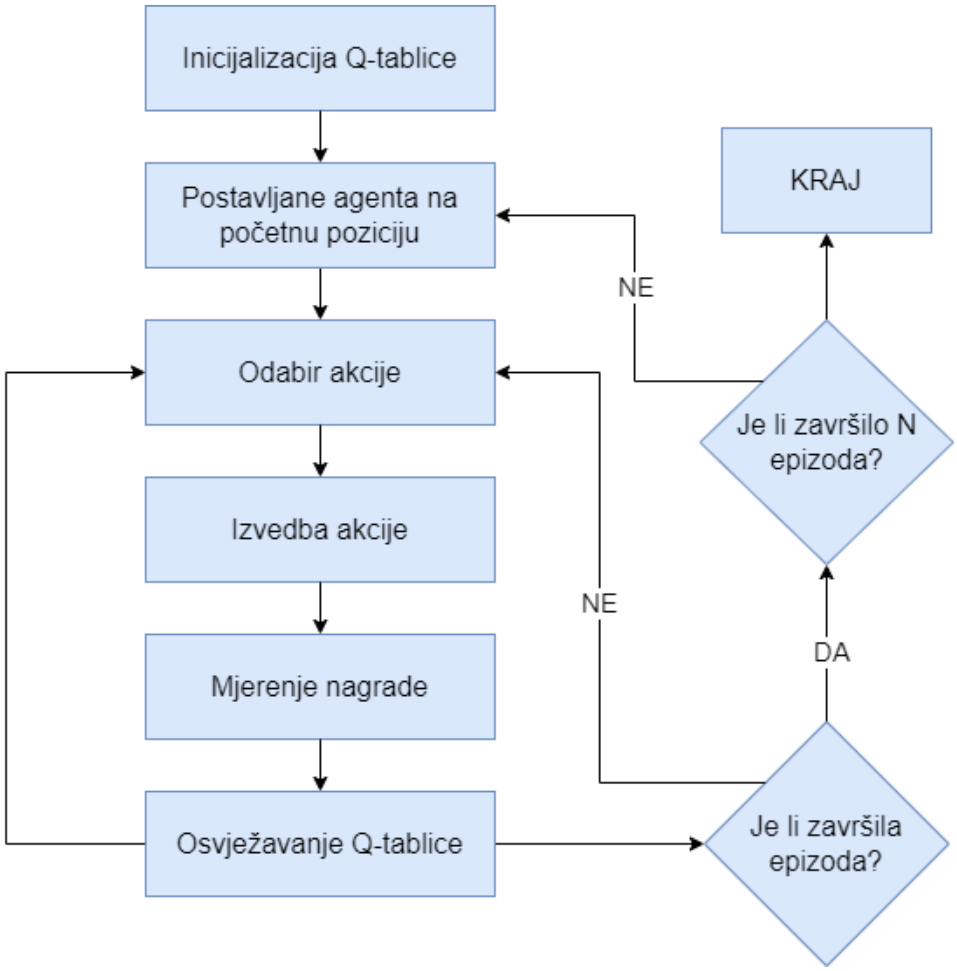
**Slika 2.4**. *Igra robota u labirintu*

Okruženje se modelira pomoću Q-tablice. Q-tablica je naziv za jednostavnu preglednu tablicu u kojoj se izračunavaju maksimalne očekivane buduće nagrade za akciju u svakom stanju. U osnovi, pomoću ove se tablice određuje koju će akciju agent poduzeti s obzirom na stanje u kojemu se nalazi. U Q-tablici, stupci su akcije, a redovi su stanja (tablica 2.1.).

Tablica 2.1. Q-tablica za igru robota u labirintu

| Akcija | ↑ | → | ↓ | ← |
|---|---|---|---|---|
| Početak | | | | |
| Prazno | | | | |
| Snaga | | | | |
| Mina | | | | |
| Kraj | | | | |

Svaka vrijednost Q-tablice će biti najveća očekivana buduća nagrada koju će robot dobiti ako poduzme tu radnju u tom stanju (Q-vrijednost). Ovo je iterativni proces jer moramo osvježiti Q-tablicu pri svakoj iteraciji. Proces algoritma Q-učenja dan je slikom 2.5. Svaki od obojenih okvira je jedan korak algoritma.




**Slika 2.5**. *Koraci algoritma Q-učenja*

U tablici 2.2. prikazana je inicijalizirana Q-tablica.

Tablica 2.2. Inicijalizirana Q-tablica

| Akcija | ↑ | → | ↓ | ← |
|---|---|---|---|---|
| Početak | 0 | 0 | 0 | 0 |
| Prazno | 0 | 0 | 0 | 0 |
| Snaga | 0 | 0 | 0 | 0 |
| Mina | 0 | 0 | 0 | 0 |
| Kraj | 0 | 0 | 0 | 0 |

Odabir i izvedba akcija su koraci koji se odvijaju sve dok se ne ispuni određeni kriterij. U primjeru robota bi to značilo da se ovi koraci ispunjavaju sve dok robot ne stane na poziciju s minom ili stigne do odredišnog cilja. Ove pozicije definiraju terminalno stanje i signaliziraju kraj jedne epizode. Znači, jedna epizoda počinje s inicijalnim stanjem i završava s terminalnim stanjem.

Treniranje se može provesti na proizvoljnom broju epizoda *N*. Na bilo kojoj poziciji se odabire akcija ($a$) u stanju ($s$) na temelju Q-tablice. Ali, kao što je ranije spomenuto, kada epizoda inicijalno započne, svaka Q-vrijednost je nula. Sada se dolazi do ranije spomenutog koncepta istraživanja i eksploatacije. U Q-učenju se obično koristi ε-pohlepno istraživanje. Parametar epsilon ε definira s kolikom vjerojatnošću će agent izabrati nasumičnu akciju. Robot ne zna ništa o svojoj okolini te će u početku epsilon vrijednosti biti veće. Robot će istraživati okolinu i nasumično birati akcije. Kako robot bude istraživao okolinu, epsilon vrijednosti se smanjuju i robot počinje eksploatirati okolinu. Tijekom procesa istraživanja, robot postupno postaje sigurniji u procjeni Q-vrijednosti. U danom primjeru, robot izabire nasumičnu akciju i događa se pomak u desno. Sada možemo ažurirati Q-vrijednosti pomoću Bellmanove jednadžbe (2-9).

$$Q_t(početak, desno) = Q_{t-1}(početak, desno) + \alpha(R(početak, desno) + \gamma \max\big(Q'(prazno, dolje), Q'(prazno, lijevo), Q'(prazno, desno)\big) - Q(početak, desno)$$

Za $\alpha = 0.1$ i $\gamma = 0.99$ vrijedi:

$$Q_t(početak, desno) = 0 + 0.1(-1 + 0.99 * 0 - 0)$$

$$Q(početak, desno) = -0.1$$

Sada kada je izračunata odgovarajuća vrijednost, osvježava se Q-tablica (tablica 2.3.).

Tablica 2.3. Prvi korak Q-učenja

| Akcija | ↑ | → | ↓ | ← |
|---|---|---|---|---|
| Početak | 0 | -0.1 | 0 | 0 |
| Prazno | 0 | 0 | 0 | 0 |
| Snaga | 0 | 0 | 0 | 0 |
| Mina | 0 | 0 | 0 | 0 |
| Kraj | 0 | 0 | 0 | 0 |

Ovaj se proces ponavlja iznova sve dok epizoda ne dođe do terminalnog stanja. Zatim se prelazi na sljedeću epizodu. Agent se postavlja na inicijalnu poziciju te algoritam nastavlja s radom. Nakon završenih *N* epizoda, učenje se zaustavlja. Rezultat učenja je Q-tablica s Q-vrijednostima temeljenima na agentovom iskustvu. Primjer kako bi mogla izgledati Q-tablica za ovu igru nakon 1000 epizoda dan je tablicom 2.4.

Tablica 2.4. Primjer rezultirajuće Q-tablice nakon 1000 provedenih epizoda

| Akcija | ↑ | → | ↓ | ← |
|---|---|---|---|---|
| Početak | 0 | -21.39 | -24.69 | 0 |
| Prazno | -32.71 | -29.73 | -44.66 | -35.91 |
| Snaga | 0.81 | 7.44 | 37.88 | 1.0 |
| Mina | 0 | 0 | 0 | 0 |
| Kraj | 0 | 0 | 0 | 0 |

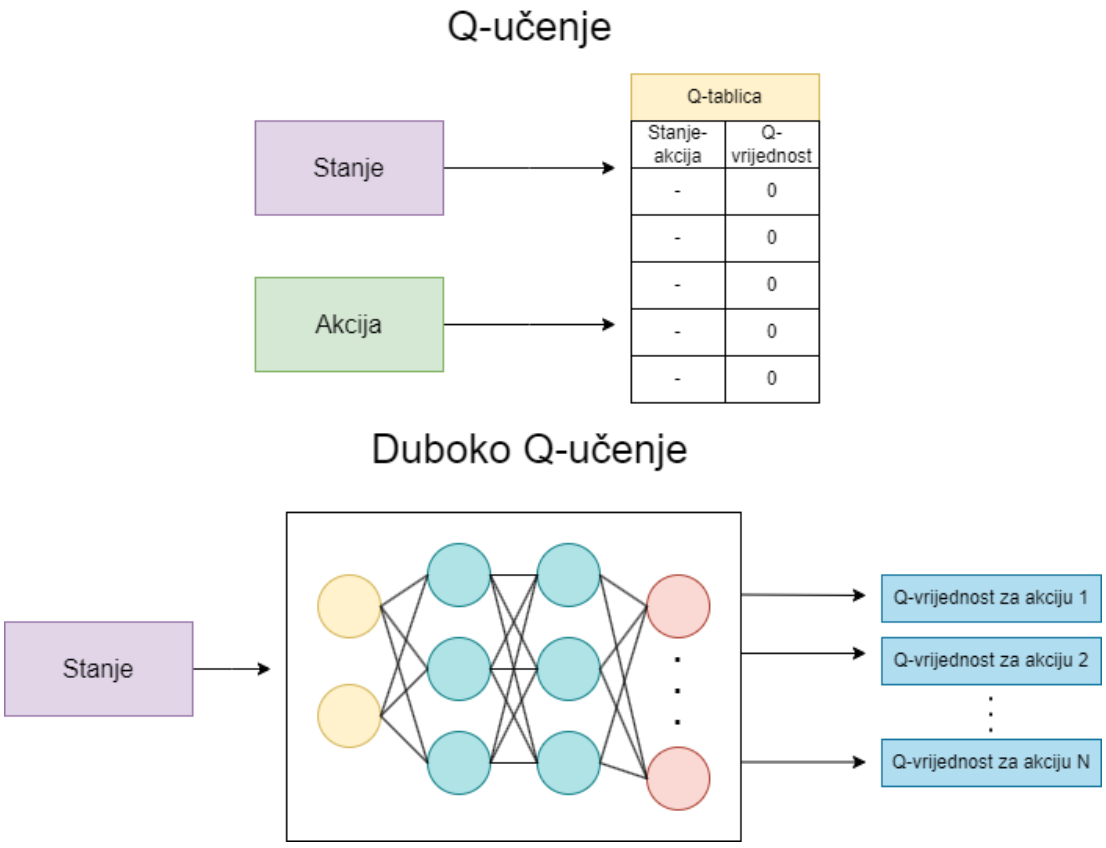
Gledajući tablicu 2.4. možemo vidjeti koje su akcije najpovoljnije u kojem stanju. Npr. za početno stanje, pošto se robot nalazi na rubu i ne može ići ni lijevo ni gore, uspoređuju se vrijednosti za kretanje u desno i kretanje prema dolje. Pošto je Q-vrijednost veća za akciju kretanja u desno, za ovo stanje se bira ta akcija. Stanja kada robot stane na poziciju s minom ili krajnjim ciljem su terminalna te se ni jedna akcija dalje ne izvodi. Stoga su Q-vrijednosti u ovim stanjima jednaka nuli. Ovo je bio primjer jedne deskriptivne tablice jer su oznake u prvom stupcu opisane riječima kako bi tumačenje okruženja bilo intuitivnije te kako bi se primjer lakše objasnio. Kada bi robot u primjeru izabirao akcije prema tablici 2.4., završio bi na mini na poziciji (1,4). Umjesto da se svaki tip pozicije gleda generalno, najbolje bi bilo da su u prvom stupcu definirane sve moguće pozicije u kojima se agent može pronaći ((0,0), (0,1), ... ,(4,5)). Algoritam bi se proveo na isti način, ali rezultati bi bili točniji. Incijalizirana tablica strukturirana na ovaj način bi izgledala kao tablica 2.5.

Tablica 2.5. Inicijalizirana Q-tablica s svim mogućim stanjima

| Akcija | ↑ | → | ↓ | ← |
|---|---|---|---|---|
| (0,0) | 0 | 0 | 0 | 0 |
| (0,1) | 0 | 0 | 0 | 0 |
| (0,2) | 0 | 0 | 0 | 0 |
| .<br>.<br>. | | | | |
| (4,5) | 0 | 0 | 0 | 0 |

Q-učenje dobro funkcionira kada imamo relativno jednostavno okruženje. Međutim, kada broj stanja i radnji koje agent može poduzeti postane značajno veći, umjesto tablica se koriste duboke

neuronske mreže za aproksimaciju. Ova vrsta učenja se naziva duboko Q-učenje. Stanje je dano kao ulaz u neuronsku mrežu, a Q-vrijednosti svih mogućih akcija su generirane i dane kao izlaz iz mreže. Usporedba Q-učenja i dubokog Q-učenja je dana slikom 2.6. [23].




**Slika 2.6.** *Usporedba Q i dubokog Q-učenja*

Važno je napomenuti da duboko Q-učenje uključuje dvije neuronske mreže: Q-mrežu (engl. *Online Network*) koja se trenira i zasebnu ciljnu mrežu (engl. *Target Network*). Ciljna mreža je kopija originalne mreže koja se ne ažurira toliko često koliko originalna NN. Ciljna mreža koristi se za izračunavanje ciljne Q-vrijednosti u svrhu stabilnosti, budući da pruža fiksnu referencu tijekom treninga. Parametri ciljne mreže povremeno se ažuriraju kako bi odgovarali onima Q-mreže kako bi se pratile promjene u procesu učenja. U primjeru robota u labirintu, neuronska mreža će predvidjeti četiri Q-vrijednosti za akcije: gore, dolje, lijevo, desno. Zatim se uzimaju ove četiri vrijednosti i uspoređuje ih se sa predviđenim vrijednostima ciljne mreže. Drugim riječima, provjeravaju se $Q_1$ i $Q - Q_{cilj1}$, $Q_2$ *i* $Q - Q_{cilj2}$, itd. gdje $Q_n$ predstavlja predviđenu Q-vrijednost Q-mreže, Q predstavlja trenutnu Q-vrijednost, a $Q_{cilj_n}$ predstavlja predviđenu vrijednost ciljne Q-mreže. NN rade na način da ažuriraju svoje težine, tako da se jednadžba vremenske razlike (2-7) mora prilagoditi. Dakle, izračunati će se gubitak uzimajući zbroj razlike kvadrata Q-vrijednosti i njihovih predviđenih Q-vrijednosti ciljne funkcije:

$$L = \sum (Q - Q_{cilj})^2 \tag{2-10}$$

Koristi se *backpropagation* algoritam kako bi se ažurirale težine mreže te minimizirao gubitak.

Treniranje Q-mreže je sustavan proces koji osposobljava agenta za optimalno donošenje odluka u dinamičnim okruženjima. Ulazni sloj u Q-mrežu mogu biti npr. elementi slike dobivene s prednje kamere vozila gdje svaki neuron predstavlja jedan element slike. Na početku se Q-mreža inicijalizira s početnim težinama, dok ciljna mreža preslikava njezinu strukturu, osiguravajući stabilnost tijekom učenja. Zatim se agent postavlja u početno stanje. Odabire se akcija na temelju trenutnog stanja, uključujući strategiju istraživanja kako bi se uravnotežilo isprobavanje novih radnji s eksploatacijom poznatih. Promatra se rezultirajuće stanje akcije i dobivena nagrada te se obično vrijednosti ($s$,$a$,$s'$,$R$($s$,$a$)) spremaju u pričuvnu memoriju. Koristi se Bellmanova jednadžba za izračunavanje Q-vrijednosti, koja predstavlja očekivanu kumulativnu nagradu za par stanje-radnja. Zatim se prilagođavaju težine Q-mreže minimiziranjem razlike između predviđenih i ciljnih Q-vrijednosti putem *backpropagation* algoritma. Povremeno se ažuriraju parametri ciljne mreže. Ovi koraci se ponavljaju $N$ epizoda sve dok se treniranje ne završi.

Jedna stvar koja može dovesti do toga da agent pogrešno razumije okolinu su uzastopna međuovisna stanja koja su vrlo slična. Na primjer, ako se razvija algoritam upravljanja autonomnim vozilom, a prvi dio ceste je samo ravna cesta bez skretanja, agent možda neće naučiti koje akcije odabrati kada naiđe na zavoj. Kako bi se spriječila ova vrsta problema, koristi se koncept nazvan ponavljanje iskustva. Početna iskustva ne prolaze odmah kroz neuronsku mrežu, već agent sprema ta iskustva u memoriju. Pod iskustvom se obično misli na parove stanje-akcija i dobivena nagrada u tom stanju. Tek kada agent dosegne određeni broj iskustava spremljenih u memoriju, tek će onda učiti iz niza tih iskustava. Iz tih iskustava, agent nasumično odabire jednoliko raspodijeljeni uzorak (engl. *batch*) iz ove serije i uči iz toga. Nasumičnim uzorkovanjem iz iskustava se razbija pristranost koja je možda proizašla iz sekvencijalne prirode određenog okruženja, npr. vožnja po ravnoj cesti.

### 2.3.2. Duboki deterministički gradijent politike

Duboko Q-učenje radi u diskretnom akcijskom prostoru, a DPG ga proširuje na kontinuirani akcijski prostor dok uči deterministička politiku [24]. Budući da je to algoritam izvan politike (engl. *off-policy*), koristi dvije odvojene politike za istraživanje i ažuriranje, stohastička politika ponašanja za istraživanje i deterministička politika za ciljno ažuriranje [25].

Algoritmi gradijenta politike možda su najpopularnija klasa algoritama za podržano učenje s kontinuiranim akcijskim prostorom. Politika se koristi za odabir radnji u MDP-u. Općenito, politika je stohastička i označava se s $\pi_\theta : \boldsymbol{S} \rightarrow P(\boldsymbol{A})$, gdje je $\boldsymbol{S}$ prostor stanja, $\boldsymbol{A}$ prostor akcija, $P(\boldsymbol{A})$ skup mjera vjerojatnosti nad $\boldsymbol{A}$ i $\theta \in \mathbf{R}^n$ je vektor od $n$ parametara, a $\pi_\theta(a_t|s_t)$ je uvjetna gustoća vjerojatnosti na $a_t$ povezana s politikom. Agent koristi svoju politiku za interakciju s MDP-om kako bi dobio parove stanje-akcija i nagrada. Vrijednosne funkcije definirane su kao očekivana ukupna umanjena nagrada $V^\pi(s) = \boldsymbol{E}[r_1^\gamma|s;\pi]$ i $Q^\pi(s,a) = \boldsymbol{E}[r_1^\gamma|s,a;\pi]$. Cilj agenta je dobiti politiku koja maksimizira kumulativnu umanjenu nagradu od početnog stanja, označenog ciljem izvedbe $J(\pi) = E[r_1^\gamma|\pi]$. Gustoća nad stanjem $s'$ nakon tranzicije vremenskog koraka $t$ iz stanja $s$ je označena s $p(s \rightarrow s', t, \pi)$. Također, umanjena distribuciju stanja označavamo s $\rho^\pi(s') \coloneqq \int_S \sum_{t=1}^{\infty} \gamma^{t-1} p_1(s) p(s \rightarrow s', t, \pi)\, ds$. Zatim se cilj izvedbe može definirati kao:

$$J(\pi_\theta) = \int_{\boldsymbol{S}} \rho^\pi(s) \int_{\boldsymbol{A}} \pi_\theta(s,a) r(s,a) da ds = \boldsymbol{E}_{s\sim\rho^\pi, a\sim\pi_\theta}[r(s,a)] \tag{2-11}$$

gdje $\boldsymbol{E}_{s\sim\rho}[\cdot]$ označava očekivanu vrijednost s obzirom na umanjenu distribuciju stanja $\rho(s)$.

Osnovna ideja ovih algoritama je prilagoditi parametre $\theta$ politike u smjeru gradijenta izvedbe $\nabla_\theta J(\pi_\theta)$. Temeljni rezultat koji leži u osnovi ovih algoritama je teorem o gradijentu politike:

$$\nabla_\theta J(\pi_\theta) = \int_{\boldsymbol{S}} \rho^\pi(s) \int_{\boldsymbol{A}} \nabla_\theta \pi_\theta(a|s) Q^\pi(s,a) da ds = \boldsymbol{E}_{s\sim\rho^\pi, a\sim\pi_\theta}[\nabla_\theta \pi_\theta(a|s) Q^\pi(s,a)] \tag{2-12}$$

Ovaj teorem ima važnu praktičnu vrijednost jer svodi izračun gradijenta izvedbe na jednostavan izračun očekivane vrijednosti.

Generalno govoreći, algoritmi gradijenta politike koriste se sa stohastičkom funkcijom politike $\pi(.|s)$. To znači da je funkcija politike $\pi(.|s)$ predstavljena kao distribucija akcija. Za dano stanje postojat će distribucija vjerojatnosti za svaku radnju u akcijskom prostoru. U DPG-u (engl. *Deterministic Policy Gradient*), umjesto stohastičke politike $\pi$, slijedi se deterministička politika $\mu(.|s)$. Za dano stanje $s$, postojat će deterministička odluka $a=\mu(s)$ umjesto raspodjele po akcijama [26].

Akter-kritičar (engl. *actor-critic*) široko je korištena arhitektura koja se temelji na teoremu o gradijentu politike. Akter prilagođava parametre $\theta$ stohastičke politike $\pi_\theta(s)$ stohastičkim gradijentnim usponom koji je dan izrazom (2-12). Umjesto nepoznate stvarne akcijska-vrijednost funkcije $Q^\pi(s,a)$ u jednadžbi (2-12), koristi se akcija-vrijednost funkcija $Q^w(s,a)$ s vektorom

parametra $w$. Kritičar procjenjuje akcija-vrijednost funkciju $Q^{\pi}(s,a) \approx Q^{w}(s,a)$ koristeći odgovarajući algoritam za procjenu politike kao što je učenje vremenske razlike (engl. *Temporal Differnce Learning*).

Ciljna funkcija stohastičkog algoritma gradijenta politike može se napisati na sljedeći način:

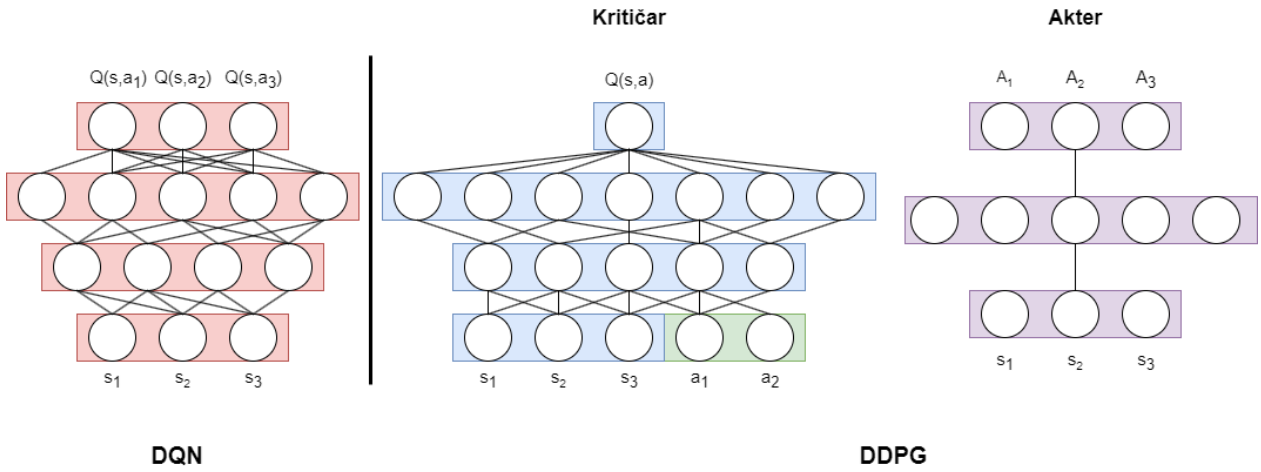
$$\nabla_{\theta} J(\theta) = E\left[\nabla_{\theta} log_{\pi_{\theta}}(s,a) Q^{\omega}(s,a)\right] \tag{2-13}$$

Za deterministička politiku tada vrijedi:

$$\nabla_{\theta} J(\theta) = E[\nabla_{\theta} \mu_{\theta}(s) \nabla_{a} Q^{\mu}(s,a)] \tag{2-14}$$

DDPG je akter-kritičar algoritam bez modela koji kombinira duboko Q-učenje i DPG. Akter-kritičar je verzija gradijenta politike vremenske razlike (engl. *Temporal Difference*, TD). U algoritmima se koriste dvije mreže: akter i kritičar. Akter odlučuje koju radnju treba poduzeti, a kritičar ga obavještava koliko je radnja dobra i kako je treba prilagoditi. Učenje aktera temelji se na pristupu gradijenta politike. Za usporedbu, kritičar procjenjuje radnju koju je proizveo akter izračunavanjem funkcije vrijednosti.

Za DDPG algoritam se koriste četiri mreže: Q-mreža $\theta^{Q}$, mreža politike $\theta^{\mu}$, ciljna Q-mreža $\theta^{Q'}$ i ciljana mreža politike $\theta^{\mu'}$. Q-mreža i mreža politike vrlo je slična jednostavnom akter-kritičar algoritmu, ali u DDPG-u akter izravno preslikava stanja u radnje umjesto ispisa distribucije vjerojatnosti kroz diskretni akcijski prostor kao u slučaju kod neuronske mreže dubokog učenja (engl. *Deep Q-Network* – DQN). Usporedba DQN-a i DDPG-a dana je slikom 2.7. Ciljne mreže su vremenski odgođene kopije svojih izvornih mreža koje polako prate naučene mreže. Korištenje ovih mreža ciljnih vrijednosti uvelike se poboljšava stabilnost učenja. Razlog tome je što u metodama koje ne koriste ciljne mreže, jednadžbe ažuriranja mreže međusobno su ovisne o vrijednostima koje je izračunala sama mreža, što je čini sklonom divergenciji [27].




**Slika 2.7.** *Usporedba DQN-a i DDPG-a*

Pseudo kod DDPG algoritma prema [28] glasi:

DDPG Algoritam:

Nasumična inicijalizacija mreže kritičara $Q(s, a|\theta^Q)$ i aktera $\mu(s|\theta^\mu)$ s težinama $\theta^Q$ i $\theta^\mu$.

Inicijalizacija ciljane mreže $Q'$ i $\mu'$ s težinama $\theta^{Q'} \leftarrow \theta^Q$ i $\theta^{\mu'} \leftarrow \theta^\mu$.

Inicijalizacija međuspremnika za ponavljanje iskustva $R$.

ZA epizoda = 1 DO M ČINITI:

Inicijalizacija slučajnog procesa $\varphi$ za istraživanje akcija.

Primanje početnog stanja promatranja $s_1$.

ZA t = 1 DO T ČINTI:

Odabiranje akcije $a_t = \mu(s_t|\theta^\mu) + \varphi_t$ prema trenutnoj politici i istraživačkom šumu.

Izvršavanje radnje $a_t$, promatranje nagrade $r_t$ i promatranje novog stanja $s_{t+1}$.

Pohrana prijelaza $(s_t, a_t, r_t, s_{t+1})$ u $R$.

Uzimanje uzoraka nasumičnog batch-a od $N$ prijelaza $(s_i, a_i, r_i, s_{i+1})$ iz $R$.

Postaviti $y_i = r_i + \gamma Q'(s_{i+1}, \mu 0(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$.

Ažuriranje mreže kritičara minimiziranjem gubitka: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$

Ažuriranje politike aktera pomoću uzorkovanog gradijenta politike:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$$

Ažuriranje ciljnih mreža:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

KRAJ PETLJE.

| |
|---|
| KRAJ PETLJE. |

Ovaj se algoritam može raščlaniti na četiri glavna dijela:

1. Ponavljanje iskustva
2. Ažuriranje akter-kritičar mreža
3. Ažuriranje ciljanih mreža
4. Istraživanje.

Kao što se koristi u dubokom Q-učenju (i mnogim drugim RL algoritmima), DDPG također koristi međuspremnik za ponavljanje radi uzorkovanja iskustva kako bi se ažurirali parametri neuronske mreže. Tijekom svake se epizode spremaju svi skupovi (engl. *tuples*) iskustva (stanje, radnja, nagrada, iduće stanje) i pohranjuju se u memoriju konačne veličine tj. međuspremnik za ponavljanje. Zatim se nasumično uzorkuju serije iskustva iz međuspremnika kada se ažuriraju vrijednosne mreže i mreže politike.

Mreža vrijednosti ažurira se slično kao što se radi u Q-učenju. Ažurirana Q-vrijednost dobiva se Bellmanovom jednadžbom:

$$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'}) \tag{2-15}$$

Međutim, u DDPG-u, Q-vrijednosti sljedećeg stanja izračunavaju se ciljnom mrežom vrijednosti i ciljnom mrežom politike. Zatim se minimizira srednji kvadrat gubitka između ažurirane Q-vrijednosti i izvorne Q-vrijednosti:

$$L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2 \tag{2-16}$$

Valja napomenuti da se izvorna Q-vrijednost izračunava pomoću mreže vrijednosti, a ne ciljne mreže vrijednosti. Za funkciju politike, cilj je maksimizirati očekivani povrat:

$$J(\theta) = E[Q(s, a)|_{s=s_t, a_t=\mu(s_t)}] \tag{2-17}$$

Kako bi se izračunao gubitak politike, uzima se derivacija ciljne funkcije s obzirom na parametar politike. Mora se imati na umu da je funkcija aktera (politike) diferencijabilna, stoga se primjenjuje pravilo ulančanog deriviranja.

$$\nabla_{\theta^\mu} J(\theta) \approx \nabla_a Q(s, a) \nabla_{\theta^\mu} \mu(s|\theta^\mu) \tag{2-18}$$

No, budući da politiku ažuriramo na način izvan politike sa serijama iskustva, uzima se srednja vrijednost zbroja gradijenata izračunatih iz serije:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q\left(s, a \middle| \theta^Q\right)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s=s_i} \tag{2-19}$$

Izrađuje se kopija parametara ciljne mreže i navodi ih da polako prate parametre naučenih mreža putem „laganih ažuriranja“ kao što je prikazano u nastavku:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'} \tag{2-20}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'} \tag{2-21}$$

gdje je $\tau \ll 1$.

U podržanom učenju za diskretne prostore akcija, istraživanje se provodi probabilističkim odabirom slučajne radnje (kao što je ε-pohlepno istraživanje). Za prostore kontinuiranih akcija, istraživanje se vrši dodavanjem šuma samoj akciji. U [28], autori koriste Ornstein-Uhlenbeckov proces za dodavanje šuma izlazu akcije:

$$\mu'(s_t) = \mu\left(s_t \middle| \theta_t^\mu\right) + \varphi \tag{2-20}$$

Ornstein-Uhlenbeckov proces generira šum koji je u korelaciji s prethodnim, kako bi se spriječilo da šum poništi ili „zamrzne“ cjelokupnu dinamiku neuronske mreže.

## 2.4. Postojeća rješenja upravljanja vozilom zasnovana na dubokom učenju

U [29], predloženi su modeli duboke Q-mreže i dvostruke duboke Q-mreže za odluku automatiziranog vozila o promjeni trake u područjima gustog prometa na temelju dubokog podržanog učenja. Predložene su i dvije nove funkcije nagrađivanja kako bi se prometnom okruženju povisila razina složenosti radi bolje simulacije prometa. Rezultati simulacije pokazali su da se DDQN (engl. *Double Deep Q-Network*) može prilagoditi promjenama u brzini vozila, promjenama u broju okolnih vozila i stilu vožnje okolnih vozila. Predložena funkcija nagrade je podigla stopu uspješnosti za 4% bez promjene u arhitekturi mreže. Mreža je postigla stopu uspješnosti promjene trake od preko 90% i imala je snažnu prilagodljivost brojnim okolinama koje vozilo nije vidjelo tijekom treniranja. Predloženo rješenje je naučeno kroz 100,000 epizoda, međutim, moguće je da bi stopa uspješnosti bila još veća s većim brojem provedenih epizoda.

U radu [30] je predložena politika odlučivanja pomoću dubokog podržanog učenja za autonomna vozila kako bi se riješili problemi ponašanja pri pretjecanju na autocesti. Prvo se uspostavlja okruženje za vožnju autocestom u kojem vozilo ima za cilj proći kroz okolna vozila učinkovitim i sigurnim manevrom. Predstavljen je hijerarhijski kontrolni razvojni okvir za kontrolu ovih vozila, što ukazuje da viša razina upravlja odlukama o vožnji, a niža razina brine o nadzoru brzine i ubrzanju vozila. Zatim se posebna metoda DRL-a nazvana DDQN (engl. *Dueling Deep Q Network*) algoritam primjenjuje za izvođenje strategije donošenja odluka na autocesti. Proveo se niz eksperimenata simulacije procjene kako bi se ocijenila učinkovitost predložene politike donošenja odluka na autocestama. Rezultati simulacije otkrivaju da politika pretjecanja temeljena na DDQN-u može učinkovito i sigurno izvršiti zadatke vožnje na autocesti. Mjerenja su se provela isključivo unutar simulatora. Prikupljena baza podataka o autocestama iz stvarnog svijeta bi se mogla koristiti za procjenu uspješnosti strategije pretjecanja.

U radu [31] se predlaže metoda kontrole odlučivanja koja se temelji na dubokom podržanom učenju u više zadataka kako bi se riješili nedostaci kontrole autonomne vožnje u složenim prometnim uvjetima. Zadatak autonomne vožnje je podijeljen u nekoliko podzadataka koji koriste predloženu metodu za smanjenje vremena učenja i poboljšanje učinkovitosti u prometu u složenim uvjetima prometa s više traka. Scenariji zadataka vožnje u pet traka izgrađen je u CARLA simulatoru. Simulacije su provedene kako bi se provjerila predložena metoda. Rezultati simulacije daju zaključak da predložena metoda povećava učinkovitost vožnje inteligentnih vozila u složenim prometnim uvjetima. Budući će rad više pozornosti posvetiti prometnim pravilima i zadacima autonomne vožnje u gradskom prometu.

U radu [32] je predloženo optimizacijski ugrađeno podržano učenje kako bi se postiglo prilagodljivo donošenje odluka u kružnom toku. Definirana je promocija, tj. modificirani agent razvojnog okvira akter–kritičar, koja ugrađuje metodu optimizacije temeljenu na modelu u podržanom učenju za izravno istraživanje kontinuiranog ponašanja u prostoru akcije. Predložena metoda može odrediti ponašanje na makrorazini (promjena trake ili ne) i ponašanja na srednjoj razini željenog ubrzanja i vremena djelovanja. U eksperimentima se učinkovitost algoritma i naučena strategija vožnje uspoređuju s donošenjem odluka koje sadrže ponašanje na makrorazini i konstantna ponašanja srednje razine željenog ubrzanja i vremena djelovanja. Kako bi se istražila prilagodljivost, simulira se izvedba u tipu kružnog raskrižja koji se nije koristio za treniranje i još dvije opasne situacije kako bi se potvrdilo da predložena metoda u skladu s promjenjivim scenarijima mijenja svoje odluke. Rezultati pokazuju da predloženi algoritam ima visoku učinkovitost.

U radu [33] se razmatra problem planiranja puta za autonomno vozilo koje se kreće autocestom. Ovaj rad predlaže razvoj politike vožnje koja se temelji na podržanom učenju. Na ovaj način, predložena politika vožnje čini minimalne ili nikakve pretpostavke o okolini, budući da nije potrebno prethodno znanje o dinamici sustava. Razmatraju se scenariji vožnje u kojima se pojavljuju vozila na cesti upravljana autonomno i neautonomno. Izvedena politika podržanog učenja prvo se uspoređuje s optimalnom politikom izvedenom putem dinamičkog programiranja, a zatim se njena učinkovitost procjenjuje prema realnim scenarijima koje generira SUMO simulator toka prometa. Različiti eksperimenti izvedeni su u simulatoru vožnje na autocesti razvijenom u *Unity* okruženju kako bi se demonstrirala učinkovitost predloženog pristupa. Rezultati potvrđuju da predložena metoda može odabrati najbolja pravila vožnje za scenarije vožnje u usporedbi s najsuvremenijim studijama i može pružiti bolje performanse u pogledu metrike brzine i promjene trake.

U radu [34] je predstavljeno moguće rješenje povećanja efikasnosti i stabilnosti podržanog učenja u kontinuiranoj kontroli brzine vozila pomoću metode determinističke promocije podržanog učenja. Razvija se evaluacija politike kod kritičara i istraživanje kod agenta što kombinira evaluaciju temeljenu na normalizaciji i vodič za pretraživanje bez modela. Predložen je deterministički promotivni algoritam podržanog učenja. Rezultati pokazuju da predložena metoda može poboljšati stabilnost i učinkovitost algoritama podržanog učenja u problemu kontinuiranog upravljanja i može postići prihvatljive performanse u uzdužnom upravljanju brzinom u promjenjivim okruženjima.

# 3. IZGRADNJA VLASTITIH ALGORITAMA ZA UPRAVLJANJE VOZILOM U SIMULATORU PRIMJENOM PODRŽANOG UČENJA

U ovom poglavlju opisana je izgradnja vlastitih algoritama za upravljanje vozilom u simulatoru primjenom podržanog učenja. Izrađena su dva algoritma upravljanja, jedan baziran na dubokom Q-učenju (DQN), a drugi na dubokom deterministčkom gradijentu politike (DDPG). Cilj je vozilom slijediti unaprijed određenu rutu izbjegavajući sudare i neplanirana skretanja s rute u urbanom okruženju unutar simulacije. Ruta je definirana kao niz putnih točaka u prostoru CARLA simulatora. Oba algoritma su izrađena u Python programskom jeziku koristeći Carla Python API uz korištenje Tensorflow i OpenCV biblioteka. Svi algoritmi će biti testirani u simulaciji korištenjem CARLA simulatora.

U potpoglavlju 3.1. detaljno je opisan proces podešavanja CARLA simulatora i ostalih potrebnih programa i biblioteka. U potpoglavlju 3.2. dan je opis definiranja agenta i okoline, kao i formulacija MDP-a. U potpoglavlju 3.3. objašnjena je arhitektura koja će se koristiti za duboko Q-učenje, a u potpoglavlju 3.4. je objašnjena arhitektura korištenog DDPG algoritma. U potpoglavlju 3.5. je opisan postupka treniranja agenta.

## 3.1. Podešavanje radnog okruženja i instalacija potrebnih biblioteka

U današnje vrijeme virtualno testiranje sve više postaje jedan od najvažnijih koncepata za izgradnju sigurne tehnologije autonomnih vozila. Korištenje foto-realistične simulacije (virtualni razvoj i validacijsko testiranje) i odgovarajući dizajn scenarija vožnje trenutni su ključevi za izgradnju sigurnog i robusnog vozila [35]. Za potrebu je ovog rada korišten CARLA simulator.

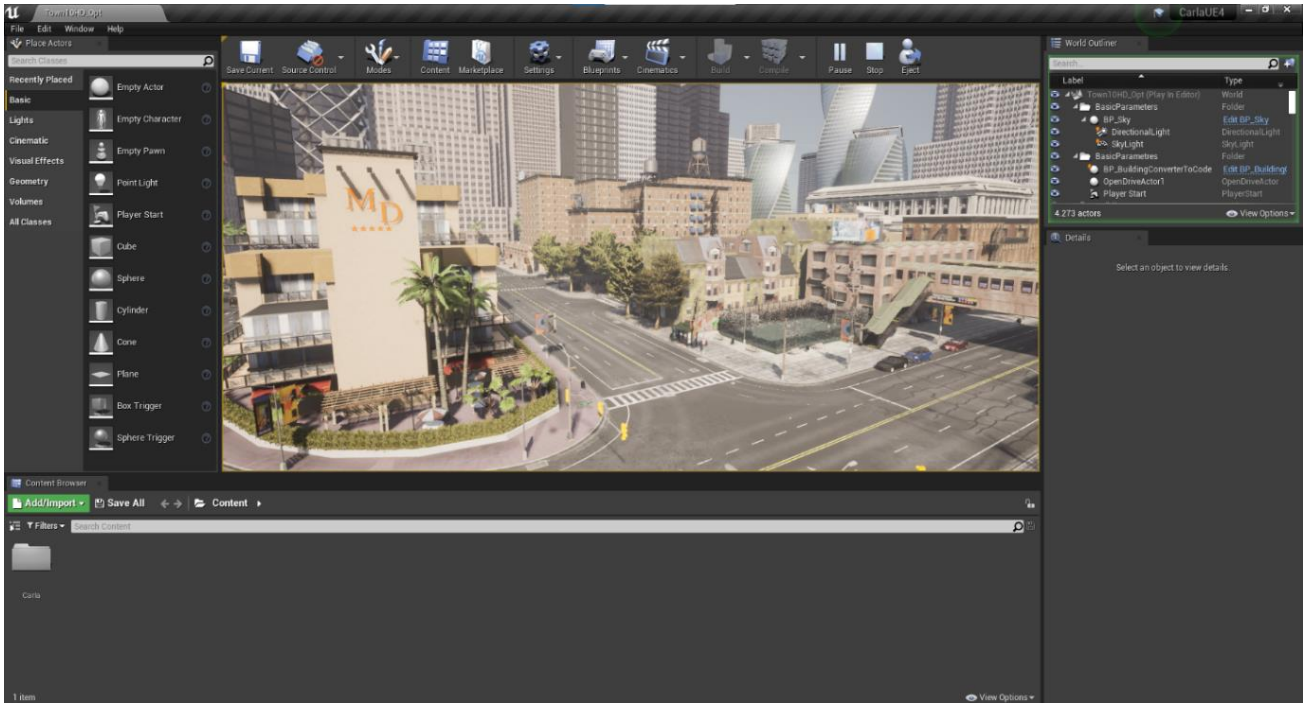
CARLA Simulator [36] je simulator otvorenog koda, temeljen na Unreal Engineu, koji omogućuje upravljanje vozilima u virtualnim prometnim scenarijima (grad, autocesta, ...). Vozila mogu biti opremljena različitim senzorima poput RGB kamere, LIDAR-a i sl. Simulator pruža cjelovit Python API koji korisniku omogućuje kontrolu svih aspekata povezanih sa simulacijom, uključujući vremenske uvjete, ponašanje pješaka, dohvaćanje podataka sa senzora u vozilu i generiranje prometa. Također, nudi brzu simulaciju za planiranje i kontrolu, gdje je prikaz onemogućen kako bi se ponudilo brzo izvršenje ponašanja na cesti i simulacija prometa kada grafika nije potrebna. Postoji i mogućnost izgradnje različitih simulacija prometnih scenarija sa Scenario Runner-om, posebnim programom kojeg sadrži ovaj simulator. Simulator se temelji na Unreal Engineu (UE4), jednom od najnaprednijih alata za stvaranje 3D-a u stvarnom vremenu, i koristi OpenDrive standard za definiranje cesta i urbanih postavki, omogućujući CARLA-i realan

izgled. CARLA ima dva glavna dijela. S jedne strane je poslužitelj koji je odgovoran za sve što je povezano sa samom simulacijom, kao što je npr. postavljanje okoline. Preporučuje se da ovaj poslužitelj radi na namjenskom GPU-u jer fluidnost simulacije ovisi o performansama GPU-a. S druge strane, klijentska strana kontrolira logiku agenta na sceni i postavlja uvjete u okolini.

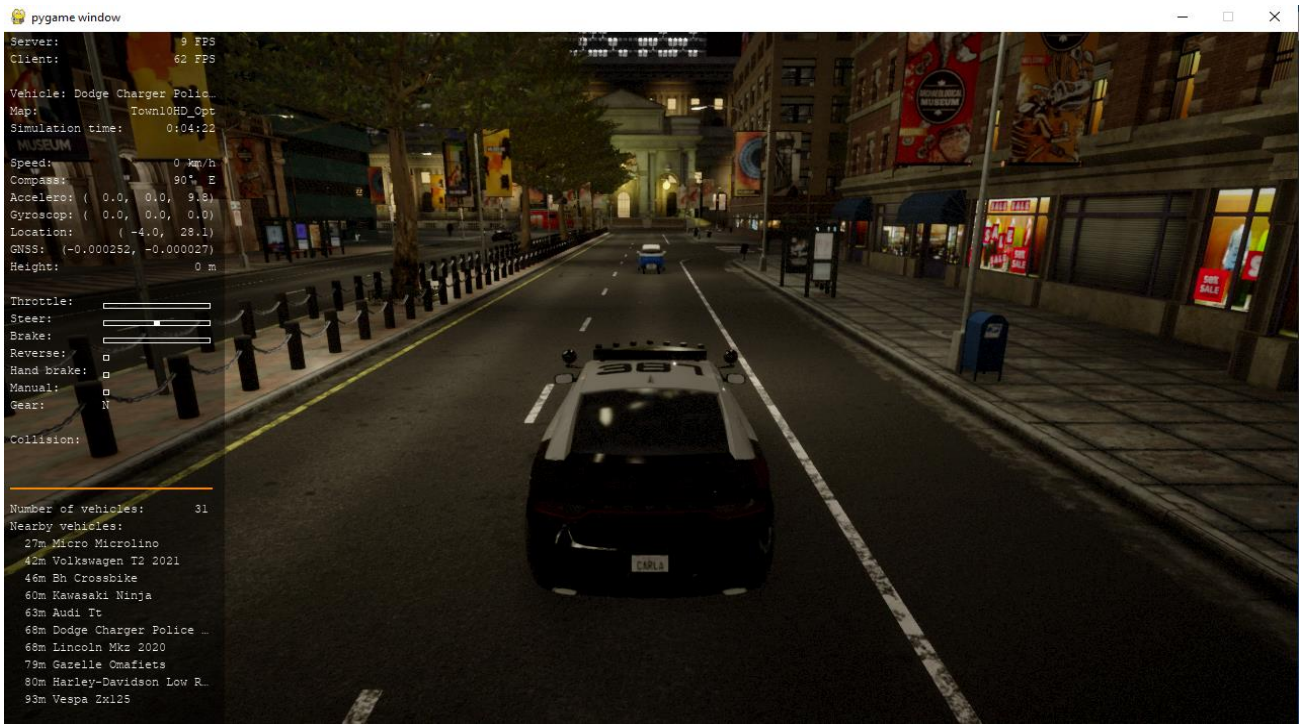
Kada se pokrene simulacija, poslužitelj je spreman za komunikaciju sa klijentima. Izgled simulatora je prikazan na slici 3.1. Klijent se spaja na poslužitelj korištenjem Python API-a. U zasebnoj konzoli, dok je aktivan poslužitelj, moguće je pokrenuti Python skriptu koja je u ulozi klijenta:

*python ime_datoteke.py*

Simulator dolazi s nekolicinom predefiniranih skripti. Kako bi se provjerila ispravnost simulatora, najlakše je pokrenuti Python skriptu *manual_control.py*. Ako je sve dobro instalirano i podešeno, otvara se novi prozor u kojemu je kreirano vozilo s kojim korisnik može samostalno upravljati pomoću tipkovnice (slika 3.2.).



**Slika 3.1**. *Izgled CARLA simulatora u Unreal Engine-u*




**Slika 3.2.** *Samostalno upravljanje vozilom unutar simulatora*

Kako bi se algoritmi lakše implementirali, koristi se TensorFlow biblioteka. TensorFlow je biblioteka otvorenog koda prilagođena Pythonu za numeričko računanje koja strojno učenje i razvoj neuronskih mreža čini bržim i lakšim.

Budući da se predloženo upravljanje temelji na slici koja se dobiva s kamere montirane na prednjoj strani vozila, potrebno je koristiti biblioteku prilagođenu za rad sa slikom i robotskim vidom. U tu svrhu se koristi OpenCV (engl. *Open Source Computer Vision Library*) biblioteka. OpenCV je biblioteka računalne podrške za računalni vid i strojno učenje otvorenog koda. OpenCV je napravljen kako bi osigurao zajedničku infrastrukturu za aplikacije računalnog vida i ubrzao korištenje strojne percepcije u komercijalnim proizvodima [37]. Detaljnije informacije o instalaciji CARLA simulatora su dane u prilogu P.3.1.1., a u prilogu P.3.1.2. je dana tablica specifikacija računala na kojemu se provelo treniranje i korištene verzije programa.

## 3.2. Formulacija Markovljevog procesa odlučivanja za problem upravljanja vozilom u CARLA simulatoru

Znanstveni rad [35] pružio je ključnu inspiraciju za modeliranje algoritama jer je autor istaknuo važne mogućnosti CARLA simulatora u rješavanju problema upravljanja vozilom u simulatoru primjenom podržanog učenja. Predloženo modeliranje MDP-a je značajno modificirano u

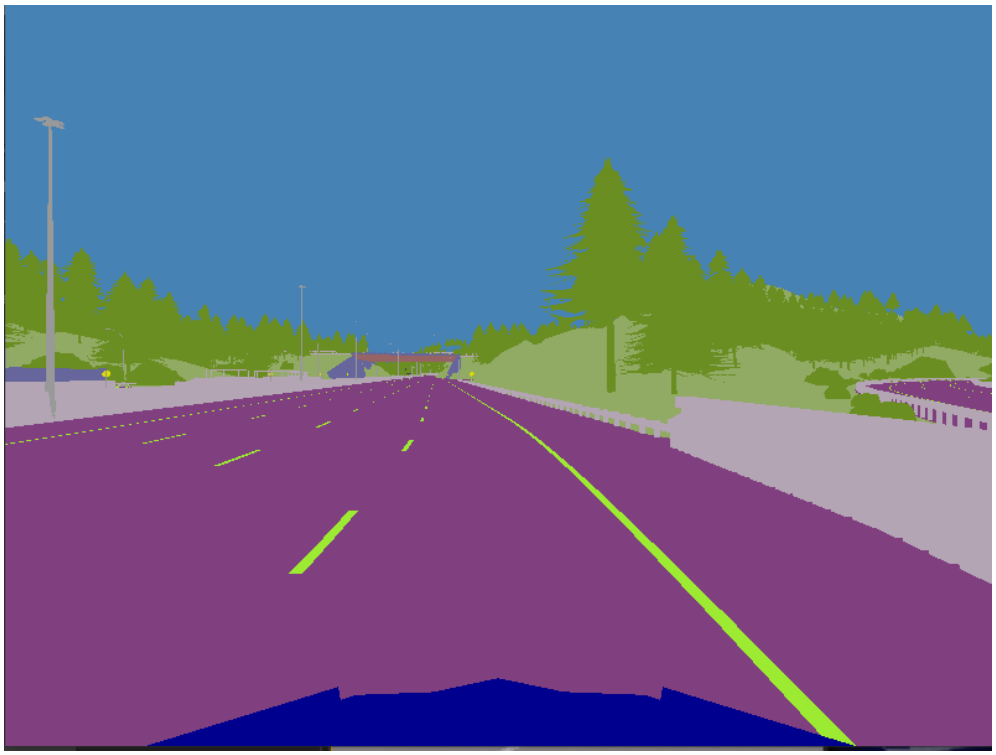
usporedbi s radom [35] kako bi se rješenje prilagodilo vlastitim potrebama i rješavanju problema koje autori nisu u potpunosti riješili.

Agent koji izvodi određene akcije je vozilo Tesla Model 3, a upravljački program kontinuirano prosljeđuje akcije vozilu. Model ovog vozila dolazi zajedno s preuzetim datotekama CARLA simulatora. Izgled vozila unutar simulatora dan je slikom 3.3. Modeli podržanog učenja pokušavaju na temelju vizualnih značajki i značajki vožnje predvidjeti optimalnu vrijednost zakreta volana u svakom stanju. Vozilom se unutar simulatora upravlja prosljeđujući numeričke vrijednosti za papučicu gasa, zakret volana i kočnicu (engl. *throttle*, *steering* i *breaking*) preko ugrađene funkcije CARLA Python API-a. Numeričke vrijednosti za papučicu gasa mogu biti u intervalu između nula (nema stiska papučice gasa) i jedan (puni stisak papučice gasa), dok kod zakreta volana mogu biti između minus jedan (puni zakret volana ulijevo) i jedan (puni zakret volana udesno). Prilikom razvoja rješenja za automatsko upravljanje vozilom nije korištena kočnica. Upravljanje brzinom vozila se vrši na način da se za početnu vrijednost papučice gasa šalje numerička vrijednost od 0.8 sve dok vozilo ne dođe do odgovarajuće maksimalne brzine. Kada dođe do maksimalne brzine, vrijednost papučice gasa se smanjuje na 0.4. Kada vozilo dođe na ispod 10% maksimalne brzine, vrijednost papučice gasa se ponovno postavlja na 0.8. Ovaj princip upravljanja brzinom vozila se ponavlja tijekom svake epizode. Vrijednost zakreta volana se dobiva kao izlaz iz izgrađenih neuronskih mreža koje su dobivene algoritmima podržanog učenja.


**Slika 3.3.** *Tesla Model 3 unutar CARLA simulatora*

Vizualne značajke vožnje se izvlače iz slike dobivene sa prednje semantičke kamere vozila. Slika dobivena s semantičke kamere vozila ima širinu 640, a visinu 480 elemenata slike. FoV (engl. *Field of View*) slike iznosi 90°. Kamera za semantičku segmentaciju kao izlaz daje sliku gdje svaki element slike sadrži oznaku kodiranu u crvenom kanalu. Ova slika prikazuje elemente u sceni različitim vrijednostima crvene boje ovisno o tome kako su označeni. Tako će primjerice pješaci na slici biti označeni jednom svjetlinom crvene boje, vozila drugom, zgrade trećom itd. Ova se izvorna slika mora pomoću *ColorConverter.CityScapesPalette.py* palete boja pretvoriti u RGB sliku koja je čovjeku lakše razumljiva. Slika dobivena koristeći ovu paletu boja je dana na slici 3.4. Najvažniji dio slike dobivene s prednje kamere je cesta koja se nalazi ispred vozila i susjedne vozne trake. Iz tog je razloga područje od interesa (engl. *Region of Interest* - RoI) samo donja polovica slike. Radi lakšeg procesiranja, nakon što se postavi područje interesa, slika se sažima na visinu i širinu od 24x24 elementa slike. Budući da je to RGB slika, ovih 24x24x3 elementa se izravnavaju (engl. *flatten*) te se s njima formira vektor vizualnih značajki.

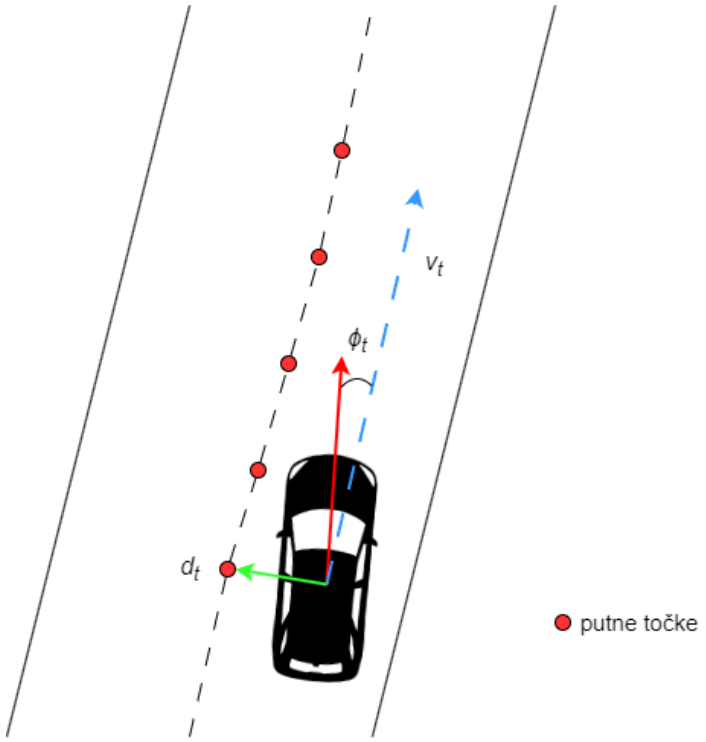
Značajke vožnje se dobivaju izravno iz podataka unutar simulacije. U svakom se trenutku mogu dobiti podaci o brzini vozila $v_t$. Kako bi se dobio kut između vozila i vozne trake $\phi_t$, prvo se formira vektor usmjerenosti vozne trake u danom trenutku te se zatim dohvaća prednji vektor usmjerenosti vozila nakon čega se može izračunati kut između ta dva vektora. Udaljenost vozila od sredine vozne trake $d_t$ se mjeri kao udaljenost dvije točke u prostoru tj. uzima se središnja točka vozila i uspoređuje se s najbližom putnom točkom rute koju vozilo prati. Prikaz vozila u usporebi sa putnim točkama se može vidjeti na slici 3.5. Zatim se formira vektor značajki vožnje $df_t = (v_t, d_t, \phi_t)$. Vektor značajki vožnje se zatim spaja s vektorom vizualnih značajki te rezultantni vektor predstavlja vektor stanja koji se predaje kao ulaz neuronskim mrežama.



**Slika 3.4.** *Slika dobivena primjenom palete boja na sliku dobivenu sa semantičke kamere*

Kao što je i prije spomenuto, problem upravljanja autonomnim vozilom se može definirati kao Markovljev proces odlučivanja (MDP). Cilj ovog rada je razviti agenta koji generira autonomnu kontrolu vozila u okviru CARLA simulatora na temelju algoritma dubokog podržanog učenja koji rješava odgovarajući MDP. Definira se agent koji promatra stanje ($s_t$) vozila kojim se upravlja i generira akciju ($a_t$). Akcija uzrokuje da se vozilo pomakne u novo stanje ($s_{t+1}$) stvarajući nagradu ($r_{t+1} = R(s_t, a_t)$) na temelju novog opažanja. Markovljev proces odlučivanja je stoga uređena četvorka ($S$, $A$, $P_a$, $R_a$) gdje je cilj pronaći dobru politiku, to jest funkciju $\pi(s)$ koju će agent izabrati kada je u stanju $s_t$.

a) Prostor stanja ($S$) - odnosi se na informacije koje se primaju iz okoline u svakom koraku algoritma. Modelira se $s_t$ kao *tuple* $s_t = (vf_t, df_t)$ gdje $vf_t$ predstavlja vektor vizualnih značajki pridruženih slici $I_t$ ili skup vizualnih značajki izdvojenih iz slike, a $df_t$ predstavlja vektor značajki vožnje koji se sastoji od brzine vozila $v_t$, udaljenosti vozila do središta trake $d_t$ i kuta između vozila i središta trake $\phi_t$, $df_t = (v_t, d_t, \phi_t)$. Slika 3.5. prikazuje ilustraciju značajki vožnje u usporedbi sa putnim točkama rute.




**Slika 3.5.** *Značajke vožnje definirane s obzirom na putne točke rute*

b) Prostor akcija (*A*) – akcije koje agent može poduzeti (zakret volana) se nalaze u rasponu [-1, 1] što je ujedno i prostor akcija.

c) Funkcija nagrađivanja $R(s_{t+1}, s_t, a_t)$ – s obzirom da je cilj vozila proći kroz središte trake bez napuštanja trake i izbjegavati sudare, funkcija nagrađivanja mora nagraditi uzdužnu brzinu i kazniti poprečnu brzinu i odstupanje od središta trake. U nastavku su predstavljene specifične vrijednosti dodijeljene nagradi *R* na deterministički način:

- *R* = -500 ukoliko dođe do sudara
- *R* = -100 ako dođe do promjene vozne trake kada je to nepotrebno
- *R* = 10 ukoliko vozilo promijeni voznu traku kada je to potrebno
- *R* = -1 ukoliko se vozilo vozi izvan svoje trake kada to ne treba
- $R = |v_t cos\phi_t| - |v_t sin\phi_t| - |v_t d_t|$ ukoliko se vozilo nalazi unutar trake koja prati rutu

Vozilo nema krajnju točku do koje treba doći, već se prioritizira vrijeme provedeno vozeći bez sudara. Iz tog razloga se svakoj vrijednosti nagrade kada vozilo vozi unutar vozne trake predviđene rutom dodaje brojčana vrijednost koja predstavlja vrijeme koje je vozilo odvozilo do tog trenutka u sekundama.

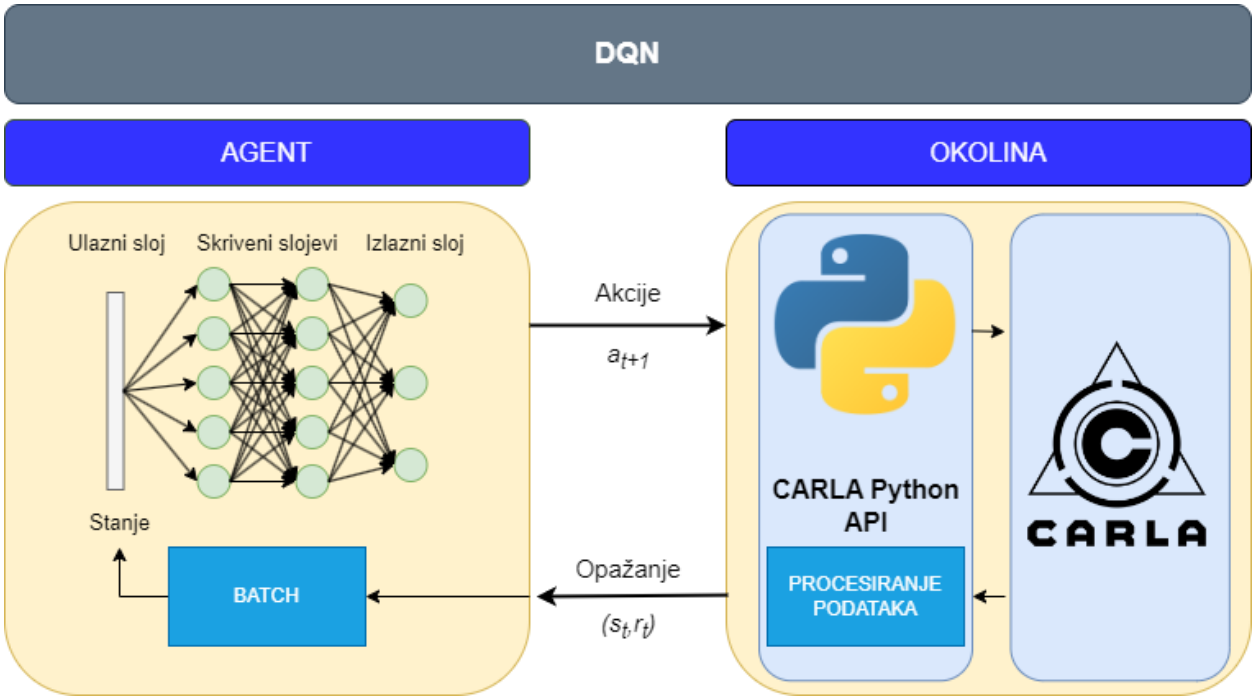
## 3.3. Arhitektura algoritma dubokog Q-učenja

Slijedeći prethodnu formulaciju MDP-a, potrebno je uspostaviti opći radni okvir onoga što će biti razvijeni DQN, jasno definirajući radnje i nagradu koja će se koristiti u algoritmu. Generalna arhitektura dubokog Q-učenja u CARLA simulatoru dana je slikom 3.6.

Predložena arhitektura dobiva vektor značajki vožnje $df_t = (v_t, d_t, \phi_t)$ izravno iz simulatora. Ovaj se vektor sastoji od brzine vozila u smjeru njegovog kretanja $v_t$, udaljenosti do središta trake $d_t$ i kuta u odnosu na smjer trake $\phi_t$.

Politika DQN-a omogućuje generiranje diskretnih akcija, stoga je potrebno kontinuirani prostor akcija diskretizirati. Uzimajući ovo u obzir, broj upravljačkih naredbi pojednostavljen je na skup od 9 diskretnih akcija u vožnji, diskretizirajući kut zakreta upravljača dok se vrijednost papučice gasa kontrolira neovisno o implementiranoj mreži kako je već objašnjeno u potpoglavju 3.2. Akcije koje DQN agent može poduzeti su zakret volana s vrijednostima [-1, -0.75, -0.50, -0.25, 0, 0.25, 0.50, 0.75, 1].

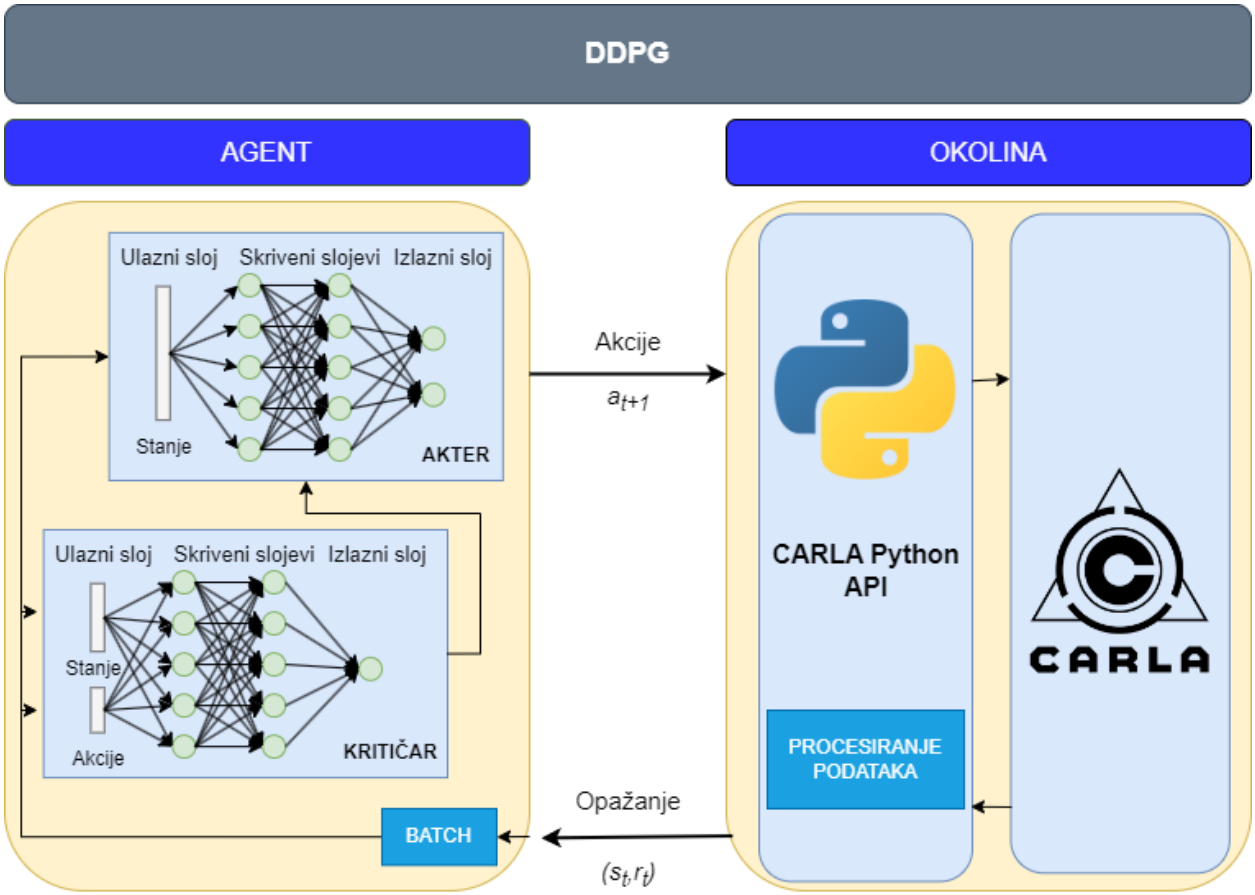
DQN se sastoji od ulaznog sloja koji je ustvari izravnati vektor stanja. Svaki element ulaznog sloja je potpuno povezan sa neuronima u sljedećem skrivenom sloju od 64 neurona sa ReLU (engl. *Rectified Linear Unit*) aktivacijskom funkcijom. Također, ovih 64 neurona je potpuno povezano sa 64 neurona u sljedećem skrivenom sloju s ReLU aktivacijskom funkcijom. 64 neurona u ovom sloju su na kraju potpuno povezana sa devet (broj mogućih akcija) neurona u izlaznom sloju sa linearnom aktivacijskom funkcijom. Također, konstruirana je ciljna mreža koja je kopija DQN-a, ali se njene težine tijekom treninga rjeđe ažuriraju. Funkcija gubitka koja se koristi je srednja kvadratna pogreška (engl. *Mean Squared Error* - MSE) između predviđenih Q-vrijednosti i ciljnih Q-vrijednosti.




**Slika 3.6.** *Dijagram arhitekture DQN algoritma*

## 3.4. Arhitektura algoritma dubokog deterministčkog gradijenta politike

Ovo potpoglavlje predstavlja osnovnu strukturu DDPG arhitekture na temelju prethodnog objašnjenja algoritma. Ovaj algoritam, kao što je prije spomenuto, sastoji se od dva dijela, aktera i kritičara, što je vidljivo na slici 3.5.




**Slika 3.7.** *Dijagram arhitekture DDPG algoritma*

Arhitektura sustava bazirana na DDPG algoritmu, kao što se može vidjeti na slici 3.5., samo mijenja agentov modul u odnosu na DQN arhitekturu. Za razliku od DQN-a, ovaj algoritam ima kontinuirani karakter, tako da akcije u ovom slučaju neće biti diskretne, već će biti u rasponu prostora akcija.

Ulazni sloj neuronske mreže kritičara čini izravnati vektor stanja spojen sa vektorom mogućih akcija koji je u ovom slučaju samo jedan element. Svaki element ulaznog sloja je potpuno povezan sa neuronima u skrivenom sloju koji sadrži 128 neurona sa ReLU aktivacijskim funkcijama. Neuroni iz ovog sloja su zatim potpuno povezani sa neuronima u sljedećem skrivenom sloju od 128 neurona sa ReLU aktivacijskim funkcijama. Neuroni u ovom skrivenom sloju su naposljetku potpuno povezani sa jednim neuronom u izlaznom sloju neuronske mreže koji ima linearnu aktivacijsku funkciju. Neuronska mreža aktera je gotovo identična kao NN kritičara. Jedina je razlika u tome što u izlaznom sloju NN aktera, neuron ima aktivacijsku funkciju tanh.

## 3.5. Postupak treniranja agenta

Agent je svaku epizodu postavljen na različite pozicije unutar mape *Town04*. Ova mapa je prikazana na slici 3.8. iz ptičje perspektive odnosno odozgo. Cestovna mreža ove mape sastoji se od male mreže kratkih ulica i raskrižja ugniježđenih između komercijalnih i stambenih zgrada, s obilaznicom u stilu "osmice" koja obilazi zgrade i obližnju planinu. Koristeći modul *GlobalRoutePlanner.py*, svakom epizodom se mijenja ruta po kojoj bi se vozilo trebalo kretati kako bi agent dobio raznolika iskustva iz kojih bi mogao bolje učiti. Jednom kada je ruta utvrđena, ona se iscrtava na mapu prateći cestu/ceste po kojima bi se vozilo trebalo voziti. Ruta je sastavljena od niza putnih točaka u prostoru CARLA simulatora sredinom ceste koje služe za lakši izračun značajki vožnje i lakše praćenje rute.

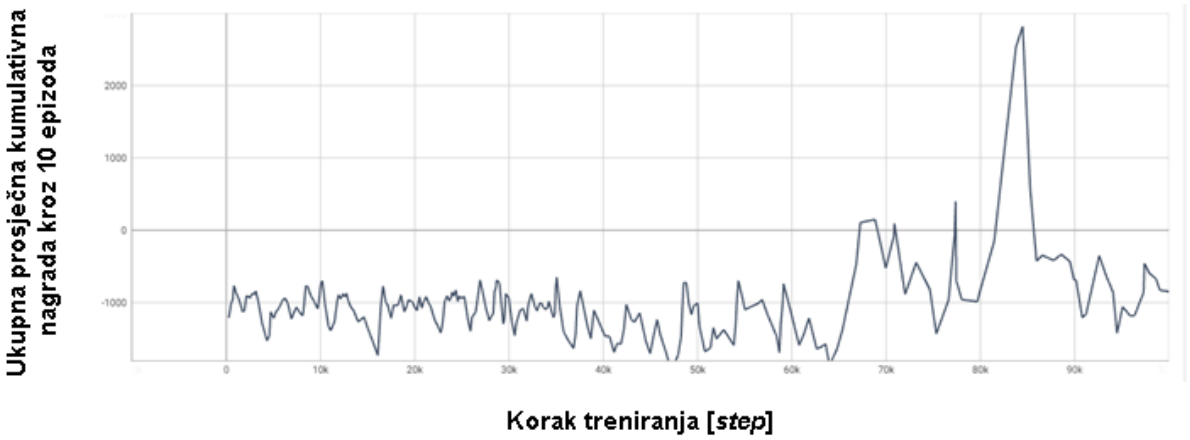
Na početku epizode se nasumično odabere jedna od početnih pozicija na mapi na kojoj će se kreirati vozilo. Na vozilo se zatim postavljaju razni senzori. Senzori postavljeni na vozilo su: prednja segmentacijska kamera, senzor napuštanja trake i senzor sudara. Inicijalizira se pričuvna memorija u kojoj će se spremati opažanja kako agent bude izabirao i izvršavao akcije. U svakom se stanju u pričuvnu memoriju spremaju opažanja kao uređena četvorka ($s$, $a$, $s'$, $R$). Veličina pričuvne memorije je postavljena tako da može sadržavati maksimalno 5000 uređenih četvorki. U početnim epizodama agent neće učiti iz ovih opažanja kako bi se spriječila pretjerana prilagodba podacima. Tek nakon što se u pričuvnu memoriju pohrani 1000 opažanja, tek onda se iz pričuvne memorije uzorkuju opažanja iz kojih se uči veličinom uzorka 32. Akcije koje agent izabire su u početku nasumične jer se koristi $\varepsilon$-pohlepno istraživanje. Parametar $\varepsilon$ je u početnim epizodama jednak 1.0 što znači da agent neće izabrati akcije koje model predvidi, već u 100% slučajeva izabire nasumične akcije. Parametar $\varepsilon$ se s epizodama linearno smanjuje sve dok ne dođe do minimalne vrijednosti od 0.1. Svakim se opažanjem ažuriraju parametri NN-a, a parametri ciljnih mreža se ažuriraju povremeno. Epizoda završava kada se vozilo sudari sa nekom preprekom ili kada istekne maksimalno vrijeme za jednu epizodu koje je postavljeno na 120 sekundi. Nakon završetka epizode, bilježe se podaci o ukupnoj nagradi dobivenoj kroz epizodu te se spremaju modeli koji su imali najbolju prosječnu nagradu u zadnjih 10 epizoda. Simulacija završava kada kada se izvrši unaprijed definiran broj epiozoda. Modeli su trenirani na 3000 epizoda. Algoritmi su najprije trenirani na mapi *Town04* bez prometa, a zatim su trenirani na istoj mapi postupno dodajući okolna vozila.

**Slika 3.8.** *Prikaz mape Town04 iz ptičje prespektive*

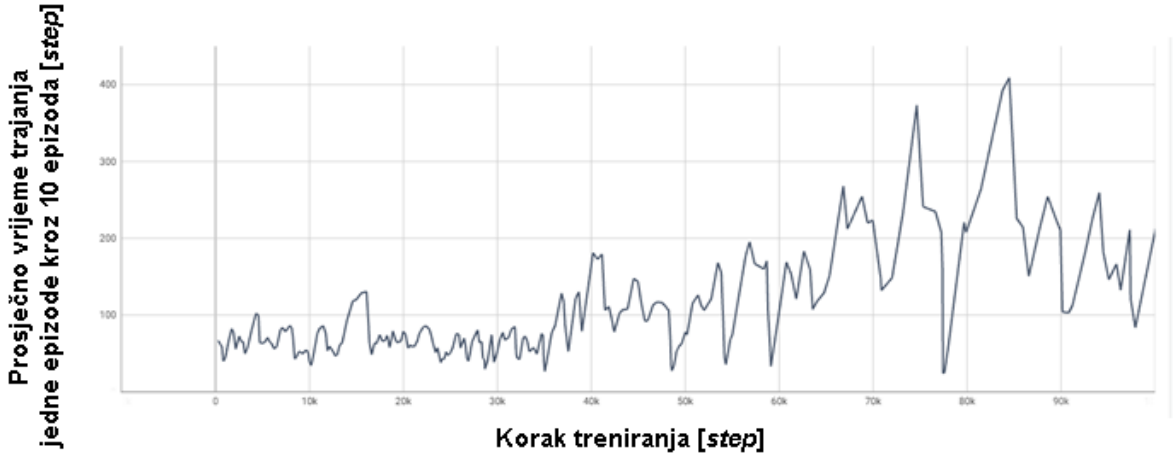
Vrijednosti hiperparametara za treniranje DQN mreže su: faktor smanjivanja $\gamma$ koji je jednak 0.99, stopa učenja $\alpha$ iznosi 0.001, a korišten je Adam optimizator. Opažanja se uzorkuju s veličinom uzorka 32. Grafički prikaz prosječne ukupne kumulativne nagrade kroz 10 epizoda tijekom vremena iskazanog u koracima treniranja (engl. *step*) prikazan je slikom 3.9., a prosječno vrijeme trajanje epizode kroz 10 epizoda tijekom koraka treniranja prikazan je slikom 3.10. Grafovi su dobiveni nakon 1000 epizoda treniranja. Iz grafičkog prikaza možemo zaključiti kako zbog visoke vrijednosti parametra $\varepsilon$ treniranje u početnom stadiju rezultira negativnim prosječnim nagradama. Pri kraju treniranja se dobiva najveća prosječna kumulativna nagrada te se vidi kako prosječna kumulativna nagrada s vremenom raste. Prosječno vrijeme trajanja epizode se također vremenom povećava.

Vrijednosti hiperparametara za obje DDPG mreže su: Adam optimizator, faktor smanjivanja $\gamma$ iznosi 0.99, parametar $\tau$ iznosi 0.005, a veličina uzorka iznosi 32. Stopa učenja kritičara iznosi 0.002, a stopa učenja aktera iznosi 0.001. Grafički prikaz prosječne ukupne kumulativne nagrade kroz 10 epizoda tijekom vremena iskazanog u koracima treniranja prikazan je slikom 3.11., a prosječno vrijeme trajanje epizode kroz 10 epizoda tijekom koraka treniranja

prikazan je slikom 3.12. Grafovi su dobiveni nakon 1000 epizoda treniranja. Sličan izgled grafova se dobio i za ovaj algoritam jer se koristila identična funkcija istraživanja. Prosječna kumulativna nagrada je u početnim epizodama negativna, a prosječno vrijeme trajanja jedne epizode relativno kratko. Pri kraju treniranja se obje vrijednosti povećavaju te se može primijetiti kontinuirani rast ovih vrijednosti kako vrijeme prolazi.

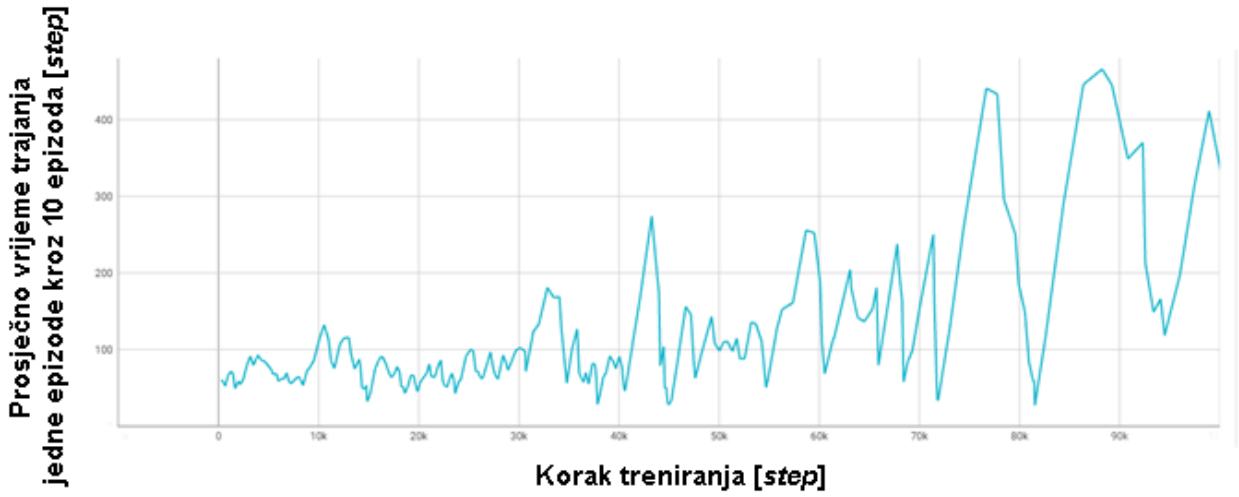



**Slika 3.9.** *Grafički prikaz prosječne ukupne kumulativne nagrade kroz 10 epizoda u ovisnosti o koraku treniranja za DQN algoritam*




**Slika 3.10.** *Grafički prikaz prosječnog vremena trajanja jedne epizode kroz 10 epizoda u ovisnosti o koraku treniranja za DQN algoritam*


**Slika 3.11.** *Grafički prikaz prosječne kumulativne nagrade u ovisnosti o vremenu za DDPG algoritam*




**Slika 3.12.** *Grafički prikaz prosječnog vremena trajanja jedne epizode u ovisnosti o vremenu za DDPG algoritam*

# 4. EVALUACIJA PERFORMANSI IZRAĐENIH ALGORITAMA ZA UPRAVLJANJE VOZILOM U SIMULATORU PRIMJENOM PODRŽANOG UČENJA

U ovome su poglavlju predstavljeni rezultati evaluacije algoritama upravljanja vozilom u simulatoru primjenom podržanog učenja. Modeli dobiveni DQN i DDPG algoritmima se testiraju na četiri različitih scenarija vožnje unutar CARLA simulatora. Evaluacija izgrađenih algoritama provedena je pomoću ukupno sedam predloženih metrika. Prvu metriku predstavlja prosječno prijeđen put i prosječno ukupno vrijeme koje je vozilo provelo vozeći kroz epizodu. Nadalje, mjeri se prosječna ukupna kumulativna nagrada dobivena kroz epizodu. Četvrta metrika je prosječan broj pogrešaka koje agent učini tijekom jedne epizode. Pogreška predstavlja akciju koja negativno utječe na promet. U pogreške pripadaju sudari te skretanja s ceste koja prati rutu onda kada to nije potrebno. Također će se pratiti i prijeđeni put i vrijeme provedeno u susjednim trakama kada to nije potrebno. Kao zadnju metriku će se pratiti uspješnost provedbe vožnje bez sudara. Tijekom testiranja modela korištena su četiri različita scenarija kao što je prikazano u tablici 4.1. Dobiveni modeli se prvo testiraju na mapi na kojoj su trenirani, a zatim se isti scenariji provode na mapi koja nije bila prisutna za vrijeme treniranja modela. Svaki scenarij je testiran 10 puta te su dobivene i prikazane vrijednosti navedenih metrika.

| Scenarij | Opis |
|---|---|
| 1 | Vozilo je postavljeno na cestu bez okolnih vozila i pješaka. U ovome se scenariju provjerava mogućnost agenta da se održava u voznoj traci zadane rute. |
| 2 | Vozilo je postavljeno na cestu na kojoj se nalazi drugo vozilo koje nepomično stoji na cesti. U ovome se scenariju provjerava agentova mogućnost prestrojavanja u drugu traku pri nailasku na statičnu prepreku i mogućnost nastavka praćenja zadane rute. |
| 3 | Vozilo je postavljeno na cestu na kojoj se vozi drugo vozilo koje je sporije od upravljanog vozila. U ovome se scenariju provjerava agentova mogućnost prestrojavanja u drugu traku pri nailasku na dinamičnu prepreku i mogućnost nastavka praćenja zadane rute. |
| 4 | Vozilo je postavljeno na cestu na kojoj se nalaze druga vozila. Generira se 80 vozila koja se slobodno kreću po mapi kako bi se simulirao stvarni promet. U ovome se scenariju provjerava agentovo donošenje odluka u kompleksnim scenarijima. |

## 4.1. Rezultati dobiveni testiranjem modela na mapi na kojoj je model treniran

Svi scenariji u ovom potpoglavlju su testirani na mapi *Town04*.

**SCENARIJ 1**

Prikaz vozila i okoline za jedan primjer iz scenarija 1 prikazan je na slici 4.1. Crni dio slike predstavlja rutu koju vozilo treba pratiti. Ruta će i u idućim scenarijima biti označena crnom bojom. Rezultati dobiveni testiranjem modela su prikazani tablicom 4.1.

Tablica 4.1. Rezultati dobiveni primjenom modela za scenarij 1

| Mjerenje | DQN | DDPG |
|---|---|---|
| prosječni ukupno prijeđen put [m] | 579.37 | 560.49 |
| prosječno ukupno vrijeme vožnje [s] | 103.40 | 100.30 |
| prosječna ukupna nagrada [brojčana vrijednost] | 62626.1 | 60000.4 |
| prosječni broj pogrešaka [brojčana vrijednost] | 28.1 | 3.6 |
| prosječni put prijeđen u susjednim trakama [m] | 71.0 | 6.28 |
| prosječno vrijeme provedeno u susjednim trakama[s] | 13.90 | 1.20 |
| uspješnost provođenja vožnje bez sudara [%] | 80 | 80 |

DQN se pokazao kao uspješan algoritam za rješavanje problema u vožnji cestom bez okolnog prometa. Vozilo se održavalo u svojoj liniji većinu vremena provedenog u vožnji, međutim u dva slučaja je skrenulo s rute i krenulo se voziti u krug što objašnjava veliki broj pogrešaka u vožnji. Problem nastaje kada se ruta prebacuje iz jedne vozne trake u drugu jer funkcija iscrtavanja rute ne uspijeva iscrtati spoj kontrolne točke između trenutne i sljedeće vozne trake. Posljedica toga je da se dio rute ne vidi na ulaznoj slici semantičke kamere te se agent odlučuje na nasumične akcije. Ovaj problem je bio prisutan i u sljedećim scenarijima, međutim bolja funkcija za iscrtavanje rute unutar CARLA API-a trenutno ne postoji. DDPG je također u dva slučaja skrenuo s rute i zabio se u ogradu, međutim nije se počeo kretati u krug. Zbog toga je i prosječna ukupna nagrada za ovaj algoritam manja u usporedbi sa DQN algoritmom.

**Slika 4.1.** *Prikaz vozila i okoline u CARLA simulatoru za primjer iz scenarija 1*

**SCENARIJ 2**

Prikaz vozila i okoline za jedan primjer iz scenarija 2 prikazan je na slici 4.2. Rezultati dobiveni testiranjem modela su prikazani tablicom 4.2.

Tablica 4.2. Rezultati dobiveni primjenom modela za scenarij 2

| Mjerenje | DQN | DDPG |
|---|---|---|
| prosječni ukupno prijeđen put [m] | 383.42 | 499.35 |
| prosječno ukupno vrijeme vožnje [s] | 73.9 | 87.6 |
| prosječna ukupna nagrada [brojčana vrijednost] | 39442.9 | 5542.2 |
| prosječni broj pogrešaka [brojčana vrijednost] | 23 | 7.3 |
| prosječni put prijeđen u susjednim trakama [m] | 40.17 | 12.22 |
| prosječno vrijeme provedeno u susjednim trakama [s] | 7.7 | 2.4 |
| uspješnost provođenja vožnje bez sudara [%] | 40 | 60 |

DQN uspijeva održavati svoju voznu traku i pri nailasku na prepreku se prestrojava u drugu voznu traku. Problem nastaje pri povratku u traku previđenu rutom gdje vozilo počinje vijugati oko trake pokušavajući se stabilizirati. Ovo ponašanje dovodi do brojnih sudara ukoliko se vozna traka nalazi na samom rubu ceste gdje ima mnogo prepreka s jedne strane. DDPG se u ovim okolnostima

uspješnije vraća u svoju traku, no u nekim slučajevima se ona promaši te dolazi do sudara s okolnim preprekama.


**Slika 4.2.** *Prikaz vozila i okoline u CARLA simulatoru za primjer iz scenarija 2*

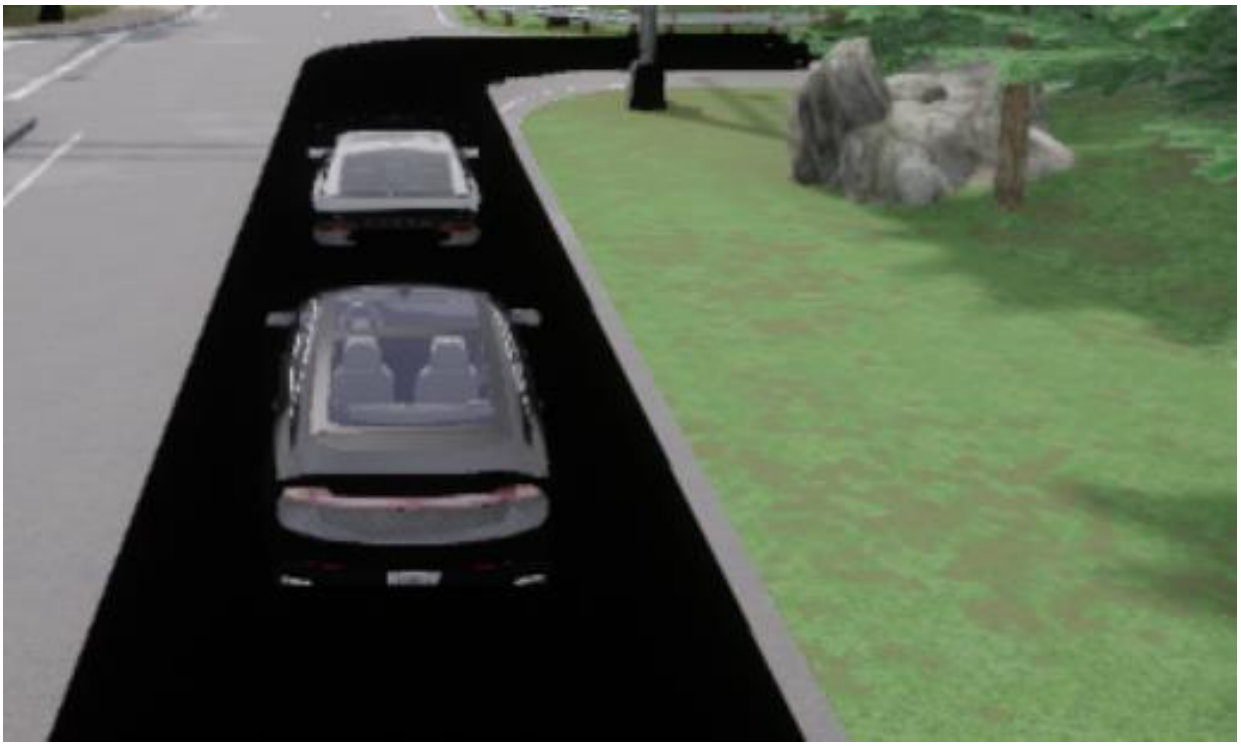
**SCENARIJ 3**

Prikaz vozila i okoline za jedan primjer iz scenarija 3 prikazan je na slici 4.3. Rezultati dobiveni testiranjem modela su prikazani tablicom 4.3.

Tablica 4.3. Rezultati dobiveni primjenom modela za scenarij 3

| Mjerenje | DQN | DDPG |
|---|---|---|
| prosječni ukupno prijeđen put [m] | 268.84 | 366.50 |
| prosječno ukupno vrijeme vožnje [s] | 31.3 | 41.4 |
| prosječna ukupna nagrada [brojčana vrijednost] | 3551.38 | 4964.58 |
| prosječni broj pogrešaka [brojčana vrijednost] | 26 | 8.6 |
| prosječni put prijeđen u susjednim trakama [m] | 55.45 | 14.20 |
| prosječno vrijeme provedeno u susjednim trakama [s] | 8.8 | 2.7 |
| uspješnost provođenja vožnje bez sudara [%] | 40 | 60 |

Oba algoritma se ponašaju na sličan način kako su se ponašala i u scenariju 2. Oba algoritma pokazuju pad u postotku uspješnosti provođenja vožnje bez sudara. Iz rezultata se vidi kako DDPG algoritam u više slučajeva uspijeva prestrojiti vozilo u susjednu traku, zaobići objekt u pokretu i vratiti se na rutu.



**Slika 4.3.** *Prikaz vozila i okoline u CARLA simulatoru za primjer iz scenarija 3*

**SCENARIJ 4**

Prikaz vozila i okoline za jedan primjer iz scenarija 4 prikazan je slikom 4.4. Rezultati dobiveni testiranjem modela su prikazani tablicom 4.4.

Tablica 4.4. Rezultati dobiveni primjenom modela za scenarij 4

| Mjerenje | DQN | DDPG |
|---|---|---|
| prosječni ukupno prijeđen put [m] | 21.3 | 24.5 |
| prosječno ukupno vrijeme vožnje [s] | 6.1 | 7.9 |
| prosječna ukupna nagrada [brojčana vrijednost] | -532.22 | -474.13 |
| prosječni broj pogrešaka [brojčana vrijednost] | 3 | 2 |
| prosječni put prijeđen u susjednim trakama [m] | 2.1 | 1.9 |
| prosječno vrijeme provedeno u susjednim trakama [s] | 1.1 | 1.0 |
| uspješnost provođenja vožnje bez sudara [%] | 0 | 0 |

Iz rezultata se može zaključiti kako oba modela nisu bila u stanju izvršiti scenarij uspješno. Oba modela su uspjela u tome da vozilo u početku nalazi u svojoj traci, međutim pri prestrojavanju se vozilo često zabijalo sa drugim vozilima u prometu. Razlog tomu je nedovoljna pokrivenost

situacija za vrijeme treniranja modela jer se za vrijeme treniranja nije koristila okolina sa tolikim brojem vozila u prometu. Također, s više epizoda treniranja bi se mogli postići bolji rezultati.

**Slika 4.4.** *Prikaz vozila i okoline u CARLA simulatoru za primjer iz scenarija 4*

## 4.2. Rezultati dobiveni testiranjem modela na mapi na kojoj model nije treniran

Svi scenariji u ovom potpoglavlju su provedeni na mapi *Town10*. Mapa predstavlja mješavinu značajki iz gradskih sredina sa neboderima, industrijskim zgradama, obalu, stambenim blokovima, javnim zgradama i bulevarima s drvoredima. Cestovna mreža mape sadrži niz različitih rasporeda raskrižja s različitim oznakama traka, prijelazima i vrstama signala. Prikaz mape iz ptičje perspektive dan je na slici 4.5.

**Slika 4.5.** *Prikaz mape Town10 iz ptičje perspektive*

**SCENARIJ 1**

Prikaz vozila i okoline za jedan primjer iz scenarija 1 prikazan je slikom 4.6. Rezultati dobiveni testiranjem modela nakon su prikazani tablicom 4.5.

Tablica 4.5. Rezultati dobiveni primjenom modela za scenarij 1

| Mjerenje | DQN | DDPG |
|---|---|---|
| prosječni ukupno prijeđen put [m] | 562.25 | 533.81 |
| prosječno ukupno vrijeme vožnje [s] | 105.12 | 102.98 |
| prosječna ukupna nagrada [brojčana vrijednost] | 42.494 | 55228.9 |
| prosječni broj pogrešaka [brojčana vrijednost] | 47 | 3.7 |
| prosječni put prijeđen u susjednim trakama [m] | 73.29 | 13.83 |
| prosječno vrijeme provedeno u susjednim trakama [s] | 14.88 | 5.54 |
| uspješnost provođenja vožnje bez sudara [%] | 50 | 50 |

**Slika 4.6.** *Prikaz vozila i okoline u CARLA simulatoru za primjer iz scenarija 1*

DQN je ostvario slične performanse kao i na mapi na kojoj je model treniran. Vozilo je uspješno moglo pratiti zadanu rutu, ali prilikom promjene vozne trake se u nekim slučajevima krenulo voziti u krug do isteka vremena epizode što objašnjava veliki broj pogrešaka u vožnji. DDPG je također ostvario slične rezultate, ali prilikom promjene vozne trake, vozilo je značajno više vijugalo kako bi se vožnja stabilizirala. Ovo ponašanje je dovelo i do većeg broja sudara jer se vozilo nije moglo na vrijeme izravnati na užim gradskim cestama. Oba algoritma su pokazala da je uspješnost algoritma lošija na mapi na kojoj se modeli nisu trenirali.

**SCENARIJ 2**

Prikaz vozila i okoline za jedan primjer iz scenarija 2 prikazan je slikom 4.7. Na slici se može vidjeti kako ruta prelazi preko kolnika na pješačku stazu i kroz zgradu. Ovo je greška u funkciji iscrtavnja rute CARLA simulatora koja iscrtava spoj između početne i krajne točke rute. Rezultati dobiveni testiranjem modela su prikazani tablicom 4.6.

Tablica 4.6. Rezultati dobiveni primjenom modela za scenarij 2

| Mjerenje | DQN | DDPG |
|---|---|---|
| prosječni ukupno prijeđen put [m] | 294.93 | 384.51 |
| prosječno ukupno vrijeme vožnje [s] | 56.8 | 67.45 |
| prosječna ukupna nagrada [brojčana vrijednost] | 3037.5 | 4262.5 |
| prosječni broj pogrešaka [brojčana vrijednost] | 29.7 | 9.2 |

| | | |
|---|---|---|
| prosječni put prijeđen u susjednim trakama [m] | 50.91 | 15.4 |
| prosječno vrijeme provedeno u susjednim trakama [s] | 8.2 | 3.2 |
| uspješnost provođenja vožnje bez sudara [%] | 30 | 40 |

Rezultati su slični kao i na mapi na kojoj su modeli trenirani, iako se prema svim vrijednostima može zaključiti kako su performanse modela nešto lošije.

**Slika 4.7.** *Prikaz vozila i okoline u CARLA simulatoru za primjer iz scenarija 2*

**SCENARIJ 3** – dinamična prepreka na cesti

Prikaz vozila i okoline za jedan primjer iz scenarija 3 prikazan je slikom 4.8. Rezultati dobiveni testiranjem modela su prikazani tablicom 4.7.

Tablica 4.7. Rezultati dobiveni primjenom modela za scenarij 3

| Mjerenje | DQN | DDPG |
|---|---|---|
| prosječni ukupno prijeđen put [m] | 280.16 | 365.28 |
| prosječno ukupno vrijeme vožnje [s] | 54.0 | 64.0 |
| prosječna ukupna nagrada [brojčana vrijednost] | 2885.6 | 4050.3 |
| prosječni broj pogrešaka [brojčana vrijednost] | 31.1 | 9.6 |
| prosječni put prijeđen u susjednim trakama [m] | 53.45 | 16.17 |
| prosječno vrijeme provedeno u susjednim trakama [s] | 8.6 | 3.3 |

| | | |
|---|---|---|
| uspješnost provođenja vožnje bez sudara [%] | 30 | 40 |

Rezultati su slični kao i na mapi na kojoj su modeli trenirani te se performanse nisu značajno promijenile. Valja naglasiti da se ovdje također kod oba modela može primjetiti kako su dobiveni rezultati nešto lošiji u usporebi s istim scenarijem provedenim na mapi na kojoj se provelo treniranje.

**Slika 4.8.** *Prikaz vozila i okoline u CARLA simulatoru za primjer iz scenarija 3*

**SCENARIJ 4**

Prikaz vozila i okoline za jedan primjer iz scenarija 4 prikazan je slikom 4.9. Rezultati dobiveni primjenom modela su prikazani tablicom 4.8.

Tablica 4.8. Rezultati dobiveni primjenom modela za scenarij 4

| Mjerenje | DQN | DDPG |
|---|---|---|
| prosječni ukupno prijeđen put [m] | 20.2 | 22.1 |
| prosječno ukupno vrijeme vožnje [s] | 5.9 | 7.5 |
| prosječna ukupna nagrada [brojčana vrijednost] | -541.57 | -499.69 |

| | | |
|---|---|---|
| prosječni broj pogrešaka [brojčana vrijednost] | 3 | 2 |
| prosječni put prijeđen u susjednim trakama [m] | 1.9 | 1.7 |
| prosječno vrijeme provedeno u susjednim trakama [s] | 1.2 | 1.1 |
| uspješnost provođenja vožnje bez sudara [%] | 0 | 0 |

Loši rezultati ovog scenarija su očekivani s obzirom na rezultate provedene na mapi na kojoj se provelo treniranje. Do sudara dolazi vrlo brzo, u prosjeku za šest sekundi, jer se vozilo pri približavanju vozilu ispred sebe pokušava prestrojiti u drugu voznu traku, bez obzira ima li u toj traci drugih vozila ili ne.

**Slika 4.9.** *Prikaz vozila i okoline u CARLA simulatoru za primjer iz scenarija 4*

# 5. ZAKLJUČAK

Autonomna vozila sve više istražuju i razvijaju zahvaljujući napretku tehnologije i inovacijama u području umjetne inteligencije. Jedan od glavnih problema u razvoju autonomnih vozila je problem upravljanja autonomnim vozilom u prometu. U ovom su diplomskom radu izgrađena dva algoritma upravljanja vozilom u simulatoru primjenom podržanog učenja. Prvi algoritam se zasniva na algoritmu dubokog Q-učenja (DQN), dok se drugi zasniva na algoritmu dubokog deterministčkog gradijenta politike (DDPG). Oba su algoritma izgrađena u Python programskom jeziku, a treniranje i testiranje modela se provelo unutar CARLA simulatora. Unutar simulatora je izgrađena vlastita okolina u kojoj je postavljeno vozilo na kojem su prikačeni razni senzori poput prednje semantičke kamere, senzora udarca i senzora napuštanja vozne trake. Agent predaje brojčane vrijednosti za papučicu gasa i kut zakreta volana preko CARLA Python API-a kako bi se moglo upravljati vozilom u simulatoru. Istrenirani modeli predviđaju brojčanu vrijednost kuta zakreta volana na temelju slike dobivene s prednje semantičke kamere vozila i značajki vožnje dobivenih iz simulatora. Testiranjem algoritama se ispitivalo agentovo donošenje odluka za određene scenarije u vožnji. Pratila se ukupna kumulativna nagrada, put i vrijeme koje je vozilo provelo vozeći bez sudara, te metrike o pogreškama koje je vozilo napravilo prateći određenu rutu. Rezultati pokazuju da su oba algoritma pogodna platforma za rješavanje problema upravljanja autonomnim vozilom.. DQN se ispostavio kao dobro rješenje za scenarije koji nisu kompleksni i gdje u okolini nema objekata u pokretu. DDPG je u tim scenarijima pokazivao znatno bolje rezultate. Iako se DDPG ispostavio kao bolje rješenje, oba algoritma nisu radila bez greške. U kompleksnijim scenarijima vožnje i vožnje na mapi na kojoj nisu trenirani, kod oba se algoritma znalo dogoditi da agent u potpunosti promaši odrediti ispravnu akciju za dano stanje. Razlog tomu je relativno mali broj odrađenih epizoda prilikom treniranja modela. Najsuvremeniji su algoritmi trenirani i do milijun, no zbog tehničkih se limitacija broj epizoda nije mogao povećati, a vjerojatno bi rezultiralo boljim performansama modela. Također, moguće poboljšanje bi se moglo dobiti finim podešavanjem vrijednosti hiperparametara trening procesa.

# LITERATURA


[1] „Weights & Biases“, *W&B*. https://wandb.ai/ivangoncharov/AVs-report/reports/The-Role-Of-Machine-Learning-In-Autonomous-Vehicles--VmlldzoyNTExMDE3 (pristupljeno 01. kolovoz 2023.).

[2] C. Bartneck, C. Lütge, A. Wagner, i S. Welsh, „Autonomous Vehicles“, 2021, str. 83–92. doi: 10.1007/978-3-030-51110-4_10.

[3] „The State of Level 3 Autonomous Driving in 2023“, *AUTOCRYPT*, 13. siječanj 2023. https://autocrypt.io/the-state-of-level-3-autonomous-driving-in-2023/ (pristupljeno 22. kolovoz 2023.).

[4] Y. Zhang, S. Balochian, P. Agarwal, V. Bhatnagar, i O. Houshia, „Artificial Intelligence and Its Applications“, *Math. Probl. Eng.*, sv. 2014, str. 10, tra. 2014, doi: 10.1155/2014/840491.

[5] R. H. Jhaveri, A. Revathi, K. Ramana, R. Raut, i R. K. Dhanaraj, „A Review on Machine Learning Strategies for Real-World Engineering Applications“, *Mob. Inf. Syst.*, sv. 2022, str. e1833507, kol. 2022, doi: 10.1155/2022/1833507.

[6] V. Nasteski, „An overview of the supervised machine learning methods“, *HORIZONS.B*, sv. 4, str. 51–62, pros. 2017, doi: 10.20544/HORIZONS.B.04.1.17.P05.

[7] „Machine Learning Paradigms - Introduction to Machine Learning“. https://www.wolfram.com/language/introduction-machine-learning/ (pristupljeno 19. srpanj 2023.).

[8] R. S. Sutton i A. G. Barto, „Reinforcement Learning: An Introduction“, 2015.

[9] M. Hausknecht i P. Stone, „Deep Recurrent Q-Learning for Partially Observable MDPs“. arXiv, 11. siječanj 2017. doi: 10.48550/arXiv.1507.06527.

[10] D. Mwiti, „10 Real-Life Applications of Reinforcement Learning“, *neptune.ai*, 21. srpanj 2022. https://neptune.ai/blog/reinforcement-learning-applications (pristupljeno 19. srpanj 2023.).

[11] A. Yang, „What is Exploration vs. Exploitation in Reinforcement Learning?“, *Medium*, 25. srpanj 2022. https://angelina-yang.medium.com/what-is-exploration-vs-exploitation-in-reinforcement-learning-a3b96dcc9503 (pristupljeno 19. srpanj 2023.).

[12] „What is Bellman Equation in Reinforcement Learning“. https://www.tutorialspoint.com/what-is-bellman-equation-in-reinforcement-learning (pristupljeno 27. srpanj 2023.).

[13] „Markov Decision Process Explained | Built In“. https://builtin.com/machine-learning/markov-decision-process (pristupljeno 19. srpanj 2023.).

[14] V. V. PV, „What is Reinforcement Learning?“:, *Medium*, 02. kolovoz 2020. https://medium.com/@vishnuvijayanpv/what-is-reinforcement-learning-e5dc827c8564 (pristupljeno 19. srpanj 2023.).

[15] D. Johnson, „Reinforcement Learning: What is, Algorithms, Types & Examples“, 27. svibanj 2023. https://www.guru99.com/reinforcement-learning-tutorial.html (pristupljeno 19. srpanj 2023.).

[16] „Markov decision process“, *Wikipedia*. 24. svibanj 2023. Pristupljeno: 27. srpanj 2023. [Na internetu]. Dostupno na: https://en.wikipedia.org/w/index.php?title=Markov_decision_process&oldid=1156832827

[17] „Deep Reinforcement Learning: Guide to Deep Q-Learning“, *MLQ.ai*, 20. travanj 2021. https://www.mlq.ai/deep-reinforcement-learning-q-learning/ (pristupljeno 27. srpanj 2023.).

[18] „What is Deep Learning? | IBM“. https://www.ibm.com/topics/deep-learning (pristupljeno 20. srpanj 2023.).

[19] „Introduction to Deep Learning“, *GeeksforGeeks*, 01. lipanj 2018. https://www.geeksforgeeks.org/introduction-deep-learning/ (pristupljeno 20. srpanj 2023.).

[20] B. R. Kiran *i ostali*, „Deep Reinforcement Learning for Autonomous Driving: A Survey“. arXiv, 23. siječanj 2021. Pristupljeno: 01. kolovoz 2023. [Na internetu]. Dostupno na: http://arxiv.org/abs/2002.00444

[21] G. Boesch, „Deep Reinforcement Learning: How It Works and Real World Examples“, *viso.ai*, 23. veljača 2023. https://viso.ai/deep-learning/deep-reinforcement-learning/ (pristupljeno 21. srpanj 2023.).

[22] „Deep Reinforcement Learning: Definition, Algorithms & Uses“. https://www.v7labs.com/blog/deep-reinforcement-learning-guide, https://www.v7labs.com/blog/deep-reinforcement-learning-guide (pristupljeno 21. srpanj 2023.).

[23] „Deep Q-Learning | An Introduction To Deep Reinforcement Learning“. https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/?ref=mlq.ai (pristupljeno 31. srpanj 2023.).

[24] L. Weng, „Policy Gradient Algorithms“, 08. travanj 2018. https://lilianweng.github.io/posts/2018-04-08-policy-gradient/ (pristupljeno 01. kolovoz 2023.).

[25] D. Karunakaran, „Deep Deterministic Policy Gradient(DDPG) — an off-policy Reinforcement Learning algorithm“, *Intro to Artificial Intelligence*, 25. studeni 2020. https://medium.com/intro-to-artificial-intelligence/deep-deterministic-policy-gradient-ddpg-

an-off-policy-reinforcement-learning-algorithm-38ca8698131b (pristupljeno 01. kolovoz 2023.).

[26] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, i M. Riedmiller, „Deterministic Policy Gradient Algorithms“.

[27] C. Yoon, „Deep Deterministic Policy Gradients Explained“, *Medium*, 23. svibanj 2019. https://towardsdatascience.com/deep-deterministic-policy-gradients-explained-2d94655a9b7b (pristupljeno 01. kolovoz 2023.).

[28] T. P. Lillicrap *i ostali*, „Continuous control with deep reinforcement learning“. arXiv, 05. srpanj 2019. Pristupljeno: 01. kolovoz 2023. [Na internetu]. Dostupno na: http://arxiv.org/abs/1509.02971

[29] S. Tang, H. Shu, i Y. Tang, „Research on decision-making of lane-changing of automated vehicles in highway confluence area based on deep reinforcement learning“, u *2021 5th CAA International Conference on Vehicular Control and Intelligence (CVCI)*, lis. 2021, str. 1–8. doi: 10.1109/CVCI54083.2021.9661204.

[30] J. Liao, T. Liu, X. Tang, X. Mu, B. Huang, i D. Cao, „Decision-Making Strategy on Highway for Autonomous Vehicles Using Deep Reinforcement Learning“, *IEEE Access*, sv. 8, str. 177804–177814, 2020, doi: 10.1109/ACCESS.2020.3022755.

[31] Y. Cai, S. Yang, H. Wang, C. Teng, i L. Chen, „A Decision Control Method for Autonomous Driving Based on Multi-Task Reinforcement Learning“, *IEEE Access*, sv. 9, str. 154553–154562, 2021, doi: 10.1109/ACCESS.2021.3126796.

[32] Y. Zhang, B. Gao, L. Guo, H. Guo, i H. Chen, „Adaptive Decision-Making for Automated Vehicles Under Roundabout Scenarios Using Optimization Embedded Reinforcement Learning“, *IEEE Trans. Neural Netw. Learn. Syst.*, sv. 32, izd. 12, str. 5526–5538, pros. 2021, doi: 10.1109/TNNLS.2020.3042981.

[33] M. Molaie, A. Amirkhani, i H. Kashiani, „Auto-Driving Policies in Highway based on Distributional Deep Reinforcement Learning“, u *2021 5th International Conference on Pattern Recognition and Image Analysis (IPRIA)*, tra. 2021, str. 1–6. doi: 10.1109/IPRIA53572.2021.9483545.

[34] Y. Zhang, L. Guo, B. Gao, T. Qu, i H. Chen, „Deterministic Promotion Reinforcement Learning Applied to Longitudinal Velocity Control for Automated Vehicles“, *IEEE Trans. Veh. Technol.*, sv. 69, izd. 1, str. 338–348, sij. 2020, doi: 10.1109/TVT.2019.2955959.

[35] Ó. Pérez-Gil *i ostali*, „Deep reinforcement learning based control for Autonomous Vehicles in CARLA“, *Multimed. Tools Appl.*, sv. 81, izd. 3, str. 3553–3576, sij. 2022, doi: 10.1007/s11042-021-11437-3.

[36] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, i V. Koltun, „CARLA: An Open Urban Driving Simulator“, u *Proceedings of the 1st Annual Conference on Robot Learning*, PMLR, lis. 2017, str. 1–16. Pristupljeno: 02. kolovoz 2023. [Na internetu]. Dostupno na: https://proceedings.mlr.press/v78/dosovitskiy17a.html

[37] „About“, *OpenCV*. https://opencv.org/about/ (pristupljeno 02. kolovoz 2023.).

[38] „CMake“. https://cmake.org/ (pristupljeno 02. kolovoz 2023.).

# SAŽETAK

U radu je detaljnije opisano podržano učenje i određeni postojeći algoritmi podržanog učenja temeljeni na dubokom učenju. Ideja rada je primijeniti opisane algoritme na problem upravljanja autonomnim vozilima u prometu unutar simulatora vožnje. U okviru ovog rada su izrađena dva rješenja za autonomno upavljanje vozilom unutar simulatora: prvo je zasnovano na algoritmu dubokog Q-učenja (DQN), a drugo na algoritmu dubokog deterministčkog gradijenta politike (DDPG). Agent dubokog učenja šalje brojčane vrijednosti za gas i zakret volana preko CARLA Python API-a do CARLA simulatora koji zatim primijenjuje primljene podatake na vozilo u simulatoru. Pritisak papučice gasa se održava u specifičnim granicama, a neuronske mreže predviđaju kut zakreta volana iz početne slike dobivene preko semantičke kamere postavljene na prednju stranu vozila. Vlastiti algoritmi su izrađeni u Python programskom jeziku, te su treniranja i testiranja provedena unutar CARLA simulatora na različitim scenarijima vožnje. Evaluacija izgrađenih algoritama provedena je pomoću predloženih metrika: prosječno prijeđen put, prosječno ukupno vrijeme koje je vozilo provelo vozeći kroz epizodu, prosječna ukupna kumulativna nagrada dobivena kroz epizodu, prosječan broj pogrešaka koje agent učini tijekom jedne epizode, prosječno prijeđen put i prosječno vrijeme provedeno u susjednim trakama kada ta akcija nije optimalna.

**Ključne riječi**: CARLA simulator, DDPG, DQN, podržano učenje, upravljanje autonomnim vozilom

# DEVELOPING AUTONOMOUS VEHICLE CONTROL ALGORITHMS IN A SIMULATOR BASED ON REINFORCEMENT LEARNING

## ABSTRACT

This master's thesis describes reinforcement learning in more detail and certain existing reinforcement learning algorithms based on deep learning. The idea of this thesis is to apply the described algorithms to the problem of autonomous vehicle control in traffic within a driving simulator. As part of this thesis, two solutions were developed for autonomous vehicle driving within the simulator: one based on the Deep Q-Learning (DQN) algorithm and the other on the Deep Deterministic Policy Gradient (DDPG) algorithm. The deep learning agent sends numerical values for throttle and steering through the CARLA Python API to the CARLA simulator, which then applies the received data to the vehicle in the simulator. The amount of throttle is maintained within specific limits, and neural networks predict the angle of rotation of the steering wheel from the initial image obtained through the semantic camera placed on the front of the vehicle. Algorithms were developed in the Python programming language, and training and testing were carried out in the CARLA simulator on different driving scenarios. The evaluation of the developed algorithms was carried out using the proposed metrics: the average distance traveled, the average total time the vehicle spent driving throughout the episode, the average total cumulative reward obtained throughout the episode, the average number of errors made by the agent during one episode, the average distance traveled and the average time spent in adjacent lanes when that action is not optimal.

**Keywords**: Autonomous Vehicle Control, CARLA simulator, DDPG, DQN, Reinforcement Learning

# ŽIVOTOPIS

Dorian Činčurak rođen je 28. siječnja 1998. godine u Osijeku. Nakon završene III. Gimnazije u Osijeku, 2016. godine ostvaruje upis na preddiplomski sveučilišni studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku. Godine 2021. stječe akademski naziv sveučilišni prvostupnik (lat. *baccalaureus*) inženjer računarstva. Iste godine upisuje diplomski sveučilišni studij automobilsko računarstvo i komunikacije i uz to postaje stipendist instituta RT-RK u Osijeku te sljedeće godine postaje stipendist tvrtke TTTech Auto. Aktivno se služi engleskim jezikom u govoru i pismu.

_____

Potpis autora

# PRILOZI

### P.3.1.1. Detalji instalacije CARLA simulatora

Prije nego što se može preuzeti i instalirati CARLA simulator, potrebno je instalirati sljedeće programe na osobno računalo:

- *Cmake* - koristi se za kontrolu procesa prevođenja izvornog koda korištenjem jednostavnih konfiguracijskih datoteka neovisnih o platformi i prevoditelju, te generiranje izvornih makefileova i radnih prostora koji se mogu koristiti u okruženju po izboru [38].
- *Git* - besplatni distribuirani sustav kontrole verzija otvorenog koda za upravljanje CARLA repozitorijima
- *Make* - alat koji kontrolira generiranje izvršnih datoteka i drugih ne-izvornih datoteka programa iz izvornih datoteka programa.
- *7ZIP* - softver za kompresiju datoteka, potreban je za automatsku dekompresiju datoteka sredstava i sprječava pogreške tijekom vremena izgradnje zbog neispravnog ili djelomičnog ekstrahiranja velikih datoteka
- *Python3 x64* - glavni skriptni jezik u CARLA-i.

Nakon što se instaliraju svi ovi programi, potrebno je preuzeti i instalirati Python ovisnosti preko pip3, službenim upraviteljem paketa i naredba pip za Python 3, upisivajući u konzolu sljedeće:

*pip3 install --user setuptools*

*pip3 install --user wheel*

Potrebno je zatim instalirati Visual Studio 2019, integrirano razvojno okruženje (IDE) koje se može koristiti za pisanje, uređivanje, uklanjanje pogrešaka i izradu koda. U radu se koristi besplatna „Community“ verzija. Prilikom instalacije je potrebno instalirati sljedeće elemente pomoću ugrađenog instalera: Windows 8.1 SDK, x64 Visual C++ Toolset i .NET framework 4.6.2.

Zatim se podešava Unreal Engine. Prvo se mora napraviti GitHub korisnički račun i povezati ga sa Unreal Engine korisničkim računom. Kako bi preuzeli modificarni UE4 *fork*, u terminalu se prebacimo na lokaciju gdje želimo spremiti podatke i kloniramo *carla* granu:

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

Nakon toga se pokreću konfiguracijske skripte Setup.bat i GenerateProjectFiles.bat nakon čega u Visual Studio-u možemo izgraditi preuzeti kod s opcijama „Development Editor“, „Win64“ i „UnrealBuildTool“.

Kada smo izgradili modificirani UE4, kloniramo „master“ granu CARLA projekta:

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

U CARLA korijenskom direktoriju se zatim preuzima najnovija sredstva komandom *Update.bat.* Sada se može prevesti Python API klijent upisujući u konzolu naredbu *make PythonAPI.* Konačno, simulator je spreman za rad. Prije svakog pokretanja potrebno je prevesti server naredbom *make launch* u korijenskom direktoriju CARLA simulatora koji automatski pokreće simulator nakon uspješnog prevođenja.

### P.3.1.2. Tablica specifikacije računala i korištenih verzija programa

| | |
|---|---|
| OS | Windows 10 |
| Procesor | Intel(R) Core(TM) i7-6700 |
| RAM | 16 GB |
| GPU | NVIDIA GeForce GTX 1070 Ti |
| CARLA | 0.9.13 |
| Python | 3.8.10 |
| tensorflow-gpu | 2.10.0 |
| opencv-python | 4.7.0.72 |
| 7-Zip | 23.01 |
| Cmake | 3.26.3 |
| Git | 2.41.0.3 |
| NVIDIA CUDA | 11.2 |
| Visual Studio | Community 2019 |

| | |
|---|---|
| .NET framework | 4.6.2. |
| Windows SDK | 8.1 |