Učenje mobilnog agenta u simulacijskom okruženju pomoću podržanog učenja i primjena u stvarnom okruženju

Zahirović, Bruno

Master's thesis / Diplomski rad

2024

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:596709

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2025-03-09**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

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

Sveučilišni diplomski studij Računarstvo
Izborni blok Robotika i umjetna inteligencija

UČENJE MOBILNOG AGENTA U SIMULACIJSKOM OKRUŽENJU POMOĆU PODRŽANOG UČENJA I PRIMJENA U STVARNOM OKRUŽENJU

Diplomski rad

Bruno Zahirović

Osijek, 2024.

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

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

Ocjena diplomskog rada na sveučilišnom diplomskom studiju

Ime i prezime pristupnika:	Bruno Zahirović
Studij, smjer:	Sveučilišni diplomski studij Računarstvo
Mat. br. pristupnika, god.	D-1258R, 07.10.2021.
JMBAG:	0165079604
Mentor:	doc. dr. sc. Petra Pejić
Sumentor:	Matej Džijan, univ. mag. ing. comp.
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	izv. prof. dr. sc. Ratko Grbić
Član Povjerenstva 1:	doc. dr. sc. Petra Pejić
Član Povjerenstva 2:	prof. dr. sc. Robert Cupec
Naslov diplomskog rada:	Učenje mobilnog agenta u simulacijskom okruženju pomoću podržanog učenja i primjena u stvarnom okruženju
Znanstvena grana diplomskog rada:	**Umjetna inteligencija (zn. polje računarstvo)**
Zadatak diplomskog rada:	Zadatak rada je u simulacijskom okruženju naučiti mobilnog agenta kako se kretati primjenom podržanog učenja te primijeniti naučeno na stvarnom mobilnom agentu. Tema rezervirana za: Bruno Zahirović Sumentor s FERIT-a: Matej Džijan

Datum ocjene pismenog dijela diplomskog rada od strane mentora:	07.09.2024.
Ocjena pismenog dijela diplomskog rada od strane mentora:	Izvrstan (5)
Datum obrane diplomskog rada:	12.9.2024.
Ocjena usmenog dijela diplomskog rada (obrane):	Izvrstan (5)
Ukupna ocjena diplomskog rada:	Izvrstan (5)
Datum potvrde mentora o predaji konačne verzije diplomskog rada čime je pristupnik završio sveučilišni diplomski studij:	12.09.2024.

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

IZJAVA O IZVORNOSTI RADA

Osijek, 12.09.2024.

Ime i prezime Pristupnika:	Bruno Zahirović
Studij:	Sveučilišni diplomski studij Računarstvo
Mat. br. Pristupnika, godina upisa:	D-1258R, 07.10.2021.
Turnitin podudaranje [%]:	9

Ovom izjavom izjavljujem da je rad pod nazivom**: Učenje mobilnog agenta u simulacijskom okruženju pomoću podržanog učenja i primjena u stvarnom okruženju**

izrađen pod vodstvom mentora doc. dr. sc. Petra Pejić

i sumentora Matej Džijan, univ. mag. ing. comp.

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

Potpis pristupnika:

SADRŽAJ

1. UVOD

Riječ 'robot' je po prvi puta predstavljena 1920. godine kroz predstavu „*Rossumovi Univerzalni Roboti"* češkog pisca Karela Čapeka [1]. Dolazi od češke riječi „*robota*" koja se prevodi kao prisilni rad te je predstavljena kao ideja koja će umjesto čovjeka odrađivati teške fizičke poslove. Prema [2], današnja najpoznatija definicija robota dana je od strane Američkog instituta za robotiku koja glasi: „Robot je multifunkcionalan, reprogramabilan manipulator dizajniran za pomicanje materijala, alata, specijaliziranih uređaja ili dijelova koristeći se pokretima integriranim varijabilnim programiranjem s ciljem izvedbe različitih zadataka." Današnje robote svrstavamo u nekoliko kategorija: čovjekoliki, mobilni, robotski manipulatori te hibridi. Primjenjuju se u raznim industrijama poput poljoprivrede, medicine, logistike i sličnih [3]. Iako se tehnike korištene u ovom radu mogu primijeniti na sve kategorije robota koji se primjenjuju u svim navedenim industrijama, ovaj rad fokusira se na primjenu tehnika na mobilnim robotima.

Mobilne se robote može klasificirati u nekoliko glavnih kategorija[4]:

1. Kopneni
    a) Mobilni robot na kotačima
    b) Hodajući mobilni roboti
    c) Mobilni roboti na gusjenični pogon
    d) Hibridni mobilni roboti
2. Zračni
3. Vodeni
4. Ostali – Hibridni

Jedan od glavnih zadataka mobilnog robota, neovisno kojoj kategoriji pripada, je od početne točke doći do svog cilja, bilo to autonomno ili pomoću intervencije čovjeka. Za sve različite kategorije i potkategorije mobilnih robota, potrebno je razviti programski kod kojim se omogućava njegovo kretanje, ovisno o njegovom pogonu i mogućnostima. To je vremenski skupa operacija koja se mora adaptirati različitom hardveru svakog pojedinačnog mobilnog robota, kao i okruženju u kojem će djelovati.

U svrhu izbjegavanja spomenute vremenski skupe operacije razvijanja programskog koda za specijalizirani hardver, korisno može biti strojno učenje, odnosno metoda strojnog učenja zvana podržano učenje. Podržano učenje je tehnika strojnog učenja zasnivana na nagrađivanju željenog i kažnjavanja neželjenog ponašanja. Koristi se u raznim industrijama, poput automatizacije robota,

procesiranja fotografija, medicini, videoigrama, kontroliranju prometa i mnogo drugih. Rezultat provedbe strojnog učenja je agent, koji ima mogućnost percipiranja svog okruženja kao i djelovanja u istom čime uči obavljati željene radnje kroz veliki broj pokušaja i pogreški. Stoga je umjesto razvijanja programskog koda za kretnju svakog različitog hardvera kojim je ostvaren mobilni robot moguće definirati okruženje u kojem će agent podržanog učenja naučiti zadanu radnju na bilo kojem modelu robota. Nakon okruženja potrebno je odrediti kakvo ponašanje robota je poželjno i pustiti agenta da samostalno dođe do rješenja koje odgovara željenom ponašanju, bez potrebe za ulaganjem vremena u razvoj programskog koda. Zanimljiva priroda rješavanja problema treniranja agenta i primjene na stvarnog robota je motivacija za ovaj diplomski rad, kao i sama korisnost njegova ostvarivanja.

Cilj ovog rada je istražiti i testirati postojeće metode kojima je moguće automatizirati rješenje problema kretanja mobilnog robota na način gdje nije potrebno ulaganje velike količine vremena i resursa na razvijanje programskog koda kojim je mobilnom robotu omogućena kretnja. Rad se fokusira na simulaciju hodajućeg četveronožnog mobilnog robota te korištenje podržanog učenja za treniranje neuronske mreže kojom se omogućuje samostalno učenje modela mobilnog robota za željenu akciju. Četveronožni robot izabran je zbog pasivne stabilnosti takve vrste robota te zbog dostupnih materijala za izradu takvog robota pomoću komponenti koje su lako nabavljive. Iz tih je razloga izabran model robota tvrtke *Petoi* zvan *Nybble*, koji je detaljnije predstavljen u četvrtom poglavlju rada. U svrhu ostvarenja ovog rada, odabrana akcija je jednostavna ali korisna za svakog mobilnog robota – kretanje unaprijed. Na kraju, potrebno je primijeniti rezultate simulacije i treniranja na stvarnog mobilnog robota, kako bi se dokazalo da je moguće ukloniti potrebu za pisanje programskog koda za svaku specifičnu akciju te svaki specifičan okoliš u kojem mobilni robot može djelovati.

U drugom poglavlju predstavljena su već dostupna rješenja prenošenja naučenog simulacijom u stvarni svijet. Treće poglavlje opisuje teorijsku podlogu podržanog učenja koje je korišteno za treniranje agenta za djelovanje u okruženju. U detalje je opisan i specifičan korišteni algoritam podržanog učenja kojim je dobiven agent, PPO (eng. *Proximal Policy Optimization*). U četvrtom je poglavlju opisan niz alata kojima je diplomski rad ostvaren, dok je u petom poglavlju iznesen slijed implementacije projekta diplomskog rada. Na kraju, šesto poglavlje predstavlja rezultate i analizu rezultata projekta diplomskog rada, gdje su u detalje opisani dobiveni rezultati te na koji način je rezultate moguće poboljšati.

1.1. Zadatak diplomskog rada

U diplomskom radu potrebno je predstaviti primjer mobilnog robota te ga u simulacijskom okruženju pomoću podržanog učenja naučiti samostalnom kretanju. Treniranjem mobilnog agenta u okruženju potrebno je dobiti model kojeg je moguće prebaciti na stvarni hardver, odnosno prebaciti naučeno iz simulacije u stvarni svijet, gdje stvarni model mobilnog robota treba imati mogućnost repliciranja naučene akcije mobilnog robota u simulacijskom okruženju, bez pisanja dodatne programske logike za naučenu akciju.

2. PREGLED PODRUČJA PRENOŠENJA NAUČENOG SIMULACIJOM U STVARNI SVIJET

Ovo poglavlje opisuje postojeće realizacije prenošenja naučenog ponašanja u simulacijskom okruženju na stvarne robote. Postojeća rješenja se analiziraju i koriste za usporedbu sa željenim ciljem diplomskog rada.

Korištenje simulacijskog okruženja kao zamjenu za stvarni hardver prilikom razvoja nekog algoritma vrlo je privlačno rješenje ako ga je moguće upotrijebiti na stvarnom hardveru, zbog smanjenja cijene i rizika razvoja – simulacijski hardver ne može se uništiti ako se prilikom razvoja dogodi greška. Još privlačnije rješenje je ako algoritam za željeno ponašanje nije potrebno razvijati. Koristeći se metodom strojnog učenja možemo dobiti željene rezultate bez ulaganja vremena u razvoj algoritma.

Prema radu *Sim-to-Real Transfer of Robotic Control with Dynamics Randomization* [5], primjena podržanog učenja u svrhu rješavanja raznovrsnih problema u području kompleksne kontrole robota koristeći simulacijsko okruženje je česta pojava, ali mnogo simulacijskih rješenja nije realizirano na stvarnom hardveru. U radu je spomenuto kako je ključni dio korištenja podržanog učenja za robusno i korisno rješenje agentovo adekvatno istraživanje okoliša u kojem djeluje te se koriste tehnikom randomizacije domene kako bi dobili dovoljno robusno rješenje za primjenu na stvarnom robotu – robotskoj ruci sa sedam stupnjeva slobode. Domena u ovom slučaju predstavlja sve parametre okoline u kojoj se agent trenira pa randomizacija domene predstavlja postavljanje nasumičnih parametara okoliša poput koeficijenta trenja i otpora zraka, kao i parametara samog robota poput snage motora i ukočenosti robota unutar nekog zadanog intervala vrijednosti. Iako je tehnika randomizacije domene vrlo korisna metoda i alat koji je korišten u ovom diplomskom radu ju podržava, nije korištena prilikom treniranja agenta u simulacijskom okruženju kako bi se smanjila kompleksnost.

Wenshuai Zhao i suradnici s finskog sveučilišta Turku u radu *Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: a Survey* [6] predstavljaju nekoliko glavnih tema koje se trenutno koriste u području prenošenja naučenog iz simulacije u stvarni svijet koristeći metodu podržanog učenja. Identifikacija sustava je metoda izgradnje preciznog matematičkog modela fizičkog sustava u svrhu što realističnije simulacije istog sustava kako bi se dobili bolji rezultati. Iako je moguće dobiti znatno bolje rezultate provođenjem identifikacije sustava, izvedba samog simulatora u kojem će se odvijati treniranje agenta koristeći podržano učenje jednako je bitna, tako da je moguće imati matematički model sustava koji je vrlo precizan i dobiti loše rezultate zbog

realizacije samog simulatora. Već spomenuta metoda randomizacije domene također je navedena u ovom radu, gdje se umjesto pažljivog i temeljitog modeliranja parametara stvarnog svijeta koriste visoko nasumični parametri simulacije kako bi se pokrili svi mogući ishodi u stvarnom svijetu te kako bi se dobio model robusniji na varijacije stvarnih parametara, poput trenja i gravitacije. Zatim, rad predstavlja metodu adaptacije domene, gdje je cilj koristiti podatke iz izvorne domene za poboljšanje performansi naučenog modela na specifičnoj domeni u kojoj su podaci teško dostupni. Predstavljeno je još nekoliko metoda, ali metoda koja je najviše vezana uz ovaj diplomski rad je takozvana *Zero-shot* metoda prijenosa. Spomenuta metoda je najizravniji način prijenosa znanja iz simulacije u stvarni svijet. Osnovna stavka uspjeha korištenja *Zero-shot* metode prijenosa je robustan i realističan simulator kako bi se model mogao primijeniti na stvarni svijet.

U radovima *Sim2Real Transfer for Reinforcement Learning without Dynamics Randomization* [7], *Humanoid-Gym: Reinforcement Learning for Humanoid Robot with Zero-Shot Sim2Real Transfer* [8] i *Test and Evaluation of Quadrupedal Walking Gaits through Sim2Real Gap Quantification* [9] predstavljeni su različiti prijenosi naučenog u simulaciji u pravi svijet. U radu [7] je riječ o robotskom manipulatoru gdje se u radu fokus stavlja na identifikaciju sustava za dobivanje što boljih rezultata. Rad [8] se fokusira na treniranje i prijenos naučenog u stvarni svijet koristeći čovjekolikog robota i *Zero-shot* metodu prijenosa iz simulacije u stvarni svijet, kao i u druge simulacije. U [9] za model robota korišten je četveronožni robot, a rad predstavlja novo razvijeni algoritam autora čiji je cilj identificirati razlike između simulacijskog okruženja i stvarnog svijeta u svrhu kompenziranja razlika za što bolje rezultate prijenosa iz simulacije u stvarni svijet.

Suradnjom između sveučilišta „ETH Zurich“ i tvrtke „Nvidia“ u radu *Learning to Walk in Minutes Using Massively Parallel Deep Reinforcement Learning* [10] predstavljen je kompletan prijenos naučenog iz simulacije u stvarni svijet. Predstavljeno je simulacijsko okruženje, ulazi i izlazi modela, simulator, parametri korišteni za simulaciju, kao i testiranje naučenog u stvarnom svijetu. Posebno zanimljivo je to što je korišteno simulacijsko okruženje koje je korišteno i u ovom diplomskom radu te je uz to model robota koji je korišten u spomenutom radu postao i jedan od temeljnih primjera korištenja podržanog učenja za treniranje agenta nad četveronoškim robotom u spomenutom simulacijskom okruženju, detaljnije opisano u četvrtom poglavlju. Na slici 2.1 prikazan je korišteni četveronoški robot *ANYmal* u pravom svijetu nakon prijenosa naučenog u simulaciji.

Slika 2.1. *Četveronožni robot ANYmal u stvarnom okruženju [10]*

3. PODRŽANO UČENJE

U ovom poglavlju je detaljno opisano podržano učenje, metoda koja je korištena za treniranje mobilnog agenta u simulacijskom okruženju u diplomskom radu. Predstavljena je teorijska pozadina iza podržanog učenja, podjela i podtipovi podržanog učenja, kao i detalji specifičnog korištenog algoritma za ostvarivanje rezultata diplomskog rada.

Prema *Reinforcement Learning: An Introduction* [11], podržano učenje se zasniva na učenju kako odraditi neku radnju, odnosno kako mapirati situacije specifičnim radnjama, u cilju maksimiziranja nagrade agenta u numeričkom obliku. Opisani proces se ne ostvaruje na način da se agentu govori koje radnje poduzimati u specifičnim stanjima, već se od agenta očekuje da ih on otkrije poduzimajući radnje i nauči koje radnje mu donose najveću nagradu. Također, agentu je u cilju maksimizirati kumulativnu nagradu na način da svaka akcija ne utječe samo na trenutačnu nagradu, već i na sve buduće nagrade. Podržano učenje se od ostalih oblika strojnog učenja poput nadziranog i nenadziranog učenja razlikuje po sljedećim dvjema karakteristikama: pristup pokušaja i pogreške (eng. *trial-and-error*) te maksimizacija kumulativne nagrade, odnosno maksimiziranje nagrade u budućnosti, ne samo za sljedeći korak.

Glavne komponente svakog algoritma podržanog učenja su:

- Agent
- Okoliš
- Politika ponašanja
- Nagrada
- Funkcija vrijednosti
- Model okoliša – opcionalno

Agent je komponenta koja djeluje u okolišu, poduzimajući akcije kojima će doći do definiranog cilja [11]. Agent mora imati mogućnost prikupiti podatke o stanju svog okoliša, kao i mogućnost poduzimanja akcija u okolišu. Također, mora imati definirani cilj ili ciljeve koji su povezani uz stanje okoliša. Prilikom korištenja metode podržanog učenja, ključan cilj je odabrati dobar omjer istraživanja i eksploatacije (eng. *exploration vs. exploitation*). Kako bi agent akumulirao što veću nagradu, mora preferirati radnje koje je već pokušao u prošlosti i koje su donijele relativno veće nagrade od ostalih, ali isto tako je potrebno te radnje i otkriti tako što će pokušati poduzeti radnje koje nije pokušao u prošlosti u slučaju da postoji bolja radnja koja povećava kumulativnu nagradu.

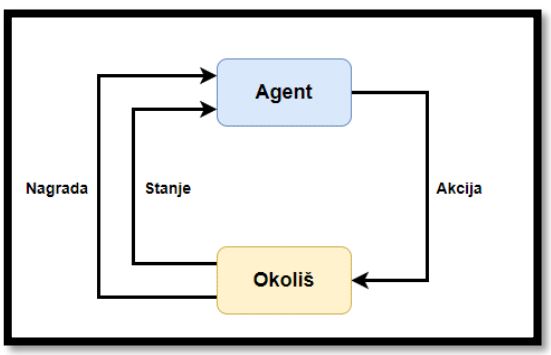
Okoliš okružuje agenta, dok agent iz njega izvlači informacije i stanja iz kojih odlučuje koju radnju poduzeti u kojem trenutku.

Politika ponašanja definira kako će se agent ponašati u danom trenutku u odnosu na trenutno stanje njegova okoliša. Može se definirati kao preslikavanje iz percipiranih stanja okoliša na radnje koje agent poduzima u tim stanjima. Politika ponašanja je temeljni dio agenta podržanog učenja jer je ona samostalno dovoljna za određivanje agentova ponašanja.

Nagrada definira cilj problema koji se rješava podržanim učenjem. U svakom poduzetom koraku agenta, okoliš agentu definira nagradu u obliku numeričke vrijednosti. Agentov krajnji cilj je skupiti što veću sumu nagrada iz okoliša – kumulativna nagrada. Nagrada tako definira dobre i loše događaje za agenta. Nagrada se koristi i za izmjenu politike ponašanja – ako radnja izabrana politikom ponašanja donosi nisku nagradu, politiku je moguće izmijeniti tako da izabere neku drugu akciju u toj situaciji u budućnosti.

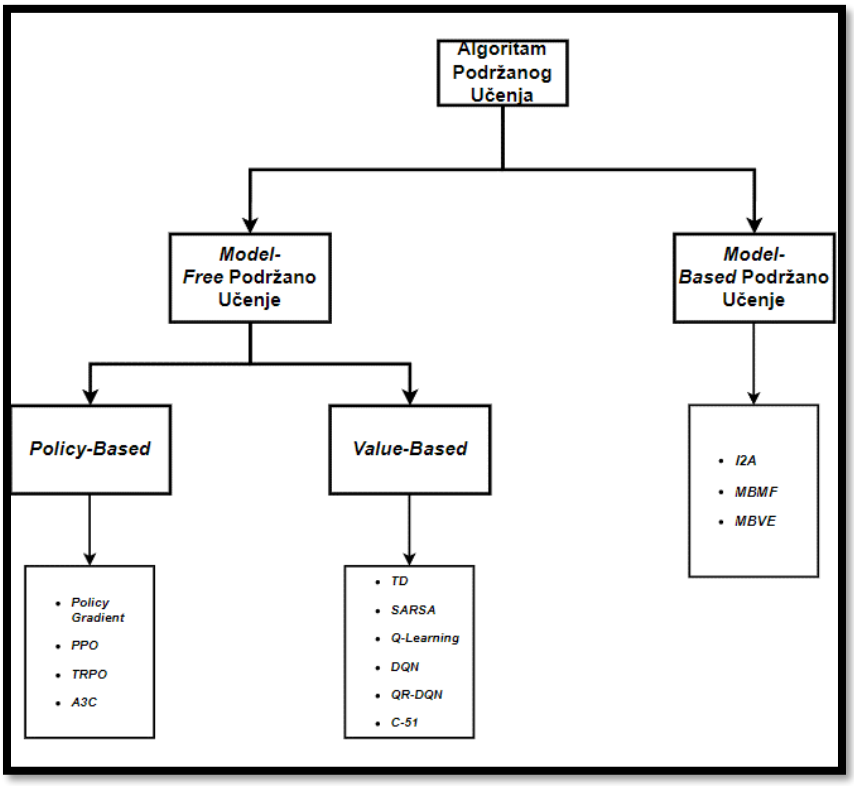
Funkcija vrijednosti za razliku od nagrade određuje što je za agenta dobro gledajući u dugoročnom smislu. Ona označava kumulativnu nagradu koju agent može očekivati u budućnosti iz trenutnog stanja. Iz nekog stanja je moguće svaki puta dobiti nisku trenutnu nagradu, ali ako je ona popraćena nizom visokih nagrada, funkcija vrijednosti će vraćati visoku vrijednost za taj niz akcija, uzimajući u obzir akumuliranu nagradu na duge staze.

Model okoliša se koristi u svrhu planiranja budućih radnji u okolišu na temelju predviđanja stanja okoliša prije nego se ta stanja dogode. Uzimajući u obzir stanje okoliša i radnju agenta, model može predvidjeti sljedeće stanje i nagradu. Podržano učenje može se tako podijeliti na metode temeljenje na modelima (eng. *model-based methods*) i metode bez modela (eng. *model-free methods*). Slika 3.1 prikazuje osnovni scenarij koji se događa kroz svaki korak metode podržanog učenja.



Slika 3.1. *Osnovni scenarij metode podržanog učenja (po uzoru na [12])*

Kako je spomenuto iznad, podržano učenje može se podijeliti na *model-based* i *model-free* metode, a isto tako, postoje i dodatne podjele ovih dviju metoda. Na slici 3.2 prikazana je potpuna podjela metoda podržanog učenja, uz specifične algoritme svake podjele.



Slika 3.2. *Taksonomija algoritama podržanog učenja [13]*

Model-based algoritmi baziraju se na izgradnji modela okoliša koristeći podatke interakcije agenta s okolišem u kojem djeluje, kako je navedeno u *Benchmarking Model-Based Reinforcement Learning* [14]. Izgradnja modela okoliša znatno smanjuje kompleksnost uzorkovanja stanja okoliša, ali izgradnja točnog i preciznog modela okoliša je kompliciran problem u specifičnim situacijama. U drugu ruku, *model-free* algoritmi se ne oslanjaju na izgradnju kompletnog modela okoliša, već koriste interakciju agenta s okolišem za direktno učenje politike ponašanja ili funkcije vrijednosti. *Model-free* algoritmi uspješno se koriste u raznim područjima, uključujući animaciju pokreta, video igrama te u robotici. Njihov nedostatak je visoka kompleksnost uzorkovanja stanja okoliša te u mnogo slučajeva ostaju limitirani na simulacijskoj domeni iz istog razloga. Ovaj rad koristi se *model-free* izvedbom algoritma podržanog učenja, detaljnije opisano u nastavku poglavlja, nakon predstavljanja samog temelja podržanog učenja – Markovljevog procesa odlučivanja.

3.1. Markovljev proces odlučivanja

Prema *A Survey of Actor-Critic Reinforcement Learning: Standard and Natural Policy Gradients* [15], Markovljev proces odlučivanja je *tuple*

$$MDP: \langle X, U, f, \rho \rangle \tag{3-1}$$

gdje X predstavlja prostor stanja, U prostor radnji, $f: X \times U \times X \rightarrow [0, \infty\rangle$ te $\rho: X \times U \times X \rightarrow \mathbb{R}$. Stohastički proces koji se pokušava kontrolirati je opisan funkcijom gustoće vjerojatnosti prijelaza stanja f. Vjerojatnost dostizanja stanja x_{k+1} u području $X_{k+1} \subseteq X$ iz stanja x_k nakon što je poduzeta radnja u_k je

$$P(x_{k+1} \in X_{k+1} | x_k, u_k) = \int_{X_{k+1}} f(x_k, u_k, x') dx'. \tag{3-2}$$

Nakon svakog prijelaza u stanje x_{k+1}, agent dobiva trenutnu nagradu

$$r_{k+1} = \rho(x_k, u_k, x_{k+1}) \tag{3-3}$$

koja ovisi o prethodnom stanju, trenutnom stanju te radnji koja je poduzeta. Radnja u_k u stanju x_k je preuzeta iz stohastičke politike $\pi: X \times U \rightarrow [0, \infty\rangle$. Cilj agenta podržanog učenja je pronaći politiku ponašanja π koja maksimizira očekivanu vrijednost funkcije g trenutnih nagrada primljenih prateći politiku π:

$$J(\pi) = E\{g(r1, r2, \ldots) | \pi\}. \tag{3-4}$$

Tokom učenja, agent mora estimirati funkciju J za danu politiku ponašanja π, odnosno mora odraditi takozvanu estimaciju politike ponašanja. Rezultat ove estimacije je vrijednosna funkcija i moguće ju je definirati dvjema jednadžbama:

$$V^{\pi}(x) = E\{\sum_{k=0}^{\infty} \gamma^k r_{k+1} | x_0 = x, \pi\} \tag{3-5}$$

i

$$Q^{\pi}(x, u) = E\{\sum_{k=0}^{\infty} \gamma^k r_{k+1} | x_0 = x, u_o = u, \pi\}, \tag{3-6}$$

gdje je $x_0 \in X$ početno stanje, $u_o \in U$ početna akcija, a $\gamma \in [0, 1\rangle$ predstavlja popusni faktor nagrade (eng. *reward discount factor*). Funkcija (3-5) ovisi samo o stanju x i pretpostavlja da je politika ponašanja π popraćena iz tog stanja. S druge strane, funkcija (3-6) također ovisi o stanju x, ali

obavlja radnju u odabranu u početnom stanju umjesto da je radnja generirana politikom ponašanja π. Nakon što se prvi prijelaz na sljedeće stanje izvede, π odlučuje odabir sljedećih radnji. Veza između ove dvije jednadžbe uz danu vrijednosnu funkciju je

$$V^{\pi}(x) = E\{Q^{\pi}(x,u)|u \sim \pi(x,\cdot)\}, \tag{3-7}$$

gdje $\pi(x,\cdot)$ predstavlja funkciju distribucije vjerojatnosti. Uz malo manipulacije, jednadžbe (3-5) i (3-6) mogu se pretvoriti u rekurzivni oblik. Za funkciju vrijednosti stanja (3-5) je to

$$V^{\pi}(x) = E\{\rho(x,u,x') + \gamma V^{\pi}(x')\}, \tag{3-8}$$

dok je za funkciju vrijednosti radnje-stanja (3-6) to

$$Q^{\pi}(x,u) = E\{\rho(x,u,x') + \gamma Q^{\pi}(x',u')\}, \tag{3-9}$$

gdje je u jednadžbi (3-8) x' dobiven iz funkcije distribucije vjerojatnosti $f(x,u,\cdot)$, a u iz funkcije distribucije vjerojatnosti $\pi(x,,\cdot)$. Vrijednost x' u jednadžbi (3-9) je dobivena koristeći funkciju distribucije vjerojatnosti $f(x,u,\cdot)$ dok je vrijednost u' dobivena iz distribucije $\pi(x',\cdot)$. Ovakve rekurzivne jednadžbe nazivaju se Bellmanove jednadžbe. Podržano učenje zasniva se na rješavanju ovakvih jednadžbi, a one opisuju kako se mijenja kumulativna nagrada za danu radnju. Prema Bellmanovim jednadžbama, kumulativna nagrada za danu radnju je jednaka nagradi za trenutnu radnju kombinirana s predviđenom nagradom svih budućih stanja. Za optimalni slučaj prosječne nagrade, Bellmanove jednadžbe mogu se zapisati ovako:

$$V^{*}(x) + J^{*} = \max_{u}\{\rho(x,u,x') + V^{*}(x')\} \tag{3-10}$$

i

$$Q^{*}(x,u) + J^{*} = \max_{u}\left\{\rho(x,u,x') + \max_{u'} Q^{*}(x',u')\right\}. \tag{3-11}$$

V^{*}predstavlja optimalnu funkciju vrijednosti stanja, Q^{*} optimalnu funkciju vrijednosti radnje-stanja, a J^{*} je optimalna srednja nagrada kada je korištena optimalna politika π^{*}, odnosno $J(\pi^{*})$.

3.2. *Model-Free* algoritmi podržanog učenja

Prema [11], *model-free* algoritmi se primarno oslanjaju na učenje – iz svakog mogućeg stanja pokušavaju pronaći koje akcije agent treba poduzeti kako bi se maksimizirala kumulativna nagrada te kroz *trial-and-error* pristup pamte koje akcije rezultiraju najvećim nagradama. Cilj je maksimizirati nagradu promatrajući posljedice akcija, za razliku od *model-based* algoritama koji

se fokusiraju na razumijevanje dinamike okoliša. *Model-free* algoritmi se dijele na dvije potkategorije. *Model-free* algoritmi bazirani na vrijednosti (eng. *value-based*) ne spremaju zasebnu politiku ponašanja. Oni politiku ponašanja izvode iz funkcije vrijednosti. Prema *Bridging the Gap Between Value and Policy Based Reinforcement Learning* [16], prednost *value-based* metoda je to što imaju mogućnost učenja na bilo kojoj putanji koja je uzorkovana iz okoliša, tako da postoji mogućnost korištenja podataka prikupljenih iz ostalih izvora, poput doprinosa stručnjaka. Ova mogućnost ih čini više učinkovitima po pitanju broja uzorkovanja potrebnih za stabilno treniranje naspram druge metode, *mode-free* algoritama bazirani na politici ponašanja (eng. *policy-based*). Algoritmi bazirani na politici ponašanja se ne bave izgradnjom funkcije vrijednosti stanja, već je fokus stavljen na izgradnju politike ponašanja, koja predstavlja vjerojatnost poduzimanja specifične akcije u trenutnom stanju tako da se maksimizira kumulativna nagrada. *Policy-based* algoritmi pogodniji su za visokodimenzionalne ili kontinuirane prostore djelovanja te uz to mogu naučiti i stohastičke politike ponašanja, koje su korisnije od determinističkih. *Policy-based* metode se još nazivaju i metode samostalnog glumca (eng. *actor-only*), a *value-based* metode još i metode samostalnog kritičara (eng. *critic-only*).

Predstavljene potkategorije *model-free* metoda su pogodne za različite primjene, a u zadnje vrijeme se sve više istražuje kombinacija ovih metoda. Takozvane glumac-kritičar (eng. *actor-critic*) metode kombiniraju prednosti obiju potkategorija [16]. Dok parametrizirani glumac donosi prednost procesiranja kontinuiranih radnji bez potrebe za optimizacijom funkcije vrijednosti, kritičar pridonosi ovoj kombinaciji tako da glumca opskrbljuje nisko-varijantnim znanjima performansi. Specifičnije, kritičarova procjena predviđenog povrata vrijednosti omogućuje glumcu ažuriranje politike s gradijentima koji imaju nižu varijancu, ubrzavajući proces učenja. Glumac-kritičar metode uobičajeno imaju bolja svojstva konvergencije optimumu u odnosu na metode zasebnog kritičara. Iz tog razloga se većina primjera dostupnih u simulacijskom okruženju korištenom u ovom radu koristi metodama glumac-kritičar, kao i izvedba samog rada. Neke od glumac-kritičar izvedbi algoritama uključuju A2C, A3C, Q-prop i GAE o kojima se može više saznati u radovima [17], [18], [19] i [20]. Također, PPO algoritam je još jedna izvedba glumac-kritičar algoritma i on se koristi za izvedbu ovog diplomskog rada te je detaljnije opisan u sljedećem potpoglavlju.

3.2.1. PPO

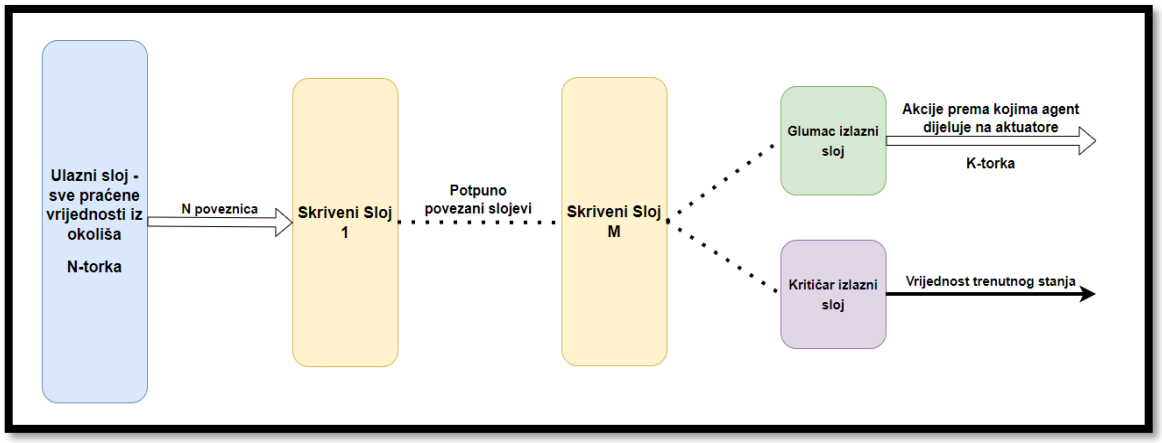
Proksimalna optimizacija politike (eng. *Proximal policy optimization*) je prema *Truly Proximal Policy Optimization* [21] jedan od najuspješnijih algoritama dubokog podržanog učenja koji postiže *state-of-the-art* performanse nad raznolikim skupom problema. PPO se smatra

nadogradnjom na dobro poznati algoritam TRPO - optimizacija politike područja povjerenja (eng. *trust policy region optimization*), opisan u radu *Trust Region Policy Optimization* [22]. Razlika naspram TRPO algoritma je ta što PPO algoritam uvelike smanjuje kompleksnost algoritma uvođenjem mehanizma isječka (eng. *clipping*) koji omogućuje PPO algoritmu da koristi optimizaciju prvog reda poput gradijentnog spusta, naspram optimizacije drugog reda koju koristi algoritam TRPO. Mehanizam isječka se koristi u svrhu sprječavanja mijenjanja politike ponašanja ako se ona nalazi izvan opsega isijecanja što povećava stabilnost i konvergenciju prema vjerojatnijim rješenjima. Mehanizam isječka može se definirati kao funkcija stanja i radnje:

$$L_t^{CLIP}(\theta) = \begin{cases} (1-\epsilon)A_t & r_t(\theta) \leq 1-\epsilon, \quad A_t < 0 \\ (1+\epsilon)A_t & r_t(\theta) \geq 1+\epsilon, \quad A_t > 0 \\ r_t(\theta)A_t, & inače \end{cases} \tag{3-12}$$

gdje je $[1-\epsilon, 1+\epsilon]$; $0 < \epsilon < 1$ opseg isijecanja, θ politika ponašanja, r_t je omjer vjerojatnosti između trenutne politike ponašanja i prošle politike ponašanja θ_{old}, a A_t predstavlja procijenjenu vrijednost prednosti u odnosu na prethodno stanje i radnju. PPO ograničava izmjenu politike ponašanja isijecajući omjer vjerojatnosti između nove i stare politike ponašanja. Uvjeti prikazani u jednadžbi (3-12) su takozvani uvjeti isijecanja (eng. *clipping conditions*). Kada je omjer vjerojatnosti između trenutne i prošle politike ponašanja r_t izvan granice za isijecanje, gradijent $L_t^{CLIP}(\theta)$ će biti nula. Time se sprječava da pomak $r_t(\theta)$ bude izvan granice isijecanja.

U izvedbi korištenog simulatora te za potrebe ovog rada, glumac-kritičar svojstva izvedena su neuronskom mrežom. Glumac i kritičar dijele skrivene slojeve neuronske mreže, kao i ulaze koji predstavljaju stanje okoliša u kojem agent djeluje. Iz neuronske mreže postoje dva izlazna sloja, jedan koji predstavlja akcije koje agent treba poduzeti u okolišu koje određuje glumac te drugi čija vrijednost predstavlja procijenjenu vrijednost trenutnog stanja, odnosno očekivanu kumulativnu vrijednost iz trenutnog stanja okoliša. Opći oblik opisane neuronske mreže prikazan je na slici 3.3.



Slika 3.3. *Glumac-Kritičar neuronska mreža*

Kako je moguće vidjeti na slici 3.3, glumac i kritičar imaju odvojene izlazne slojeve te koriste istu neuronsku mrežu. Ulazni stoj sastoji se od N vrijednosti prikupljenih iz okoliša, koji ulaze u potpuno povezane skrivene slojeve. Izlazni sloj glumca sastoji se od K vrijednosti koje predstavljaju distribuciju vjerojatnosti, ali s obzirom da se sve odvija u kontinuiranom prostoru radnje, te vrijednosti se direktno vežu na specifične radnje koje agent, u kontekstu ovog rada, primjenjuje na aktuatore modela robota. Robot ovog rada koristi se pozicijskim pogonom, pa je potrebno pomoću vrijednosti dobivene od strane glumca izračunati pozicije aktuatora koje agent može primijeniti na aktuatore robota:

$$T = T_{old} + (Ca \cdot dt), \tag{3-13}$$

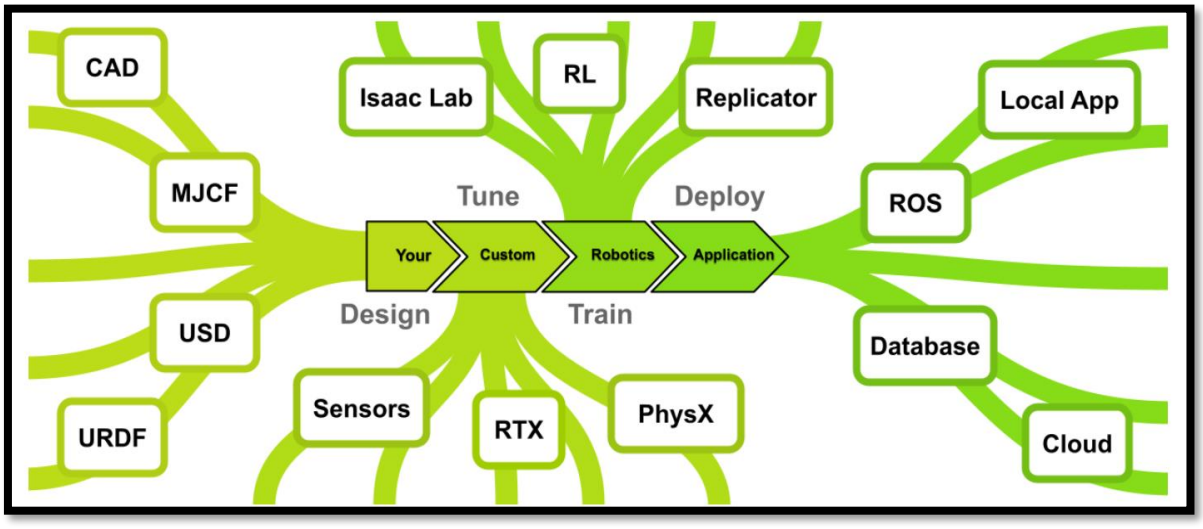
gdje je T vektor pozicija aktuatora koje agent primjenjuje u trenutnom koraku, T_{old} vektor pozicija aktuatora u prošlom koraku, Ca skalirani vektor radnji i dt vremenski korak simulacije. Specifičnosti neuronske mreže, poput hiperparametara i strukture mreže, bit će predstavljeni u petom poglavlju koje se bavi samom izvedbom zadatka rada, dok će u sljedećem poglavlju biti predstavljeni korišteni alati te datoteke potrebne za provedbu rada.

4. RAZVOJNI ALATI I POMOĆNE DATOTEKE

Ovo poglavlje opisuje korištene alate te datoteke koje opisuju robota korištenog za eksperimentalni dio rada. Opisuje se simulator korišten za provedbu algoritma podržanog učenja, model robota, alat za programiranje upravljača robota te alat za pretvorbu modela neuronske mreže u oblik koji je moguće koristiti na upravljaču, uz pomoćne skripte koje su potrebne kako bi se sve spojilo u jednu cjelinu.

4.1. Isaac Sim

Prema dokumentaciji [23], *Isaac Sim* je softverska platforma izgrađena od temelja za mogućnost podržavanja sve više robotiziranog i automatiziranog svijeta i to je proizvod tvrtke *Nvidia*. Cilj *Isaac Sim* platforme je na što lakši način omogućiti dizajniranje, podešavanje, treniranje i prijenos autonomnih agenta na stvarne robote. U ovom radu korištene su mogućnosti pretvorbe URDF datoteka (eng. *Universal Robot Description File*) u USD (eng. *Universal Scene Description*) datoteke kojima simulator upravlja, a isto tako je *Isaac Sim* simulacijsko okruženje kojim se koristi alat *Isaac Gym* za provođenje algoritma podržanog učenja. Platforma pruža fleksibilne API-je za C++ i Python programske jezike te ju je moguće u projekt integrirati u različitim stupnjevima, ovisno o potrebama projekta. Na slici 4.1 prikazana je sistemska arhitektura *Isaac Sim* softverske platforme.

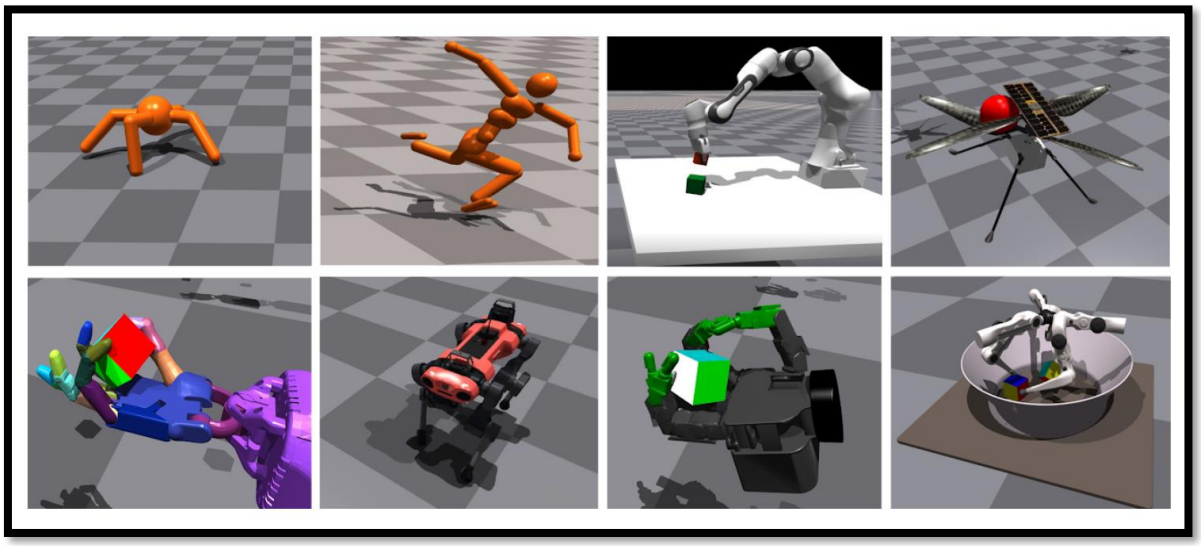


Slika 4.1. *Sistemska arhitektura Isaac Sim softverske platforme* [23]

Isaac Sim koristi se *PhysX*, softverom za simuliranje fizike na grafičkoj jedinici za procesiranje (GPU). Koristeći *PhysX* je stoga moguće postići masivno paralelno treniranje unutar alata *Isaac Gym*, koje uvelike ubrzava proces treniranja pomoću podržanog učenja.

4.2. Isaac Gym

Isaac Gym je simulacijsko okruženje fizike za istraživanje koristeći podržano učenje [24] te je kao i *Isaac Sim* proizvod tvrtke *Nvidia*. *Isaac Gym* okruženje moguće je koristiti za različite zadatke u svijetu robotike, koristeći se snagom GPU-a za paralelizirano učenje. Simulacija fizike kao i treniranje neuronske mreže politike ponašanja odvija se direktno na GPU-u te direktno komuniciraju koristeći *PyTorch* [25] tenzore bez potrebe za prolaženje kroz procesor, eliminirajući uska grla predstavljena niskim brojem jezgri procesora. Samim time se postižu mnogo kraća vremena treniranja za kompleksne robotske zadatke na jednom GPU-u. Slika 4.2 predstavlja različite robote u procesu treniranja koristeći *Isaac Gym*.



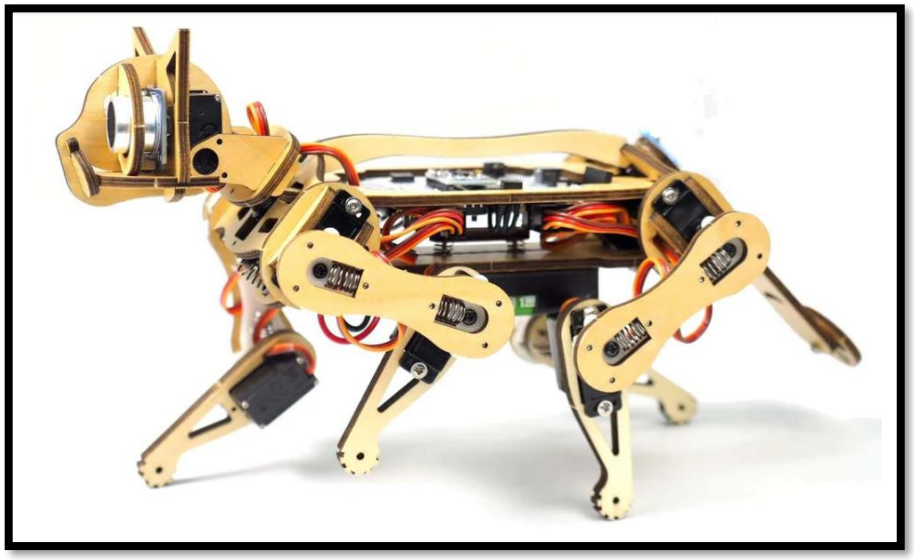
Slika 4.2. *Primjeri treniranja različitih modela robota s različitim zadacima* [26]

Koristeći *Isaac Gym*, postoji mogućnost treniranja agenta u simulatoru i prebacivanje modela u stvarni svijet, na stvarnog robota, kako je to prikazano u radu [10], što je i cilj ovog rada, tako da je *Isaac Gym* pogodno rješenje za treniranje i izgradnju modela agenta koji će upravljati stvarnim robotom, predstavljenim u sljedećem potpoglavlju.

4.3. Petoi Nybble

Tvrtka *Petoi* je proizvođač futurističkih bioničkih robota-ljubimaca za djecu i odrasle s primjenama u različitim područjima. Roboti tvrtke *Petoi* su slobodno dostupni, s visokim performansama i uz to su programibilni [27]. Robot *Nybble* tvrtke *Petoi* je četveronoški robot koji poprima oblik mačke. *Nybble* se prodaje kao zaseban proizvod, ali svi modeli i dijelovi besplatno su dostupni kao 3D modeli tako da je robota moguće izgraditi samostalno. Uz dopuštenje osnivača

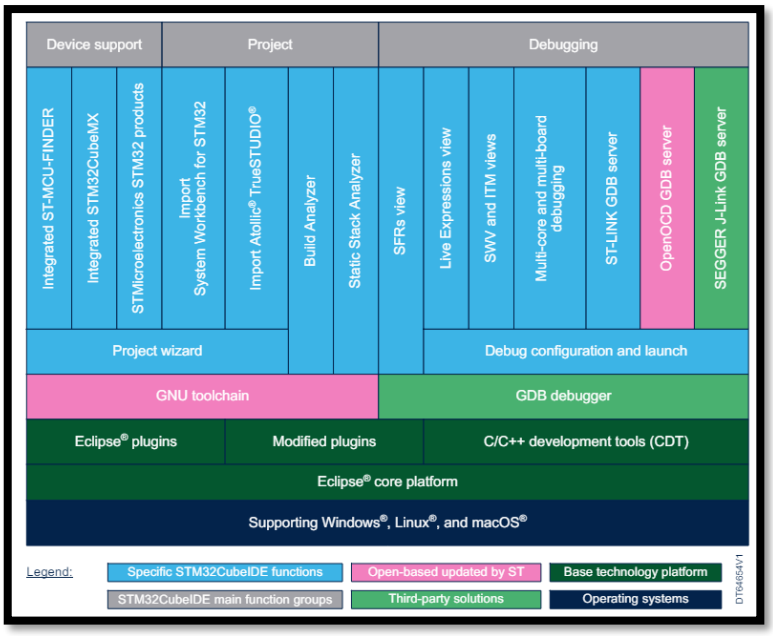
tvrtke *Petoi*, model *Nybble* robota korišten je u ovom diplomskom radu, koristeći dostupne materijale za 3D ispis strukturnih dijelova robota, uz koje su korišteni MG90S servo motori kao aktuatori robota te NUCLEO-L476RG razvojna pločica tvrtke *STMicroelectronics* kao mikroupravljač na kojeg je prevedena neuronska mreža agenta koji upravlja aktuatorima robota. Prikaz modela robota *Nybble* nalazi se na slici 4.3.



Slika 4.3. *Prikaz modela robota Nybble* [27]

4.4. STM32 CUBE IDE

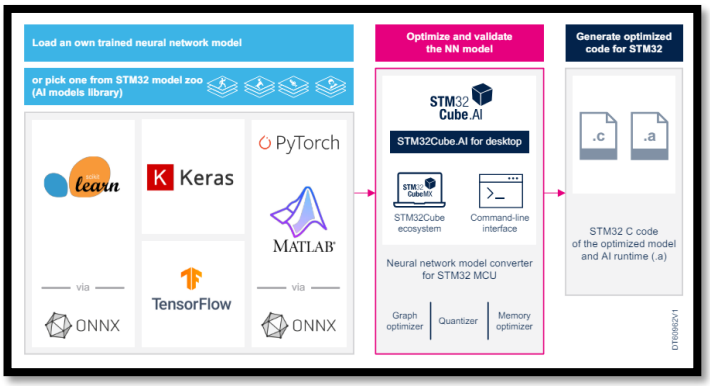
Tvrtka *STMicroelectronics* u [28] opisuje *STM32CubeIde* kao sveobuhvatan alat za razvoj na više operacijskih sustava koji sadrži naprednu platformu razvoj C/C++ softvera koristeći proizvode bazirane na *STM32* arhitekturi. Sadrži mogućnost konfiguracije perifernih uređaja, generacije programskog koda, kompilacije programskog koda te povezivanja i otklanjanja pogrešaka ujedinjeno na jednom mjestu. Koristeći *STM32CubeIde* moguće je projekt pripremiti za odabrani mikroupravljač uz grafičko sučelje za podešavanje svih potrebnih perifernih uređaja, nakon čega je moguće generirati programski kod koji slijedi prethodnu konfiguraciju za brz i učinkovit početak razvoja. Kako je spomenuto u prošlom potpoglavlju, odabrana je NUCLEO-L476RG razvojna pločica koja služi kao upravljač robota *Nybble*, na koju je prevedena neuronska mreža koja upravlja radnjama robota, koristeći alat tvrtke *STMicroelectronics* koji je opisan u sljedećem potpoglavlju. Slika 4.4 prikazuje ključne značajke *STM32CubeIde* razvojnog alata.



Slika 4.4. *Ključne značajke razvojnog alata STM32CubeIde* [28]

4.5. STM32 X-CUBE-AI

X-CUBE-AI tvrtke *STMicroelectronics* je softver za generiranje optimiziranog C programskog koda za *STM32* mikroupravljače i sučelja za neuronske mreže [29]. Ovaj softver koristi se u sklopu razvojne platforme *STM32CubeIde* kao dodatni modul, kojim se omogućuje učitavanje raznih modela neuronskih mreža, optimizacija i validacija neuronske mreže uz računanje potrebnih resursa koji su potrebni kako bi neuronska mreža mogla biti korištena na mikroupravljaču te konačno generiranje optimiziranog C programskog koda neuronske mreže te svih potrebnih sučelja kako bi se inferencija mogla odvijati na mikroupravljaču. *X-CUBE-AI* podržava razne okvire za duboko strojno učenje, poput *Keras*-a, *TensorFlow*-a te *PyTorch*-a korištenog u ovom radu. Na slici 4.5 prikazan proces kojim se dolazi do generacije optimiziranog programskog koda neuronske mreže koristeći alat *X-CUBE-AI*.



Slika 4.5. *Prikaz tijeka rada u alatu X-CUBE-AI* [29]

4.6. Pomoćne skripte i alati

Kako bi se cijeli projekt mogao uklopiti, dostupne datoteke je potrebno pretvoriti u točne oblike. Za potrebe treniranja, potrebno je stvoriti spomenutu USD datoteku koja opisuje scenu zajedno s modelom robota. Koristeći platformu *Isaac Sim*, moguće je učitati URDF datoteku te iz nje stvoriti potrebnu USD datoteku za potrebe treniranja, ali dostupni materijali tvrtke *Petoi* sadrže opis robota *Nybble* u obliku XACRO datoteke koja omogućuje dodavanje makro naredbi unutar URDF datoteke, koja je često korištena u ROS metaoperacijskom sustavu korištenom za razvoj robotskih sustava. S obzirom da *Isaac Sim* ne podržava XACRO format, potrebno je dostupnu datoteku opisa *Nybble* robota pretvoriti u URDF format kako bi se mogla stvoriti USD datoteka. U projekt je dodana skripta kojom se to omogućuje, *xacro2urdf.py*, koja kao ulaz prima putanju do XACRO datoteke i stvara novu datoteku URDF oblika koju je moguće koristiti unutar *Isaac Sim* platforme.

Nakon što je model neuronske mreže istreniran koristeći *PyTorch*, izlazna datoteka modela je u PTH formatu. Datoteka PTH sadrži rječnik stanja *PyTorch* modela, uključujući težine (eng. *weights*), sklonosti (eng. *biases*) te ostale parametre modela. Iako alat *X-CUBE-AI* podržava okvir dubokog učenja *PyTorch*, ne podržava PTH format datoteke modela, već je potrebno učitati datoteku u ONNX formatu. Iz tog razloga je potrebno nakon uspješnog treniranja modela koristiti pomoćnu skriptu *pthToONNXConverter.py* dodanu u projekt, koja kao ulaz prima ime eksperimenta i putanju do datoteke u PTH formatu i stvara novu datoteku u ONNX formatu istog imena. Nakon pretvorbe datoteke u ONNX format, model je moguće učitati u alat *X-CUBE-AI*.

Spomenute skripte pisane su u *Python* programskom jeziku i obje su dostupne na *GitHub* repozitoriju projekta[1] navedenom u prilogu rada.

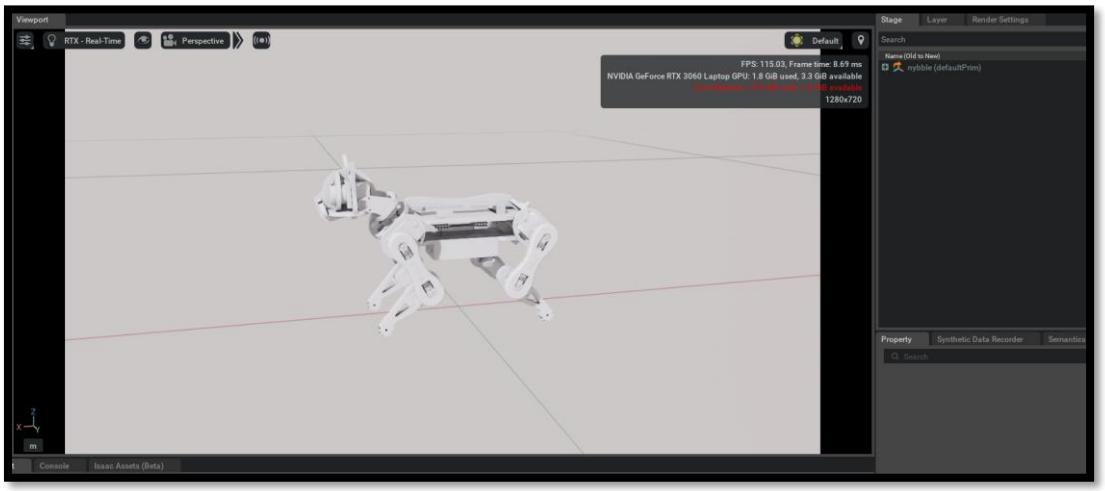
[1] https://github.com/bruno-zahirovic/OmniIsaacGymEnvs-NybbleSim2Real

5. IMPLEMENTACIJA PROJEKTA

U ovom poglavlju predstavljen je tijek izvedbe projekta, od ubacivanja modela robota u simulator, treniranja agenta za upravljanje robotom, podešenim parametrima treniranja, do izgradnje robota u stvarnom svijetu, prebacivanja neuronske mreže na upravljačku ploču robota uz simulaciju ulaznih parametara neuronske mreže i prilagođavanje izlaza neuronske mreže.

5.1. Uvoz modela robota u simulator

Kako je spomenuto u četvrtom poglavlju, dostupnu datoteku opisa modela robota *Nybble* potrebno je pretvoriti u URDF format kako ju se moglo uvesti u *Isaac Sim* za kreiranje USD datoteke koju je zatim moguće koristiti prilikom procesa treniranja. *Isaac Sim* ima ugrađeni alat za uvoz URDF datoteka te je jednostavno uvesti model robota pomoću spomenutog alata. Kada je model uvezen u *Isaac Sim*, potrebno je maknuti sve ostale dijelove scene te osigurati da je model robota korijenski član scene, za potrebe alata *Isaac Gym*. Na slici 5.1 prikazan je uvezeni model robota *Nybble* u program *Isaac Gym*.



Slika 5.1. *Model robota Nybble unutar Isaac Sim softverske platforme*

Kako je moguće vidjeti na slici 5.1, jedini element scene je sam model robota i pored imena se može vidjeti kako mu je dodano svojstvo *defaultPrim*, koje označava kako je to korijenski član, što je potreban atribut kako bi se kreiranje scena prilikom pripreme treniranja algoritma moglo obaviti unutar alata *Isaac Gym*. Moguće je i primijetiti da robotu nedostaje rep naspram originalnog modela. Rep je uklonjen jer je uočeno prilikom treniranja kako agent nauči koristiti rep kao oslonac, pa je rep maknut kako ne bi utjecao na hodanje i stabilnost robota. USD datoteka prikazana na slici 5.1 dostupna je na *GitHub* repozitoriju projekta.

5.2. Proces treniranja agenta

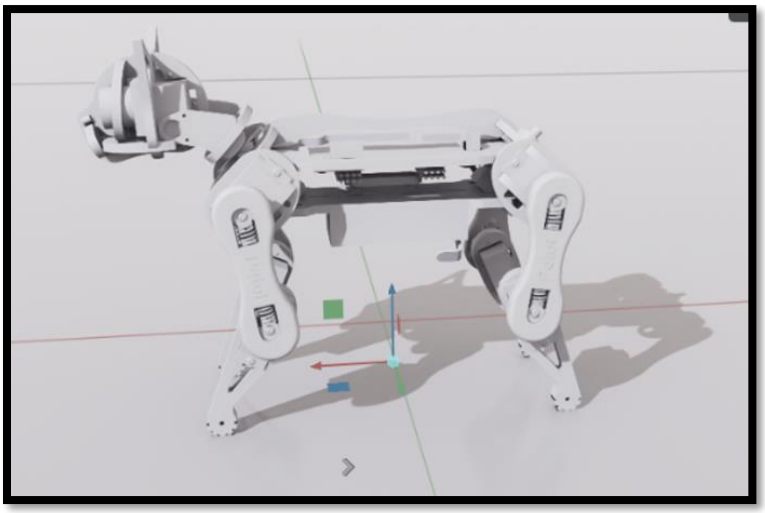
Nakon kreiranja USD datoteke opisane u prošlom potpoglavlju, dodane su nove datoteke u mapu za robot *Nybble* po uzoru na primjer *ANYmal*. Također, dodana je mogućnost pokretanja skripte *rlgames_train.py* u repozitoriju *Isaac Gym* alata koristeći parametar *task=Nybble*, kojim je moguće pokrenuti treniranje i testiranje agenta podržanog učenja koristeći model robota *Nybble*. Kako je objašnjeno u trećem poglavlju, korišteni algoritam podržanog učenja je PPO metoda i potrebno je definirati hiperparametre PPO algoritma, koji se nalaze u tablici 5.1. Hiperparametri PPO algoritma preuzeti su od primjera *ANYmal* te su izmijenjeni za potrebe treniranja robota *Nybble*.

Tablica 5.1. *Hiperparametri PPO algoritma*

Hiperparametar	*Vrijednost*
γ	*0.99*
τ	*0.95*
ϵ	*0.2*
e	*0.0001*
α	*promjenjiv*
KL	*0.008*
n_{steps}	*24*
minibatch_size	*32768*
mini_epochs	*5*
bound_loss_coeff	*0.001*
max_epochs	*1000*

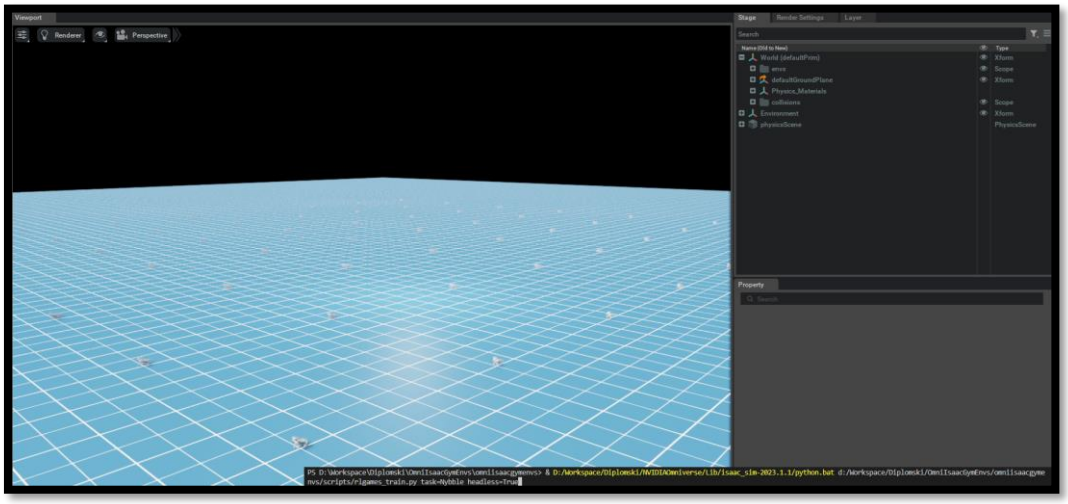
Još jedna bitna definicija je struktura neuronske mreže koju dijele kritičar i glumac, koja ima tri skrivena sloja, gdje prvi skriveni sloj ima 256, drugi 128 a treći 64 neurona, što je identična struktura kao i kod primjera robota *ANYmal*. Struktura je zadovoljavajuća i testiranjem je utvrđeno kako se povećanjem broja neurona i skrivenih slojeva ne povećavaju performanse modela. Također, uz hiperparametre PPO algoritma, potrebno je definirati željenu ciljnu brzinu kojom agent kontrolira model robota i početnu poziciju robota. Eksperimentalnom analizom utvrđeno je kako se najbolji rezultati dobivaju kada je robotu zadana brzina između *0.2 m/s* i *0.4 m/s*.

Još je potrebno definirati agentovu nagradu, kao i kod svakog primjera podržanog učenja. Definirana nagrada jednaka je kao i kod primjera *ANYmal*, a u obzir uzima razne vrijednosti: linearnu brzinu u *x* i *y* smjeru, linearnu brzinu u *z* smjeru, kružnu brzinu u *z* smjeru, ubrzanje zglobova robota te poziciju zglobova robota u odnosu na početnu poziciju. Sama programska definicija nagrade dostupna je na *github* repozitoriju projekta. Početna pozicija s kojom se dobiju najbolji rezultati prikazana je na slici 5.2.



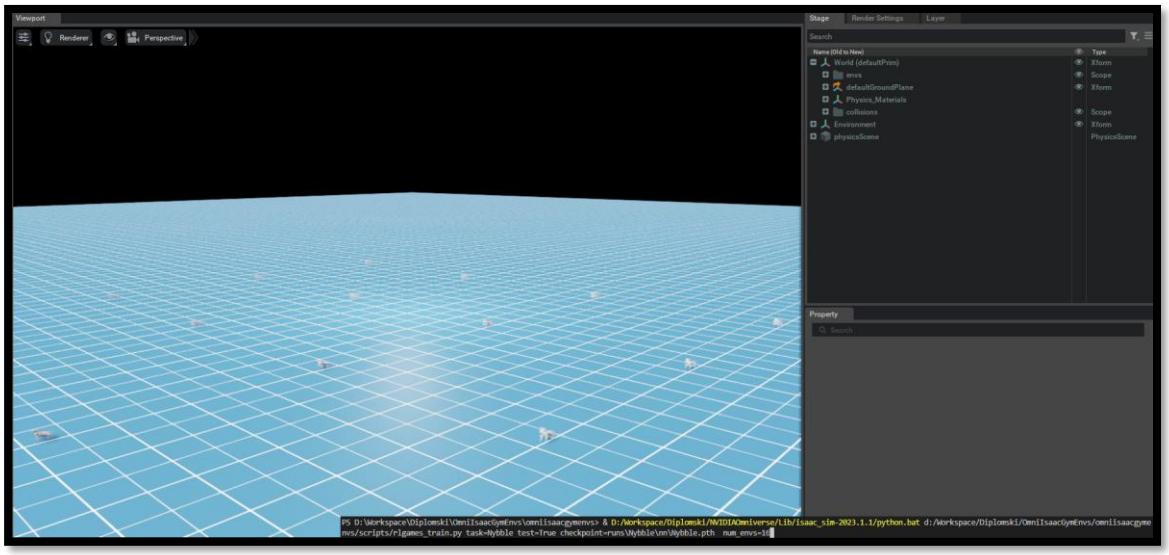
Slika 5.2. *Model robota Nybble u početnoj poziciji za trening*

Sve izmjene i potrebni parametri dostupni su na repozitoriju rada, a treniranje je moguće započeti pozivajući skriptu *rlgames_train.py* uz parametar *task=Nybble*. Za povećanje performansi, moguće je skripti također predati i parametar *headless=True* koji ne pokreće grafičko sučelje i animacije koje zauzimaju velike količine resursa GPU-a, kako bi se proces treniranja znatno ubrzao. Slika 5.3 prikazuje paralelni proces treniranja 128 modela bez *headless* konfiguracije, uz prikaz pravilnog pokretanja skripte treniranja u *headless* konfiguraciji. Prilikom stvarnog pokretanja treniranja, broj modela poveća se na 4096 paralelnih simulacija kako bi se ubrzao proces treniranja te kako bi tisuću epoha bilo dovoljno za stabilno hodanje robota.

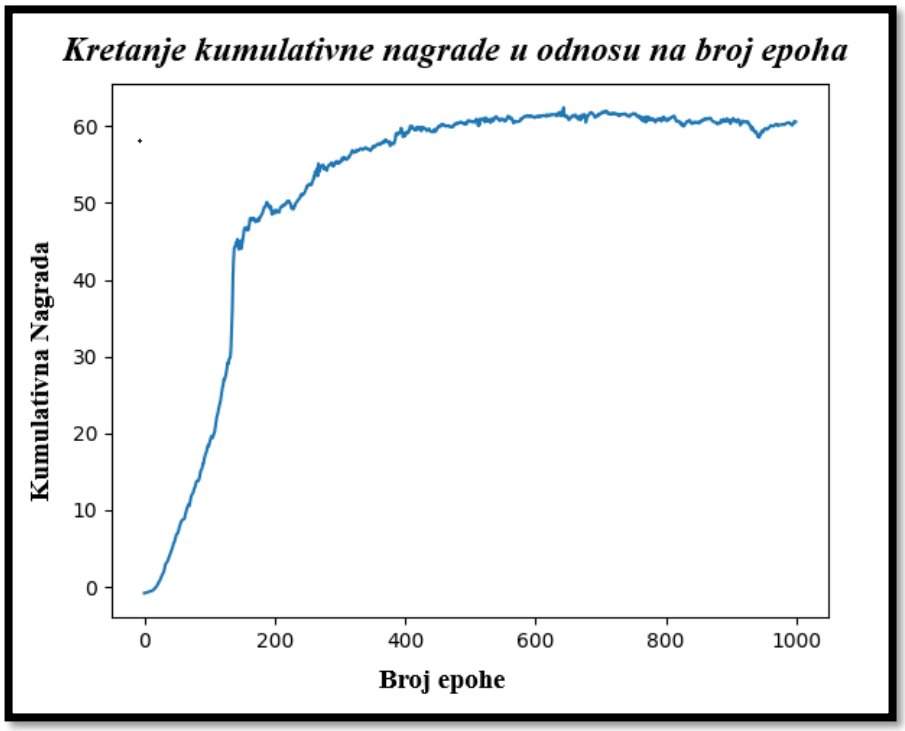


Slika 5.3. *Proces treniranja agenta nad robotom Nybble*

Nakon odrađenog procesa treniranja, sprema se model kojim se postiže najveća kumulativna nagrada i taj model je moguće isprobati na proizvoljnom broju okoliša, prikazano na slici 5.4, a napredak kumulativne nagrade u odnosu na broj epoha prikazan je na slici 5.5.



Slika 5.4. *Prikaz pokretanja najboljeg modela agenta*



Slika 5.5. *Kretanje kumulativne nagrade agenta*

Treniranje je odrađeno na laptopu s *AMD Ryzen 5 5600H* procesorom, *NVIDIA GeForce RTX 3060 Laptop GPU* grafičkoj kartici koja ima 6GB memorije te uz 16GB memorije sustava. Prolazak kroz tisuću epoha s treniranjem 4096 paralelnih okoliša u *headless* načinu treniranja na navedenom hardveru traje oko 25 minuta, odnosno u prosjeku 1.493 sekunde po epohi.

5.3. Prevođenje modela u programski kod

Nakon što je proces treniranja izvršen, potrebno je pretvoriti datoteku modela u odgovarajući oblik te ju uvesti u *STM32CubeIde* projekt, čime se generira programski kod kojim je moguće izvesti inferenciju nad ulaznim podacima. Odabirom mikroupravljača omogućeno je da sažimanje modela nije potrebno provesti, s obzirom da odabrani mikroupravljač ima dovoljno memorije za prevođenje modela bez potrebe za sažimanjem. Model neuronske mreže zauzima 209.57 KiB od dostupnih 1024 KiB memorije mikroupravljača. Koristeći generirani kod, moguće je definirati glavnu petlju programa. Glavna petlja je vrlo jednostavna, potrebno je dohvatiti ulazne podatke, izvesti inferenciju nad modelom, obraditi izlaze kako bi ih se pretvorilo u ciljeve za svaki aktuator robota te zatim obrađene izlazne podatke primijeniti na same aktuatore robota. Prije programske petlje potrebno je također inicijalizirati sve periferne uređaje sustava. Prikaz glavne programske petlje nalazi se na slici 5.6, a vizualizacija neuronske mreže na mikroupravljaču prikazana je na slici 5.7.

```
1:      int main(){
2:         SYSTEM_Initialize();
3:         while(1){
4:            acquire_and_process_data(in_data); //Prikupljanje podataka
5:            aiRun(in_data, out_data); // Inferencija
6:            post_process(out_data); // target_position = ciljevi aktuatora
7:            NYBBLE_SetPosition(target_position); // Postavljanje ciljeva
8:            TimeSync(); // Stanka u programu kako bi robot bio stabilan
9:         }
10:     }
```

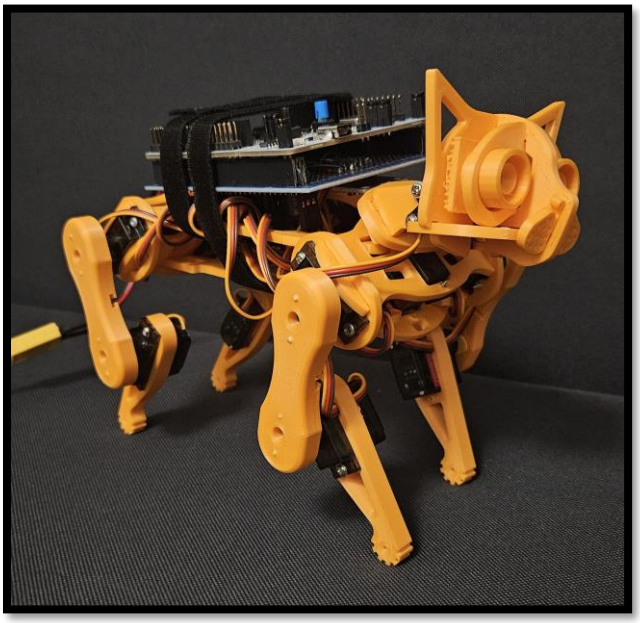
Slika 5.6. *Glavna programska petlja na mikroupravljaču robota Nybble*

Slika 5.7. *Vizualizacija slojeva neuronske mreže na mikroupravljaču*

Projekti koji provode prenašanje naučenog u simulaciji na stvarni svijet se u većini slučajeva koriste vrlo specijaliziranim hardverom kojim je moguće dobaviti sve potrebne ulaze koje model treba za odrađivanje inferencije. Takav specijaliziran hardver je skup te za ovaj rad nije korišten, tako da su korišteni ulazni parametri preuzeti iz same simulacije koji su preuzeti prilikom testiranja modela. Sav kod koji je korišten na mikroupravljaču zajedno sa generiranim sučeljima inferencije i parametrima modela nalazi se na repozitoriju rada.

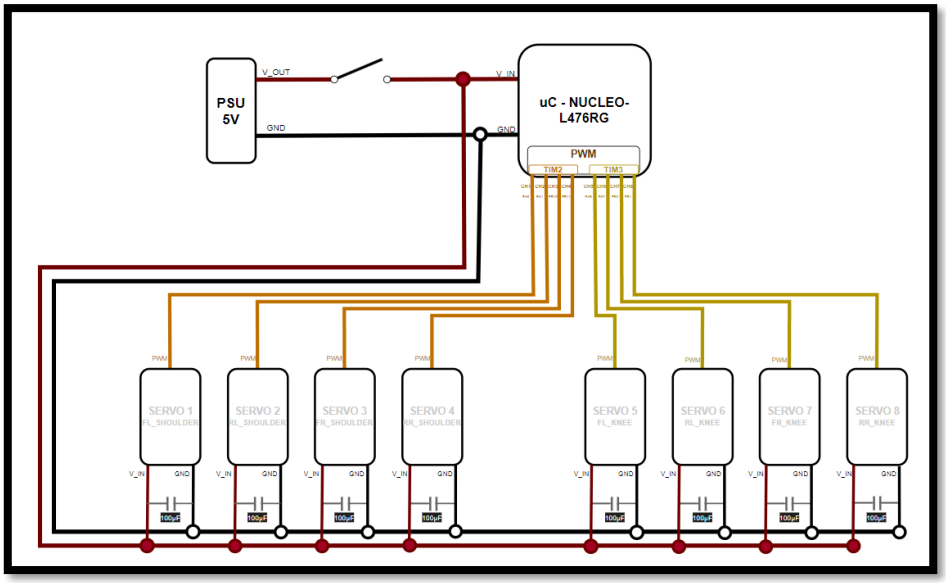
5.4. Izgradnja robota Nybble

Koristeći 3D pisač te MG90S servo motore, robota *Nybble* moguće je izgraditi s datotekama dostupnim na internetu. Slika 5.8 prikazuje sastavljeni model robota *Nybble*.



Slika 5.8. *Sastavljeni model robota Nybble*

Dizajnirana je i upravljačka pločica kako bi se na jednostavan način moglo povezati aktuatore robota s potrebnim signalima kojima kontrolira mikroupravljač. Dijagram upravljačke pločice nalazi se na slici 5.9.

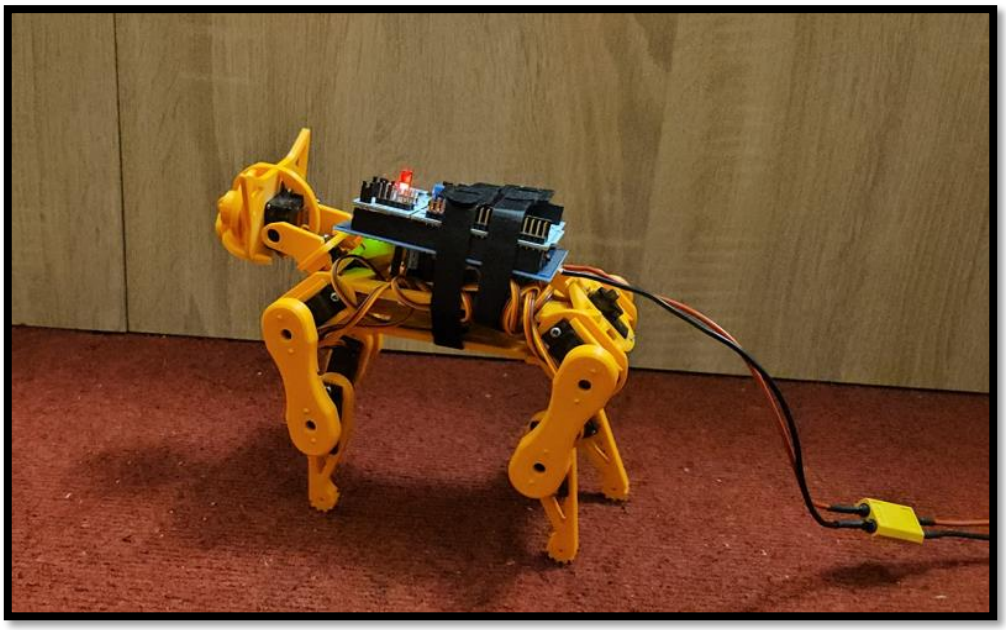


Slika 5.9. *Upravljačka pločica robota Nybble*

Kompilacijom programskog koda i učitavanjem koda na mikroupravljač se projekt smatra ostvarenim. Rezultati projekta diplomskog rada dani su u šestom poglavlju.

6. REZULTATI I ANALIZA

Rezultati su dostupni na *github* repozitoriju diplomskog rada unutar pod-direktorija *Results*, s obzirom da je za prikaz rezultata potrebna snimka robota. U pod-direktoriju *Results* se nalaze snimke testiranja mobilnog robota *Nybble* u stvarnom svijetu kao i prikaz istreniranog robota u simulatoru, dok je na slici 6.1 prikazan isječak iz videozapisa u kojem se istrenirani robot kreće u stvarnom svijetu.



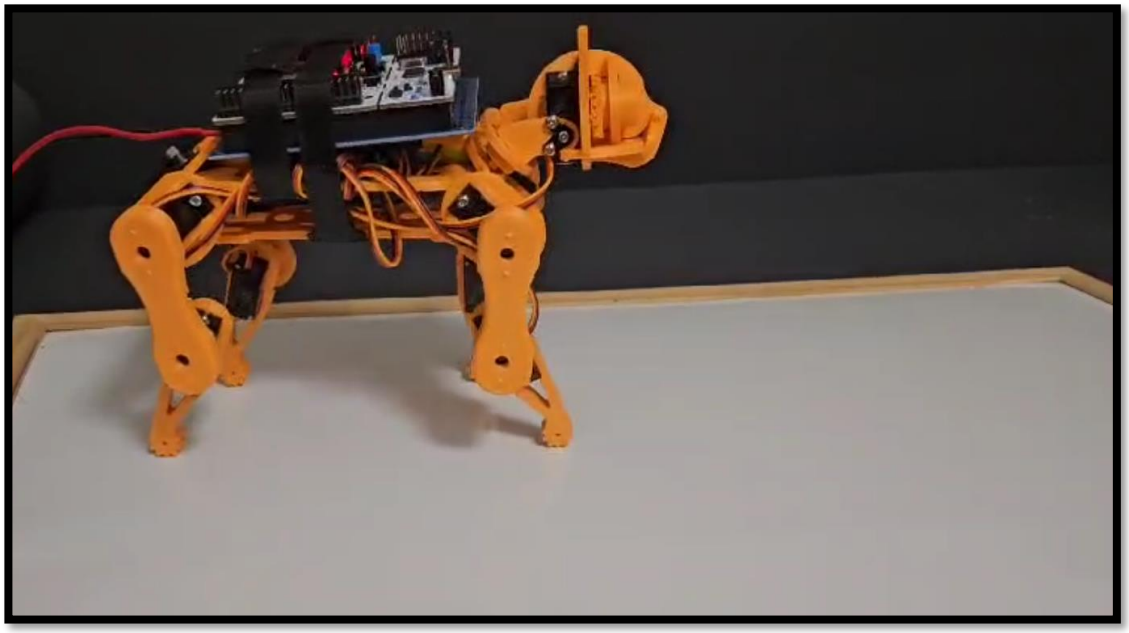
Slika 6.1. *Isječak iz videozapisa testiranja prebačenog agenta na 3D ispisani model robota Nybble na tepihu*

Iz rezultata je moguće zaključiti kako je treniranje agenta u simulacijskoj okolini i prijenos istog u pravi svijet uspješno obavljen. Iako rezultati nisu optimalni i moguće je vidjeti kako robot ne slijedi akcije agenta unutar simulatora savršeno, s ograničenjima korištenog hardvera se rezultat smatra prihvatljivim. Bez informacije o poziciji servo motora svakog zgloba, početnu poziciju nije moguće podesiti na način da je jednaka kao u simulacijskom okruženju. Isto tako je problem što kod postavljanja servo motora zglobova robota na željene pozicije, agent nema mogućnost očitati grešku prilikom postavljanja. Još jedan veliki problem je i kvaliteta samog 3D ispisa dijelova robota. Nestabilnost strukture uvodi mnogo nasumičnih pokreta koje agent ne može ispraviti jer nema pristup tim informacijama. Opisani problemi najuočljiviji su prilikom postavljanja zglobova robota na početnu poziciju i početka dohvaćanja podataka iz simulatora. Moguće je uočiti da robotu treba nekoliko sekundi kako bi krenuo stabilno hodati nakon postavljanja zglobova na početnu poziciju, ali se nakon toga stabilnost poveća i svih 200 ulaza iz simulatora prođe kroz

neuronsku mrežu bez da robot izgubi ravnotežu i padne. Nakon inferencije svih 200 ulaza, robot ponovno dođe u početnu poziciju i opisani slijed se ponavlja.

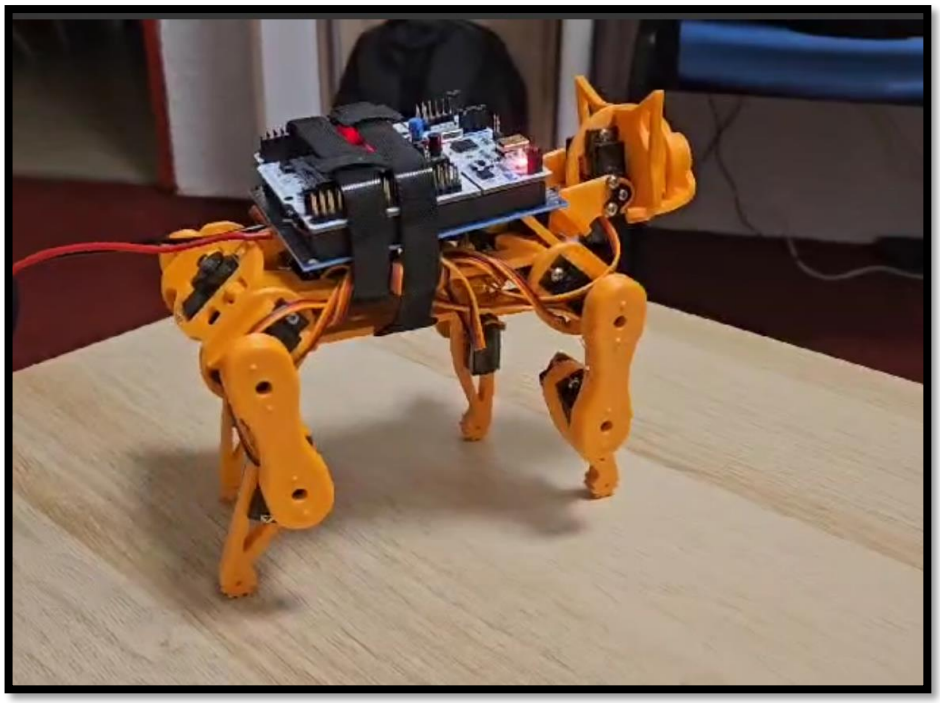
Inferencija modela neuronske mreže se obavlja na mikroupravljaču u stvarnom vremenu i traje između 6 i 7 milisekundi, što je dovoljno brzo kako bi se programska petlja izvršavala unutar zadanog vremena simulacije. Iako je moguće pokretati programsku petlju na brzini simulacije, potrebno ju je usporiti zbog stabilnosti robota i mogućnosti korištenog hardvera. Pokretanjem programske petlje na brzini simulacije robot gubi stabilnost i često pada, iz razloga što korišteni servo motori zglobova robota nemaju dovoljno vremena za postavljanje tražene pozicije od strane agenta.

Robot ima mogućnost hodanja na podlogama s većim koeficijentom trenja između robota i podloge, poput tepiha, dok na podlogama s manjim koeficijentom trenja između robota i podloge, kao što je plastika, teže ostvaruje kretanje unaprijed. Videozapis ispitnog slučaja kretanja na plastičnoj podlozi također se nalazi na *github* repozitoriju, a slika 6.2 prikazuje isječak iz iste.



Slika 6.2. *Isječak iz videozapisa testiranja prebačenog agenta na 3D ispisani model robota Nybble na plastičnoj podlozi*

Zadnje testiranje rezultata obavljeno je na drvenoj površini, gdje je moguće vidjeti da se malim povećanjem koeficijenta trenja između robota i površine postižu bolji rezultati, ali ne kao na tepihu. Na slici 6.3 prikazan je isječak iz videozapisa testiranja na drvenoj površini, a isti je, kao i ostali, dostupan na *github* repozitoriju projekta.



Slika 6.3. *Isječak iz videozapisa testiranja prebačenog agenta na 3D ispisani model robota Nybble na drvenoj podlozi*

Iako je moguće povećati koeficijent trenja između robota i plastične površine, kao i drvene, koristeći neki drugi materijal za dijelove robota koji dodiruju podlogu, isto tako je moguće i koristiti razne načine poboljšavanja ponašanja agenta, poput randomizacije domene. To također zahtjeva i unaprjeđenje korištenog hardvera i mogućnost stvarnog percipiranja okoliša od strane agenta za ulaz u neuronsku mrežu kako bi agent mogao reagirati na različite varijable okoline u kojoj se nalazi.

7. ZAKLJUČAK

U ovom diplomskom radu cilj je prijenos naučenog unutar simulacije u stvarni svijet. Korišten je izrađeni četveronožni robotski model *Nybble*, čiji je model istreniran za kretanje unaprijed unutar simulacijskog okruženja pomoću metode podržanog učenja. Dobiveni agent u obliku neuronske mreže provedbom algoritma podržanog učenja PPO prebačen je na mikroupravljač, kojim je agentu omogućeno upravljanje 3D ispisanim modelom robota *Nybble*. Za treniranje agenta korišteni su alati *Isaac Gym* i *Isaac Sim* tvrtke *Nvidia*, a za prevođenje modela neuronske mreže agenta korišten je *STM32Cube IDE* alat tvrtke *STMicroelectronics* uz dodatak *X-CUBE-AI*.

Analiza rezultata obavljena je vizualno, s obzirom da je cilj bio ostvariti kretanje unaprijed bez potrebe za pisanje logike unutar programskog koda za isto. Videozapisi rezultata dostupni su na *github* repozitoriju projekta diplomskog rada.

Rezultati su analizirani na različitim podlogama i kako je moguće vidjeti na videozapisima rezultata, povećanjem trenja između podloge i robota povećavaju se i performanse agenta. Iako je moguće koristiti napredniji algoritam prilikom razvoja agenta, poput randomizacije domene, najveći nedostatak je to što agent nema mogućnost percipiranja stvarne okoline u kojoj se nalazi, već su ulazni podaci iz simulatora pohranjeni u memoriji mikroupravljača. Kad bi agent imao mogućnost percipiranja okoline, poput trenutne pozicije, ubrzanja i brzine robotskih zglobova, agent bi mogao percipirati razliku naspram zadanih pozicija zglobova stvarnog robota te ispraviti pogrešku i time poboljšati rezultate na različitim podlogama.

Kako bi se performanse agenta poboljšale na raznim površinama, prvo je potrebno koristiti sofisticiraniji hardver poput servo motora s povratnim podacima o trenutnom položaju zajedno sa senzorom ubrzanja i žiroskopom, ali isto tako potrebno je i poboljšati strukturnu stabilnost robota. Koristeći opisani hardver s velikom preciznošću i povećanjem stabilnosti robota agent bi bio u mogućnosti dohvaćati korisne podatke iz okoline i samim time bi se performanse na različitim površinama znatno poboljšale. Nakon poboljšanja hardvera i analiziranja rezultata moguće je započeti daljnji razvoj agenta koristeći tehnike poput randomizacije domene ako je to potrebno.

Iz rezultata je moguće zaključiti kako su hardverski nedostatci poput nedostatka čitača pozicije motora te loša kvaliteta 3D ispisanih dijelova robota razlog zašto rezultati nisu bolji. Ulaganjem u bolji hardver znatno je moguće povećati performanse istreniranog agenta, ali i s ostvarenim rezultatima je moguće vidjeti kako pristup prenošenja naučenog iz simulacije u stvarni svijet može biti vrlo koristan i zanimljiv.

LITERATURA

[1] Urednici enciklopedije *Britannica*, “R.U.R.”, Enciklopedija *Britannica* [online], studeni 2014. Dostupno na: https://www. britannica.com/topic/RUR [12.06.2024.]

[2] R. Ravi, A. Kos, "A Comprehensive Study of Mobile Robot: History, Developments, Applications, and Future Research Perspectives" Applied Sciences vol 12, no. 14, str. 1, 2022.

[3] Intel, Types of Robots: How Robotics Technologies Are Shaping Today's World [online], dostupno na https://www.intel.com/content/www/us/en/robotics/types-and-applications.html [12.06.2024]

[4] F. Rubio, F. Valero, C. Llopis-Albert, A review of mobile robots: Concepts, methods, theoretical framework, and applications, International Journal of Advanced Robotic Systems, br. 2, sv. 16, str. (1-2), Travanj 2019.

[5] X.B. Peng i sur., Sim-to-Real Transfer of Robotic Control with Dynamics Randomization, International Conference on Robotics and Automation (ICRA), Brisbane, Australia, 21-25 svibnja 2018.

[6] W.Zhao, J.P. Queralta, T. Westerlund, Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: a Survey, Symposium Series on Computational Intelligence (SSCI), Canberra, Australia, 01-04 prosinca 2020.

[7] M. Kaspar, J.D.M. Osorio, J. Bock, Sim2Real Transfer for Reinforcement Learning without Dynamics Randomization, International Conference on Inteligent Robots and Systems (IROS), Las Vegas, USA, 25 - 29 listopada 2020.

[8] X. Gu, Y.J. Wang, J. Chen, Humanoid-Gym: Reinforcement Learning for Humanoid Robot with Zero-Shot Sim2Real Transfer, International Conference on Robots and Automation (ICRA), Yokohama, Japan, 13-17 svibnja 2024.

[9] P. Akella, W. Ubellacker, A.D. Ames, Test and Evaluation of Quadrupedal Walking Gaits through Sim2Real Gap Quantification, siječanj 2022.

[10] N.Rudin i sur., Learning to Walk in Minutes Using Massively Parallel Deep Reinforcement Learning, Conference on Robot Learning (CoRL), London, UK, 08-11 studenog 2021.

[11] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, Second Edition, MIT Press, Massachusetts, 2018.

[12] C. Szepesvári, Algorithms for Reinforcement Learning, Springer Nature, Švicarska, 2022.

[13] E. Akankasha i sur., Review on Reinforcement Learning, Research Evolution and Scope of Application, Fifth International Conference on Computing Methodologies and Communication (ICCMC), Erode, India, 08-10 travnja 2021.

[14] T. Wang i sur., Benchmarking Model-Based Reinforcement Learning, srpanj 2019.

[15] I. Grondman i sur., A Survey of Actor-Critic Reinforcement Learning: Standard and Natural Policy Gradients, IEEE Transactions on Systems, Man, and Cybernetics, Part C, vol. 42, br. 6, str. 1291 - 1307, studeni 2012.

[16] O. Nachum i sur., Bridging the Gap Between Value and Policy Based Reinforcement Learning, Conference on Neural Information Processing Systems (NIPS), Long Beach, CA, US, 04-07 prosinca 2017.

[17] S. Huang i sur., A2C is a special case of PPO, svibanj 2022.

[18] M. Babaeizadeh i sur., GA3C: GPU-based A3C for Deep Reinforcement Learning, studeni 2016.

[19] S. Gu i sur., Q-Prop: Sample-Efficient Policy Gradient with An Off-Policy Critic, International Conference on Learning Representations (ICLR), Toulon, Francuska, 24-26 travnja 2017.

[20] K Shao i sur., Visual Navigation with Actor-Critic Deep Reinforcement Learning, International Joint Conference on Neural Networks (IJCNN), Rio de Janeiro, Brazil, 08-13 srpnja 2018.

[21] Y. Wang, H. He, X. Tan, Truly Proximal Policy Optimization, 35. Uncertainty in Artificial Intelligence Conference, PLMR 115, str. 113-122, 2020.

[22] J. Schulman i sur., Trust Region Policy Optimization, 32. International Conference on Machine Learning, PLMR 37, str. 1889-1897, 2015.

[23] What is Isaac Sim? [online], Nvidia, dostupno na https://docs.omniverse.nvidia.com/isaacsim/latest/index.html [30.06.2024.]

[24] Isaac Gym, Nvidia, dostupno na https://developer.nvidia.com/isaac-gym [30.06.2024.]

[25] PyTorch documentation, The Linux Foundation, dostupno na https://pytorch.org/docs/stable/index.html [30.06.2024.]

[26] V. Makoviychuk i sur., Isaac Gym: High Performance GPU-Based Physics Simulation For Robot Learning, dostupno na https://sites.google.com/view/isaacgym-nvidia [30.06.2024.]

[27] About Us, Petoi, dostupno na https://www.petoi.com/pages/about [30.06.2024.]

[28] STM32CubeIDE user guide, STMicroelectronics, dostupno na https://www.st.com/resource/en/user_manual/um2609-stm32cubeide-user-guide-stmicroelectronics.pdf [30.06.2024]

[29] X-CUBE-AI:Artificial intelligence (AI) software expansion for STM32Cube, STMicroelectronics, dostupno na https://www.st.com/resource/en/data_brief/x-cube-ai.pdf [30.06.2024.]

SAŽETAK

U ovom diplomskom radu provedeno je treniranje agenta koji upravlja četveronožnim mobilnim robotom pomoću metode podržanog učenja, odnosno još specifičnije, PPO algoritmom. Odabrani četveronožni robot je robot tvrtke *Petoi* zvani *Nybble*. Agent je u obliku neuronske mreže prebačen na mikroupravljač čime je agentu omogućeno upravljanje 3D ispisanom verzijom robota. Agent ima mogućnost upravljanja osam nožnih zglobova robota i treniranjem je naučeno da ih pomiče na način da se robot kreće unaprijed. Za treniranje agenta korišteni su alati *Isaac Sim* i *Isaac Gym* tvrtke *Nvidia*, dok je za prevođenje neuronske mreže na mikroupravljač korišteno razvojno okruženje *STM32Cube IDE* tvrtke *STMicroelectronics* koristeći dodatak *X-CUBE-AI*. Spomenuto razvojno okruženje korišteno je i za razvijanje programskog koda u svrhu prilagođavanja ulaza i izlaza neuronske mreže te za pisanje programskog koda za upravljanje servo motorima robota, napisano u C programskom jeziku. Zbog ograničenja hardvera agent ulazne podatke potrebne za inferenciju neuronske mreže ne skuplja iz stvarnog okoliša, već su ulazni podaci prikupljeni iz simulacijskog okruženja i pohranjeni unutar memorije mikroupravljača. Rezultatima je pokazano kako je prijenos naučenog iz simulacije u stvarni svijet moguć čak i s jeftinim hardverom te iako rezultati nisu optimalni velikim dijelom je to krivnja samog hardvera.

Ključne riječi: četveronožni robot, Isaac Gym, podržano učenje, prijenos naučenog simulacijom u stvarni svijet, PPO

ABSTRACT

This master's thesis addresses the problem of training an agent to control a quadruped robot using a machine learning method called reinforcement learning, and more specifically the PPO algorithm. The chosen quadruped robot model is *Nybble* by the *Petoi* company. After successful training, the agent was transferred onto a microcontroller in the form of a neural network to have the ability to control a 3D printed version of *Nybble*. The agent has the ability to control *Nybble*'s eight leg joints and with training it learned how to move them in such a way to achieve forward movement. The agent was trained using the *Isaac Sim* and *Isaac Gym* tools by *Nvidia*, and the *STM32Cube IDE* by *STMicroelectronics* was used to translate the agent neural network to the microcontroller using the *X-CUBE-AI* addon. The IDE was also used to develop the source code responsible for adapting the inputs and outputs of the neural network, as well as to write the drivers to control the *Nybble*'s servo motors, all written in C programming language. Because of the hardware limitations, the agent doesn't collect actual environment data necessary for neural network inference while on the real-world model. The data has been previously collected in the simulation and stored into the microcontroller's internal memory. The results show that the simulation to reality transfer is possible even with cheap hardware, and even though the results aren't optimal, most of it can be blamed on the hardware itself.

Keywords: Isaac Gym, PPO, quadruped robot, reinforcement learning, simulation to reality transfer

ŽIVOTOPIS

Bruno Zahirović rođen je 22. srpnja 1999. godine u Našicama. Nakon pohađanja Osnovne škole kralja Tomislava Našice, upisuje prirodoslovnu matematičku gimnaziju u Srednjoj školi Isidora Kršnjavog Našice. Obrazovanje u srednjoj školi završava 2018. godine te iste godine upisuje sveučilišni preddiplomski smjer računarstva na Fakultetu Elektrotehnike, Računarstva i Informacijskih Tehnologija Osijek. Preddiplomski studij završava 2021. godine i upisuje diplomski studij računarstva na istom fakultetu, izborni blok Robotika i umjetna inteligencija, iste godine. Programski jezici koje poznaje su C, C++ i Python.

PRILOG

1. Diplomski rad u PDF formatu
2. *GitHub* repozitorij projekta s programskim rješenjem i rezultatima diplomskog rada (https://github.com/bruno-zahirovic/OmniIsaacGymEnvs-NybbleSim2Real)