Simulacija hodajućeg robota

Bogadi, Hrvoje

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij, izborni blok Robotika i umjetna inteligencija (DRB)

SIMULACIJA HODAJUĆEG ROBOTA

Diplomski rad

Hrvoje Bogadi

Mentor: Robert Cupec

Sumentor: Matej Džijan

Osijek, 2022.

SADRŽAJ

1. UVOD

Značajnim napretkom algoritama strojnoga učenja, pogotovo u području podržanog učenja, tijekom zadnjih nekoliko godina omogućile su se nevjerojatne razine samostalnosti i efikasnosti robotskih sustava. Odmicanje od tradicionalnih prediktivnih metoda upravljanja i razvoj pomoću dubokog podržanog učenja, omogućili su samostalnu prilagodbu sustava na kretanje kroz okolinu u kojoj se nalazi bez značajne analize sklopovlja robota i mukotrpnog dizajna eksplicitnih modela sustava kretanja. U središtu je ovoga rada razvoj upravo takve metode, koja unutar simulacije, sustavu mobilnog robota omogućava stabilizaciju u okolini te kretanje kroz prostor na zahtjevan i kontroliran način, uz mogućnost savladavanja prepreka. Iako će se u ovome radu zadržati unutar okvira simulacije, kratko će biti govora i o izazovima prijenosa takvog sustava u stvarni svijet.

1.1. Pregled diplomskog rada

Kroz ovaj će diplomski rad u drugom poglavlju ukratko biti predstavljen pregled područja rada na zadanu temu te će se ukratko ponuditi i opisati najznačajniji radovi i povijesni razvoj teme. U trećem poglavlju pokrit će se teorijska pozadina podržanoga učenja, što je to, na kojim se konceptima temelji te će se u potpoglavljima 3.1, 3.2 i 3.3 prikazati tri velike i glavne podjele algoritama podržanoga učenja te način na koji je svaka od njih koncipirana, kao i razlozi njihova nastanka i problemi koje rješavaju. U četvrtom će poglavlju biti objašnjena osnovna postavka praktičnog dijela ovog rada te osnovni alati kojima se koristi, kao i način na koji je koncipirana izvedba algoritamskog rješenja te će se, u konačnici, u petom poglavlju čitatelju ponuditi pregled i opis ostvarenih rezultata.

2. PREGLED PODRUČJA RADA

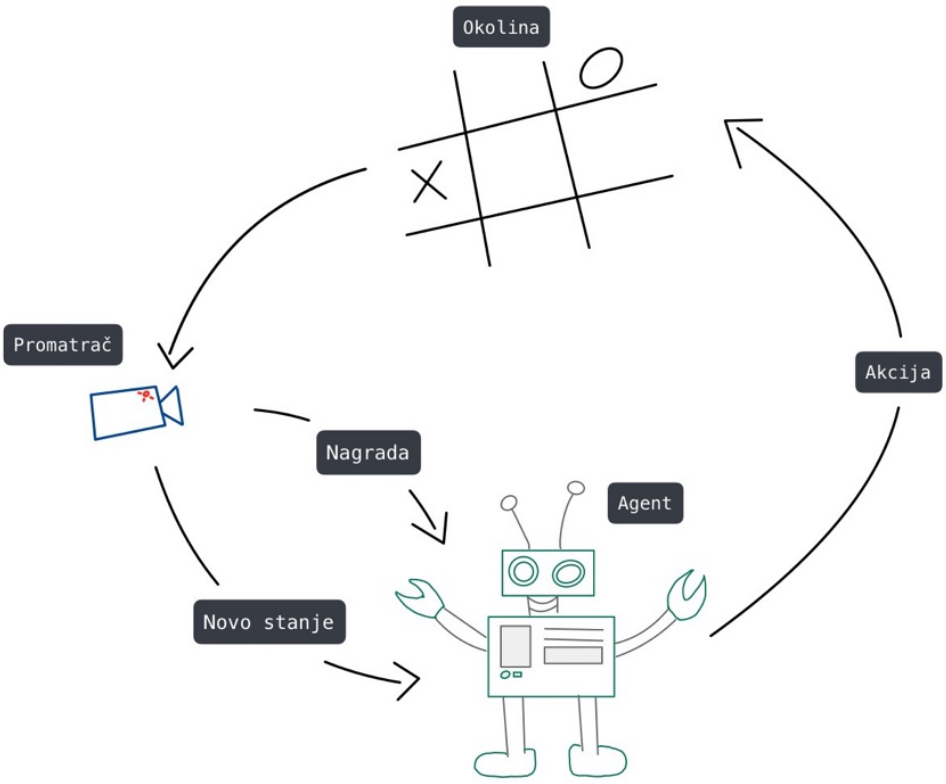
Tijekom zadnjih nekoliko godina, znanstvena istraživanja na području mobilne robotike i sustava kretanja premjestila su fokus s do tada popularnih metoda koje su iskorištavale pristupe poput estimacije stanja (engl. s*tate estimation*), optimizacije putanje (engl. *trajectory optimization*), planiranja pozicije gaza (engl. *foot placement planning*) i sl., koji su redovito zahtijevali identifikaciju složenog matematičkog modela, na metode podržanog učenja i *black box* modela. Iako na prvi pogled primamljive, zbog svoje jednostavnosti i izbjegavanja potrebe za poznavanjem intrinzičnih parametara sustava, RL (engl. *Reinforcement learning* – podržano učenje) metode su se pokazale su se teškima za optimiranje i skupima s gledišta računalnih resursa tijekom simulacije i učenja, a u pogledu prenošenja na stvarne robotske sustave i financijski skupi zbog velike vjerojatnosti oštećenja sustava tijekom učenja na način pokušaj-pogreška. Jedan od značajnih radova koji je pokazao kako je moguće stvoriti dovoljno dobar RL model hodajućeg robota u stvarnom svijetu bez prethodnog dizajna modela bio je „*Learning to Walk via Deep Reinforcement Learning*“ [1]. U to vrijeme postojale su već metode koje su se koristile *soft Q-learning-om* [2] ili *soft actor-critic-om* [3] i postizale *state-of-the-art* rezultate na području robotike [4] [5], uz ograničenja pri izboru hiperparametara (više o temi u [1]), no autori prvoga rada uspjeli su riješiti i taj problem, time generalizirajući i olakšavajući primjenu ovakvih metoda. Daljnjim razvojem povećavala se robustnost postojećih algoritama pa je tako 2021. godine objavljen rad „*Reinforcement Learning for Robust Parameterized Locomotion Control of Bipedal Robots*“ [6], u kojemu autori razvijaju bipedalni robotski sustav „*Cassie*“ kojim demonstriraju mogućnost normalnog hoda, promjene visine hoda, trčanja pa čak i otpornost na guranje i udarce. Tu do izražaja dolazi upravo dizajn robotskog sustava (odnosno samo 3D modela na kojemu se sustav temelji) i njegovo učenje hoda u simulaciji te zahtjevnost prenošenja funkcionalnosti iz simulacije u pravi svijet. Koristeći se PPO (engl. *Proximal Policy Optimization*) algoritmom i pametnim postavkama simulacije za prijenos u stvarni svijet, autori su postigli do sada neviđenu robusnost sustava i otpornost na smetnje do te mjere da je robot Cassie mogao hodati i uz kvar jednoga od motora. Postalo je vrlo očito da je takav pristup ne samo moguć nego i vrlo povoljan. Veliki je problem tada predstavljao velik broj iteracija potreban za treniranje mreže i učenje robota hodu u simulaciji te količina dostupne računalne snage. Rad autora s „ETH Zürich“-a u suradnji s tvrtkom „Nvidia“, uspješno je premostio i ovu prepreku. „*Learning to Walk in Minutes Using Massively Parallel Deep Reinforcement Learning*“ [7] iskorištava masovni paralelizam na jednoj GPU. Dodatno, razvijaju „*game inspired curriculum*“, kojim omogućavaju optimizaciju parametara i trening

na tisuće robota istovremeno te na kraju pokazuju da je sustav iz simulacije moguće prenijeti u stvarni svijet bez većih poteškoća.

Ovaj diplomski rad koristi se upravo njihovim otvorenim kodom i simulacijom kako bi efikasno ostvario postavljene ciljeve razvijanja hodajućeg robota koristeći se RL algoritmima.

3. PODRŽANO UČENJE

Glavna je prednost podržanog učenja u odnosu na ostale metode upravljanja i učenja upravo uspješno ostvarivanje željenog ponašanja sustava bez potrebe za poznavanjem parametara i modela sustava. Na najosnovnijoj razini, podržano učenje vrlo je intuitivan i ljudima konceptualno blizak proces. Ideja je stvoriti sustav u kojemu u zadanoj okolini postoji agent koji, izvršavanjem određenih akcija mijenja stanje okoline, a čija se ponašanja promatraju te se ona poželjna potiču kroz davanje nagrade agentu, a neželjena obeshrabruju kažnjavanjem agenta, odnosno uklanjanje nagrade. Okolina je generalno bilo koje okruženje u kojemu se od agenta očekuje izvršavanje akcije kako bi postigao zadani cilj. Na primjeru igre križić-kružić, agent bi bio jedan od igrača, okolina ploča na kojoj se igra, stanje trenutne pozicije križića i kružića na ploči, a akcija agenta postavljanje križića na ploču. Kroz igru, agent bi bivao nagrađen svaki puta kada uspije pobijediti u igri (a potencijalno i u određenoj mjeri kada odigra neriješenu partiju), a kažnjen kada izgubi. Ponavljanjem ovoga postupka kroz veliki broj igara moguće je naučiti agenta da igra križić-kružić na razini čovjeka. U sljedećim će poglavljima ovaj postupak biti nešto detaljnije i formalnije opisan.



Slika 3.1 - Životni ciklus RL algoritma

3.1. Tablične metode

Prethodno ugrubo opisani proces (Slika 3.1) formalno se temelji na konačnom Markovljevom procesu odlučivanja (engl. *Finite Markov Decision Process - MDP*). Markovljev proces odlučivanja ekstenzija je Markovljevih lanaca, a možemo ga opisati kao četvorku vrijednosti (S, A, P_a, R_a) gdje su:

- $\boldsymbol{S}$ – skup postojećih stanja zvan ***prostor stanja***
- $\boldsymbol{A}$ – skup mogućih akcija zvan ***prostor akcija*** (alternativno, moguće je iskazati A_s, koji označava set mogućih akcija iz stanja $s \in S$)
- $\boldsymbol{P_a(s', r \mid s, a)}$ – vjerojatnost da će akcija *a* u stanju *s* i trenutku *t* rezultirati prelaskom u stanje s' i nagradom *r* u trenutku *t+1*
- $\boldsymbol{R_a(s, s')}$ – trenutna nagrada ili očekivana trenutna nagrada pri prelasku iz stanja s u stanje s' kao posljedica akcije a.

Agent i okolina tada međudjeluju na način da agent kreće iz stanja S_0, odabire akciju A_0 te dobiva nagradu R_1 i prelazi u stanje S_1 iz kojeg ponovno ima na izbor akciju A_1. Ponavljanjem tog postupka stvara se sekvenca zvana ***putanja*** (engl. *Trajectory*), kojom agent dolazi do poželjnoga cilja odnosno konačnog stanja. Funkcija $\mathbf{P_a}$ daje nam informaciju o vjerojatnosti pojavljivanja određenih vrijednosti iz skupa $\boldsymbol{S}$ i $\boldsymbol{R_a}$ u trenutku *t* s obzirom na prethodno stanje i akciju, a zadana je kao:

$$p(s', r \mid s, a) = P(S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a) \qquad (3\text{–}1)$$

Iz jednadžbe ((3–1) jasno je vidljivo da stanje S_t ovisi jedino i isključivo o prethodnom stanju, što znači, kako bi uspješno postigli napredak pri izvođenju našeg algoritma, dano stanje u trenutku *t* mora sadržavati informaciju o svim prethodnim postupcima. Ukoliko je ovo svojstvo stanja zadovoljeno, kažemo da stanje ima ***Markovljevo svojstvo*** (engl. *Markov property*). Dobar primjer ovakvih stanja upravo je igra križić-kružić, u kojoj svako aktualno stanje igraču ploče nudi pregled svih akcija koje su se dogodile do toga trenutka. Ključan aspekt tabličnih metoda podržanog učenja upravo je nužnost zadovoljavanja Markovljevog svojstva, a metode aproksimacije bit će prikazane u nastavku, gdje ovo nije nužan uvjet.

Važno je u ovome trenutku istaknuti veliku dilemu podržanog učenja, problem ***istraživanja i iskorištavanja*** (engl. *Exploration vs. Exploitation*). Kasnije ćemo vidjeti na koji način agent odlučuje koju je akciju najbolje izabrati kako bi maksimizirao svoju nagradu i došao

do optimalnog načina rada. Upravo se na taj postupak maksimizacije trenutne nagrade odnosi izraz iskorištavanje. Smisleno je očekivati da će agent uvijek htjeti izabrati najveću nagradu jer bi to značilo da najbolje radi, no vrlo brzo dođemo do problema iz kojeg proizlazi navedena dilema. Naime, ukoliko agent svaki puta izabire akciju s kojom postiže trenutnu najveću nagradu (ovo se zove pohlepni agent (engl. *Greedy*)), postoji mogućnost da s ciljem maksimizacije trenutne nagrade agent nikada neće saznati za dolaženje u neko od stanja koje će mu trenutno pružiti manju nagradu, ali kroz nekoliko koraka rada, ultimativno omogućiti veću nagradu. Dakle potrebno je uvesti mehanizam istraživanja kojim agent u određenim trenucima s određenom vjerojatnosti ne izabire najbolji potez za trenutnu nagradu, s ciljem istraživanja drugih mogućnosti i potencijalnog razvoja prema boljem ukupnom učinku te maksimizacije ukupne nagrade. U tabličnim se metodama ovo uglavnom implementira na način da agent u nasumičnim trenucima s određenom vjerojatnošću, umjesto najbolje trenutne akcije, bira jednu od gorih opcija.

Način na koji agent odabire akcije temelji se na maksimiziranju očekivane nagrade $\mathbb{E}(G_t)$ koju prima kroz kretanje između stanja,

$$G_t \doteq R_{t+1} + R_{t+2} + \cdots + R_T \tag{3–2}$$

gdje je *T* konačni korak.

Iz jednadžbe (3–2) možemo zaključiti da ovakav postupak ima smisla samo ako je zadani proces konačan, odnosno kada se djelovanje agenta prirodno razlama u epizode (na primjer partije šaha, igre križić kružić, prolazak kroz labirint, premještanje objekta…). Tada kraj epizode možemo promatrati kao dolazak u konačno stanje pri čemu se dodjeljuje prikladna nagrada (ovisno o tome je li agent i u kojoj je mjeri uspješan u obavljanju zadatka) te započinje nova epizoda, potpuno neovisna o ishodima prethodne. S druge strane, postoje zadaci koji nemaju čvrsto definiran kraj i ne razlamaju se tako lako u epizode. Takve zadatke nazivamo kontinuiranima, a neki su od primjera tvornički proces proizvodnje, mobilni robot s dugim životnim vijekom i slično. Vrlo lako zaključujemo da bi tada konačan korak bio $T = \infty$, što bi posljedično značilo da je i očekivana nagrada koju pokušavamo maksimizirati beskonačna, a naša metoda nikako nije primjenjiva. Kako bi izbjegli ovaj problem, uvodimo pojam *faktor sniženja* (engl. *Discounting*). Jednadžba nagrade sada glasi:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{3–3}$$

gdje je $0 \leq \gamma \leq 1$ *stopa sniženja* (engl. *Discount rate*). Koristeći se ovakvom jednadžbom nagrade, osiguravamo se da je nagrada u *k*-tom koraku γ^{k-1} puta manja od početne nagrade. Time omogućavamo kontinuitet izvođenja algoritma, a istovremeno osiguravamo vrijednost nagrade koja nije beskonačna. Ako raspišemo jednadžbu (3–3), ispoljava se važno svojstvo jednadžbe nagrade i povezanost nagrada kroz korake izvođenja algoritma.

$$\begin{aligned} G_t \doteq\ & R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots) \\ &= R_{t+1} + \gamma(G_{t+1}) \end{aligned} \tag{3–4}$$

Odabirom faktora $\gamma = 0$, agent gleda samo trenutne nagrade kao dominantne i maksimizira samo trenutne nagrade te pokušava izabrati A_t samo tako da maksimizira dobit R_{t+1}. To u određenim slučajevima može biti i dobro, ali generalno, maksimizirajući samo nagradu u idućem koraku, potencijalno zatvaramo pristup budućim nagradama i smanjujemo dugoročnu nagradu. Ovaj efekt sve je manje izražen što se više približavamo $\gamma = 1$.

Postavlja se pitanje kako agent sada može znati koju bi akciju bilo najbolje izabrati. Većina algoritama podržanog učenja temelji se na estimaciji ***funkcije vrijednosti*** (engl. *Value function*). Funkcije vrijednosti funkcije su stanja koje govore o tome koliko je dobro za agenta da bude u danome stanju, što je pak određeno budućim očekivanim nagradama. Kako bi uopće mogli znati kolike su očekivane nagrade u budućnosti, moramo imati nekakvo znanje o akcijama koje će agent iz zadanoga stanja najvjerojatnije poduzeti. Uvodeći pojam ***politike*** (engl. *Policy*) kao funkciju koja na temelju zadanoga stanja daje vjerojatnosti poduzimanja svake od dostupnih akcija, dobivamo informaciju o akcijama koje će agent poduzeti u budućnosti. Tada možemo definirati funkciju vrijednosti kao

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a\right], \text{za svaki } s \in S \tag{3–5}$$

v_π i tada funkcija vrijednosti stanja za politiku π, a $\mathbb{E}_{pi}[\cdot]$ označava očekivanu vrijednost nagrada prateći politiku π. Koristeći se izrazom (3–4), v_π možemo raspisati kao

$$v_\pi(s) \doteq \mathbb{E}[R_{t+1} + \gamma G_{t+1} \mid S_t = s] = \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)\,[r + \gamma v_\pi(s')], \text{za sve } s \in S \quad (3\text{–}6)$$

Jednadžba (3–6) naziva se ***Bellmanova jednadžba*** za v_π, a ističe povezanost funkcije vrijednosti stanja *s* sa sljedećim stanjima.

Zaključno, rješavanje problema podržanog učenja koristeći tablične metode svodi se na pronalaženje optimalnog rješenja Bellmanove jednadžbe, odnosno optimalne politike i funkcije vrijednosti za dani problem, odnosno maksimiziranja nagrade kroz izvršavanje pojedinih epizoda. Kažemo da je politika π' jednaka ili bolja od politike π ako se, prateći politiku π' ostvari jednaka ili veća nagrada kroz sva stanja. Logično je zaključiti da je $\pi' \geq \pi$ ako i samo ako je $v'_\pi(s) \geq v_\pi(s)$. Za svaki problem postoji barem jedna politika za koju ovo vrijedi te tu politiku nazivamo ***optimalna politika***, a funkciju vrijednosti koja ju prati ***optimalna funkcija vrijednosti***, koju možemo zapisati kao $v_*(s) \doteq \max_\pi v_\pi(s)$.

Postoji nekoliko poznatih metoda kojima se rješava problem pronalaska optimalne politike agenta, kao što su: Dinamičko programiranje, Monte-Carlo metode, Temporal-Difference metode i n-step Bootstraping metode. Više o svakoj od ovih metoda može se pronaći u knjizi autora Sutton i Barto, *Reinforcement Learning: An Introduction* [8]. Kako nisu u središtu ovoga rada, navedene metode neće biti detaljnije objašnjene.

3.2. Metode aproksimacije

Iako su matematički potpuno opisane i rješive, agenti kroz podržano učenje rijetko u stvarnosti uspješno dolaze do optimalnih politika. Čak i ako imamo potpuno opisan i osmotriv prostor događanja, zbog opsega problema često nije moguće unutar jednog životnog vijeka izračunati optimalne politike. Dok za neke probleme poput već spomenute igre križić-kružić optimalno rješenje nije problem, već pri pokušaju rješavanja problema poput igre šah dolazi se do prevelikog broja stanja i velikog porasta računalnih zahtjeva.

Već u samoj temeljnoj, Bellmanovoj jednadžbi (3–6), možemo vidjeti zašto velik prostor stanja predstavlja znatan problem. Kako bismo izračunali optimalnu vrijednost

pojedinog stanja, potrebno je znati i višestruko iterirati kroz sva ostala stanja. Čak i ako bismo ispunili memorijske zahtjeve za takav problem, vrijeme bi izračuna u svakom koraku bilo preveliko, čak i za najbolja računala. Također, postoje problemi u kojima pojedina stanja nisu u potpunosti osmotriva ili ne zadovoljavaju već spomenuto Markovljevo svojstvo, nisu sva uvijek dostupna i pojavljuju se nova, do sada neviđena stanja. Neki od primjera tih problema bili bi igra Backgammon, koja ima oko 10^{20} stanja, igra Go sa oko 10^{170} stanja, upravljanje procesom poput helikoptera koji ima kontinuirani prostor stanja… Primjećujemo da ne samo da ne možemo dobiti prave optimalne politike i funkcije vrijednosti, nego nam je i potrebno dobro svojstvo generalizacije kako bismo mogli djelovati u novim, do sada nepoznatim stanjima. Tablične metode tada postaju beskorisne i potreban nam je novi pristup.

Srećom, tijekom zadnjih nekoliko godina problem generalizacije i aproksimacije funkcije vrlo je uspješno riješen na puno načina pa se tako i uz malo prilagodbe lako primjenjuje na probleme podržanog učenja. Problem aproksimacije funkcije jedna je od glavnih tema nadgledanog učenja, koje je vrlo istaknuto u algoritmima strojnog učenja. Teoretski, svi algoritmi nadgledanog učenja primjenjivi su pri aproksimaciji funkcije vrijednosti u podržanom učenju, ali naravno, u praksi su neki nešto bolji izbor od drugih.

No što je uopće cilj aproksimiranja u podržanom učenju? Uvodimo pojam ***srednje kvadratne pogreške vrijednosti*** $\overline{VE}$ (engl. *Mean Squared Value Error*):

$$\overline{VE}(\boldsymbol{w}) \doteq \sum_{s \in S} \mu(s)[v_\pi(s) - \hat{v}(s, \boldsymbol{w})]^2 \tag{3–7}$$

gdje je μ distribucija stanja $\mu(s) \geq 0, \sum_s \mu(s) = 1$ kojom označavamo koliko želimo uzeti u obzir važnost pogreške pojedinog stanja, a $\overline{VE}$ mjera je koja nam govori koliko aproksimirana vrijednost stanja odstupa od prave vrijednosti stanja. $\mu(s)$ često se izračunava kao udio vremena koje agent provodi u stanju s u odnosu na ostala stanja. Više o računanju ovog parametra i razlici između epizodalnog i kontinuiranog slučaja može se pronaći u [8] na stranici 199.

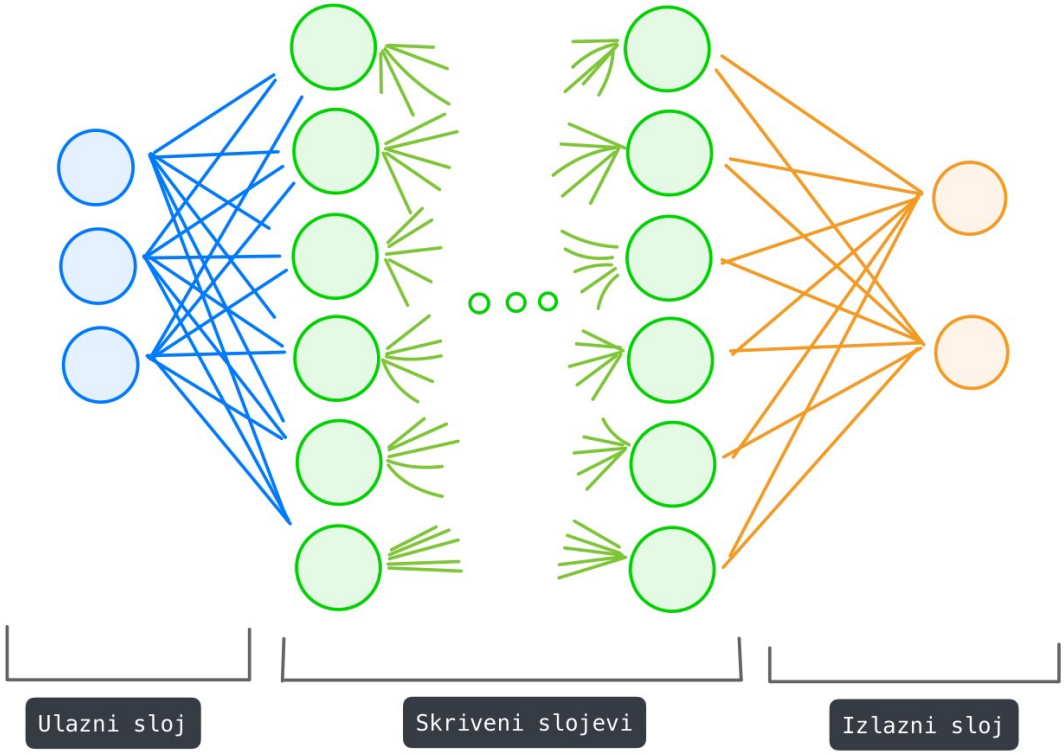
Postoji puno metoda kojima se koristi za aproksimaciju ove funkcije, od kojih su neke metode stohastičkog gradijenta (gradijentna Monte Carlo metoda, semi-gradijentni TD(0)), linearne metode, nelinearni funkcijski aproksimatori (umjetne neuronske mreže) i sl. Više o ovim metodama može se pronaći u [8], a u nastavku će, zbog raširenosti, popularnosti i uspjeha

koji se ostvaruje koristeći se ovom metodom, ukratko biti pojašnjeno kako funkcioniraju neuronske i duboke neuronske mreže

3.2.1. Neuronske mreže i duboke neuronske mreže

Duboko podržano učenje kombinira koncepte podržanog učenja s dubokim neuronskim mrežama te unazad nekoliko godina, ostvaruje vrhunske rezultate. Pokazalo se da koristeći se upravo ovom metodom i iskorištavanjem mogućnosti dubokih neuronskih mreža za prepoznavanje uzoraka na različitim razinama apstrakcije ulaznih podataka, agent može uspješno odrediti kako djelovati. Tako je, na primjer, u slučaju igranja igara agent utreniran ovom metodom uspio postići performanse jednake ljudskima ili bolje (na nekoliko igara s konzole Atari 2600, u igri Go, igrajući poker, Quake III, StarCraft II i u mnogim drugima). Napretkom i razvojem algoritama došlo je i do značajnog napretka na područjima upravljanja robotima, autonomnih vozila, zdravstva, financija i mnogih drugih.

Neuronske mreže (također poznate kao umjetne neuronske mreže) podskup su algoritama strojnog učenja. Generalno gledano, sastoje se od 3 glavnih skupova slojeva: ulazni sloj, skriveni slojevi i izlazni sloj. Svaki od slojeva sastoji se od jedne ili više jedinica koje nazivamo neuroni. Neuroni svakog od slojeva povezani su s neuronima idućeg sloja (ako su povezani svaki sa svakim, stvorila se potpuno povezana mreža, a ako su povezani samo neki s nekima, stvorila se djelomično povezana mreža). Motivacija za stvaranje ovakvog sustava bio je ljudski mozak (pa odatle i naziv neuronske mreže i neuroni), a sličnost i načelnu shemu neuronske mreže možemo vidjeti na slici 3.2. Mreže s tri ili više skrivenih slojeva nazivaju se duboke neuronske mreže.



Slika 3.2 - Arhitektura neuronske mreže

Ulazni sloj i izlazni sloj možemo matematički prikazati vektorima, gdje je ulazni sloj vektor $u = [u_1, u_2, u_3, \dots, u_n]^T$, a predstavlja vrijednosti ulaznih podataka (na primjer crno bijelu sliku veličine 16x16 predstavili bismo kao vektor od 256 elemenata s vrijednostima 0 ili 1 – po jedan element za svaki piksel), a vektor $y = [y_1, y_2, y_3, \dots, y_n]^T$ predstavlja izlazne neurone gdje svaki predstavlja vjerojatnost pojavljivanja određenog događaja ili pripadnosti nekoj klasi (na primjer, ako radimo prepoznavanje rukom pisanih brojeva, vektor y imao bi 10 elemenata, a posljedično i neurona u izlaznom sloju mreže, gdje bi svaki od elemenata predstavljao vjerojatnost pripadnosti ulaznih podataka određenoj klasi).

No kako ova nakupina neurona i veza uspijeva postići ikakve rezultate? Zamislimo da je svaki čvor mreže (neuron) jedan zasebni model linearne regresije koji se sastoji od ulaznih podataka, težine (engl. *Weight*), praga (engl. *Threshold*) i izlaza. Tada ulaz (u) u neuron matematički možemo opisati formulom kao sumu umnoška svih ulaza (x) i njihovih pripadajućih težina (w) zbrojenu s dodijeljenom pristranosti (b) (engl. *Bias*) (jednadžba (3–8)).

$$u = \sum_{i=1}^{n} w_i * x_i + b \tag{3–8}$$

Izlaz (*y*) iz neurona možemo predstaviti zadanom aktivacijskom funkcijom dodijeljenoj neuronu (neke od najpoznatijih aktivacijskih funkcija su prag, linearna, sigmoid, ReLu…). Zbog jednostavnosti, na jednadžbi (3-2) napisana je samo prag funkcija.

$$y(u) = \begin{cases} 1, & u \geq prag \\ 0, & u < prag \end{cases} \tag{3–9}$$

Izlaz svakog od neurona iz jednog sloja predstavlja ulaze u neuron idućeg sloja, osim ako je u pitanju zadnji, izlazni sloj, kada su izlazi neurona upravo rezultati rada mreže.

Sada kada znamo kako dobiti nekakav izlaz na temelju ulaza, nameće se pitanje kako treniramo mrežu i učimo ju da dobije izlaz koji očekujemo i kako uopće znamo je li to izlaz koji tražimo? Veliku ulogu u vrijednostima izlaznog sloja očito imaju težine svake od veza između neurona. Kako bismo ih mogli podesiti, potrebno je procijeniti valjanost izlaza koji je generiran. Jedan od načina korištenje je srednjom kvadratnom pogreškom (engl. *Mean square error*) (jednadžba (3–10)).

$$MSE(w, b) = \frac{1}{2n} \sum_{n} \left|\left| y(u) - \hat{y} \right|\right|^2 \tag{3–10}$$

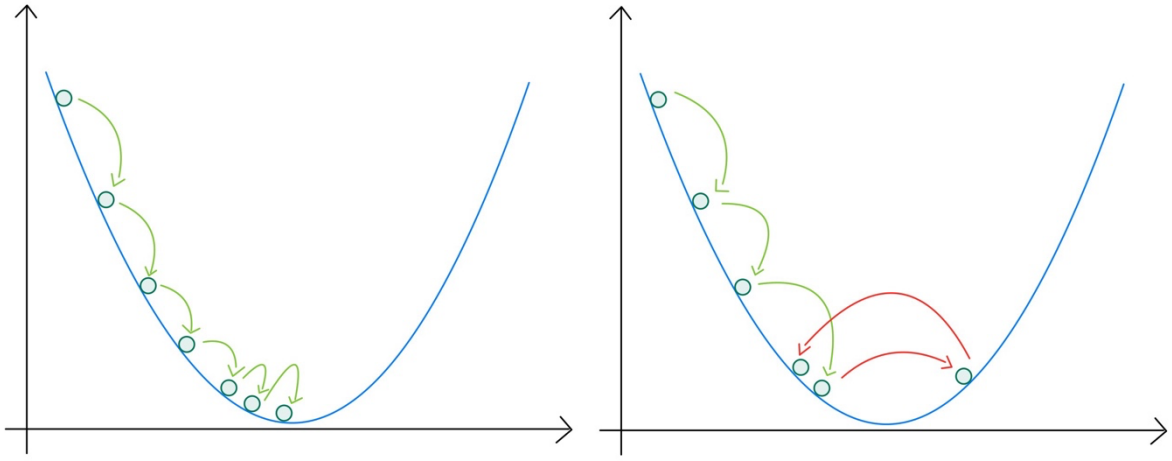
y(u) predstavlja izlaz iz naše neuronske mreže, dok $\hat{y}$ predstavlja očekivani izlaz. Cilj je treninga mreže, koristeći se ovakvim kriterijem (odnosno funkcijom troška) postići da MSE izlaza i očekivanog izlaza konvergira u nulu.

Funkciju troška možemo minimizirati koristeći se metodom gradijentnog spusta, odnosno računajući parcijalne derivacije funkcije troška po težinama i po pristranostima (jednadžba 3-4). Taj nam podatak govori o iznosu i smjeru gradijenta u kojemu se trebamo kretati kako bismo daljnje smanjili funkciju troška. Tada svakoj od težina i pristranosti možemo dodati izračunate vrijednosti umanjene za faktor nazvan stopa učenja (engl. *Learning rate*). Ponavljanjem ovog postupka možemo lako doći do lokalnog minimuma funkcije troška te na takav način postići željene izlaze iz neuronske mreže.

$$MSE'(w,b) = \begin{bmatrix} \frac{dMSE}{dw} \\ \frac{dMSE}{db} \end{bmatrix} = \begin{bmatrix} \Delta w \\ \Delta b \end{bmatrix} \quad (3\text{–}11)$$

Tada svaku od težina iz mreže možemo zapisati kao $w_i = w_i + \Delta w * LR$, a svaku od pristranosti kao $b_i = b_i + \Delta b * LR$ gdje LR predstavlja stopu učenja. Izbor stope učenja važan nam je zbog brzine spuštanja po funkciji troška. Ako izaberemo veliku brzinu spusta, brzo ćemo doći u okolinu minimuma funkcije, ali postoji vrlo velika vjerojatnost da ćemo sam minimum promašiti i divergirati oko njega. Ukoliko izaberemo malenu brzinu spusta, zasigurno ćemo moći doći u minimum, ali pitanje je koliko će nam vremena trebati da ga dosegnemo. Sigurnost dostizanja minimuma nije nam važna ako ne stignemo u jednom životu doći do njega. Često je poželjno izabrati prilagodljivu brzinu učenja kako bismo što brže došli u neku okolinu minimuma pa onda smanjili brzinu spusta kako bismo bili sigurni da ćemo u doći u minimum. Na slici 3.3 prikazani su dobro izabrani LR i prevelik LR.

Kvalitetan izbor stope učenja posebno dolazi do izražaja pri minimiziranju funkcija s više lokalnih minimuma (koje su već same po sebi jako velik problem neuronskim mrežama), gdje će manji LR potencijalno zapeti u ne-globalnom minimumu, a prevelik LR preskočiti globalni minimum i možda se nikada ne vratiti u njega.



Slika 3.3 Utjecaj izbora stope učenja na pronalazak minimuma funkcije

Iako takvi univerzalni aproksimatori funkcija djeluju kao izvrstan način za rješavanje problema aproksimacije funkcija vrijednost, postoje problemi i kod njih. Osim odabira stope učenja, moguće je da mreža zapne u jednom od lokalnih minimuma te tako neće pronaći

optimalnu funkciju. Također, koristeći se algoritmom unazadne propagacije (engl. *Backpropagation*), pojavljuje se problem izbora broja skrivenih slojeva, gdje pri odabiru pretjerano velikog broja dolazi do overfittinga (engl. Overfitting) gdje se mreža pretjerano prilagođava ulaznim podacima na kojim je trenirana te gubi na sposobnosti generalizacije. Vrijedi spomenuti da se često koriste i konvolucijske neuronske mreže koje bolje toleriraju višedimenzionalni ulazni prostor poput slika.

3.3. Metode gradijenta politike (engl. *Policy Gradient Methods*)

Svi algoritmi podržanog učenja o kojima je do sada bilo govora bili su takozvani algoritmi temeljeni na vrijednosnoj funkciji (engl. *Value-Based algorithms*). To je značilo da za određivanje najbolje akcije ne pratimo direktno politiku, nego računamo koliko je dobro da se agent u pojedinom trenutku nađe u pojedinom stanju te koju akciju tada da napravi na temelju nagrade koju dobije, odnosno akcija je uvjetovana i poznata tek nakon izračuna funkcije vrijednosti. Druga porodica algoritama su algoritmi temeljeni na politici (engl. *Policy-Based algorithms*), koji se eksplicitno koriste funkcijom politike kako bi odredili vjerojatnosti odabira svake od mogućih akcija iz pojedinog stanja. Pri tome se ne uzima u obzir vrijednosna funkcija stanja (ali se njome može koristiti kako bi se došlo do optimalnih parametara politike) nego jedino funkcija politike koja estimira i povezuje vjerojatnost poduzimanja pojedine akcije sa stanjima. Kažemo da politika više nije deterministička ($\pi: s \rightarrow a$) nego stohastička.

Označimo vektor parametara politike sa $\boldsymbol{\theta} \in \mathbb{R}^{d'}$. Tada vjerojatnost poduzimanja akcije a, ako je okolina u stanju s u trenutku t sa parametrom $\boldsymbol{\theta}$, možemo napisati kao $\pi(a\,|s, \boldsymbol{\theta}) = \Pr\{A_t = a\,|S_t = s, \boldsymbol{\theta}_t = \boldsymbol{\theta}\}$. Kako bismo postigli željene rezultate koristeći se metodama gradijenta politike i maksimizirali performanse, potrebno je izvršiti stohastički gradijentni uspon:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha\widehat{\nabla J(\boldsymbol{\theta}_t)} \tag{3–12}$$

gdje je $\widehat{\nabla J(\boldsymbol{\theta}_t)}$ očekivanje skalarne vrijednosti u odnosu na parametar politike $\boldsymbol{\theta}$, odnosno jednostavnije rečeno, očekivana nagrada $J(\boldsymbol{\theta}) = \mathbb{E}_\pi[r(\tau)]$ (gdje τ označava putanju), a α stopa učenja. Da bi se agentu omogućilo istraživanje, važno je napomenuti da politika ni u jednom trenutku ne smije postati deterministička (odnosno mora vrijediti $\pi(a\,|s, \boldsymbol{\theta}) \in (0, 1), \forall s, \forall a, \forall \boldsymbol{\theta}$). Dakle, korištenjem metode gradijenta politike umjesto odabira jedne direktne akcije, uzorkujemo akcije iz razdiobe vjerojatnosti koja se prilagođava

po vektoru parametara politike $\boldsymbol{\theta}$ s ciljem da povisimo vjerojatnost odabira onih akcija koje nam svojim djelovanjem daju visoku nagradu.

3.3.1. Teorem gradijenta politike

Uzmimo sada u obzir epizodalni slučaj gdje definiramo parametar performansa kao

$$J(\boldsymbol{\theta}) \doteq v_{\pi_\theta}(s_0) \tag{3–13}$$

gdje $v_{\pi_\theta}(s_0)$ označava vrijednost početnog stanja epizode. Sada možemo, koristeći se ***teoremom gradijenta politike*** (dokaz teorema može se pronaći u [8]), uvesti izraz (3–14) kao analitički izraz za gradijent performanse s obzirom na parametar politike.

$$\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a q_\pi(s,a) \nabla \pi(a\,|s, \boldsymbol{\theta}) \tag{3–14}$$

$\mu(s)$ tada predstavlja faktor opisan u (3–7), q_π predstavlja drugu sumu iz Bellmanove jednadžbe (3–6), a π predstavlja politiku koja odgovara parametarskom vektoru $\boldsymbol{\theta}$. Važno nam je samo da parametar performanse bude proporcionalan stvarnom gradijentu jer razliku između proporcionalnosti i jednakosti možemo nadoknaditi parametrom α iz (3–12). Daljnjim matematičkim izvodom i prilagodbom zadane funkcije ([8], stranice 326 i 327), možemo doći do REINFORCE algoritma, koji će biti baza na kojoj su izgrađeni svi ostali algoritmi gradijenta politike:

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha G_t \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta}_t)}{\pi(A_t|S_t, \boldsymbol{\theta}_t)} \tag{3–15}$$

Jednadžbom (3–15) pokazujemo da je svako ažuriranje parametra politike proporcionalno nagradi G_t (3–3) i vektoru koji označava omjer gradijenta vjerojatnosti poduzete akcije i vjerojatnosti poduzimanja te akcije, što znači da dani vektor u parametarskom prostoru ima smjer takav da maksimalno povećava vjerojatnost poduzimanja akcije A_t pri budućim posjetima stanja S_t te da se vektor parametra politike u ovom smjeru povećava u proporciji sa nagradom koja slijedi i obrnuto proporcionalno vjerojatnosti poduzimanja akcije. Radi kasnije

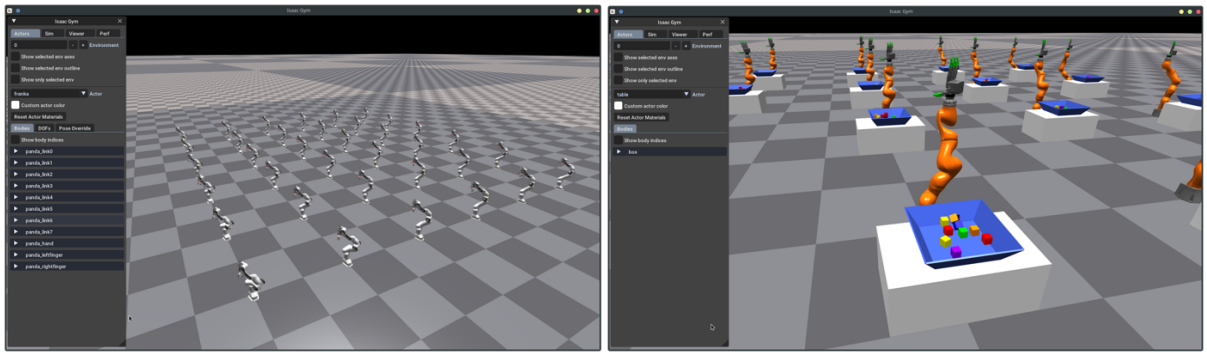
lakše programske implementacije, vrijedi istaknuti da (3–15), prateći svojstvo gradijenta po kojemu je $\nabla \ln x = \frac{\nabla x}{x}$ možemo zapisati kao

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha G_t \nabla \ln (\pi(A_t|S_t, \boldsymbol{\theta}_t)) \quad (3–16)$$

Glavni je izazov koji se ističe pri izvođenju gradijenta politike velika varijanca gradijenta. Neki od algoritama koji su temeljeni na gradijentu politike su: A2C, ACER, TRPO, PPO i drugi, a više o njima može se pronaći u [8] [9] [10] [11].

4. IMPLEMENTACIJA

Vodeći se radom „*Learning to Walk in Minutes Using Massively Parallel Deep Reinforcement Learning*“ [7] i otvorenim kodom koji su autori ponudili, za ovaj će diplomski rad također biti korišten simulator Isaac Gym [12] tvrtke Nvidia, koji je napravljen upravo u svrhu razvoja robotskih rješenja. Na slici 4.1. mogu se vidjeti primjeri simulacije robotskih manipulatora ponuđenih u sklopu instalacije programskog paketa Isaac Gym-a, Franka robotski manipulator i Kuka robotski manipulator sa šakom nalik ljudskoj, kojemu je zadatak podignuti predmete iz kutije. Simulacija ima dovoljno vjernu fiziku da se većina algoritama i događaja iz nje može dovoljno dobro prenijeti u pravi svijet bez pretjerano velikih prilagodbi.



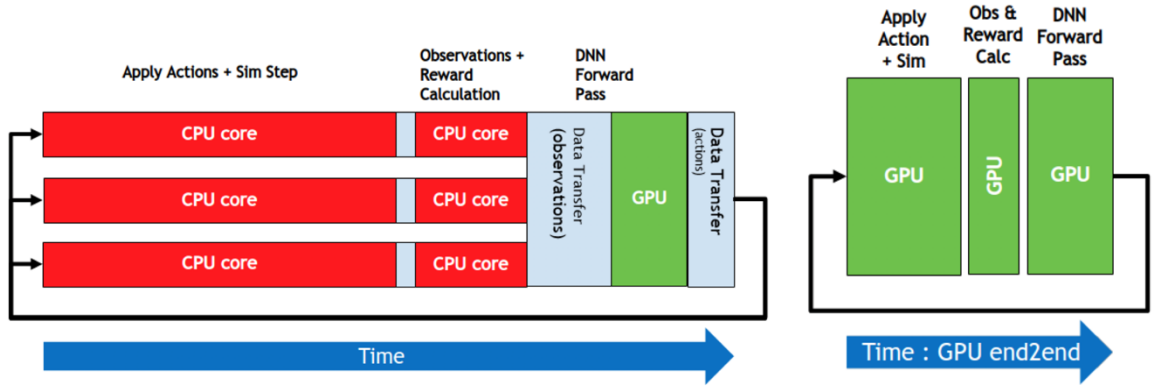
Slika 4.1 Primjeri iz programske biblioteke Isaac Gym

Kao temeljna robotska platforma za razvoj algoritama izabran je robot ANYmal C [13] (Slika 4.2), koji su na raspolaganje stavili autori rada (uz robote ANYmal B, Cassie i A1). Kroz sljedećih nekoliko poglavlja bit će opisane osnovne postavke simulacije, način implementacije i kojim se postupcima došlo do zadanog cilja učenja robota da samostalno održava ravnotežu, hoda, prelazi preko prepreka i zahtjevnog terena te dosegne korisnički zadane ciljne koordinate.

Slika 4.2 ANYmal C robot [13]

4.1. Isaac Gym

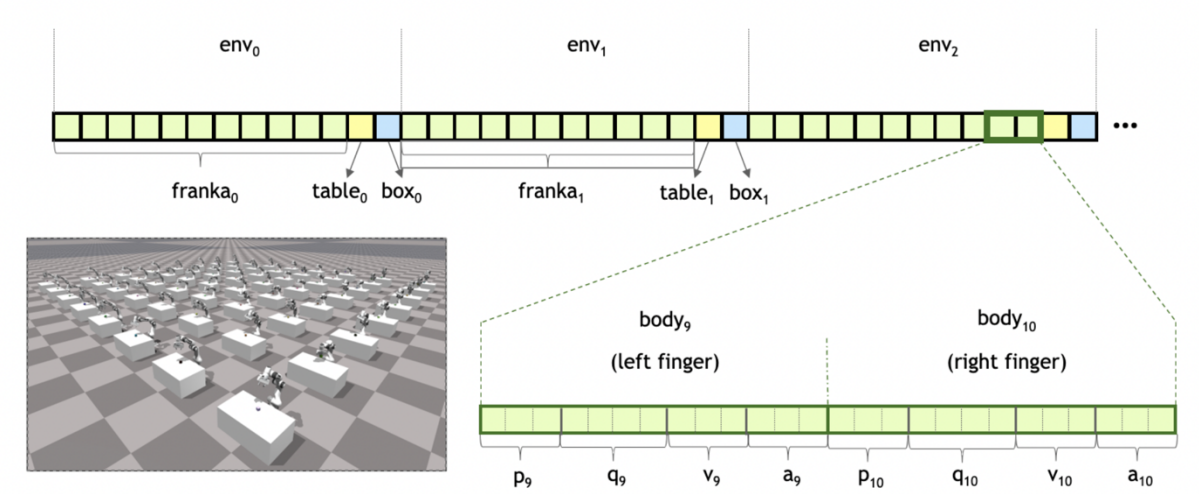
Porastom popularnosti i uspjeha RL-a pojavila se potreba za kvalitetnim simulatorom koji omogućava ne samo efikasno i brzo implementiranje nego i brz trening i izvođenje algoritama. Osim toga, sve većom uporabom podržanog učenja na području robotike pojavila se potreba za kvalitetnom simulacijom jer iznalaženje politika na stvarnim robotima često može biti dugotrajno i vrlo skupo. Upravo su ove probleme uspjeli riješiti autori rada *Isaac Gym: High Performance GPU-Based Physics Simulation For Robot Learning* [14], stvorivši alat Isaac Gym. Posebnost je ovog simulatora, u odnosu na neke druge vrlo uspješne i poznate simulatore (MuJoCo [15], PyBullet [16], V-Rep [17]), upravo brzina izvođenja algoritama i mogućnost korištenja velikim brojem agenata istovremeno. Autori su uvidjeli problem pri pokretanju dijela simulacije na CPU-u i prijenosu podataka između CPU-a i GPU-a te ostvarili 2-3 puta brže izvođenje pri obavljanju kontinuiranih zadataka. Koristeći se NVIDIA PhysX-om kao temeljem GPU-a, ubrzane programske podrške omogućavaju korisnicima iskorištavanje velike mogućnosti paralelizma pri radu na grafičkim karticama. Ovim postupkom omogućili su pokretanje simulacija na običnim desktop računalima i eliminirali potrebu za ogromnim podatkovnim centrima sa stotinama i tisućama centralnih procesorskih jedinica.



Slika 4.3 a) Prikupljanje znanja na tradicionalnim CPU simulacijama. b) Prikupljanje znanja u Isaac Gym [14]

Kroz proceduralni API u Python programskom jeziku, korisniku se omogućava stvaranje okoline i učitavanje agenta koristeći se URDF ili MJCF podatkovnim oblicima. Svaki je agent predstavljen kao zasebna cjelina sa svojim vlastitim okruženjem u okviru kojega može djelovati neovisno o drugim agentima te mu je omogućeno stvaranje više jednakih okruženja koji će istraživati i učiti svaki za sebe. Korisniku se također prepušta odabir hoće li algoritam učenja pokrenuti na CPU-u ili GPU-u.

Podaci u Isaac Gym-u predstavljeni su tenzorima unutar kojih se nalaze detalji o svakom od okruženja i agenata u njima za svakog od agenta.



Slika 4.4 Primjer tenzora unutar Isaac Gym-a na temelju primjera Franka manipulatora [14]

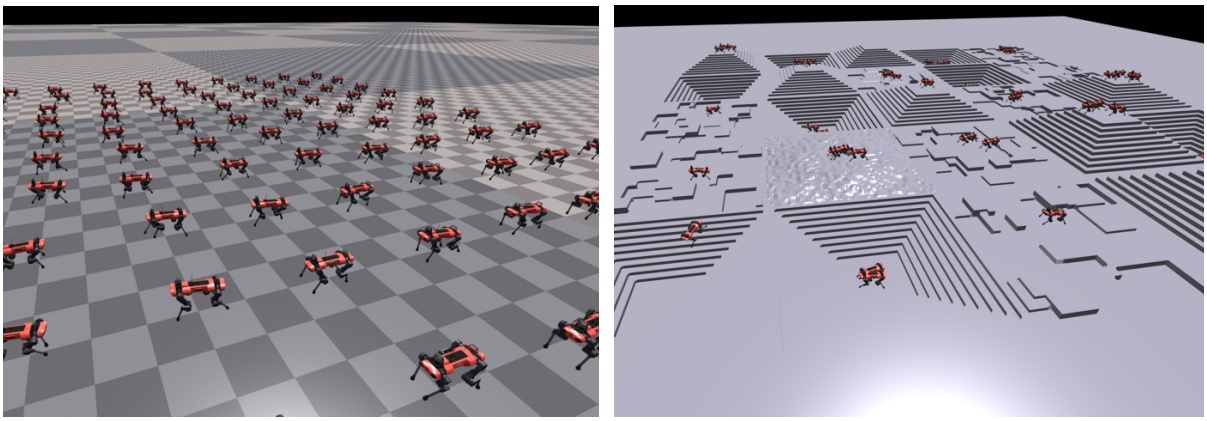
Kako bismo pristupili svojstvima agenta, možemo se poslužiti isječkom „Kod 4-1“ gdje vidimo kako prva tri elementa tenzora čine x, y i z koordinate odabranog krutog tijela, daljnja četiri elementa čine elementi rotacije predstavljeni kvaternionom, daljnja tri elementa čine linearne brzine krutih tijela, a nakon toga slijede kutne brzine istoga tijela.

```
root_p = root_states[..., 0:3]     # Pozicije krutih tijela
root_q = root_states[..., 3:7]     # Rotacije krutih tijela u kvaternionima
root_v = root_states[..., 7:10]    # Linearne brzine krutih tijela
root_a = root_states[..., 10:13]   # Kutne brzine krutih tijela
```

Kod 4-1 Pristup podacima jednog od tijela instance agenta [14]

4.2. Simulacija i temeljne postavke

Simulacija i učenje su koncipirani u dva dijela. Prvi dio podrazumijeva omogućavanje samokontrole robotu u jednostavnim uvjetima, na ravnom terenu bez ikakvih prepreka. Drugi dio simulacije podrazumijeva korištenje terenom s preprekama u obliku stepenica prema gore, prema dolje i neravnomjernim terenom. Na slikama 4.3. mogu se vidjeti primjeri i postavke robota ANYmal C-a u obje zadane okoline. Robot je u simulaciji predstavljen korištenjem .urdf datotekom, a vjeran je prikaz fizičke izvedbe ANYmal C-a. Simulacija se izvodi na osobnom računalu sa CPU „*AMD Ryzen 5 2600*“ i „*NVIDIA GeForce GTX 1050 Ti – 4GB*“.



Slika 4.5 Postavke ANYmal C robota u zadanim okolinama

Cilj je učenja robotskog agenta uspješno praćenje naredaba zadanih X i Y brzina te rotacije oko Z-osi bazom tijela robota.

Kao algoritam učenja, implementiran je PPO na način da omogućava potpuni rad na grafičkoj kartici. PPO algoritam definiran je kao

$$\theta_{t+1} = \arg\max_{\theta} \mathbb{E}_{s,a\sim\pi_{\theta_t}}[L(s,a,\theta_t,\theta)\,]$$

$$L(s,a,\theta_t,\theta) = \min\left(\frac{\pi_\theta(a\mid s)}{\pi_{\theta_t}(a\mid s)}A^{\pi_{\theta_t}}(s,a), clip\left(\frac{\pi_\theta(a\mid s)}{\pi_{\theta_t}(a\mid s)}, 1-\epsilon, 1+\epsilon\right)A^{\pi_{\theta_t}}(s\mid a)\right) \quad (4\text{–}1)$$

gdje je ϵ hiperparametar koji ugrubo opisuje koliko udaljena nova politika smije biti od prošle.

Kao vrlo važan parametar autori rada ističu veličinu serija (engl. *Batch size*). „Sa premalo podataka gradijenti postaju prezašumljeni i algoritam ne može dobro učiti. Sa previše podataka uzorci postaju repetitivni i algoritam ne može dobiti više informacija iz njih.“ [7] Ako predstavimo veličinu batcha kao B = $n_{robota} * n_{koraka}$ gdje je n_{koraka} broj koraka koji svaki robot napravi pri ažuriranju politike, a n_{robots} broj robota koji su istovremeno simulirani, vidimo da uslijed povećanja broja robota moramo smanjiti broj koraka. Ipak, autori rada tvrde da broj koraka ne može biti proizvoljno malen jer algoritam u tom slučaju ne uspijeva konvergirati k optimalnom rješenju.

Nagrada koju robot prima pri radu temelji se na sposobnosti da prati naredbe zadane brzine kretanja po X i Y koordinatama prostora te brzine rotacije oko Z-osi te se istovremeno

pokušavaju smanjiti zakretni momenti zglobova robota kako bi se spriječilo trzanje i nagli pokreti pa je nagrada definirana kao [14]:

$$R = c_1 R_{vel,xy} + c_2 R_{vel,yaw} + c_3 R_{torque} \tag{4–2}$$

Drugi je veliki aspekt koji se pokušava ostvariti u ovome radu mogućnost robota da dostigne zadane koordinate na karti. Zbog mogućnosti robota da premosti prepreke i svojstva terena tako da nema nikakvih robotu nepremostivih prepreka (previsokih prepreka ili rupa u okruženju), ovo je implementirano na vrlo jednostavan način.

```
if not reached_destination:
    # Setting robot rotation
    required_yaw = robot_navigation.get_destination_direction(current_positions[robot_index,
:], destination)

    roll, pitch, yaw = robot_navigation.quaternion_to_euler(env.root_states[robot_index, 3],
                                                            env.root_states[robot_index, 4],
                                                            env.root_states[robot_index, 5],
                                                            env.root_states[robot_index, 6])

    print("Angle robot: {}, angle target: {}".format(roll, required_yaw))
    print("Angle difference: {}".format(required_yaw - roll))

    env.commands[robot_index, 3] = required_yaw
    env.commands[robot_index, 0] = 0.0
    env.commands[robot_index, 1] = 0.0

    delta_angle = required_yaw - roll

    if delta_angle > 0.5 or delta_angle < -0.5:
        env.commands[robot_index, 0] = 0
        env.commands[robot_index, 1] = 0
    else:
        env.commands[robot_index, 2] = 0
        env.commands[robot_index, 0] = 0.8
        env.commands[robot_index, 1] = 0

else:
    print("DESTINATION REACHED")
    env.commands[robot_index, 0] = 0
    env.commands[robot_index, 1] = 0
    env.commands[robot_index, 2] = 0
```

Kod 4-2 Dostizanje zadane točke u prostoru

Cilj je zadanoga koda (Kod 4-2) dosegnuti zadanu ciljnu točku. Funkcija *get_destination_direction(...)* računa kut između robota i ciljne točke u globalnom

koordinatnom sustavu koristeći trenutni položaj robota i poziciju ciljne točke na način opisan u nastavku (Kod 4-3). Kontrola robota vrši se pomoću naredbi (engl. *Commands*) gdje se za svakog od robota zadanih određenim indeksom mogu postaviti naredbe brzine kretanja po X koordinati s obzirom na vlastiti koordinatni sustav (*env.commands[robot_index, 0]*, odnosno robot se kreće ravno), brzine kretanja po Y koordinati s obzirom na vlastiti koordinatni sustav (*env.commands[robot_index, 1]*, odnosno robot se kreće postrance) i kutna brzina robota koja se postavlja ili direktno (*env.commands[robot_index, 2]*) ili koristeći zadani kut zakreta s obzirom na globalni koordinatni sustav (engl. *Heading*) (*env.commands[robot_index, 3]*) iz kojeg se onda programski izvodi kutna brzina dokle god robot ne dosegne zadani kut. *required yaw* u kodu predstavlja upravo zadnju od naredbi, odnosno kut koji robot treba imati s obzirom na globalni koordinatni sustav kako bi dosegnuo ciljnu točku krećući se samo pravocrtno, a *roll* predstavlja trenutni kut robota s obzirom na globalni koordinatni sustav.

Nakon što se izračuna i zada potreban kut (odnosno kada se robot orijentira tako da uđe unutar granica pogreške od *+/- 0,5 rad* (*delta_angle*), robot se prestaje rotirati te se pravocrtna brzina robota po X koordinati *env.commands[robot_index, 0]* postavlja na 0,8m/s. Robot se tako kreće sve dok ne dostigne zadanu ϵ okolinu ciljne točke kada se sve naredbe brzina postavljaju na 0. Ukoliko se dogodi da u bilo kojem koraku simulacije robot izađe iz zadanih granica pogreške kuta, brzina kretanja se postavlja na 0 te se robot ponovno rotira dok ne smanji pogrešku kuta do tolerantne razine.

Kut koji robot treba ostvariti kako bi njegova putanja bila usmjerena direktno prema točki računa se pomoću pitagorinog poučka, odnosno uzima se udaljenost po x-osi, udaljenost po y-osi te se između njih računa $\phi = atan2(\Delta x, \Delta y)$. Koristi se *atan2*-om umjesto *atan*-om zbog očuvanja informacije o smjeru (*atan2* računa kutove između -180 i 180 stupnjeva, dok *atan* računa kutove samo između -90 i 90 stupnjeva)(Kod 4-3).

```
def get_destination_direction(current_robot_position, destination) -> float:
    '''
    Calculate destination based on robot coordinates
    Input:  current_robot_position [(x, y, z) robot position],
            destination [(x, y, z) destination coordinates]
    '''

    delta_x = destination[0] - current_robot_position[0]
    delta_y = destination[1] - current_robot_position[1]

    return math.atan2(delta_y, delta_x)
```

Kod 4-3 Funnkcija izračuna kuta za koji je potrebno zakrenuti robota

Zbog ograničenja korištenja sustavom Eulerovih kutova, u robotici i općenito računalnim simulatorima, tvorci simulatora najčešće se odluče kutove predstaviti sustavom kvaterniona (zapis kuta kao proširenje kompleksnog broja s realnim i 3 imaginarna dijela, $q = q_r + q_i\vec{\imath} + q_j\vec{\jmath} + q_k\vec{k}$ koji jednoznačno određuje rotaciju). Kako bi se odredio kut za koji se robot treba zakrenuti oko Z-osi na intuitivan i ljudima razumljiviji način, potrebno je prvo preći iz sustava kvaterniona kojim se simulator koristi u Eulerov sustav kutova (Kod 4-4)((4–3).

```
def quaternion_to_euler(qw, qx, qy, qz):
    roll = math.atan2((2 * (qw * qx + qy * qz)), 1 - 2 * (qx ** 2 + qy ** 2))
    pitch = math.asin(2 * (qw * qy - qz * qx))
    yaw = math.atan2(2 * (qw * qz + qx * qy), 1 - 2 * (qy ** 2 + qz ** 2))

    return roll, pitch, yaw
```

Kod 4-4 Prelazak iz kvaterniona u Eulerove kutove

$$\begin{bmatrix}\phi\\ \theta\\ \psi\end{bmatrix} = \begin{bmatrix} atan2\big(2(q_0q_1 + q_2q_3), 1 - 2(q_1^2 + q_2^2)\big)\\ \text{asin}\big(2(q_0q_2 - q_3q_1)\big)\\ atan2\big(2(q_0q_3 + q_1q_2), 1 - 2\,(q_2^2 + q_3^2)\big)\end{bmatrix} \quad (4\text{–}3)$$

Robot se po dolasku u zadanu ϵ okolinu oko točke zaustavlja i ostaje nepomično stajati na mjestu.

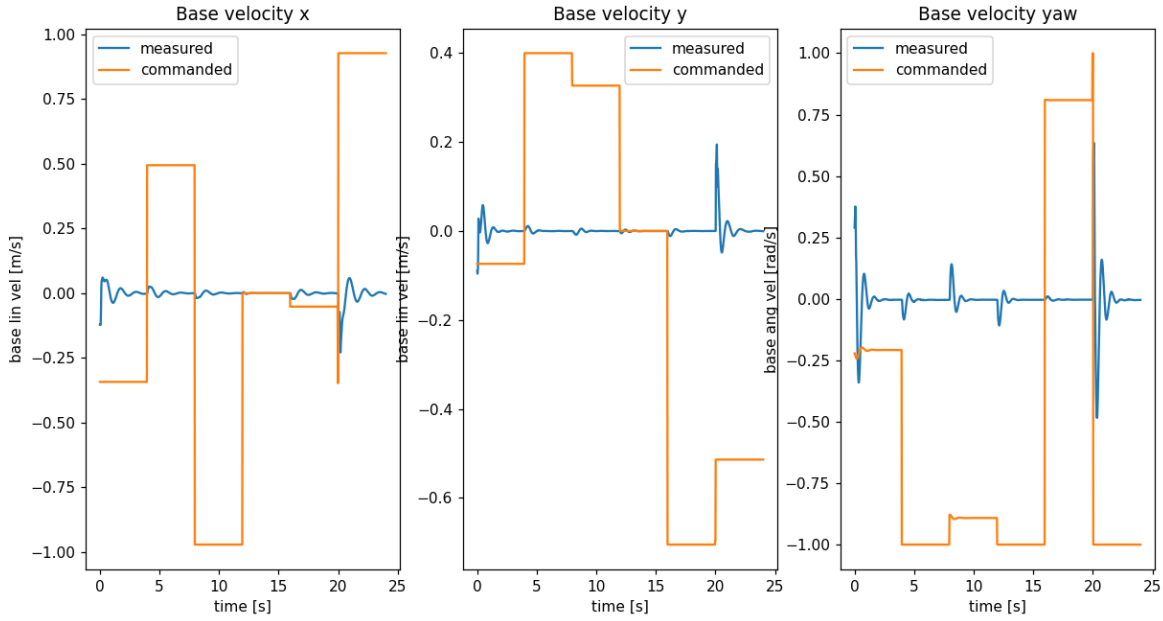
```
def reached_destination(current_robot_position, destination, epsilon) -> bool:
    '''
        Check if robot is less than epsilon units away from the destination
    '''

    dist = math.sqrt(
        (destination[0] - current_robot_position[0]) ** 2 + (destination[1] -
current_robot_position[1]) ** 2)

    if dist < epsilon:
        return True
    return False
```

Kod 4-5 Provjera dolaska u ϵ okolinu zadane točke

5. OSTVARENI REZULTATI

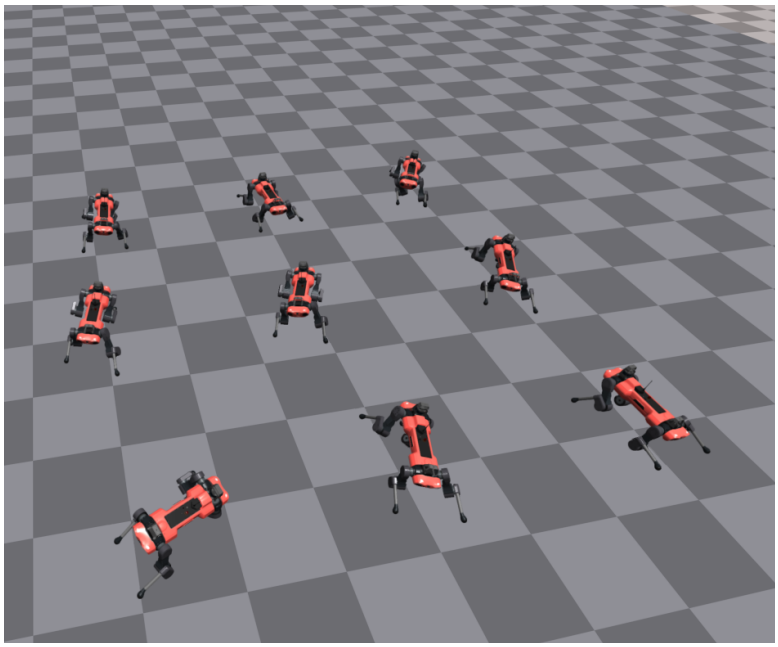
5.1. Slučaj s ravnim okruženjem

Trening sustava u ravnom okruženju pokrenut je s 1024 okruženja s agentima, potpuno kroz GPU. Vrijeme koje je bilo potrebno za izvršenje simulacije i treninga bilo je 20 minuta, pri čemu je izvedeno 1200 potpunih iteracija algoritma. Zbog usporedbe, prvo su na slici Slika 5.1 prikazani grafovi kvalitete praćenja zadanih naredbi prije treninga. Kao referentni agent na kojem se provjeravaju rezultati, uvijek se uzima samo jedan, isti primjerak (uvijek onaj koji je prvi instanciran).



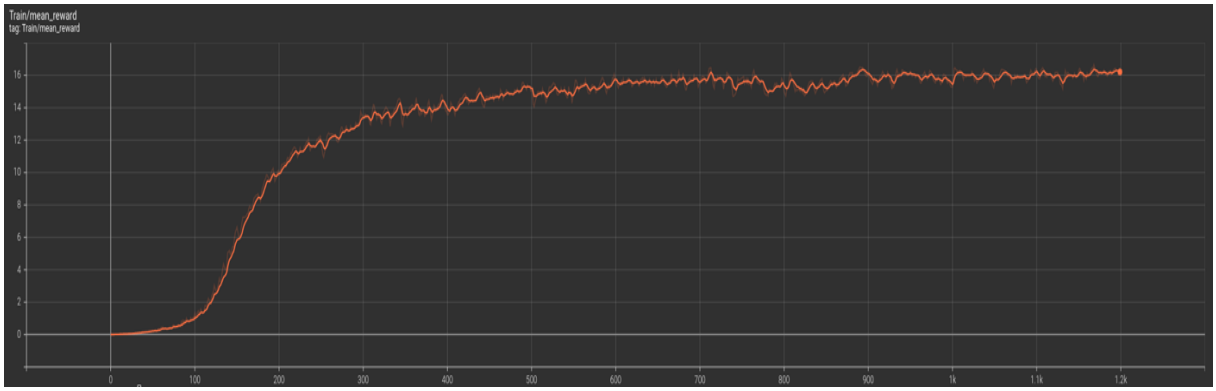
Slika 5.1 Prikaz kvalitete rada robotskog agenta prije treninga na ravnom terenu

Pokretanjem algoritma u ovom trenutku možemo vidjeti da roboti uglavnom stoje nepomično na mapi, a neki čak ni ne mogu uspješno održavati ravnotežu pa padaju (Slika 5.2).



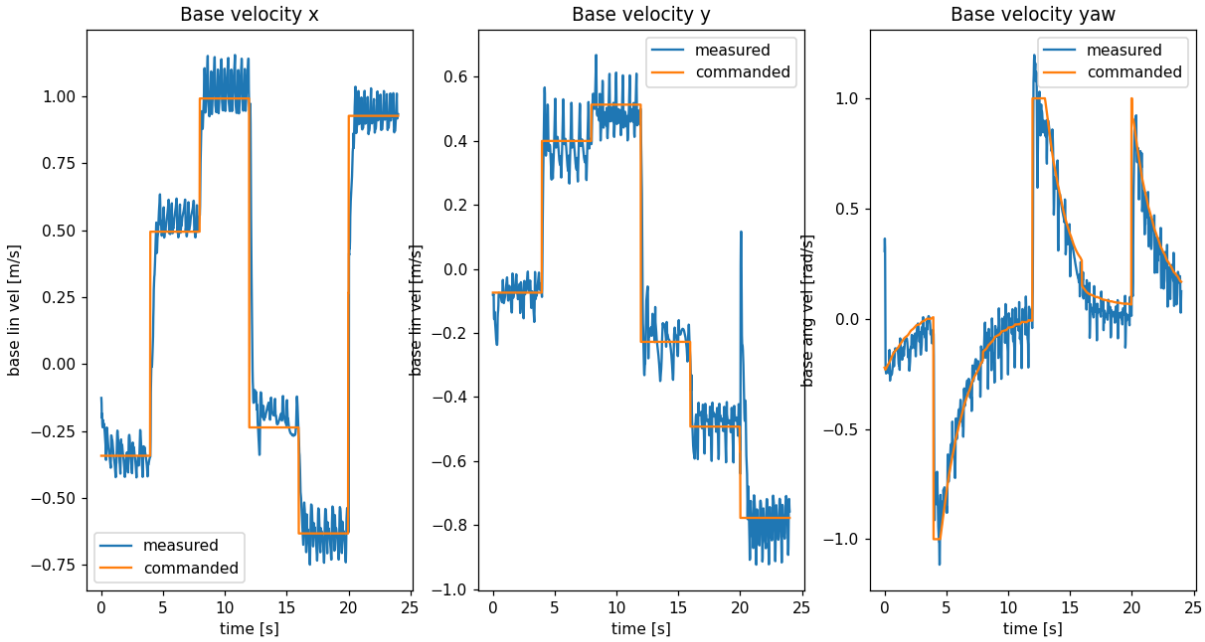
Slika 5.2 Prikaz robota u simulaciji prije treninga

Napredak kroz trening praćen je alatom Tensorboard, a na slici Slika 5.3 prikazano je ostvarivanje nagrade treningom algoritma kroz vrijeme.



Slika 5.3 Prosječna nagrada koju agenti prikupljaju treningom prikazana s obzirom na iteracije algoritma

Vidimo da se nagrada uspješno stabilizirala, što znači da je učenje agenta došlo do otprilike optimalnog rješenja (ili barem dovoljno dobroga da mu je teško dalje napredovati), što je i vidljivo na Slika 5.4. Vidimo da agent vrlo dobro prati naredbe zadane brzine baze po x i y koordinatama te zakreta oko z-osi.

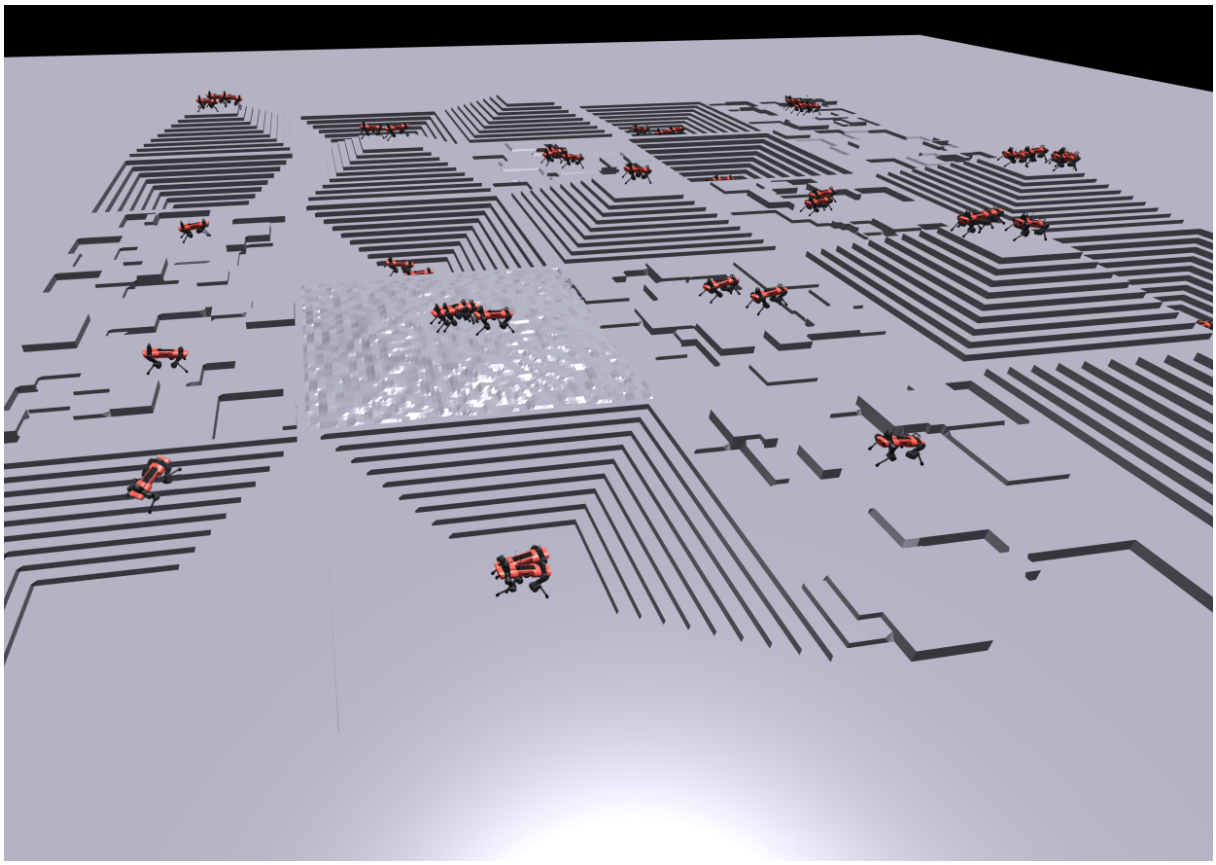


Slika 5.4 Praćenje naredbi utreniranoga agenta

Sada, također promatrajući agenta, subjektivno vidimo da robot gotovo nikada ne pada i kreće se na način koji je očekivan s obzirom na nasumično zadane brzine baze po osima i zakrete. Uz ovako poslušnog agenta, implementacijom koda Kod 4-2 vrlo lako dolazimo do ostvarenja zadanog cilja dolaska robota do točke zadane u prostoru (pogotovo zbog jednostavnosti ravnoga terena).

5.2. Slučaj sa neravnim okruženjem

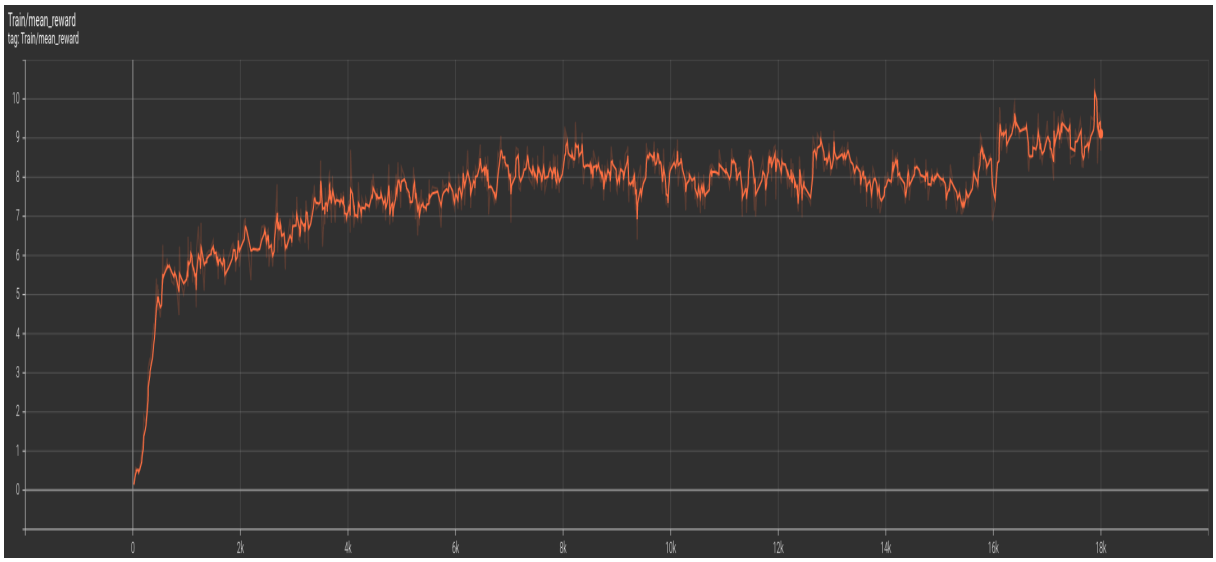
Postavke u slučaju neravnoga terena vrlo su slične onima u slučaju ravnoga terena. Jedna važna razlika bila je ograničenost rada algoritma dostupnom grafičkom memorijom. Dok je algoritam koji je trenirao agenta na ravnome terenu vrlo uspješno podnosio 1024 okruženja s robotima, algoritam s neravnim terenom tolerirao je maksimalno 512 okruženja. Ovaj problem jednim dijelom nastaje i zbog problema s alociranjem memorije koji postoji u sustavu Isaac Gym-a.



Slika 5.5 Prikaz agenta u neravnom okruženju s preprekama

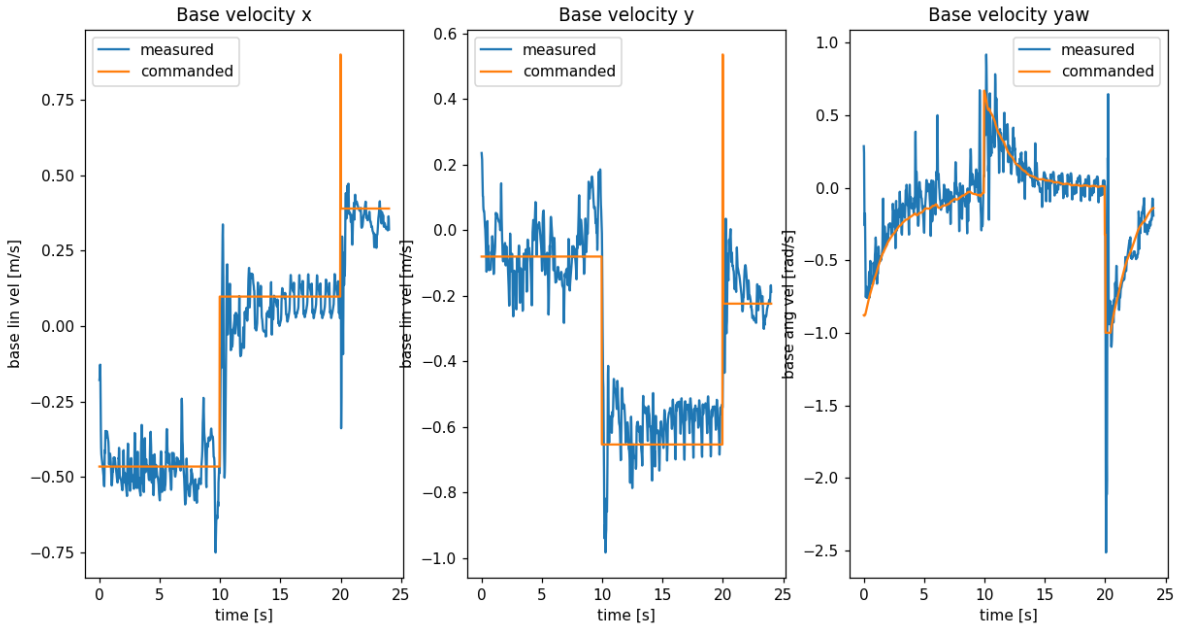
S obzirom na manji broj agenta koji istovremeno rade u odnosu na program s ravnim terenom i veću zahtjevnost treniranja algoritma u ovakvom okruženju, broj iteracija povećan je na 18000, što je zauzvrat produžilo vrijeme treniranja sustava na kvalitetnih 4 sata i 53 minute. Dakako, treba uzeti u obzir i starost korištene grafičke kartice te istaknuti kako današnje prosječno računalo sa prosječnom modernom grafičkom karticom, ovaj algoritam može podnijeti bez problema sa puno više agenta i puno manjim vremenom treninga. Autori rada [7] navode da su koristili 4096 robota i postigli 1500 ažuriranja politike u manje od 20 minuta.

Kao i u prethodnom slučaju, slike u nastavku redom prikazuju prosječnu nagradu koju agent dobiva kroz etape treninga te sposobnost agenta da prati naredbe nakon treninga. Ostatak algoritma za dolazak do zadane pozicije u svijetu implementira se na isti način, neovisno o načinu treninga agenta.



Slika 5.6 Prosječna nagrada koju agenti prikupljaju treningom prikazana u odnosu na iteracije algoritma

Možemo vidjeti da je prosječna nagrada u ovom slučaju znatno manja nego što je bila kod slučaja na ravnom terenu. Ovo se može opravdati većom kompleksnosti terena, ali i činjenicom da je prosječna nagrada sastavljena od nekoliko faktora nagrađivanja pa tako treba uzeti u obzir da se teren na kojem se roboti kreću postepeno kroz rad algoritma i učenje povisuje kako bi se robot polagano prilagođavao različitim terenima (više o ovome može se pronaći u [7]). Ipak, na kraju vidimo da robot poprilično dobro može pratiti zadane naredbe, unatoč tome što još uvijek postoji prostor za dodatnim treningom.



Slika 5.7 Praćenje naredbi utreniranog agenta

Iako vidimo da je robot poprilično nesiguran u svoje radnje (što se manifestira velikom zašumljenošću podataka na grafovima) i dalje radi dovoljno dobro da postigne zadani cilj dolaska do zadane točke bez da se ozlijedi (odnosno da se prevrne ili udari jednim od članaka u samog sebe). Za očekivati je da bi daljnjim treningom ili povećanjem broja agenta koji se istovremeno treniraju, agenti postigli kvalitetu praćenja naredbi gotovo istu kao u slučaju s ravnim terenom.

6. ZAKLJUČAK

Kroz ovaj je rad ponuđen pregled područja podržanog učenja i prikazane su razlike u osnovnim metodama te kada je koja metoda korisna. Pokazano je također kako je moguće stvoriti kvalitetan sustav koji može na visokoj razini rješavati probleme i pratiti zadane naredbe bez posjedovanja potpunog znanja o okolini u kojoj taj sustav djeluje te bez modeliranja okoline. Uspješno je ostvaren cilj treniranja agenta metodama podržanog učenja da prelazi prepreke u zadanom okruženju i uspješno dolazi do točke zadane u njegovoj okolini. Mjesta za napredak svakako ima, kako u simulaciji kroz rad agenta tako i pri prebacivanju agenta u stvarni svijet.

Uzevši u obzir način na koji agent dolazi do zadanog cilja, svakako je moguće implementirati jednu od metoda kojom će se agentu omogućiti snalaženje u prostoru. Implementacijom SLAM algoritama agent bi već postao dovoljno samostalan i snalažljiv da bi mogao obavljati zadatke u stvarnome svijetu, što je, naravno, samo jedan korak u razvoju kvalitetnog robotskog rješenja primjenjivog u industriji i svakodnevici.

Veliki faktor zbog kojeg su se uopće razvile ovakve metode simulacije robotskih sustava svakako je mogućnost učenja agenta u simulacijskom okruženju koje nudi rješavanje problema visoke cijene i smanjenje mogućnosti štete na robotima za vrijeme razvoja i učenja pa je tako smisleno pretpostaviti da bi idući veliki korak bio saznati koliko se dobro znanje agenta u simulatoru može primijeniti na puno manje predvidljiv stvarni svijet. Na ovu temu već postoje radovi i pokazano je upravo kroz rad koji je baza ovome [7] ,kako to zapravo jest moguće i već ostvareno. To je svakako jedan velik korak za razvoj robotike i mobilne robotike te označava vrlo uzbudljivu budućnost i ostavlja prostor za inovacije i nova fascinantna postignuća!

7. LITERATURA

[1] T. Haarnoja, S. Ha, A. Zhou, J. Tan, G. Tucker i S. Levine, Learning to Walk via Deep Reinforcement Learning, Berkeley: Berkeley Artificial Intelligence Research, University of California, 2019.

[2] T. Haarnoja, H. Tang, P. Abbeel i S. Levine, Rein- forcement learning with deep energy-based policies, International Conference on Machine Learning (ICML), 2017.

[3] T. Haarnoja, A. Zhou, P. Abbeel i S. Levine, Soft actor-critic: Off-policy maximum entropy deep reinforce- ment learning with a stochastic actor, In International Conference on Machine Learning (ICML), 2018.

[4] T. Haarnoja, V. Pong, A. Zhou, M. Dalal, P. Abbeel i S. Levine, Composable deep reinforcement learning for robotic manipulation, International Conference on Robotics and Automation, 2018.

[5] A. R. Mahmood, D. Korenkevych, G. Vasan, W. Ma i J. Bergstra, Benchmarking reinforcement learning algorithms on real-world robots, Conference on Robot Learning (CoRL), 2018.

[6] Z. Li, X. Cheng, X. B. Peng, P. Abbeel, S. Levine, G. Berseth i K. Sreenath, Reinforcement Learning for Robust Parameterized Locomotion Control of Bipedal Robots, International Conference on Robotics and Automation (ICRA), 2021.

[7] N. Rudin, D. Hoeller, P. Reist i M. Hutter, Learning to Walk in Minutes Using Massively Parallel Deep Reinforcement Learning, 2021.

[8] R. S. Sutton i A. G. Barto, Reinforcement Learning: An Introduction, Cambridge, Massachusetts; London, England: The MIT Press, 2018.

[9] J. Schulman, F. Wolski, P. Dhariwal, A. Radford i O. Klimov, »Proximal Policy Optimization Algorithms,« OpenAI, 2017.

[10] J. Schulman, S. Levine, P. Moritz, M. Jordan i P. Abbeel, »Trust Region Policy Optimization,« University of California, Berkeley, Department of Electrical Engineering and Computer Sciences, Berkley, California, 2017.

[11] Z. Wang, V. Mnih, N. d. Freitas, V. Bapst, R. Munos, N. Heess i K. Kavukcuoglu, »Sample Efficient Actor-Critic With Experience Replay,« Published as a conference paper at ICLR 2017, 2017.

[12] Nvidia, [Mrežno]. Available: https://developer.nvidia.com/isaac-gym. [Pokušaj pristupa 30 Lipnja 2022].

[13] ANYbotics, »ANYbotics,« [Mrežno]. Available: https://www.anybotics.com/anymal-legged-robot/?from=mergeek.com. [Pokušaj pristupa 30 Lipnja 2022].

[14] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa i G. State, »Isaac Gym: High Performance GPU-Based Physics Simulation For Robot Learning,« 2021.

[15] E. Todorov, T. Erez i Y. Tassa, »Mujoco: A physics engine for model-based control,« 2012.

[16] E. Coumans i Y. Bai, »Pybullet, a python module for physics simulation for games, robotics and machine learning,« 2016.

[17] E. Rohmer, S. P. Singh i M. Freese, »V-rep: A versatile and scalable robot simulation framework,« 2013.

SAŽETAK

Cilj je ovog diplomskog rada ostvarenje mogućnosti stabilnog hodanja i savladavanja prepreka robotskog agenta te dolazak do cilja koristeći metode podržanog učenja. Kroz rad je dan kratak pregled područja te teorijska podloga podržanog učenja te implementacija algoritama za rješenje zadanoga problema. Koristeći već postojeću implementaciju algoritma *Proximal Policy Optimization* (PPO) i simulator Isaac Gym robotski je agent uspješno naučen da prati zadane naredbe i dolazi do cilja zadanog u obliku koordinata na karti.

KLJUČNE RIJEČI

Robotika, mobilna robotika, podržano učenje, simulacija, Proximal Policy Optimization

ABSTRACT

The goal of this thesis is to realize the ability of a robot agent to walk and cross obstacles and reach a given goal using reinforcement learning methods. This paper provides a brief overview of the scientific field and the theoretical background of reinforcement learning, as well as the implementation of algorithms to solve the given problem. Using the existing *Proximal Policy Optimization* algorithm implementation and the Isaac Gym simulator, the robotic agent was successfully trained to follow commands and arrive at a target position provided to it in the form of coordinates in an environment map.

KEY WORDS

Robotics, mobile robotics, reinforcement learning, simulation, Proximal Policy Optimization

ŽIVOTOPIS

Hrvoje Bogadi je rođen 18. siječnja 1998. u Vinkovcima u Hrvatskoj. U Vinkovcima se školuje do završetka gimnazije u kojoj se ističe kroz natjecanja na području informatike i osvajanjem nagrade za najbolje napisani jezični ispit njemačkog jezika (DSD program, certifikat *Deutsches Sprachdiplom der Kultusministerkonferenz*) te timskom pobjedom na Euroscola debati organiziranoj programom Europske unije o tada aktualnim pitanjima EU. Po završetku srednje škole upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija u Osijeku gdje 2020. godine završava preddiplomski studij s titulom sveučilišnog prvostupnika računarstva (univ. bacc. ing. comp.) te se nastavlja školovati na diplomskom studiju Računarstva, smjeru Robotika i umjetna inteligencija. U sklopu školovanja na diplomskom studiju ističe se kao najbolji student na svome smjeru te dobiva Dekanovu nagradu za uspjeh.