# Podržano duboko učenje za hvatanje objekata robotskom rukom

**Žmak, Luka**

SVEUČILIŠTE U ZAGREBU
**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

ZAVRŠNI RAD br. 1539

# PODRŽANO DUBOKO UČENJE ZA HVATANJE OBJEKATA ROBOTSKOM RUKOM

Luka Žmak

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

ZAVRŠNI RAD br. 1539

# PODRŽANO DUBOKO UČENJE ZA HVATANJE OBJEKATA ROBOTSKOM RUKOM

Luka Žmak

Zagreb, lipanj 2024.

**SVEUČILIŠTE U ZAGREBU**
**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

Zagreb, 4. ožujka 2024.

# ZAVRŠNI ZADATAK br. 1539

| | |
|---|---|
| Pristupnik: | **Luka Žmak (0036542692)** |
| Studij: | Elektrotehnika i informacijska tehnologija i Računarstvo |
| Modul: | Računarstvo |
| Mentor: | prof. dr. sc. Ivan Marković |

Zadatak: **Podržano duboko učenje za hvatanje objekata robotskom rukom**

Opis zadatka:

Hvatanje predmeta robotskom rukom bavi se problemom postavljanja izvršnog člana robotske ruke u pozu koja omogućuje čvrsti prihvat objekta, omogućujući podizanje bez klizanja. Donedavno, hvatanje predmeta zahtijevalo je implementaciju algoritama specifičnih za pojedine zadatke, što je zahtjevan proces koji je potrebno ponoviti u slučajevima promjene objekta hvatanja ili okoline robota. U posljednjih nekoliko godina, duboke metode učenja značajno su unaprijedile različita područja, uključujući robotski vid. To je potaknulo istraživače da istraže primjenu podržanog učenja u zadacima robotskog hvatanja. Cilj je ovog završnog rada objasniti ključne značajke potpornog učenja te istražiti kako različiti parametri prilikom treniranja modela za autonomniju robotske ruke utječu na kvalitetu hvatanja.

Rok za predaju rada: 14. lipnja 2024.

Mentor: prof. dr. sc. Ivan Marković

Voditelj rada: dr. sc. Luka Petrović

## Sadržaj

# Uvod

Kako bi roboti postigli širu funkcionalnost, važno je ovladati vještinom hvatanja. Hvatanje opisuje kako postaviti robotski završetak (*engl. end-effector*) kako bi čvrsto uhvatio objekt, omogućujući podizanje bez klizanja. Iako je sama definicija jednostavno za shvatiti, implementacija takvog programa je puno teža.

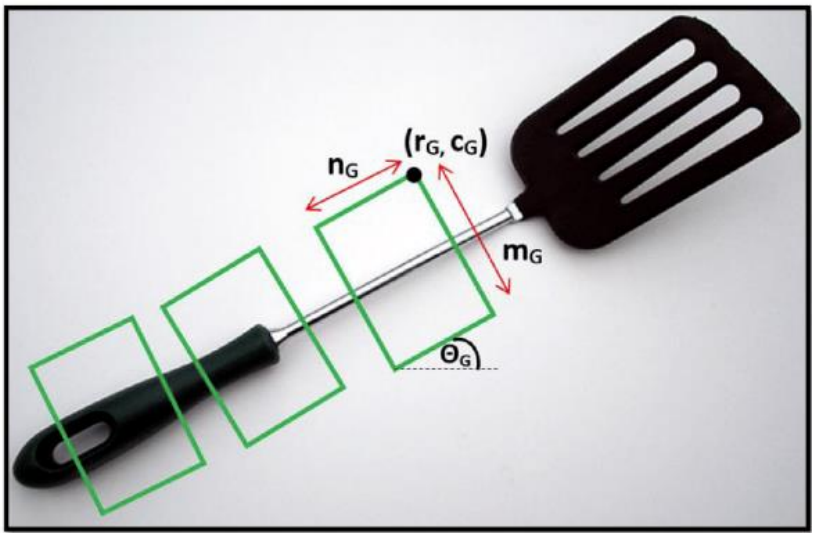
Donedavno, hvatanje predmeta zahtijevalo je stvaranje algoritama specifičnih za pojedine end-effectore i pojedine predmete hvatanja, što je dugotrajan proces koji nije otporan na promjene objekta hvatanja i okolinu robota. Potrebno je uzeti u obzir sve moguće situacije u kojima se hvatač (*engl. gripper*) može naći u odnosu na objekt hvatanja.

Zatim se razvojem strojnog učenja krenulo na ideju generalizacije ovog postupka. Korištenjem metoda nadziranog učenja, otvorila se mogućnost treniranja modela pomoću slika na kojima su labelirane optimalne pozicije hvatanja. (Slika 1)[12] Finalni model na temelju slike predmeta određuje gdje i kako robotski gripper treba biti postavljen. Takva metoda pruža dobre krajnje rezultate, ali potrebna je velika količina podataka za treniranje.

Jedan od mogućih načina prikupljanja podataka je da se prati kako čovjek (koji je ekspert za upravljanje nekim strojem) preko daljinskog upravljača upravlja kretnjama robota. No, na taj način model bi zapravo oponašao čovjeka i nikada ne bi uspio postići bolje i preciznije rezultate, a nadmašivanje mogućnosti čovjeka je upravo ono čemu robotika teži.

U posljednjih nekoliko godina, duboke metode učenja značajno su unaprijedile različita područja, uključujući računarni vid. To je potaknulo istraživače da istraže primjenu podržanog učenja (*engl. reinforcement learning - RL*) u zadacima robotskog hvatanja. Prilikom treniranja agenta koji se bazira na podržanom učenju, podaci na temelju kojih se trenira agent nisu unaprijed poznati, već agent sam stvara podatke iz kojih uči na temelju pokušaja i pogreške.

U ovom završnom radu bavio sam se proučavanjem podržanog učenja i njegove primjene u hvatanju predmeta robotskom rukom. Za istraživanje ove teme koristio sam jedan već napisani kod, koji je poslužio kao osnova za razumijevanje algoritama podržanog učenja u kontekstu robotske manipulacije. Ovaj pristup omogućio mi je da detaljno analiziram strategiju učenja i njegovu primjenu u praktičnom problemu.



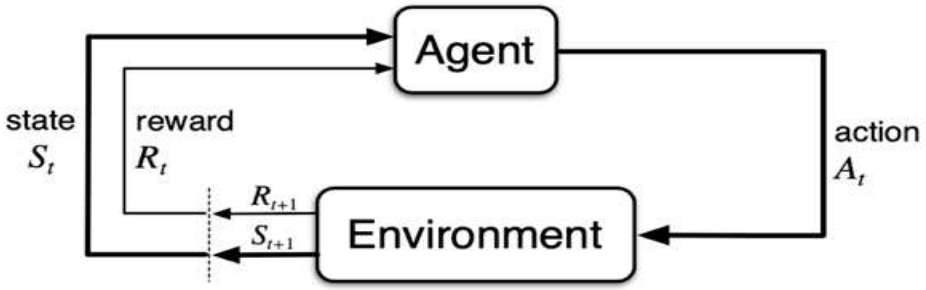

Slika 1 – primjer labelirane fotografije koja služi kao primjer za treniranje modelu nadziranog učenja[12]

# 1. Podržano učenje

## 1.1. Definicija i osnovni koncepti

Podržano učenje je vrsta strojnog učenja u kojem program uči strategiju rješavanja nekog problema stalnom interakcijom sa svojom okolinom. Taj program se naziva agent. Agent je u interakciji s okolinom na način da na temelju svoje strategije bira akciju koju zatim izvršava te za nju dobiva nagradu. Za akcije koje su se pokazale kao bitne ili su vodile ka dobrom rješenju agent biva nagrađen, dok za akcije koje su rezultirale krivim rješenjem biva kažnjen. Takav model nas podsjeća na nas ljude. Ono što dobro napravimo ponavljamo, ono što pogriješimo izbjegavamo.

Ovaj način rada pobliže ću objasniti koristeći sliku. (Slika 2) [9]



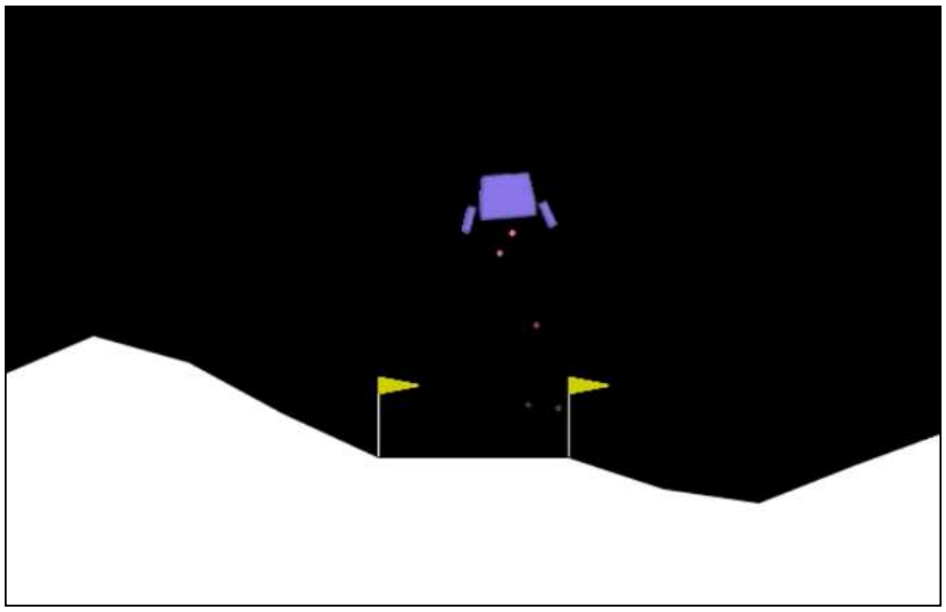

Slika 2 – prikaz interakcije agenta s okolinom [9]

Agent (model) percipira stanje okoline $S_t$ u trenutku $t$ te na temelju znanja o tom stanju bira jednu od dostupnih akcija $a_t$ iz skupa akcija $A_t$ koje su mu dostupne u tom trenutku. Potom agent izvodi izabranu akciju $a_t$ i na taj način djeluje na okolinu i mijenja joj stanje u $S_{t+1}$ te dobiva nagradu $r_{t+1}$. Nagrada je neki realni broj. Ovaj postupak se ponavlja sve dok okolina ne dođe u jedno od svojih terminalnih (završnih) stanja $S_T$. Terminalna stanja su ona iz kojih agent više ne može izvoditi daljnje akcije zbog toga što je ili izvršio svoj zadatak ili je uništen (npr. zaglavio je, isteklo je vrijeme predviđeno za obavljanje zadatka…).

Agent je u konstantnoj interakciji s okolinom u kojoj radi razne akcije od kojih je za neke nagrađen, dok je za druge kažnjen. Na taj način, agent stvara politiku (strategiju donošenja odluka) kojom odlučuje koju akciju primijeniti tako da dobije maksimalnu nagradu. Ovakav pristup omogućuje robotu da naučenu politiku uspješno primijeni na dosad neviđeni predmet u nepoznatoj okolini. Model na početku ne zna koji mu je točan cilj. Ne zna je li to hvatanje predmeta ili mahanje rukom ili nešto treće. Model samo zna da treba na kraju sakupiti što je veću moguću nagradu a kako do toga doći otkriva sam.

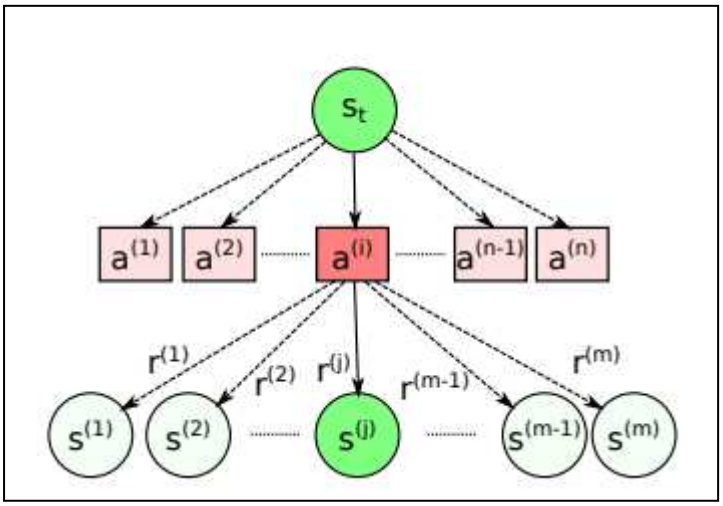
## 1.2. Okoline i Markovljev proces odlučivanja

Okoline s kojima je agent u interakciji mogu biti determinističke i nedeterminističke. Deterministička okolina opisuje situaciju u kojoj, u trenutku $t$ izvođenjem akcije $a_t$ iz stanja $S_t$, možemo sa sigurnošću odrediti koje će biti sljedeće stanje $S_{t+1}$ u kojem će se agent nalaziti. Uzmimo za primjer agenta koji upravlja letjelicom koja treba sletjeti na Mjesec. (Slika 3)[10] Letjelica ima na raspolaganju 2 motora – lijevi i desni te je za očekivati da će se izvršavanjem akcije paljenja lijevog motora, sljedeće stanje letjelice promijeniti na način da se pomakne udesno. U nekakvoj simulaciji mogli bi modelirati da je takvo ponašanje zagarantirano. No, u stvarnosti moguće je da paljenjem lijevog motora ispušteni zrak iz motora se odbije o neku neravnu površinu i da takav potisak zakrene letjelicu u lijevo (protivno očekivanom ponašanju).



Slika 3 – okolina u kojoj letjelica pokušava sletjeti na površinu Mjeseca[10]

Iz tog razloga, za stvarne probleme modeliraju se nedeterminističke okoline. (Slika 4)[11] U takvim okolinama agent u trenutku $t$ i stanju $S_t$ također ima na raspolaganju niz akcija od kojih bira jednu, no sada nije moguće sa 100%-tnom sigurnošću odrediti koje će biti iduće stanje $S_{t+1}$. Sada agent prelazi u jedno od stanja $S^{(1)}…S^{(m)}$.




Slika 4 – model nedeterminističke okoline[11]

Okoline koje se koriste u podržanom učenju moraju zadovoljavati svojstvo da ih je moguće opisati Markovljevim procesom odlučivanja. To znači da je sljedeće stanje okoline $S_{t+1}$ modelirano kao vjerojatnost koja isključivo ovisi o trenutnom stanju $S_t$ i akciji koju agent izvršava $a_t$.

$$p(S_{t+1} \mid S_t, a_t) \tag{1}$$

Sljedeće stanje ne ovisi o nikakvim ostalim parametrima, poput prijašnjih stanja, prijašnjih akcija… Ako uzmemo prethodni primjer sa letjelicom, vjerojatnost da će se letjelica zakrenuti udesno paljenjem lijevog motora je velika, dok je vjerojatnost da će skrenuti lijevo puno manja.

## 1.3. Politika

Izvršavajući akcije, agent od okoline prima nagrade. Što je nagrada veća, to znači da je akcija dobro usmjerila agenta ka ispunjenju svog cilja. Sumu svih nagrada koje je agent priskrbio izvođenjem niza akcija počevši od trenutka $t$ računamo:

$$R_t = r_{t+1} + r_{t+2} * \gamma + r_{t+3} * \gamma^2 + \ldots + r_T * \gamma^{T-t-1} \quad (2)$$

gdje je $r_{t+1}$ nagrada dobivena u stanju $s_{t+1}$, a faktor γ je je broj iz interval [0,1] koji označava kolika je težinska vrijednost nagrade dobivene izvođenjem akcije u nekom trenutku. Primijećujemo da se taj težinski faktor smanjuje što je akcija izvedena u kasnijem trenutku *t*. Uporabu parametra γ možemo opisati jednostavnim primjerom. Ako nam netko ponudi 100 eura sada ili 1000 eura za 10 godina, vjerojatno ćemo izabrati prvu opciju jer više vrednujemo nagrade u trenutnom trenutku.

Način na koji agent bira akciju koju će izvršiti naziva se politika (*engl. policy*) i označava se grčkim slovom π. Politika π treba birati akcije na način da je u terminalnom stanju $s_T$ suma svih nagrada $R_t$ što veća. Suma svih nagrada očito ovisi o izboru politike π, no kako na početku procesa učenja algoritma ne možemo znati koliko će ta suma iznositi, bitno je uvesti pojma vrijednosti stanja. Vrijednost stanja se označava s $v_\pi(s)$ te predstavlja očekivani iznos sume svih nagrada koje agent dobiva počevši od stanja *s* i birajući akcije slijedeći politiku π. Vrijednost stanja se računa:

$$v_\pi(s) = \mathbb{E}[\sum_{k=0}^{\infty} r_{k+t+1}\gamma^k | S_t = ] \quad (3)$$

Ako ovaj izraz raspišemo uz pretpostavku da je okolina nedeterministička i poslužimo se izrazom (1) dobivamo:

$$v_\pi(s) = \sum_a \pi(a \mid s) \sum_{s'} p(s' \mid s, a)\left[r(s, a, s') + \gamma * v_\pi(s')\right] \quad (4)$$

Sličan pojam vrijednosti stanja je vrijednost akcije-stanja $q_\pi(s, a)$ i označava očekivani iznos sume svih nagrada koje agent dobiva počevši od stanja *s* u kojem izvršava akciju *a* i zatim bira naredne akcije slijedeći politiku π.

$$q_\pi(s, a) = \mathbb{E}[\sum_{k=0}^{\infty} r_{k+t+1}\gamma^k | S_t = , a_t = a ] \quad (5)$$

Jednadžbe (4) i (5) se također nazivaju i Bellmanove jednadžbe.

## 1.4. Bellmanove jednadžbe

Bellmanove jednadžbe pružaju rekurzivnu dekompoziciju funkcija vrijednosti uz pretpostavku da na raspolaganju imamo model okoline. Model okoline opisuje skup svih mogućih stanja, svih mogućih akcija, determinističnost djelovanja akcije (ako u stanju *s*

djelujemo akcijom *a*, znamo li sigurno u koje će stanje prijeći okolina *p(s'|s,a),* te nagrade koje dobivamo u određenom stanju. U nekom stanju *s*, agent može poduzeti nekoliko akcija *a* i dobiti odgovarajuću nagradu *r*. Time postoji više mogućih vrijednosti ukupnih suma nagrada *R*. Tada je maksimalna vrijednost stanja *s* u nedeterminističkoj okolini:

$$v^*(s) = \max(a) \sum_{s'} p(s' \mid s, a) \left[ r(s, a, s') + \gamma * v_\pi(s') \right] \quad (6)$$

gdje se zapravo traži najveća *q*(s, a)* na prostoru svih mogućih akcija *a*. Možemo reći da je vrijednost stanja jednaka najvećoj q-vrijednosti stanja gledano po svim akcijama koje se mogu izvesti u stanju *s*.

$$v^*(s) = \max q^*(s, a) \quad (7)$$

Tada je optimalna politika $\pi^*$ ona koja bira akcije koje maksimiziraju optimalne vrijednosti stanja, odnosno maksimiziraju q-vrijednosti.

$$\pi^*(s) = \text{argmax}(a) \sum_{s'} p(s' \mid s, a) \left[ r(s, a, s') + \gamma * v(s') \right] \quad (8)$$

$$\pi^*(s) = \text{argmax}(a)\, q^*(s, a) \quad (9)$$

Iterativnim korištenjem Bellmanovih jednadžbi dobiti ćemo optimalne vrijednosti stanja. Ovaj postupak se naziva iteriranje vrijednosti (*engl. value iteration*).

Alternativni postupak izračuna optimalnih vrijednosti i optimalnog stanja naziva se iteracija politike (*engl. policy iteration*). Postupak se sastoji od biranja neke nasumične politike $\pi_0$ koja svakom stanju pridjeljuje neku akciju, te zatim 2 koraka koja iterativno ponavljamo do konvergencije. Prvi korak je vrednovanje politike koji se obavlja prema izrazu:

$$v_{\pi(i)}(s) = \sum_{s'} p(s' \mid s, a) \left[ r(s, a, s') + \gamma * v_{\pi(i)}(s') \right] \quad (10)$$

Sada se vrijednost stanja ne određuje više po svim mogućim akcijama kao u izrazu (4), već samo po jednoj akciji koja je prema izračunu politike $\pi(i)$ optimalna. Drugi korak je ažuriranje politike koji se provodi na način da za svako stanje provjeri sve akcije te ako utvrdi da izvođenjem neke od njih daje veću očekivanu nagradu, pridjelit će stanju novu akciju. Na taj način se pomoću vrijednosti stanja politike $\pi(i)$, određuje nova politika $\pi(i+1)$. Postupak staje kada se za niti jedno stanje ne promijeni akcija, odnosno kada je politika $\pi(i)$ jednaka politici $\pi(i+1)$.

Problem s prethodnim pristupima je u tome da se temelje na pretpostavci da poznajemo model okoline, tj. ne znamo što očekivati od okoline (vjerojatnosti prijelaza stanja za određenu akciju, vrijednosti nagrada za stanje…).

Kada nam je model okoline crna kutija, otvaraju nam se dvije mogućnosti za učenje optimalne politike $\pi^*$. Prva je da kroz mnoštvo iteracija u kojima puštamo agenta da izvodi nasumične akcije, polako gradimo model izračunavajući vjerojatnosti prijalaza i pripadne nagrade. Na temelju tih podataka možemo ponovno izračunati vrijednosti stanja te prema njima ponovno pustiti agenta da radi i sve bolje procijenjuje vjerojatnosti prijalaza i nagrada sve dok algoritam ne konvergira u $\pi^*$. Ovakav pristup se naziva učenje temeljeno na modelu (*engl. model-based learning*) te je računski vrlo zahtijevan pa se u praksi izbjegava. Puno češći su pristupi koji se ne temelju na modelu (*engl. model-free learning*). Primjeri su algoritam SARSA te Q-učenje.

## 1.5. Q - učenje

Cilj Q-učenja je naučiti Q-funkciju (funkciju vrijednosti akcije) koja određuje koliko je određena akcija u određenom stanju korisna u smislu dobivanja maksimalne ukupne buduće nagrade *R*. Algoritam koristi iterativni postupak za ažuriranje q-vrijednosti na temelju iskustava koja agent stječe tijekom interakcije s okolinom. Algoritam se temelji na sljedećoj osnovnoj formuli za ažuriranje:

$$q(s,a) = (1-\alpha) * \mathrm{q(s,a)} + \alpha * (r(s,a,s') + \gamma * \max q(s',a')) \qquad (11)$$

gdje je α stopa učenja (*engl. learning rate*), koja određuje koliko brzo se ažuriraju q-vrijednosti. Ovaj iterativni pristup konvergira do optimalnih q-vrijednosti $q^*(s,a)$ za veliki broj iteracija. To znači da agent treba istraživati prostor akcija i stanja i tako ažurirati svoje q-vrijednosti sve dok ne dođe do onih optimalnih. No, kada je broj stanja i akcija velik, postaje računski nemoguće istražiti cijeli taj prostor i dovoljno kvalitetno procijeniti optimalne $q^*(s,a)$. Ovaj problem rješavamo tako da za procjenu q-vrijednosti koristimo neuralnu mrežu koju nazivamo Q-mreža. Treniramo ju tako da u svakoj iteraciji podešavamo njene težine smanjujući srednju kvadratnu grešku Bellmanove jednažbe (5).

$$\mathrm{y} = r(s,a,s') + \gamma * \max q(s',a';w) \qquad (12)$$

$$\mathrm{L} = (\mathrm{y} - q(s,\ a;\ w))^2 \qquad (13)$$

Parametre mreže ažuriramo nakon izvođenja svake akcije. Problem ovakvog pristupa je što se *q(s, a)* procijenjuje na temelju *q(s', a')* koju model još nije naučio ili nije naučio dovoljno precizno. Njegova procjena *q(s, a)* se temelji na trenutnim procjenama ostalih q-vrijednosti koje nisu nužno dobre. Na taj način algoritam može u stanju s favorizirati akciju $a_1$ naspram akcije $a_2$ jer ima veću procijenjenu q-vrijednost iako je ta procjena temeljena na ostalim lošim procjenama.

Korištenje isključivo jedne neuralne mreže za procjenu q-vrijednosti pokazalo se kao nestabilna metoda jer agent politiku gradi na temelju procjena q-vrijednosti koje u isto vrijeme uči. Ovaj problem rješavamo koristeći dodatnu ciljnu Q-mrežu (*engl. target network*) i spremnik iskustva (*engl. experience buffer)*. Ciljna Q-mreža ima istu arhitekturu kao i ona glavna, a na početku dijele i iste težine. Ciljnu Q-mrežu označavat ćemo s $Q_T$. Algoritam kreće tako da se izvrši nekoliko akcija na temelju politike koju opisuje glavna neuralna mreža Q. Iskustvo sakupljeno tim akcijama se sprema u spremnik iskustva u obliku (s, a, r, s′). Zatim se iz spremnika iskustva nasumično bira uzorak (*engl. batch*) iskustava. Za svaki uzorak (s, a, r, s′), ciljna vrijednost y se računa kao:

$$y_T = r(s, a, s') + \gamma * \max q_T(s', a'; w_T) \tag{14}$$

a težine w glavne mreže Q ažuriramo minimizirajući grešku:

$$\mathrm{L} = y_T - q(s, a; w))^2 \tag{15}$$

Također, svakih C koraka ažurirat ćemo i težine $Q_T$ mreže koristeći tzv. *soft update*. To znači da ćemo za ažuriranje vrijednosti koristiti sljedeći izraz:

$$w_T \leftarrow \tau * w + (1 - \tau) * w_T \tag{16}$$

Ciljna mreža Q smanjuje korelaciju između Q-vrijednosti i ciljeva, čime se izbjegava problem "lovljenja vlastitog repa" u ažuriranjima.

Pri samom početku treniranja neuronske mreže i dalje ostaje problem da će mreža birati akcije na temelju svojih procjena q-vrijednosti koje su opet bazirane na drugim lošim početnim procjenama. Zato je bitno u početku učenja agentu dopustiti da puno istražuje (*engl. exploration*) – da izabire akcije neovisno o njihovoj trenutnoj q-vrijednosti. U kasnijim fazama učenja, agent bi trebao manje istraživati i više iskorištavati (*engl. exploatation*) već naučene q-vrijednosti. U podržanom učenju taj koncept se naziva kompromis između istraživanja i iskorištavanja. Taj kompromis se rješava određivanjem parametra $\boldsymbol{\varepsilon}$ koji označava vjerojatnost koliko često će agent izabrati nasumičnu akciju koju

će izvršiti, a 1 - $\boldsymbol{\varepsilon}$ koliko često će izabrati akciju za koju zna da je dobra jer ima najveću q-vrijednost za neko stanje. Ovakva politika se naziva $\boldsymbol{\varepsilon}$-pohlepna politika i kreće tako da se pri početku učenja vrijednost $\boldsymbol{\varepsilon}$ postavi na 1, što znači da će agent u stanju $s$ birati nasumične akcije bez obzira na to koje je stanje. Kako treniranje odmiče, $\boldsymbol{\varepsilon}$ vrijednost će se polako povećavati i na taj način u odabiru akcije za stanje $s$ veću prednost dati akciji za koju procjenjuje da maksimizira q-vrijednost.

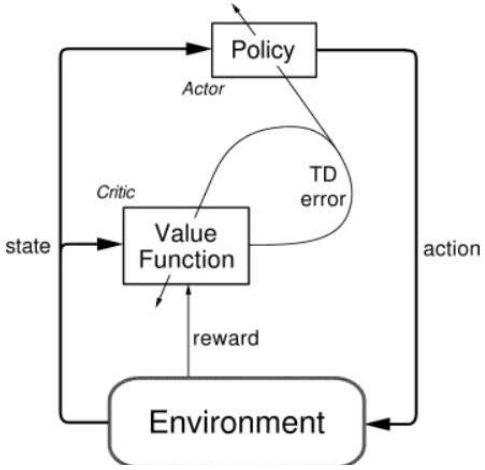
# 2. Akter – kritičar model

## 2.1. Arhitektura akter-kritičar modela

Akter – kritičar model je koncept podržanog učenja koji objedinjuje metode iteracije vrijednosti i iteracije politike. Iteracija politike se očituje u objektu aktera, dok se iteracija vrijednosti očituje u objektu kritičara. Ovaj pristup je osmišljen kako bi se riješila ograničenja koja se pojavljuju kada se svaka metoda koristi zasebno.

Model (agent podržanog učenja) sastoji se od dvije glavne komponente: aktera i kritičara. Ove dvije komponente rade zajedno kako bi omogućile agentu učenje i donošenje odluka u složenim okruženjima. Akter uči politiku kako bi mogao donijeti odluku koju akciju izvršiti, a kritičar vrednuje akciju koju je akter izabrao tako što izračunava vrijednost akcije. (Slika 5).[13]

Obje komponente rade na principu Q-učenja, odnosno postoje dvije ciljne kompomente, jedna za aktera, a druga za kritičara. Na početku, ciljne komponente su jednake svojim glavnim komponentama, no kako se glavne komponente tijekom treniranja mijenjaju ka sve boljem aproksimiranju optimalne politike, odnosno vrijednosti stanja, tako se i ciljne komponente polako prilagođavaju da prate ove promjene (soft update). Ciljne komponente se koriste kako bi izračunale ciljnu vrijednost u funkciji pogreške (*engl. loss function*). Kao što je rečeno u prethodnom poglavlju, ovo osigurava da ciljne vrijednosti i vrijednosti glavne komponente nisu ovisne o istim parametrima te na taj način se pridonosi da je treniranje stabilnije. Za ažuriranje parametara ciljnih komponenti koristi se formula (16).




Slika 5 – arhitektura akter-kritičar modela[13]

## 2.2. Kontekstualizacija modela za hvatanje objekta

Okolina u kojoj agent zasnovan na akter-kritičar modelu djeluje se sastoji od robotske ruke i predmeta hvatanja. na čijem se završetku (*engl. end-effector*) nalazi hvatač (*engl. gripper*). Hvatač se sastoji od 2 paralelna prsta koji imaju mogućnost pomaka po samo jednoj osi te se na taj način mogu međusobno približavati i udaljavati kako bi vršili akciju hvatanja. (Slika 6)[14] S druge strane, sama robotska ruka na kojoj se nalazi hvatač ima puno više mogućnosti. Može mijenjati poziciju u svakom smjeru, tj. može se pomicati po svim osima (x, y, z). Također, ima mogućnost promjene orijentacije u prostoru, odnosno može se zakretati u smjeru osi x (*engl. roll*), osi y (*engl. pitch*) i osi z (*engl. yaw*). Ovime je opisan prostor stanja kao prostor svih mogućih kombinacija pozicije robotske ruke, orijentacije (smjera) u kojem se hvatač može naći (x, y, z, roll, pitch, yaw) te širine na kojoj se nalaze prsti hvatača. Na isti način je opisan i prostor akcija koje agent može napraviti u nekom stanju, a to je promijeniti poziciju ili orijentaciju robotske ruke za proizvoljni pomak. Funkcija nagrade je takva da odredi stanje okoline na temelju informacija o terminalnom stanju, uspjehu pokreta i visini na kojoj se nalazi hvatač. Ako je okolina u terminalnom stanju (zadana je naredba hvatanja, odnosno skupljanja prstiju), onda provjerava je li hvatanje uspjelo. To se provjerava pomoću informacija o visini na kojoj se hvatač nalazi i udaljenosti između prstiju. Ako je uspjelo agentu okolina vraća veliku pozitivnu nagradu, no ako hvatanje nije uspjelo okolina vraća veliku negativnu nagradu. U slučaju da stanje nije terminalno, agent prima nagradu koja je ovisna o udaljenosti od predmeta. Na ovaj način, agentu se sugerira da se mora približiti objektu da bi naredba zatvaranja prstiju hvatača mogla stvarno uhvatiti objekt.

Slika 6 – primjer jednog industrijskog grippera s 2 prsta na robotskoj ruci[14]

U spremnik iskustva se pohranjuju uzorci koji se sastoje od stanja, akcije, nagrade i sljedećeg stanja. Spremnik iskustva ima svojstvo biranja nasumičnih uzoraka koji će služiti za treniranje agenta u jednoj iteraciji algoritma podržanog učenja.

Što se tiče faze istraživanja (exploration), pomoću parametra $\boldsymbol{\varepsilon}$ se kontrolira hoće li izvršena akcija biti ona koju predlaže akter ili će to biti nasumične vrijednosti x, y, z, roll, pitch i yaw. Također, moguća je i treća opcija, a to je da se odredi akcija koju predlaže akter te se onda vrijednosti koje ju definiraju malo promijene. To je dobro koristiti kad se akter već istrenirao do neke mjere, ali ne možemo još biti sigurni u ispravnost njegovih odluka.

## 2.3. Akter

Akter određuje koju akciju izvršiti na temelju trenutnog stanja takvu da maksimizira očekivanu ukupnu nagradu. Možemo reći da akter zapravo modelira politiku koju agent slijedi za odabir akcija. Politika je funkcija koja mapira svako stanje na vjerojatnosti izvođenja određene akcije. U mnogim modernim implementacijama, politika se predstavlja neuronskom mrežom koja kao ulaz uzima trenutno stanje iz uzorka iskustva, a kao izlaz proizvodi distribuciju vjerojatnosti za sve moguće akcije. Akcija na čijem izlaznom neuronu je najveća vjerojatnost je ona koja je optimalna za izvršavanje u tom stanju.

Neuronsku mrežu aktera označavat ćemo s $\pi$, dok ćemo njegovu ciljnu mrežu označavati s $\pi_T$. Neuronsku mrežu kritičara označavat ćemo s Q, a njegovu ciljnu mrežu označavat ćemo s $Q_T$.

Korak treniranja neuronske mreže $\pi$ u jednoj iteraciji prikazat ću pseudokodom:

```
za (s, a, r, s') u uzorak:
        akter_akcija = π(s)
        akter_gubitak = - Q(s, akter_akcija)
        w_π ← w_π − α * ∇_{w_π} akter_gubitak
```

Možemo primijetiti da pseudokod prati teoretsku ideju koja stoji iza modela akter-kritičar. Neuronska mreža $\pi$ koja modelira politiku, bira akciju za neko stanje, a zatim mreža Q, koja modelira funkciju vrijednosti akcije, ocijenjuje izabranu akciju tako da izračunava očekivanu ukupnu nagradu. Vrijednost funkcije vrijednosti akcije je negirana iz razloga što je cilj aktera maksimizirati očekivanu ukupnu nagradu. Funkcija vrijednosti akcije q(s, a) procjenjuje koliko je dobra određena akcija a u stanju s. Dakle, želimo pronaći politiku $\pi$ koja

maksimizira očekivanu q-vrijednost. Međutim, u većini optimizacijskih algoritama, kao što je gradijentni spust (*engl. gradient descent*), minimiziramo funkciju gubitka. Kako bismo maksimizirali q(s, a), moramo minimizirati negativnu vrijednost –q(s, a). Težine neuronske mreže ažuriramo tako da trenutno težine pomaknemo u smjeru gradijenta funkcije:

$$-\nabla_{w_\pi} Q(s, \pi(s))$$

pomnožen s faktorom stope učenja α koji određuje brzinu kojom se parametric mreže mijenjaju.

## 2.4. Kritičar

Procjenjuje očekivanu buduću ukupnu nagradu para stanja i akcije, što je bitno za ocjenu učinka politike i usmjeravanje njezina poboljšanja. Ovo je ključno u podržanom učenju za procjenu koliko je dobra određena akcija u danom stanju. Koristi Temporal Difference (TD) grešku za procjenu koliko je akcija koju je izveo akter bila korisna, tj. koliko je približila agenta postizanju cilja. Temporal Difference greška je razlika između očekivane nagrade i stvarno primljene nagrade. Jednako kao i za aktera, oblikovanje kritičara se često radi pomoću neuronske mreže. Neuronska mreža Q kao ulaz prima dvije informacije – o trenutnom stanju i akciji koja se poduzima. Za razliku od neuronske mreže π kojoj je izlaz distribucija vjerojatnosti odabira neke akcije, izlaz mreže Q je samo jedan broj koji predstavlja očekivanu nagradu. Za procjenu stvarne primljene nagrade koristi se ciljna mreža $Q_T$. Korak treniranja neuronske mreže Q u jednoj iteraciji prikazat ću pseudokodom:

```
za (s, a, r, s') u uzorak:
```

akter_ciljna_akcija = $\pi_T(s')$

kritičar_ciljna_qval = r + $\gamma * Q_T(s'$, akter_ciljna_akcija)

kritičar_očekivana_qval = Q(s, a)

kritičar_gubitak = (kritičar_ciljna_qval - kritičar_očekivana_qval)$^2$

$w_Q \leftarrow w_Q - \alpha * \nabla_{w_Q}$kritičar_gubitak

Ovaj pseudokod ilustrira kako kritičar u koristi ciljne i očekivane q-vrijednosti za ažuriranje težina svoje neuronske mreže. Kritičar koristi te vrijednosti kako bi izračunao gubitak, a zatim prilagođava svoje parametre kako bi minimizirao taj gubitak. Ovaj postupak

omogućuje kritičaru da kontinuirano poboljšava svoje procjene q-vrijednosti, što zauzvrat pomaže akteru da donosi bolje odluke u biranju akcija za neko stanje.

Sada možemo primijetiti kako akter koristi mrežu kritičara i q-vrijednosti koje ona pruža kako bi optimizirao svoje težine, a kritičar koristi ciljne mreže i aktera i kritičara kako bi optimizirao svoje.

Iako ciljne mreže za aktera i kritičara na početku algoritma dijele iste težine kao i pripadajuće glavne mreže, tijekom algoritma treniranja njihove težine, a time i procjene koje pružaju se počinju razlikovati. Nakon svake iteracije u kojoj su se optimizirale težine glavnih mreža, slijedi ažuriranje ciljnih mreža koristeći mehanizam soft update radi stabilnosti tijekom treninga.

$$w_{\pi_T} \leftarrow \tau * w_{\pi} + (1 - \tau) * w_{\pi_T}$$

$$w_{Q_T} \leftarrow \tau * w_{Q} + (1 - \tau) * w_{Q_T}$$

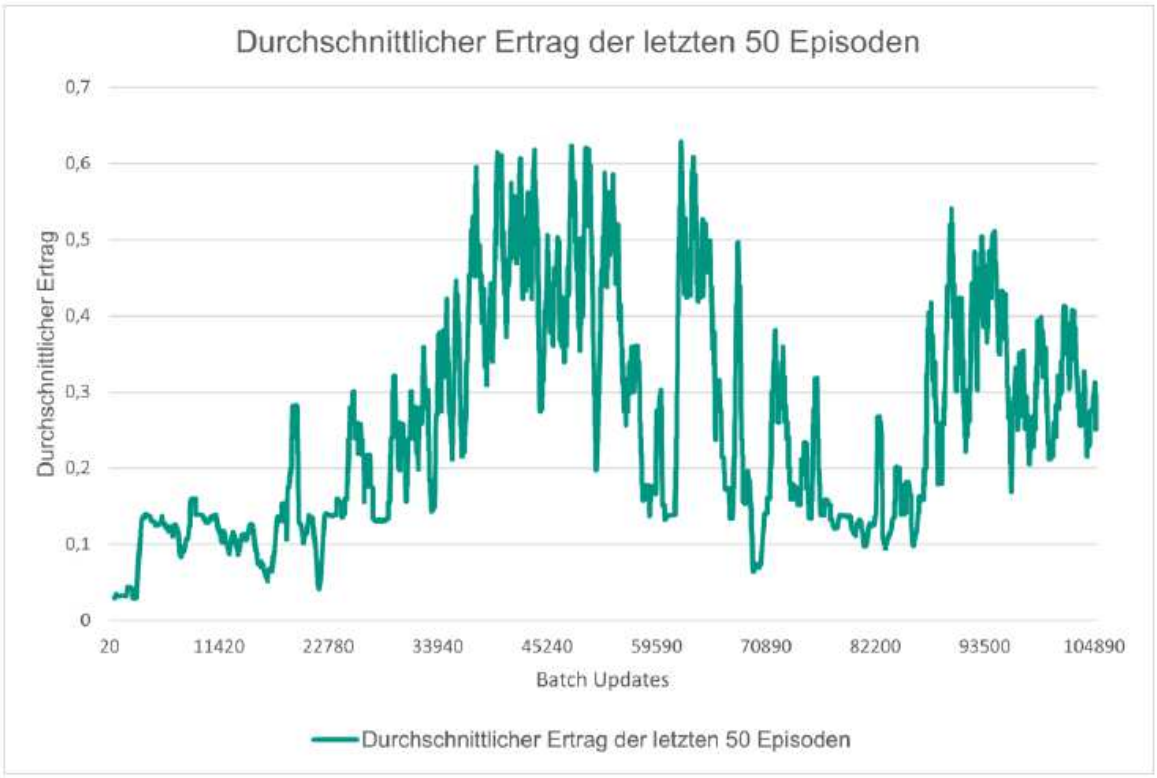
Faktor $\tau$ (tau) određuje proporciju koliko će se težine glavne neuronske mreže nadodati na težine ciljne mreže.

# 3. Rezultati

Za donošenje zaključaka koristiti ću rezultate iz rada [5] koji je istraživao primjenu podržanog učenja s dubinskom kamerom na end-effectoru robota kako bi naučio hvatati objekte. U tom radu, za učenje je korišten algoritam zasnovan na ideji akter-kritičar, a sustav je testiran i u simulaciji i u stvarnom okruženju.

Prema rezultatima tog rada, treniranje u simulaciji bilo je nestabilno, ali su postignuti uspjesi. Prosječna ukupna nagrada i uspjeh hvatanja praćeni su u intervalima od posljednjih 50 iteracija učenja.

Prosječna ukupna nagrada pokazuje pozitivan trend rasta do 50000 iteracija, nakon čega do 60000-te iteracije dolazi do naglog pada za više od 3 puta. (Slika 7)[5] Zatim dolazi do još jedne faze rasta pa pada prosječne ukupne nagrade te zadnje faze rasta do otprilike 90000-te iteracije.



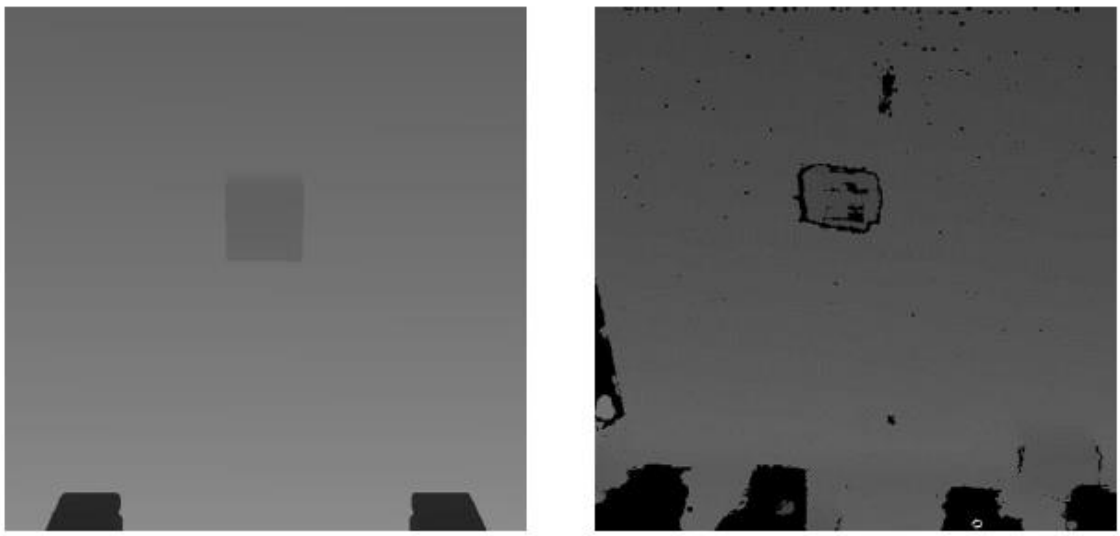

Slika 7 – prosječna nagrada posljednjih 50 iteracija tijekom treniranja modela [5]

Pisac ovog rada primijetio je da faze pada prati ponašanje agenta takvo da približavanjem objektu krene previše zakretati hvatač oko z-osi i tim pokretom pomakne objekt hvatanja te ga ne uspije pravilo smjestiti između prstiju. Ovakvo ponašanje je objašnjeno time što

procjene q-vrijednosti divergiraju zbog loše politike, a politika će biti loša ako su procjene q-vrijednosti netočne. Autori rada [15] predlažu da se težine mreže π rjeđe ažuriraju nego težine mreže Q. Također je predloženo da se ažuriranje politike i ciljnih mreža odgodi dok se greška q-vrijednosti ne smanji što je više moguće. Vrlo sličan trend prati i uspjeh hvatanja.

Testiranje u simulacijskom okruženju pokazalo je lošiji uspjeh nego tijekom treniranja. Izabran je agent nakon 94130 ažuriranja koji u tom trenutku tijekom treniranja daje prosječnu nagradu u iznosu 0.43 u posljednjih 50 epizoda, a prosječni uspjeh hvatanja bio je 28%. Isti taj agent tijekom testiranja daje lošije rezultate – prosječnu nagradu u iznosu 0.31 i uspjeh hvatanja 16%. Ovo pokazuje očiti znak prenaučenosti modela, odnosno model loše generalizira na nove primjere koje još nije vidio. Prenaučenost se očituje u tome da se hvatač uvijek pomiče prema istoj apsolutnoj poziciji neovisno o poziciji predmeta hvatanja. Za ovaj problem autor predlaže 2 potencijalna rješenja: nastavak treniranja ili promjena arhitekture neuronskih mreža tako da se svojstva dubinskih slika bolje percipiraju i više utječu na odabir akcije. (Slika 8)[5]

Pri testiranju u stvarnom okruženju, rezultati su također pokazali da model nije adekvatno reagirao na dubinske slike. Hvatač se pomicao se na sličnu poziciju iznad stola, što je rezultiralo uspješnim hvatanjem samo kada je kocka bila na toj poziciji.



Slika 8 – primjer dubinskih slika korištenih za treniranje i testiranje modela;
dubinska slika u simulaciji (lijevo) i dubinska slika realnog okruženja (desno)[5]

# Zaključak

U ovom radu napravilo se detaljno istraživanje korištenja metode podržanog učenja za problem hvatanja objekta. Fokus je bio na teoretskom objašnjenju različitih metoda podržanog učenja, s naglaskom na Q-učenje i akter-kritičar metode. Rezultati su pokazali da podržano učenje predstavlja dobar odabir za rješavanje ovog problema, prvenstveno zbog svoje sposobnosti da uči iz interakcija sa okolinom bez potrebe za velikim brojem unaprijed labeliranih slika, što je glavni nedostatak metoda nadziranog učenja.

Okolina u kojoj djeluje agent podržanog učenja mora biti oblikovana prema Markovljevom procesu odlučivanja, odnosno odluka o akciji koju će agent izvršiti mora ovisiti samo o trenutnom stanju, a ne o bilo kakvim parametrima iz prethodnog djelovanja agenta.

Posebno dobre rezultate daju modeli oblikovani prema akter-kritičar modelu, koji kombinira prednosti modela temeljenih na politici (akter) i modela temeljenih na vrijednosti stanja (kritičar). Rezultati su pokazali da akter-kritičar metoda još bolje funkcionira kada se koristi s ciljnom mrežom, koja stabilizira proces učenja smanjujući varijacije u procjeni vrijednosti.

# Literatura


[1] Čupić, M. „*Podržano učenje",* (2020.) (poveznica: http://java.zemris.fer.hr/nastava/ui/rl/rl-20200401.pdf)

[2] Goodfellow, I.; Bengio, Y.; Courville, A. "*Deep Learning"*; MIT Press: Cambridge, MA, USA, (2016.)

[3] S. Ekvall and D. Kragic. "*Interactive grasp learning based on human demonstration*". In: Proceedings - IEEE International Conference on Robotics and Automation 2004.4 (2004), pp. 3519–3524. issn: 10504729. doi: 10.1109/robot.2004.1308798.

[4] S. Bhagat and H. Banerjee, "*Deep reinforcement learning for soft flexible robots: Brief review with impending challenges*", *Robotics*, vol. 8, no. 4, pp. 1-36, 2019.

[5] Müller, V. "*Greifen von Objekten mit einem 7-DoF-Roboterarm mithilfe von Deep Reinforcement Learning*"; Diplomski rad. Institut für Angewandte Informatik und Formale Beschreibungsverfahren (AIFB) (2022.)

[6] Yazici B. "*Branch Dueling Deep Q-Networks for Robotics Applications*"; Diplomski rad. Technical University of Munich; (2020.)

[7] *Actor-Critic Algorithm in Reinforcement Learning;* GeeksForGeeks (22.3.2024.) (poveznica: https://www.geeksforgeeks.org/actor-critic-algorithm-in-reinforcement-learning/) pristupljeno: 7.6.2024.

[8] Dhanoop Karunakaran; "*The Actor-Critic Reinforcement Learning algorithm*"; Medium (30.9.2020.) (poveznica: https://medium.com/intro-to-artificial-intelligence/the-actor-critic-reinforcement-learning-algorithm-c8095a655c14) pristupljeno 7.6.2024.

[9] Yazici B. "*Branch Dueling Deep Q-Networks for Robotics Applications*"; Diplomski rad. Technical University of Munich; (2020.) str. 14

[10] Andrew Ng; "Unsupervised Learning, Recommenders, Reinforcement Learning "; Machine Learning Specialization; Coursera

[11] Čupić, M. „*Podržano učenje",* str.6 (2020.) (poveznica: http://java.zemris.fer.hr/nastava/ui/rl/rl-20200401.pdf)

[12] Jiang, Y.; Moseson, S.; Saxena, A. Efficient grasping from RGBD images: Learning using a new rectangle representation. In Proceedings of the 2011 IEEE International Conference on Robotics and Automation, Shanghai, China, 9–13 May 2011; pp. 3304–3311

[13] https://inst.eecs.berkeley.edu/~cs188/sp20/assets/files/SuttonBartoIPRLBook2ndEd.pdf

[14] https://www.automate.org/robotics/blogs/the-emergence-of-smart-collaborative-robot-grippers (pristupljeno 1.6.2024.)

[15] Fujimoto, S.; van Hoof, H; Meger, D; „*Addressing Function Approximation Error in Actor-Critic Methods*" (2018.); (poveznica: https://proceedings.mlr.press/v80/fujimoto18a/fujimoto18a.pdf)

# Sažetak

Ovaj rad istražuje primjenu podržanog učenja u hvatanju objekata robotskom rukom koristeći neuronske mreže unutar akter-kritičar modela. Model se oslanja na Q-učenje, gdje i akter i kritičar posjeduju svoje ciljne mreže za stabilnije učenje. Akter odabire akcije temeljem politike koja se optimizira kroz proces učenja, dok kritičar procjenjuje vrijednost tih akcija koristeći Bellmanove jednadžbe za izračunavanje vrijednosti stanja. Akcije su pomak pozicije i orijentacije robotske ruke, a stanje je trenutna konfiguracija pozicije, orijentacije i udaljenosti između prstiju robotskog hvatača. Cilj je maksimizirati zbroj svih nagrada kroz vrijeme, prilagođavajući politiku aktera kako bi se postigle optimalne akcije.

# Summary

This paper explores the application of reinforcement learning in object grasping with a robotic arm using neural networks within the actor-critic model. The model relies on Q-learning, where both the actor and the critic possess their target networks for more stable learning. The actor selects actions based on a policy that is optimized through the learning process, while the critic evaluates the value of those actions using Bellman equations to calculate the state values. The actions involve adjusting the position and orientation of the robotic arm, and the state is defined by the current configuration of the position, orientation, and distance between the fingers of the robotic gripper. The goal is to maximize the sum of all rewards over time, adjusting the actor's policy to achieve optimal actions.