ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

TUDOMÁNYOS DIÁKKÖRI DOLGOZAT

MEGERŐSÍTÉSES TANULÁSI ALGORITMUSOK ÉRTÉKELÉSE ROBOTIKUS SZIMULÁCIÓS KÖRNYEZETBEN

Szerző: **Farkas Bálint Károly**
mechatronika MSc. szak, II. évf.

Konzulens: **Dr. Széll Károly**
egyetemi docens

Székesfehérvár, 2025

Tartalomjegyzék

Bevezetés

A robotika szerepe az iparban megkérdőjelezhetetlen. Napjainkban egyre több, korábban manuálisan végzett feladat automatizálása történik robotok alkalmazásával. Ezzel egyidejűleg a mesterséges intelligencia is egyre nagyobb teret nyer és óriási fejlődésen megy keresztül, egyre szélesebb körű problémákra alkalmazzák a különböző modelleket. A kutatók pedig egyre nagyobb figyelmet szentelnek a robotika és a mesterséges intelligencia közös metszetére. Ennek egyik oka az iparban és kutatási környezetben található égető igény a robotok nagyfokú autonóm viselkedésére. Az autonóm viselkedés a robotikában azt jelenti, hogy a rendszer képes önálló döntéshozatalra, a hibák detektálására és kijavítására, és képes adaptálódni a külső környezeti változásokhoz, mindezt emberi beavatkozás nélkül. E képességek egyik ígéretes megközelítése a megerősítéses tanulás (Reinforcement Learning, RL), amely lehetővé teszi, hogy a robot interakciók során tanulja meg a kívánt viselkedést [1]. A robotkarok, központi szerepet töltenek be a tárgymanipulációs feladatokban. A robotkarok célja, hogy különböző tárgyakat mozgassanak át egyik pozícióból a másikba, miközben szenzoros visszacsatolás segítségével képesek pontos, megbízható működésre. Az ilyen rendszerek hatékonyságát jelentősen növelik a korszerű szenzorok, ilyenek például a mélységkamerák, illetve taktilis érzékelők. A kutatás-fejlesztés fázisban kiemelt szerepet kapnak a valósághű szimulációs környezetek, amelyek hozzájárulnak a rendszer biztonságos és költséghatékony teszteléséhez. Ezen túlmenően egyes szimulációs szoftverek segítségével képesek vagyunk szintetikus adatokat generálni, valamint jelentősen gyorsítják az RL-algoritmusok betanítását [2]. Jelen dolgozat az NVIDIA által fejlesztett IsaacSim szimulációs környezetre épít, amely fizikailag hiteles, fotórealisztikus szimulációt biztosít komplex robotikai rendszerek számára. Jelen dolgozat célja, hogy az egyetemen is megtalálható hardverelemekkel felszerelt robotikai rendszert modellezzen és vizsgáljon egy szimulációs környezetben. A vizsgálat során több meglévő megerősítéses tanulási algoritmus kerül alkalmazásra annak érdekében, hogy a robot autonóm módon képes legyen egy adott tárgy manipulálására. A kutatás középpontjában az áll, hogy egyes modellek milyen hatékonysággal képesek a feladat elsajátítására és végrehajtására szimulált környezetben. Célom a különböző modelleket kiértékelni különböző szempontok alapján. A dolgozat elsőre részletezi a probléma hátterét és elemzését, majd összefoglalja az irodalmi forrásokat, melyekben megismerkedünk a terület state-of-the-art megoldásaival. Ezt követően bemutatja a rendelkezésre álló hardvereket, majd következik a szimulációs környezet kialakítása és az alkalmazott megerősítéses tanulási modellek bemutatása. Végül a kísérleti eredmények ismertetésére kerül sor, melyeket követően a dolgozat levonja a következtetéseket, és javaslatot tesz a jövőbeli irányokat illetően.

1. Probléma felvetése és elemzése

1.1. Probléma felvetése

A robotikai rendszerek egyik központi eleme a manipuláció, ennek legnagyobb kihívása, hogy a feladatot hatékonyan, megbízhatóan, autonóm módon hajtsa végre a rendszer. A tárgymozgató műveletek alapvető szerepet játszanak az ipari automatizálásban, a logisztikai alkalmazásokban, az összeszereléseknél, valamint a modern kutatásokban, amilyen például a LAPP framework, melyet Wolf Ádám fejlesztett ki, melyben gyógyszeripari laboratóriumban alkalmazható autonóm mobil manipulátorok fontosságát is taglalja [3]. Ezekhez a feladatokhoz azonban elengedhetetlen, hogy a robot képes legyen a környezete érzékelésére, a döntéshozatalra, valamint az adaptív cselekvésre, mindezt emberi beavatkozás nélkül. Jelen kutatás központi problémája annak vizsgálata, hogy a jelenleg egyetemen megtalálható rendszert - amelyben egy UR5e típusú robotkar, egy Robotiq 2f-85 megfogó és egy Intel RealSense D435 mélységkamera található – milyen megerősítéses tanulási algoritmussal célszerű feltanítani, ezeknek az algoritmusoknak összevetése, értékelése. A feladat során a robotnak az aktuális állapottér alapján kell meghatároznia az optimális cselekvéssorozatot úgy, hogy a tanulási folyamat során a rendszer jutalmazás alapján fejlődni tudjon.

A probléma különösen releváns a logisztikában és az iparban, ahol a robotrendszereknek dinamikusan változó környezetben kell stabilan működniük. Egyre több területen, ahol robotokat alkalmaznak, jelentősen megnőtt az igény az autonóm viselkedés iránt. A megerősítéses tanulás olyan potenciális eszközt kínál ezen problémák megoldására, amely a hagyományos, szabályalapú programozással szemben rugalmasabb és adaptívabb működést tesz lehetővé. Ugyanakkor az ilyen RL-alapú rendszerek tanítása jelentős kihívásokkal is jár, ilyen kihívások közé tartozik például a nagy adatigény, az érzékenység a környezet reprezentációjára, valamint a konvergenciaproblémák.

A kutatási feladat több megoldandó problémából áll: robotikai szempontból a robotkar dinamikus vezérlése, a megfogó működtetése jelenti. Szimulációs oldalról pedig a cél egy realisztikus és fizikailag hiteles környezet felépítése, amely lehetővé teszi a tanulási folyamat végrehajtását. Ez magában foglalja az objektumok fizikai paramétereinek beállítását, ütközések szimulációját. Algoritmikus szempontból a kutatás középpontjában a megfelelő állapottér- és akciótér-definíció, a jutalomfüggvény kialakítása, a tanítási hiperparaméterek optimalizálása, valamint a különböző algoritmusok értékelése áll. A sikeres manipuláció kulcsa a tanulási folyamat megfelelő vezérlésében rejlik: a modellnek fokozatosan kell megtanulnia az optimális viselkedést, miközben képes marad általánosítani új, korábban nem látott konfigurációkra is.

1.2. Kutatási cél megfogalmazása

A kutatás célja tehát annak vizsgálata, hogy a különböző RL algoritmusok milyen paraméterekkel hogyan teljesítenek a szimulációban, ezeknek az algoritmusoknak összevetése. A cél nem a valós robotra történő azonnali áttérés, hanem egy realisztikus és fejleszthető szimulációs modell létrehozása, amely alapul szolgál további kutatások sikeréhez. A probléma pontosabb megértéséhez és a lehetséges megközelítések feltérképezéséhez azonban elengedhetetlen a témához kapcsolódó szakirodalom áttekintése. Ennek áttekintése a következő fejezetben kerül bemutatásra.

2. Irodalmi áttekintés

2.1. Megerősítéses tanulás elméleti alapjai

A megerősítéses tanulás a gépi tanulás egy olyan ága, amelyben egy ágens megtanul döntéseket hozni azzal, hogy a környezetével interaktál. Az ágens tapasztalati úton, próbálgatásokkal tanulja meg, hogy hogyan cselekedjen optimálisan. A tanulás során az ágens visszacsatolást kap a környezetétől jutalom (reward) formájában, amelyet az elvégzett akciói után kap. A cél az, hogy a döntéseket az ágens úgy optimalizálja, hogy a lehető legnagyobb kumulatív jutalmat érje el [1].

2.1.1. Alapfogalmak és struktúra

Ahhoz, hogy képesek legyünk beszélni a témáról, fontos tisztáznunk a legfontosabb alapfogalmakat:

Állapottér (S): az a reprezentáció, amellyel az ágens interakciót folytat.

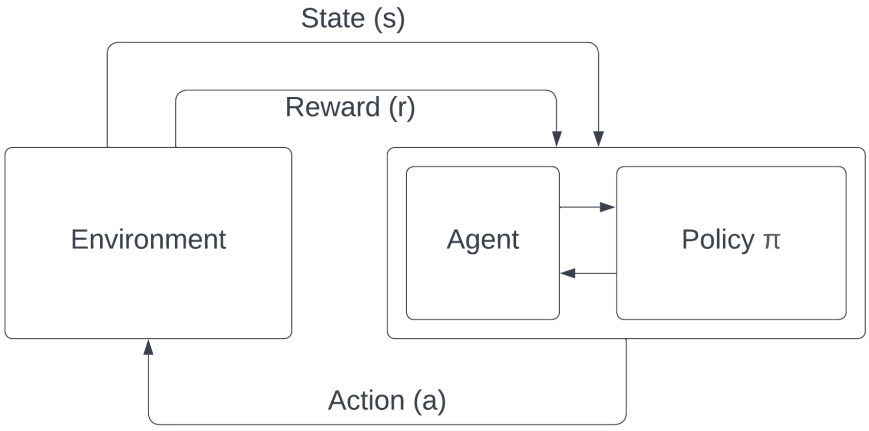
Akciótér (A): az összes lehetséges mozdulat vagy döntés, amelyet az ágens képes tenni.

Stratégia (π): ennek segítségével képes az ágens eldönteni, hogy az éppen aktuális állapotban milyen akciót tegyen.

Jutalomfüggvény (R): az adott akció következményeként kapott numerikus visszacsatolás.

Állapotátmenet (P): annak valószínűsége, hogy az akció hatására az ágens egyik állapotból a másikba kerül.

Diszkontálási tényező (γ): a jövőbeli jutalmak súlyozását szabályozza.



1. ábra. Markov Döntési Folyamat [1]

A tanulás célja egy olyan optimális stratégia (π) megtalálása, amely az ágens számára az összes elérhető állapotból indulva maximális jutalmat eredményez hosszú távon. Ez egyensúlyt igényel a felfedezés (exploration: új akciók kipróbálása azok hatásainak megismerése érdekében) és a kihasználás (exploitation: az ismert akciók közül annak a kiválasztása az azonnali jutalom maximalizálása érdekében) között.

Markov döntési folyamat (Markov Decision Process, MDP): A megerősítéses tanulás matematikai keretét a Markov döntési folyamat (MDP) adja, ennek blokkábráját mutatja az 1 ábra. Az MDP egy 5 elemből álló struktúra: (S, A, P, R, γ).

A megerősítéses tanulás során két alapvető értékfüggvényt különböztetünk meg: az **állapotérték-függvényt** $V_\pi(s)$ 2.1 és az **állapot-akció értékfüggvényt** $Q_\pi(s,a)$ 2.2, melyek definíciói az alábbiak:

$$V_\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \,\middle|\, s_0 = s \right] \tag{2.1}$$

$$Q_\pi(s,a) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \,\middle|\, s_0 = s, a_0 = a \right] \tag{2.2}$$

ahol:

- $s \in \mathcal{S}$: a környezet egy állapota (state),
- $a \in \mathcal{A}$: az ágens által választható akció (action),
- π: a stratégia (policy), amely meghatározza az ágens viselkedését,
- $R(s_t, a_t)$: az adott időpillanatban kapott jutalom,
- $\gamma \in [0,1]$: diszkontálási tényező, amely csökkenti a jövőbeli jutalmak súlyát,
- $\mathbb{E}_\pi[\cdot]$: a várható érték, feltételezve, hogy az ágens a π stratégiát követi.

A $V_\pi(s)$ tehát azt mutatja meg, hogy egy adott állapotból indulva milyen várható kumulatív jutalom érhető el a stratégia követésével. Ezzel szemben a $Q_\pi(s,a)$ azt fejezi ki, hogy mekkora a várható kumulatív jutalom, ha az ágens az adott állapotban egy konkrét akciót hajt végre, majd onnantól a π stratégiát követi. Ezek a függvények visszavezethetők az ún. Bellman-egyenletekre, amelyek a várható értékeket rekurzívan fejezik ki.

2.1.2. Klasszikus RL algoritmusok

A megerősítéses tanulás első generációját, a klasszikus algoritmusokat diszkrét állapot- és akcióterekre alkalmazzuk, mert ezek az algoritmusok csak ezeket az eseteket tudják kezelni. Néhány ilyen algoritmus:

Temporal Difference (TD) Learning: a tanulás a jutalom és a becsült érték különbségén alapul.

Q-Learning: optimális Q-függvényt tanítja, függetlenül attól, hogy milyen stratégiát követ az ágens, garantált konvergenciát biztosít vége, diszkrét problémák esetén.

Ezek az algoritmusok előnyei közé tartozik, hogy az elméletük egyszerű, analitikusan kezelhetőek, de nem használhatóak hatékonyan folytonos állapot- vagy akciótereken.

2.1.3. Modern megközelítések

A modern megközelítések a klasszikus megközelítéseket kiterjesztik mély neurális hálózatokkal, amelyek segítségével a Q-függvényeket vagy a stratégiát folytonos, nagy dimenziójú térben lehet approximálni.

Deep Q-Network (DQN): Q-Learning kiterjesztése mély neurális hálózattal [4], [5].

Proximal Policy Optimization (PPO): legszélesebb körben alkalmazott algoritmus a robotikában. Célja, hogy a tanulás során a stratégia ne változzon túl gyorsan, ezzel létrehozva a stabilitását az algoritmusnak [6].

Soft Actor-Critic (SAC): Entrópia maximalizálást alkalmaz, elősegíti a felfedezést (exploration), ezáltal robusztusabb és gyorsabban konvergál [7].

Deep Deterministic Policy Gradient (DDPG) és Twin-Delayed DDPG (TD3): képes nagy dimenziós akcióterek kezelésére, azonban érzékeny a hiperparaméterekre. A TD3 a DDPG továbbfejlesztett változata, kettős értékbecslést alkalmaz, így stabilabb és megbízhatóbb tanulást biztosít [8].

Trust Region Policy Optimization (TRPO): Szigorú lépéskorlátokat alkalmaz a stratégia stabilitásának megőrzése érdekében [9].

Dolgozatomban ezek közül a PPO, SAC, DDPG, TD3 és a TRPO algoritmusok tesztelését tűztem ki célul.

A megerősítéses tanulás egy rugalmas és hatékony megközelítése az autonóm viselkedés kialakításának, különösen olyan környezetben, ahol nem áll rendelkezésünkre előzetes adat vagy pontos modell. A klasszikus algoritmusok remek megoldások diszkrét problémákra, míg a modernebb megközelítések tökéletesek a folytonos terekben alkalmazható problémákra, amelyek közé például a robotikai alkalmazások is tartoznak. A dolgozat során választott algoritmusok – PPO, SAC, DDPG, TD3, TRPO – a modern algoritmusok közé tartoznak és különösen alkalmasak a szimulált robotmanipulációs feladatok megoldására.

2.2. RL alapú manipulációs megközelítések

A megerősítéses tanulás az utóbbi években egyre népszerűbb lett a manipulációs feladatok automatizálásában is. Ez a módszer különösen hatékony olyan feladatokra, amelyeknél a robotnak több lépésen keresztül kell döntéseket hoznia, és a helyes döntések következményei csak később jelentkeznek. A tárgymanipuláció tipikusan ilyen probléma. Manipulációs feladatokra jellemzően model-free (modell nélküli) algoritmusokat alkalmaznak, mivel a robotkar és a környezet pontos fizikai modellje nem ismert. Ilyen algoritmusok közé tartoznak a kutatásom során alkalmazott algoritmusok is.

Számos kutatás foglalkozott RL alapú tárgymegfogással és mozgatással, a teljesség igénye nélkül a leghíresebbeket felsoroltam:

NVIDIA RVT és RVT2: Ezeknek a projekteknek a célja, hogy egy objektumot stabilan képesek legyenek megfogni, ezen felül az RVT2-es a precízebb feladatok ellátására lett kifejlesztve [10], [11].

OpenAI Rubik-kocka kirakó robotja: OpenAI cég egy Shadow Robot kér segítségével megtanította egy robotnak a Rubik-kocka kirakását kizárólag megerősítéses tanulással [12].

QT-Opt (Google DeepMind és Google Robotics): Egy robusztus manipulációs rendszer, amely mély megerősítéses tanulási algoritmusok segítségével tanul meg tárgyakat megfogni változatos környezetben [13].

Ezek a munkák bizonyítják, hogy az RL alapú megközelítések képesek és hatékonyak ezeknek a feladatoknak az ellátására. A kutatásomban én is ezt a módszert alkalmazom egy UR5e robotkar

kockamanipulációs feladatának megoldására szimulációs környezetben.

2.3. Szimuláció alapú tanítás a robotikában

A megerősítéses tanulás nagyon sok interakciót kíván a környezettel, mire egy használható modell kialakulna, ez valós roboton nagyon költséges lenne, és nagyon sok időbe telne, ezen felül a be nem tanított modell akár biztonsági kockázatot is jelenthet, ha még nem tudja megfelelően korrigálni a hibáit. Ezeken felül a tanítás során szeretnénk egy olyan környezetet kialakítani, amely ismételhető, és determinisztikus, hogy a tanítás stabilan menjen, egyben mérhető is legyen a kutatásom célkitűzését segítve. A fizikai tanítással szemben a szimulációnak számos nagy előnye van, melyeket lentebb fejtjük ki.

Gyorsított tanítás: gyakran a szimulációs környezetben lehetőségünk van gyorsítani a tanítást, gyorsabban lefuttatni az adott etapot, vagy képesek vagyunk egyszerre több robotot letenni, amelyek egymás klónjai, és ezeken egyidejűleg tanítjuk az ágenset, majd ezeket a feltanított stratégiákat tudjuk különböző módokon a valós robotra rátölteni. Ezeknek a módoknak a vizsgálatával foglalkozik Horváth et al. [14].

Szintetikus adatgenerálás: szintetikus adatgenerálás alatt azt értjük, amikor szükségünk van adatra, viszont azt a valós világból nem tudjuk begyűjteni, vagy nem éri meg, mert sok időbe telne, és ezért a szimulált világban levő kameraképet, vagy egyéb szenzor adatait gyűjtjük össze, és generálunk annotált adatot. Ez a folyamat akkor célszerű, ha a környezetünk képes realisztikus adatokat generálni, az általunk kiválasztott környezet, az Isaac Sim képes ilyen realisztikus megjelenítésre, egyben el van látva olyan API-kal, melyekkel képesek vagyunk a domén randomizálására, ezáltal sokkal változatosabb tanító adatokat kapva [15].

Könnyű replikáció és hibakeresés: a valós környezethez képest szimulációban könnyebben tudjuk replikálni az adott állapotokat, és a hibákat, így tesztelve az algoritmusunkat, ami kutatásom szempontjából előnyt jelent.

2.3.1. RL benchmarkok manipulációs feladatokhoz

A különböző algoritmusok fejlődésével egyre égetőbb lett a szükség ezeknek az algoritmusoknak az összevetésére egy szabványosított benchmark környezetben. Az ilyen környezetek lehetővé teszik, hogy szabályozott módon mindegyik algoritmust ugyan olyan környezetben tudjunk tesztelni, és elemezni, ezáltal segítve a kutatókat a legmegfelelőbb algoritmusok alkalmazásában. Manipulációs feladatokhoz leggyakrabban alkalmazott benchmarkok az alábbiak:

OpenAI Gym környezetek: Ilyen például a FetchReach vagy a FetchPickAndPlace, ezek egyszerűbb, jól kontrollálhatók, különböző feladatokra vannak felépítve [16].

RLBench: Az Imperial College London által lett kifejlesztve, több manipulációs feladatot kínál [17].

Isaac Gym Manipulation Benchmarks: NVIDIA által készített környezetek, nagy skálázhatóságot kínálnak, ezáltal főleg a nagyméretű tanulási feladatok tesztelésére tökéletes [18].

A benchmarkok fejlődése nagyban hozzájárult a robotika és megerősítéses tanulási algoritmusokra irányuló kutatáshoz. Saját kutatásomban saját szimulációs környezetet hozok létre, amely egyetemi hardverkörnyezettel szoros párhuzamban van, viszont ugyan azokat az elveket fogom követni, mint a már szinte sztenderdnek nevezhető benchmarkok.

2.4. Saját megközelítés helye az irodalomban

A megerősítéses tanulás alkalmazása robotmanipulációs feladatokra dinamikusan fejlődő kutatási terület, amely az elmúlt években számos új megközelítést és algoritmust eredményezett. A szakirodalomban több munka foglalkozik az autonóm tárgymozgatási feladatok RL-alapú megoldásával, azonban az egyes algoritmusok hatékonyságának és általánosíthatóságának szisztematikus összehasonlítása kevésbé kidolgozott. Az OpenAI kutatói [19] például a domain randomization technikát alkalmazták annak érdekében, hogy a szimulációban tanított ágensek sikeresen működjenek valós környezetben is, míg Levine és munkatársai [20] vizuális visszacsatolású grasping feladatokon alkalmaztak megerősítéses tanulást, főként mély neurális háló stratégiák tanításával. Noha ezek a kutatások jelentős előrelépést jelentenek a területen, jellemzően egy-egy kiválasztott algoritmusra fókuszálnak, és a különböző megerősítéses tanulási módszerek közvetlen összevetése ritkán szerepel elsődleges célként. A jelen dolgozat ebbe a kutatási környezetbe illeszkedik azzal, hogy nem egyetlen algoritmus alkalmazására koncentrál, hanem több, különböző típusú megerősítéses tanulási modell teljesítményét vizsgálja meg azonos feltételek mellett. A modellek értékelése az alábbi szempontok mentén történik:

- átlagos epizódjutalom (mean episode reward),
- tanulási sebesség (time to convergence),
- cselekvő hálózat átlagos vesztesége (mean policy loss),
- kritikus hálózat átlagos vesztesége (mean critic loss),
- stratégia entrópiája (policy entropy).

A választott megközelítés újdonsága abban rejlik, a kutatás relevanciája mellett, hogy a szakirodalomban a különböző megerősítéses tanulási algoritmusok közvetlen összehasonlítása sokszor eltérő feladatokon, eltérő implementációkkal és paraméterezéssel történik, így nehéz objektív következtetéseket levonni. Jelen dolgozat célja ezért egységes környezetben, azonos hardver- és szimulációs konfigurációk mellett értékelni az algoritmusokat, ezzel hozzájárulva a megerősítéses tanulás alkalmazási lehetőségeinek jobb megértéséhez a robotmanipuláció területén.

Összefoglalva a kutatásom egy valós, de egyszerűbb kérdést tesz fel: különböző megerősítéses tanulási algoritmusok hogyan viselkednek ugyan olyan szimulációs konfigurációkkal. A megközelítésem hozzájárul ahhoz, hogy a realisztikus szimulációk egyre elterjedtebbek legyenek az megerősítéses tanulás alapú robotikai kutatásokba, későbbiekben pedig a legmegfelelőbb modell átültethető lesz a valós környezetbe is.

3. Kutatási környezet és eszközpark bemutatása

3.1. Hardverpark bemutatása

A szimulációs környezetet a rendelkezésre álló hardverpark alapján állítottuk össze, ebbe az alábbi eszközök tartoznak bele:

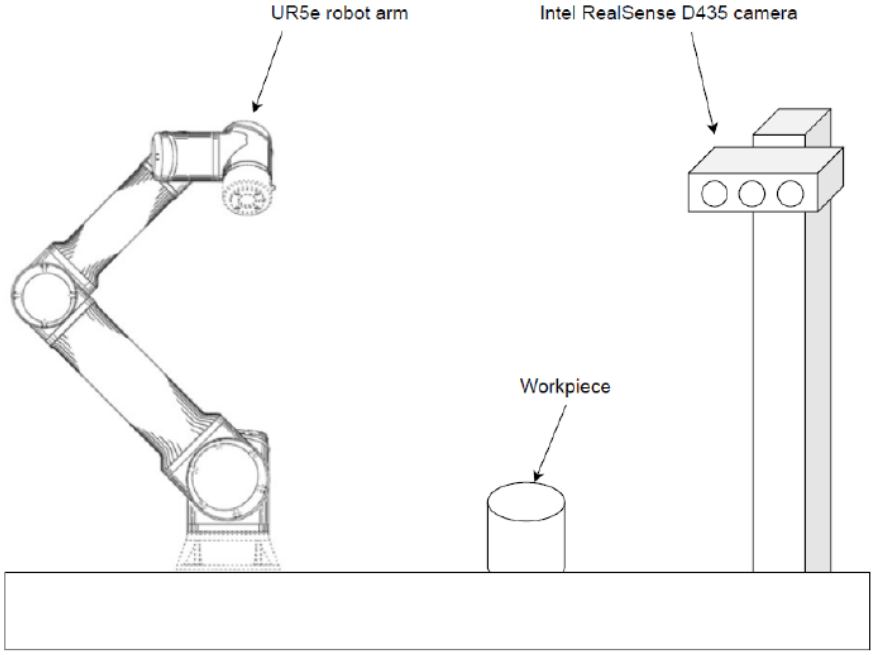
PC NVIDIA GPU-val felszerelve: a megerősítéses tanulás algoritmusok futtatásához elengedhetetlen a párhuzamos, GPU-alapú számítási kapacitás.

UR5e robotkar: a robotkar egyik fő előnye a nyitott szoftverkörnyezet, hiszen egy linux alapú operációs rendszerrel van felszerelve a robotvezérlő, amelyhez kívülről könnyen hozzá lehet férni, például a ROS2 (Robot Operating System) rendszerrel, és nagyon könnyű a kommunikáció kiépítése a robotvezérlővel, ezért kutatásokban előszeretettel alkalmazzák.

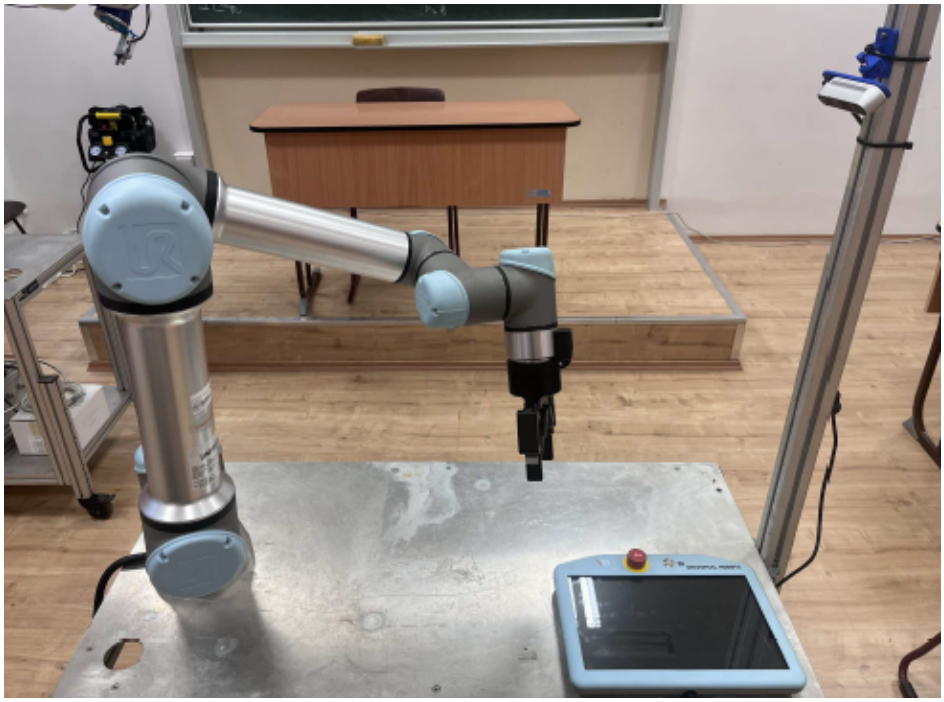
Robotiq 2f-85 megfogó: ez a robotkar végén, az úgynevezett flange-en helyezkedk el, ezzel képes a robotkar manipulálni a különböző tárgyakat. Ennek a megfogónak az egyik nagy előnye, hogy képesek vagyunk megadni a kívánt állapotát, hogy mennyire legyenek összezárva az ujjai, ezen felül a sebességét és az erejét is tudjuk állítani, így képesek vagyunk törékeny tárgyak manipulálására is. Ezen felül ez az eszköz is kompatibilis a ROS-szal, így a valós hardveren végzett kutatások gördülékenyebben tudnak menni.

Intel RealSense D435 kamera: ez egy RGB-D kamera, ami azt jelenti, hogy egyrészt az RGB képet adja vissza, másrészt pedig a távolságokat, ezáltal képesek vagyunk pozíciókat is meghatározni a robotkar számára. Ez az eszköz is képes a ROS2-vel való kommunikációra.

A fent felsorolt eszközök kiválóak kutatások elvégzésére, ezek az eszközök széles körben elterjedtek a robotikai kutatásokban, így kézenfekvő választást jelentettek a jelen kutatáshoz. A hardverelemek elhelyezésének terve 2 ábra, és a valós hardverkörnyezet pedig a 3 ábra szemlélteti.



2. ábra. Hardverelemek elhelyezésének terve



3. ábra. Valós hardverkörnyezet

A szimuláció felépítésénél figyelembe vettem ezeket a hardverelemeket, és direkt úgy alakítottam ki a szimulációs környezetet, hogy ezáltal tudjam validálni a valós környezet megfelelőségét a további kutatásokhoz.

3.2. Szimulációs környezet bemutatása

A hardverpark áttekintését követően a következő lépés a szimulációs szoftver kiválasztása és a virtuális környezet felépítése volt, amely részletesen a következő alfejezetekben kerül bemutatásra. Első lépésként meg akartam határozni, hogy milyen eszközök állnak rendelkezésemre, és hogy melyiket lenne célszerű alkalmaznom figyelembe véve a jövőbeli terveinket is.

3.2.1. Rendelkezésre álló szimulációks szoftverek

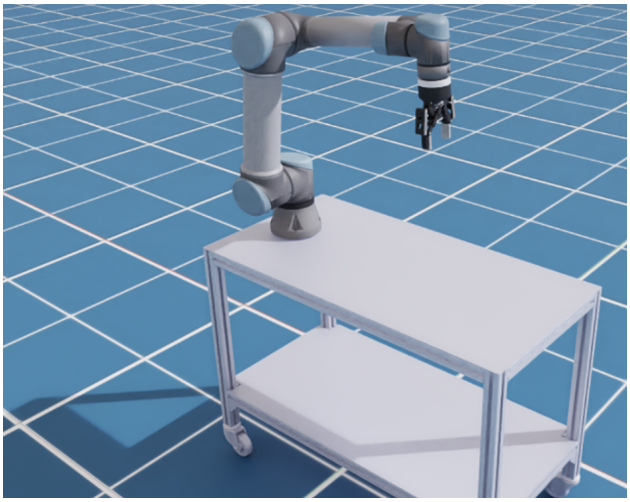
Kezdetnek összeszedtem a fontosabb szimulációs szoftvereket, amelyek szóba jöhettek, amikor a projektemet elkezdtem: Gazebo, MuJoCo, Isaac Sim + Isaac Lab [18]. Ezek közül az Isaac Sim és Isaac Lab kombinációt választottam, mivel ez a környezet képes realisztikusan mind a fizikai, mind a vizuális szimulációra, ezen felül több megerősítéses tanulási algoritmus használatára is lehetőség van, és a környezet elkészítése után ezeket már gördülékenyen lehet átváltani. Megfontolandó további környezet a MuJoCo, mely szintén alkalmas a realisztikus fizikai szimulációra, és nagyon hatékonyan lehet különböző algoritmusokat implementálni, tanítani és lejátszani.

Az Isaac Sim az NVIDIA által fejlesztett, fizikailag hiteles és fotorealisztikus robotikai szimulációs környezet. A rendszer PhysX fizikai motorra és a USD (Universal Scene Description) formátumra épül. A szimuláció képes valós időben kezelni komplex jeleneteket, szenzorokat, va-

lamint támogatja a ROS2 kompatibilis vezérlést is. Az Isaac Sim egy fejlett Python API-val rendelkezik, amely nagyban megkönnyíti a szimulált környezet felépítését és automatizált vezérlését. A szimulációs szoftver mellett az Isaac Lab keretrendszert alkalmaztam, amely a RL környezetek fejlesztését és integrálását biztosítja. Az Isaac Lab egy nyílt forráskódú könyvtár, amely előre definiált környezeteket, RL-agent interfészeket és tanulási sémákat kínál. Segítségével egyszerűen hozhatók létre új környezetek, konfigurációk. A két rendszer együttes alkalmazása lehetővé teszi a valós robotikai feladatokhoz hű szimulációs modell kialakítását, a fizikai és vizuális visszacsatolások együttes szimulálását, és a tanulási ciklus teljes körű vezérlését Python nyelven keresztül A választás tehát a valós robotikai rendszerekkel való jövőbeli kompatibilitás szempontjából volt indokolt.

3.2.2. Szimulációs környezet létrehozása

Miután kiválasztottam a szimulációs szoftvert, létre kellett hoznom a környezet mását. Ehhez a legelső lépés a modellek felkeresése, illetve elkészítése volt. Az Isaac Lab-be behívható modellek kiterjesztése USD (Universal Scene Description) fileformátum, célom pedig elsőre az volt, hogy mind a robotkarról, mind az asztalról egy-egy külön USD file készüljön. Ennek megtételére több lehetőség állt rendelkezésemre, az egyik, hogy a .stp kiterjesztésű file-t beimportáljam a szimulációba, ezt a módszert alkalmaztam a robotkart tartó asztalnál, ennek a modelljét pedig én készítettem el Autodesk Inventor szoftverben, majd kiexportáltam .stp file formátumba, majd ezt a modellt importáltam be Isaac Sim-be, itt meg tudtam adni a fizikai tulajdonságait az asztalnak, majd elmentettem USD fileformátumban. A következő modell maga a robot volt, felszerelve a megfogóval. Mivel kutatásokban eléggé gyakori az általam alkalmazott konfiguráció (UR5e robotkar és Robotiq 2f-85 megfogó) ezért nem kellett elkészíteni az URDF (Unified Robot Description Format) file-t, hanem az interneten megtalálható URDF-ek közül választhattam. Az URDF egy XML formátum, amely a különböző robotok ábrázolására tökéletes. Ebben megtalálhatók a kinézetért felelős file-ok, mesh-ek formájában, ezen felül a különböző tengelyek is meg vannak adva, amelyek lehetnek rotációsak és lineárisak is. Az általam alkalmazott URDF file-ban már be volt importálva a megfogó is, így már csak be kellett importálnom Isaac Sim környezetbe, amelyet az URDF importer segítségével tettem meg. Ezután finomhangoltam a robot tulajdonságait, majd ezt is kimentettem USD file formátumba. A vizualizáció kedvéért készítettem egy harmadik USD file-t, hogy megmutathassam, hogy hogyan néz ki ez a két környezet összefésülve. Az eredményt a 4 ábra mutatja.



4. ábra. Szimulációs környezet

Következő lépésben szükséges volt az Isaac Lab-ben feldolgozható formátumra alakítani a USD modelleket, ami azt jelentette, hogy Python API-ban kellett egy környezethez hozzáadnunk ezeket az építőelemeket. Legelső lépésben a robot konfigurációját adtam meg. Ez hozzájárul ahhoz, hogy a későbbiekben a megerősítéses tanulási algoritmusnak megfelelő akcióteret tudjunk képezni.

4. Alkalmazott megerősítéses tanulási algoritmusok

Kutatásomban fő célkitűzésem a különböző megerősítéses tanulási algoritmusok implementálása és tesztelése, mérése. Ebben a fejezetben bemutatom a választott algoritmusokat, megmutatom az előnyeiket, hátrányaikat, működésüket. Majd bemutatom az probléma reprezentációját, úgymint állapottér, akciótér, jutalomfüggvények.

4.1. Választott algoritmusok bemutatása

Kutatásomban kitűztem, hogy a legelterjedtebb algoritmusokat szeretném megvizsgálni, hogy milyen eredményeket hoznak az általam kialakított szimulációs környezetben. Ez az alfejezet ezeket az algoritmusokat fogja sorban bemutatni. Az algoritmusokhoz az skrl könyvtárat használtam. A könyvtár kiválasztását indokolja, hogy Isaac Lab környezethez már ki van alakítva a keretrendszer a könyvtár használatára, és csak a környezet reprezentációját és a különböző algoritmusok konfigurációját kellett létrehoznom, amelyben az algoritmusok hiperparaméterei foglalnak helyet.

4.1.1. Trust Region Policy Optimization

A Trust Region Policy Optimization (TRPO) egy on-policy megerősítéses tanulási algoritmus, amelyet azért hoztak létre, hogy a stratégia stabilan és megbízhatóan változzon a tanulás során. A TRPO célja, hogy minden tanulási lépésben maximalizálja az ágens által megszerezhető várható jutalmat, miközben a stratégia változásának mértékét egy előre meghatározott korláton belül tartja (Trust Region) [9]. Az pszeudokódját az 1. algoritmus mutatja be.

Az algoritmus előnyei közé tartozik a magas stabilitás, mert a stratégia csak kis mértékben változik, ugyanakkor hátránya, hogy számításigényes lehet, ezért nagyobb környezetekben vagy hosszú epizódok során nem feltétlenül alkalmas.

1. Algorithm: Trust Region Policy Optimization [9]

Input: initial policy parameters θ_0, initial value function parameters ϕ_0
Hyperparameters: KL-divergence limit δ, backtracking coefficient α, max. number of backtracking steps K
for $k = 0, 1, 2, \ldots$ **do**
    Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
    Compute rewards-to-go: $\hat{R}_t$
    Compute advantage estimates $\hat{A}_t$ (e.g., GAE) using value function V_{ϕ_k}
    Estimate policy gradient:

$$\hat{g}_k = \frac{1}{|\mathcal{D}_k|} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\big|_{\theta_k} \hat{A}_t$$

    Use the confugate gradient algorithm to compute

$$\hat{x}_k \approx \hat{H}_k^{-1} \hat{g}_k$$

    where $\hat{H}_k$ is the Hessian of sample average KL-divergence
    Update the policy by backtracking line search with

$$\theta_{k+1} = \theta_k + \alpha^j \sqrt{\frac{2\delta}{\hat{x}_k^T \hat{H}_k \hat{x}_k}} \hat{x}_k$$

    where $j \in \{0,1, \ldots, K\}$ is the smallest index satisfying KL constraint
    Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_\phi \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2$$

    typically via some gradient descent algorithm.
end for

4.1.2. Proximal Policy Optimization

A Proximal Policy Optimization (PPO) az OpenAI által kifejlesztett algoritmus, a TRPO továbbfejlesztéseként hozták létre, hogy elődjéhez képest a stabilitást megtartsák, viszont a számítási igényét jelentősen csökkenthessék. A PPO célja, hogy korlátozza a stratégia frissítésének mértékét, de bonyolult optimalizációs feladatok nélkül tegye ezt meg. A PPO egy módosított veszteségfüggvényt alkalmaz, amely a stratégia frissítését egy úgynevezett clipped objective segítségével szabályozza. Ez a veszteségfüggvény a régi és az új stratégia valószínűségaránya alapján bünteti azokat a frissítéseket, amelyeknél egy előre meghatározott küszöbértéknél nagyobb a változás. Ezért a stratégia frissítései korlátozott mértékűek maradnak, ami növeli a stabilitást, miközben elkerüli a TRPO által használt bonyolult számításokat, ezáltal kevesebb számítási igényt támasztva [6]. A

PPO algoritmus működési elve a 2. algoritmusban látható.

2. Algorithm: PPO-Clip [6]

Input: initial policy parameters θ_0, initial value function parameters ϕ_0
for $k = 0, 1, 2, \ldots$ **do**
    Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
    Compute rewards-to-go $\hat{R}_t$
    Compute advantage estimates $\hat{A}_t$ (e.g. GAE) using value function V_{ϕ_k}
    Update policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min \left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} \hat{A}^{\pi_{\theta_k}}(s_t, a_t), g(\epsilon, \hat{A}^{\pi_{\theta_k}}(s_t, a_t)) \right)$$

    typically via stochastic gradient ascent (e.g. Adam optimizer)
    Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2$$

    typically via gradient descent algorithm.
end for

Előnyei közé tartozik a magas stabilitás, az egyszerű implementáció, ezek miatt az egyik legszélesebb körben alkalmazott algoritmusok közé tartozik a modern megerősítéses tanulási algoritmusok között.

4.1.3. Soft Actor-Critic

A Soft Actor-Critic (SAC) egy off-policy megerősítéses tanulási algoritmus, amelyet kifejezetten folytonos akcióterekhez fejlesztettek ki. A SAC célja, hogy a tanulás során az ágens ne csak a várható jutalmat maximalizálja, hanem mellette fenntartsa a stratégia entrópiáját is, elősegítve a felfedezést (exploration) [7]. A SAC előnyei közé tartozik, hogy magas stabilitású, jó általánosítóképességgel rendelkezik, gyorsan képes tanulni. Működési elvét a 3. algoritmus mutatja be.

3. Algorithm: Soft Actor-Critic [7]

Input: initial policy parameters θ, Q-function parameters ϕ_1, ϕ_2, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\phi_{\text{targ},1} \leftarrow \phi_1$, $\phi_{\text{targ},2} \leftarrow \phi_2$
repeat
    Observe state s and select action $a \sim \pi_\theta(\cdot|s)$
    Execute a in the environment
    Observe next state s', reward r, and done signal d to indicate whether s' is terminal
    Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
    if s' is terminal **then** reset environment state
    end if
    if it's time to update **then**
        for j in range **do**
            Sample a batch $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
            Compute targets for Q-functions:

$$y(r, s', d) = r + \gamma(1 - d)\left(\min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}'|s')\right), \quad \tilde{a}' \sim \pi_\theta(\cdot|s')$$

            Update Q-functions by one step of gradient descent using

$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_{\phi_i}(s, a) - y(r, s', d)\right)^2, \quad \text{for } i = 1, 2$$

            Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} \left(\min_{i=1,2} Q_{\phi_i}(s, \tilde{a}_\theta(s)) - \alpha \log \pi_\theta(\tilde{a}_\theta(s)|s)\right)$$

            Update target networks with

$$\phi_{\text{targ},i} \leftarrow \rho \phi_{\text{targ},i} + (1 - \rho)\phi_i, \quad \text{for } i = 1, 2$$

        end for
    end if
until convergence

4.1.4. Twin Delayed Deep Deterministic Policy Gradient

A Twin Delayed Deep Deterministic Policy Gradient (TD3) egy megerősítéses tanulási algoritmus, amelyet folyamatos akciótereknél stabil és megbízható tanításra fejlesztettek ki. Elődje a DDPG (Deep Deterministic Policy Gradient), ennek hátránya volt, hogy nagyon érzékeny volt a tanulás során fellépő hibákra, túl optimistán értékelte az akciókat, és nehéz volt stabilan tartani. A TD3 ezeken a hibákon javít a következő megoldásokkal. Két kritikus hálót használ, mindig a kisebb Q értéket veszi figyelembe. Ennek hatása, hogy csökken a túlbecslés. Késleltetetten frissíti a straté-

giát. Ennek hatása, hogy stabilabb lett a tanulás. Zajt ad az akciókhoz tanulás során. Ennek hatása, hogy a hálózat ne csak a pontos értékekre, hanem a környező értékekre is megfelelően működjön [8].

A TD3 algoritmus pszeudokódja a 4. algoritmusban látható.

A TD3 algoritmus hatékony mintafelhasználást tesz lehetővé, miközben magas stabilitást és pontos akcióértékelést biztosít. Ez fontos lehet olyan feladatoknál, ahol például robotkart szeretnénk irányítani, ahol az apróbb hibák is jelentős problémákat okozhatnak.

---

4. Algorithm: Twin Delayed DDPG [8]

Input: initial policy parameters θ, Q-function parameters ϕ_1, ϕ_2, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ},1} \leftarrow \phi_1$, $\phi_{\text{targ},2} \leftarrow \phi_2$
repeat
    Observe state s and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{\text{Low}}, a_{\text{High}})$, where $\epsilon \sim \mathcal{N}$
    Execute a in the environment
    Observe next state s', reward r, and done signal d to indicate whether s' is terminal
    Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
    if s' is terminal **then** reset environment state
    end if
    if it's time to update **then**
        for j in range(however many updates) **do**
            Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
            Compute target actions:
                $a'(s') = \text{clip}(\mu_{\theta_{\text{targ}}}(s') + \text{clip}(\epsilon, -c, c), a_{\text{Low}}, a_{\text{High}}), \quad \epsilon \sim \mathcal{N}(0, \sigma)$
            Compute targets:
                $y(r, s', d) = r + \gamma(1 - d) \min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', a'(s'))$
            Update Q-functions by one step of gradient descent using:
                $\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_{\phi_i}(s, a) - y(r, s', d)\right)^2 \quad$ for $i = 1,2$
            if j mod `policy_delay` $= 0$ **then**
                Update policy by one step of gradient ascent using:
                    $\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_{\phi_1}(s, \mu_\theta(s))$
                Update target networks with:
                    $\phi_{\text{targ},i} \leftarrow \rho\phi_{\text{targ},i} + (1 - \rho)\phi_i \quad$ for $i = 1,2$
                    $\theta_{\text{targ}} \leftarrow \rho\theta_{\text{targ}} + (1 - \rho)\theta$
            end if
        end for
    end if
until convergence

4.2. RL probléma reprezentációja

A megerősítéses tanulás során az ágensnek nagyon fontos, hogy valamilyen módon kapcsolódjon a környezetéhez. Az ágens közvetlenül nem látja az egész környezetet, csupán egy absztrakcióját. Mivel az ágens döntéseinek alapja a környezetében megtett megfigyelések, ezért kulcsfontosságú a környezet reprezentációja. A reprezentáció döntően befolyásolja, hogy az ágens meg tudja-e tanulni az adott feladatot, vagy hogy milyen gyorsan képes megtanulni a feladatot. Egy megerősítéses tanulási feladatban szükséges így létrehozni a reprezentációt, amelyet külön részegységekre bontunk:

Állapottér (Observation space): Ez adja meg, hogy mi a státusz, ezek azoknak az adatoknak az összessége, melyek alapján az ágens a döntést hozza.

Akciótér (Action space): Ez az ágens által kiadható parancsok összessége.

Jutalomfüggvény (Reward function): Ezekben a függvényekben van leírva, hogy mi alapján pontozzuk az ágens cselekvéseit. Ezek alapján fogja tudni, hogy éppen megfelelően cselekedett-e, esetleg elvétette a feladatot.

Felfüggesztés (Termination): Ebben kell leszögeznünk, hogy mikor tekinthetünk egy epizódot befejezettnek.

Környezet inicializációja (Environment initialization): Ebben adjuk meg, hogy az epizódok hogyan induljanak. Itt lehetőségünk van a domén randomizációjára, ezáltal stabilabb rendszert kiépítve, viszont így több időre van szüksége az ágensnek a tanulásra.

Curriculum learning: Ha túl nehéz a feladat elsőre, akkor lehetőségünk van kis lépésekre bontani, majd ezeket folyamatosan nehezítve tanítani az ágenst.

Speciális tanítási technikák: Ilyenek például a parancsok, amelyekkel még jobban tudjuk finomítani a szimulációt.

Kutatásomnál nagyon fontos, hogy ezek mind pontosan és jól legyenek megadva, különben az ágens nem fogja megtanulni az adott feladatot, vagy éppen pontatlanul fogja végrehajtani. A szimulációban nagy és folytonos állapot- és akciótérrel találkozhatunk, ehhez hozzájárul, hogy a megfogási feladat már elég komplex, ezért számomra érdemes használatba vennem a tanítási folyamat fokozatos nehezítését, amelyet curriculum learning-nek hívnak. Kutatásomban figyelmet fordítottam arra is, hogy randomizáljam a környezetet, ezáltal az ágenset rávettem a robusztusabb stratégia létrehozására. A következő alfejezetek részletesebben bemutatják az állapot- és akcióteret, a jutalomképzést, az alkalmazott curriculum learning stratégiát, az epizódok lezárásának feltételeit, a környezet inicializációját, valamint a speciális parancsok használatát.

4.2.1. Állapottér

Az állapottér (observation space) adja vissza az ágens megfigyeléseit a környezetéről, ezek alapján hozza meg a megfelelő döntést. Jelen kutatásban az állapottér célja, hogy a robotkar, a megfogó és a manipulálandó tárgy pillanatnyi konfigurációját pontosan és tömören reprezentálja. Az állapottér a következő elemekből épül fel: csuklópozíciók, csuklósebességek, objektum pozíciója, célpozíció: parancs által meghatározott pozíció, megfogás után ide kell mozgatnia a kockát a robotkarnak, utolsó akció. Az állapotteret a 4.1 táblázat mutatja be.

A fenti információk egyetlen vektorrá kerülnek összeillesztésre, majd normalizáltam ezeket a jellemzőket, és ezt a vektort kapja meg az ágens bemenetként a stratégia modelljéhez. Ez a repre-

4.1. táblázat. Állapottér definíciója

Megfigyelés típusa	Leírás	Dimenzió
Ízületi pozíciók	UR5e robot hat ízületének aktuális szögállása	6
Ízületi sebességek	UR5e robot hat ízületének szögsebessége	6
Objektum pozíciója	A kocka pozíciója a robot bázisához viszonyítva	3
Célpozíció (command)	Az `object_pose` által generált célkoordináta	3
Utolsó akció	Az előző lépésben végrehajtott akció értékei	7
Összesen		**25**

zentáció biztosítja, hogy az ágens rendelkezzen mind a saját aktuális állapotával, mind az objektum elhelyezkedésére vonatkozó információval, ezáltal lehetővé téve az értelmes döntéshozatalt.

4.2.2. Akciótér

Az ágens az aktuális állapot alapján akciókat hajt végre, az akciótér elemei az ágens kimeneteinek felelnek meg. Az akciótér a következő komponensekből áll:

Robotkar vezérlés: A UR5e robotkar hat csuklójára pozícióparancsokat küldünk. Ez egy hatdimenziós vektor, ahol minden elem az adott izület pozícióját jelöli.

Gripper vezérlés: A megfogó bináris pozícióparancsot kap, ezáltal vagy kinyitja, vagy bezárja a megfogót.

Ennek megfelelően a teljes akciótér egy 7-dimenziós vektor. Ez a felépítés lehetővé teszi, hogy az ágens képes legyen mozgatni a robotkart és a megfogót. Az akcióteret a 4.2 táblázat mutatja be.

4.2. táblázat. Akciótér definíciója

Akció komponens	Típus	Méret	Leírás
Kar ízületi parancs	Folytonos	6	Ízületi elmozdulás parancs, skálázva 0.05-tel
Gripper parancs	Bináris	1	Nyitás (0.0) vagy zárás (41°) mindkét ujjra
Összesen		**7**	

4.2.3. Jutalomfüggvény

Az ágens tanulása szempontjából elengedhetetlen a megfelelő jutalomfüggvény létrehozása, ez adja a visszacsatolást az ágensnek a saját viselkedéséről. A jutalom célja, hogy a kívánt viselkedés – jelen esetben a kocka sikeres megfogása és célpontra helyezése – kialakulását elősegítse, míg a nem kívánatos cselekvéseket büntesse. A tanítás során alkalmazott összetett jutalomfüggvény több komponensből áll:

Közelítés jutalmazása: Ez a komponens a robot end-effectorának és az objektum pozíciójának távolságát igyekszik csökkenteni. A távolság alapján exponenciálisan csökkenő jutalomkeretet hozva létre. Ez segít az ágensnek az objektum közelébe mennie.

Megfogás ösztönzése: Ez a komponens jutalmaz abban az esetben, ha a megfogó zárt állapotban van, miközben az objektum a megfogóujjak között helyezkedik el. Ez elősegíti az időben történő megfogást.

Emelés jutalmazása: Ez a komponens jutalmat ad, ha az objektum egy meghatározott magasság fölé emelkedik. Ezzel az ágens megtanulja az emelés fontosságát a manipulációs folyamat részeként.

Célkövetés: Ez két komponensből áll, az objektum pozícióját hasonlítják össze az epizód elején definiált célponttal. A két komponens különböző szórásértékekkel és súlyozással működik, így egyszerre biztosítva durvább és finomhangolt célkövetési képességet.

Büntető tagok: Két büntetést is beletettem, az első az akciók közötti időbeli változást bünteti, csökkentve ezzel a hirtelen, instabil mozgásokat, a második pedig a robotkar csuklósebességeit bünteti, ezzel elősegítve a finomabb vezérlést.

A jutalom komponenseket a 4.3 táblázat mutatja be. A jutalmak megfelelő súlyozása kulcsfontosságú a tanulási folyamat stabilitása és konvergenciája szempontjából. A tanítás során alkalmazott curriculum learning tovább finomítja ezeket a súlyokat a tanulási szakasz előrehaladásával arányosan.

4.3. táblázat. Jutalomstruktúra komponensei

Jutalom komponens	**Funkció**	**Súly**	**Paraméterek**
reaching_object	Kocka megközelítése	10.0	std = 0.1
gripper_closed_near_object	Gripper zárva van objektum közelében	25.0	threshold = 0.04
lifting_object	Kocka megemelése egy küszöbszint fölé	40.0	min_height = 0.3
object_goal_tracking	Objektum közelítése célpozícióhoz	16.0	std = 0.2, min_height = 0.1
object_goal_tracking_fine_grained	Finomított célkövetés	5.0	std = 0.05, min_height = 0.1
action_rate	Akcióváltozás büntetése	$-1e\text{-}4$	
joint_vel	Ízületi sebességek büntetése	$-1e\text{-}5$	

4.2.4. Curriculum Learning alkalmazása

A curriculum learning célja, hogy a tanulási folyamatot egyszerűbb feladatokkal indítsa, majd fokozatosan növelje a feladat nehézségét. Ez nagyon hasznos olyan feladatoknál, ahol az ágens elsőre nem képes még alapvető feladatokat sem ellátni. A fokozatos terhelés révén azonban hatékonyabban sajátít el egyre nehezedő feladatokat. Jelen környezetben a jutalomfüggvény büntető komponenseinek időbeli súlyozását hoztam létre. A tanítás korai szakaszában a két büntetőkomponenst kisebb súllyal vesszük figyelembe, majd az idő előrehaladtával ezeket egyre nagyobb súllyal büntetjük. A curriculum learning értékeit a 4.4 táblázat mutatja be.

4.4. táblázat. Jutalomkomponensek súlyának módosítása a tanulás során

Jutalom komponens	Kezdeti súly	Módosított súly	Változtatás ideje (lépés)
action_rate	−1e-3	−1e-4	4500
joint_vel	−1e-3	−1e-5	4500

4.2.5. Epizód befejezési feltételek

Egyik fontos része a tanításnak az epizódok végének meghatározása. Ez megadja, hogy milyen helyzeteket tekintünk lezárt tanulási egységnek. Jelen környezetben kétféle terminációs feltételt szabtam:

Idő alapú megszakítás (timeout): Egy epizód maximális hossza 5 másodperc. Ha ez alatt nem történik sikeres manipuláció, az epizód automatikusan lezárul. Ez biztosítja, hogy az ágens ne végezzen értelmetlen vagy elhúzódó műveleteket.

Objektum leejtése: Ha az objektum leesik az asztalról, akkor az epizód szintén megszakításra kerül. Ez a feltétel a szabályok megsértését vagy a manipuláció sikertelenségét jelzi.

A két terminációs feltétel kombinációja biztosítja, hogy a tanulás fókuszált és hatékony módon történjen. A leírt logika segít az ágensnek, hogy a sikeres manipulációt minél kevesebb lépésből és megbízhatóan hajtsa végre.

4.2.6. Környezet inicializációja

Az epizódok kezdetén a környezet inicializációja is fontos szerepet játszik. Az inicializáció során meghatározhatjuk a szimuláció változatosságát, egy biztosítja a stratégia általánosító képességét is. A tanítás során a következő inicializálási stratégiák kerültek alkalmazásra:

Robot és környezet elemeinek pozíciója: A robotkar, az asztal és a többi statikus elem rögzített helyzetből indul minden epizódban.

Objektum pozíciója: Az objektum kezdőpozícióját minden epizódindításnál kis mértékben véletlenszerűen módosítottam. Az objektum sebessége minden esetben nullázott.

Ez a stratégia hozzájárul ahhoz, hogy a tanulási folyamat során az ágens fokozatosan alkalmazkodjon különböző kezdőfeltételekhez, így robusztusabb és megbízhatóbb viselkedésre legyen képes a továbbiakban.

4.3. Implementáció módja

A kutatási feladat megvalósítása során a már említett Isaac Sim és Isaac Lab szimulációs platformokat használtam a robotikai környezet létrehozására, míg a megerősítéses algoritmusok implementációját az skrl könyvtár segítségével valósítottam meg. A fejlesztés Python programozási nyelven történt, Ubuntu 22.04 operációs rendszeren. A fent említett környezeti konfigurációt saját fejlesztésű osztályokkal alakítottam ki, különös figyelmet fordítva az állapottér, akciótér, jutalomfüggvények és a curriculum learning elemeinek testreszabására. A tanítási folyamat lebonyolításához már az Isaac Lab-ben megtalálható szkriptet alkalmaztam, viszont ez csak PPO algoritmussal működött, ezért saját tanító szkriptet kellett írnom, mely képes használni a fent felsorolt algo-

4.5. táblázat. Legfontosabb hiperparaméterek összehasonlítása

Hiperparaméter	**TRPO**	**TD3**	**PPO**	**SAC**
seed	42	42	42	42
rollouts	16	16	24	16
learning_epochs	8	–	8	–
nn_layers	256, 128, 64	256, 128, 64	256, 128, 64	256, 128, 64
actication_function	elu	elu	elu	elu
mini_batches	2	–	4	–
discount_factor	0.99	0.99	0.99	0.99
lambda	0.95	–	0.95	–
learning_rate	–	0.0001	0.0001	0.0003
grad_norm_clip	0.5	1.0	1.0	1.0
entropy_loss_scale	–	–	0.001	–
kl_threshold	–	–	0.0	–
ratio_clip	–	–	0.2	–
value_clip	–	–	0.2	–
timesteps	100000	100000	100000	100000

ritmusokat, mint a TRPO, PPO, SAC és TD3. Ezeknek az algoritmusoknak a hiperparamétereit beállítottam, finomhangoltam, a fontosabb hiperparaméterek a 4.5 táblázatban láthatók.

5. Kísérleti eredmények

A kutatási munka során különböző RL algoritmusok teljesítményét és viselkedését vizsgáltam meg egy szimulált környezetben. A kísérletek célja az volt, hogy objektív összehasonlítást nyújtsak a TRPO, PPO, SAC és TD3 algoritmusok között, figyelembe véve a tanulási teljesítményt, a stabilitást és a tanulási dinamika jellemzőit. A kísérletek során a modellek azonos környezetben és lehetőleg azonos konfigurációs beállítások mellett kerültek betanításra, biztosítva az eredmények összehasonlíthatóságát. Következőkben részletesen bemutatom a vizsgált teljesítménymutatókat, az algoritmusok tanulási görbéit, majd számszerűen összehasonlítom és értékelem az elért eredményeket.

5.1. Vizsgált paraméterek

A kísérlet eredményeinek értékelése során több szempontból is vizsgáltam az alkalmazott megerősítéses tanulási algoritmusokat. A paraméterek kiválasztása során fő célom volt, hogy a tanulás hatékonyságát, stabilitását és megbízhatóságát tudjam értékelni. Az alábbi fő mutatókat határoztam meg értékelés szempontjából:

Átlagos epizódjutalom (mean episode reward): Ez a mutató azt mutatja meg, hogy az egyes epizódokban mennyi volt a jutalmak átlaga egy adott időablakon belül. Ez indikátora a tanulás előrehaladásának és a feladat sikeres végrehajtásának.

Tanulási sebesség (time to convergence): Az a lépésszám, amely alatt az ágens az adott feladatot stabilan magas teljesítménnyel képes végrehajtani. A konvergencia megmutatja a tanulás hatékonyságát.

Cselekvő hálózat átlagos vesztesége (mean policy loss): Az actor hálózat tanítási vesztesége megmutatja a tanulási folyamat stabilitását. Nagy fluktuációk instabil tanulásra utalnak.

Kritikus hálózat átlagos vesztesége (mean critic loss): Megmutatja az értékbecslés minőségét a SAC és TD3 algoritmusok esetében.

Stratégia entrópiája (policy entropy): A TRPO, PPO és SAC algoritmusok esetében a stratégia entrópiája azt mutatja meg, hogy a stratégia mennyire bizonytalan.

A fent bemutatott mérőszámok segítségével képesek vagyunk összevetni az egyes algoritmusokat.

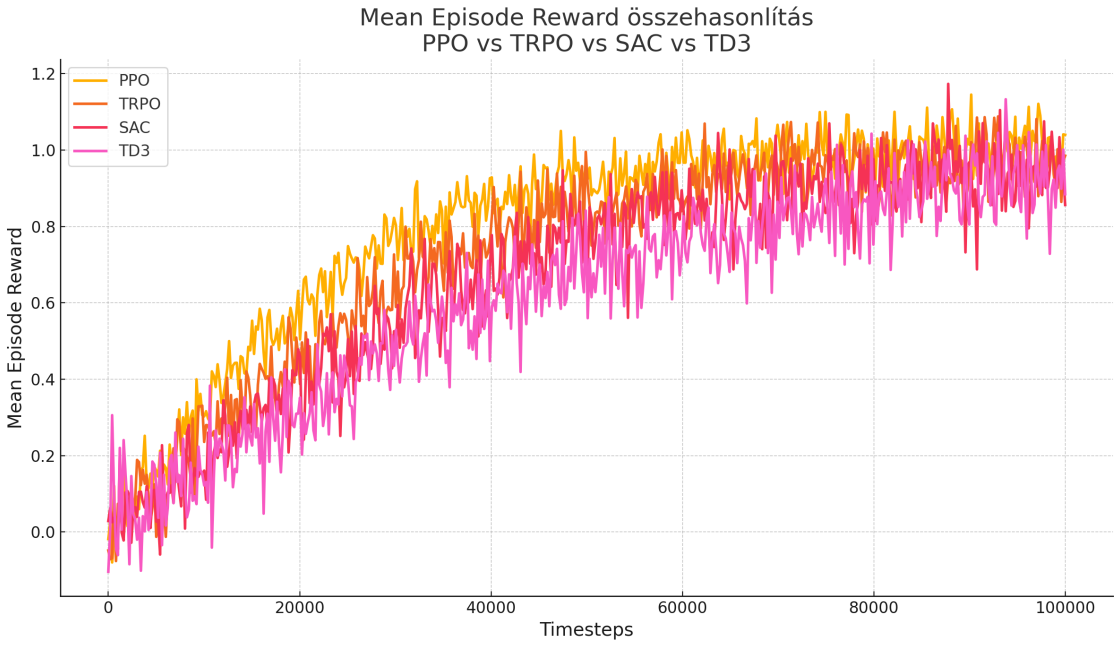
5.2. Tanulási görbék bemutatása és értékelése

Ebben a fejezetben bemutatom a tanítás során készített tanulási görbéket, ezeket az Isaac Lab beépített TensorBoard-jával készítettem a tanítás során logolt adatok alapján. Minden mutató esetén az egyes algoritmusok eredményeit egy közös ábrán ábrázoltam, ez alapján sokkal könnyebb kiértékelni az algoritmusokat. Minden ábrán a görbéket mozgóátlag alkalmazásával simítottam, hogy könnyebben értelmezhető trendeket kapjak. A következő alfejezetekben részletesen bemutatom az egyes tanulási görbéket, és kiértékelem az algoritmusok közötti hasonlóságokat és különbségeket.

5.2.1. Átlagos epizódjutalom alakulása és tanulási sebesség

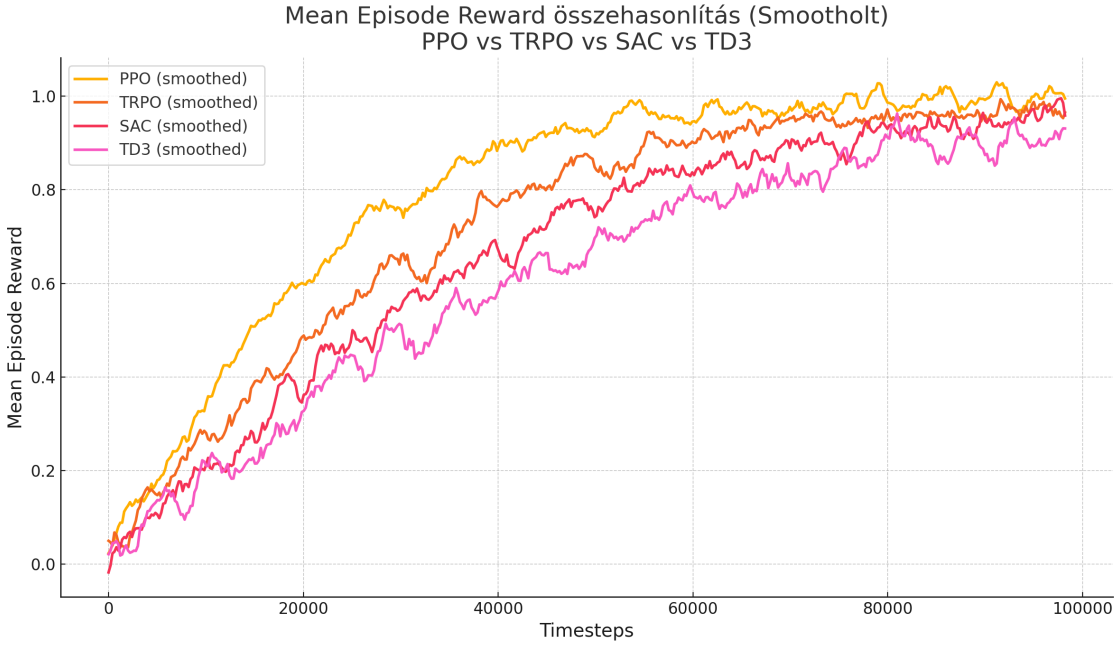
A tanítás során készítettem egy nyers ábrát, mely az átlagos epizódjutalom alakulását mutatja nyers logolt adatokból, külön színnel jelölve a különböző algoritmusokat, ezeket a görbéket a 5 ábra

szemlélteti. A görbéken látható, hogy a tanulás kezdeti szakaszában a TD3 és a SAC algoritmus nagy ingadozást mutatott, míg a PPO és a TRPO stabilabb viselkedést mutattak.



5. ábra. Átlagos epizódjutalom alakulása - nyers görbék

Az adatok trendjének meghatározására készítettem egy simított görbéket bemutató ábrát, amelyen több megfigyelés is tehető (6 ábra).



6. ábra. Átlagos epizódjutalom alakulása - simított görbék

Az első észrevétel a tanulási sebességgel kapcsolatos. A PPO mutatta a leggyorsabb tanulási sebességet: 30000 – 40000 lépés körül már magas átlagos jutalomszintet érve el, majd ezt stabilan is tudta tartani. Ezután következett a TRPO algoritmus, mely hasonlóan jó teljesítményt ért el, viszont lassabb konvergenciát mutatott. A PPO és a TRPO algoritmus után a SAC algoritmus következik tanulási sebességet illetően, az előző kettő társához képest lassabb konvergenciát mutatott, de végül sikerült stabilan magas jutalomszintet elérnie. A TD3 algoritmus mutatta a leglassabb tanulási sebességet, és a tanítás során végig alacsonyabb jutalomszintet produkált a többi algoritmushoz képest. Összességében elmondható, hogy mindegyik algoritmus sikeresen megtanulta a feladatot, az objektum megfogását és felemelését, viszont stabilitás és tanulási sebesség terén is a PPO és a TRPO algoritmusok mutatták a legjobb értékeket. A tanulási görbék alapján az egyes algoritmusok által elért maximális átlagos epizódjutalom értékei a 5.1 táblázat tartalmazza. Ezeket az értékeket a simított görbék alapján becsültem meg.

5.1. táblázat. Átlagos epizódjutalom és konvergencia ideje

Algoritmus	**Maximális átlagos epizódjutalom**	**Konvergencia ideje (timesteps)**
PPO	~ 1.05	$\sim 50\,000$
TRPO	~ 1.00	$\sim 60\,000$
SAC	~ 0.95	$\sim 70\,000$
TD3	~ 0.90	$\sim 80\,000$

Az adatok alapján összefoglalva a következők mondhatók el:

PPO algoritmus érte el a legmagasabb epizódonkénti jutalmat, és a konvergencia is itt történt a leggyorsabban.

TRPO algoritmus szintén magas epizódjutalmat ért el, de a konvergencia kissé lassabb volt.

SAC algoritmus stabil tanulási folyamatot mutatott, de alacsonyabb maximális jutalmat ért el.

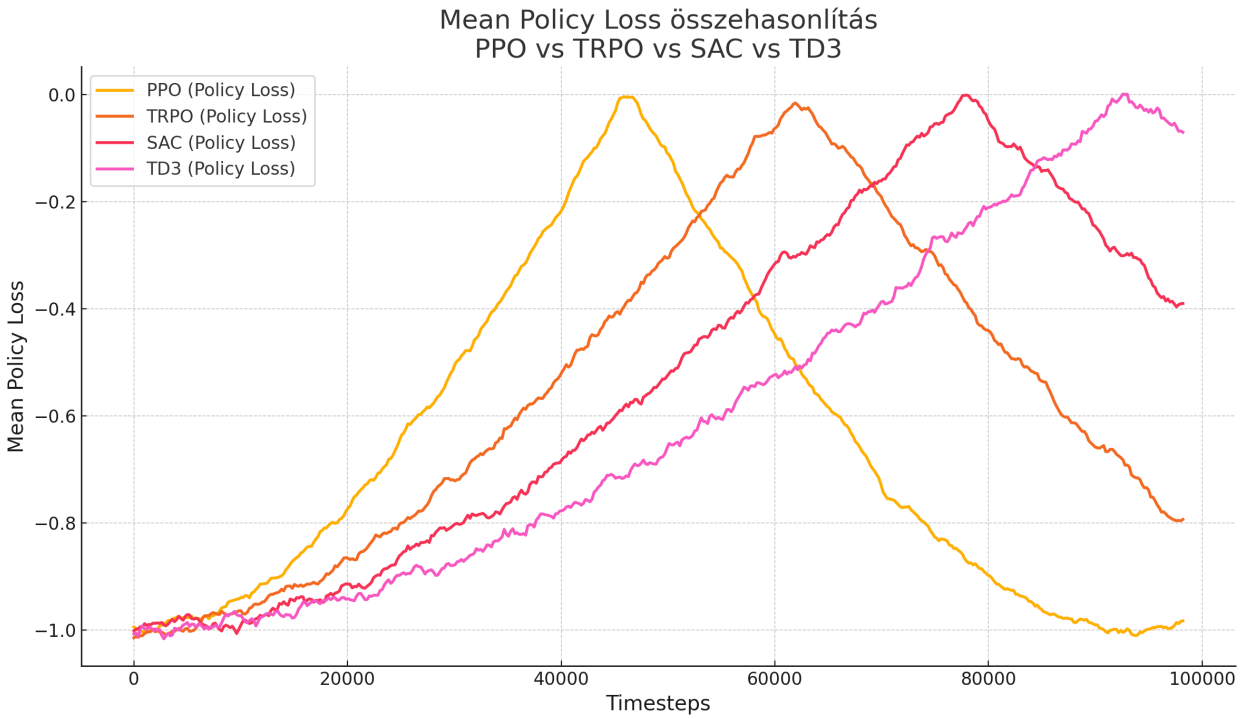
TD3 algoritmus rendelkezett a legalacsonyabb maximális jutalomértékkel és a leglassabb tanulási folyamattal.

Összességében a PPO algoritmus bizonyult a leghatékonyabbnak a mutatók alapján.

5.2.2. Cselekvő hálózat vesztesége (Mean Policy Loss)

A tanulási folyamat stabilitásának értékelése érdekében elkészítettem az egyes algoritmusokhoz tartozó actor hálózatok veszteségfüggvényeit (mean policy loss). A 7 ábra mutatja be a TRPO, PPO, SAC és TD3 algoritmusok actor hálózatának veszteségfüggvényeit az idő függvényében.

A görbéken látható, hogy a PPO algoritmus esetében a veszteség fokozatosan növekszik, majd 45000 lépés környékén elkezd csökkenni. Ez mutatja, hogy a stratégia a korai fázisban nagyban módosul, majd a tanulás előrehaladtával stabilizálódik. TRPO algoritmus esetén is hasonlóképpen alakul, annyi eltéréssel, hogy a görbe maximuma időben később helyezkedik el, ez is mutatva a tanulás sebességét. A SAC algoritmus maximuma az előző kettő mögött helyezkedik el. Ez összhangban van azzal, hogy a SAC algoritmus egy off-policy algoritmus, mely kiegyensúlyozottabb változásokat eredményez. A TD3 algoritmus értékei szintén hasonlóképpen alakultak, viszont az ingadozások nagyobb mértékűek voltak, különösen a későbbi szakaszokban. Összefoglalva, a függvények nem mutattak jelentősebb oszcillációt, ami azt jelzi, hogy a tanulási folyamat



7. ábra. Cselekvő hálózat veszteségfüggvényei - simított görbék

során az actor hálózatok stabil módosulásokon mentek keresztül. A PPO és a TRPO algoritmusok esetén a veszteségfüggvények jól lekövetik a jutalomgörbék alakulását is. A cselekvő hálózat veszteségfüggvényeinek (policy loss) alakulása alapján az algoritmusok által elért jellemző átlagos veszteségértékeket a 5.2 táblázat mutatja be.

5.2. táblázat. Cselekvő hálózatok veszteségjellemzőinek összehasonlítása

Algoritmus	**Minimális veszteség érték**	**Maximális veszteség érték**	**Általános stabilitás**
PPO	~ -1.2	~ -0.01	Jó, kisebb ingadozások
TRPO	~ -1.0	~ -0.03	Jó, folyamatos változás
SAC	~ -1.0	~ -0.01	Stabil, sima trend
TD3	~ -1.1	~ -0.01	Kissé nagyobb ingadozások

A számszerűsített adatok alapján az alábbi megfigyeléseket tettem:

PPO és TRPO algoritmusoknál fokozatosan növekedett, majd stabilizálódott. Kisebb ingadozások jellemezték.

SAC algoritmus esetén a görbe a legsimább, változások fokozatosak, tanulás stabilitását jelzi.

TD3 algoritmusnál nagyobb az ingadozás, különösen a tanulás későbbi szakaszaiban.

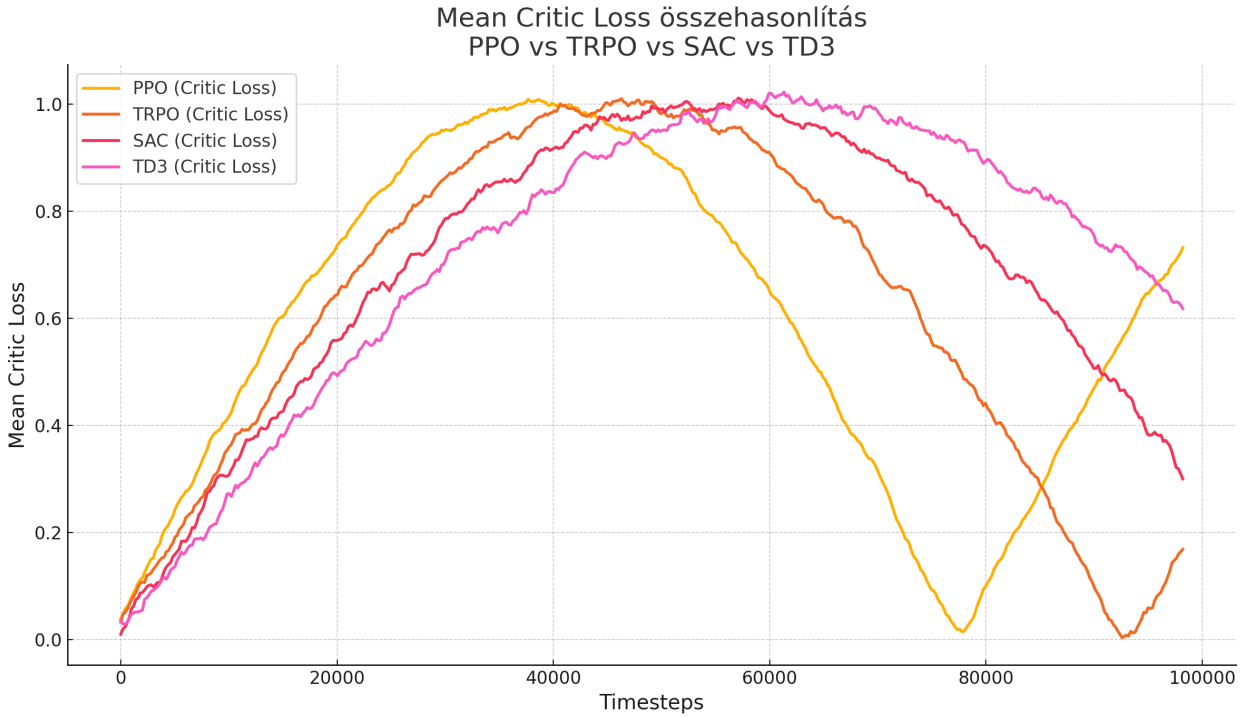
Összességében elmondható, hogy mindegyik algoritmus érétkei ésszerű határok között mozogtak, instabilitás nem volt jelentős, ami a tanulási folyamat sikerességét mutatja be.

5.2.3. Kritikus hálózat vesztesége (Mean Critic Loss)

Következő lépcsőben a kritikus hálózat veszteségfüggvényeit vizsgáltam. Ezeknek az alakulása megmutatja, hogy milyen stabil az értékbecslés a tanulás során, ez pedig befolyásolja a tanulás sikerességét, különösen az értékalapú (off-policy) algoritmusok esetében. Minden algoritmusnál értelmezhető a kritikus hálózat vesztesége, azonban az egyes algoritmusoknál különböző módon számoljuk:

PPO és TRPO: a value hálózat a környezeti állapotok értékét becsüli meg. A veszteség a jósolt értékek és a célértékek közötti négyzetes hiba (MSE) minimalizálására irányul.

SAC és TD3: két különálló kritikus hálózat van (Q1 és Q2) tanul egymással párhuzamosan a cél Q-érték felé. Itt is négyzetes hibán alapul a veszteség. Ennek megfelelően a kritikus hálózat vesztesége elsősorban az értékbecslés pontosságának indikátora. A 8 ábra a TRPO, PPO, SAC és TD3 algoritmusok kritikus hálózatának veszteségfüggvényeit mutatja be idő függvényében.



8. ábra. Kritikus hálózat vesztesége - simított görbék

Az ábra alapján a PPO és TRPO algoritmusok esetében a kritikus háló veszteségértékei kezdetben növekednek, majd egy maximum pont elérése után csökkenni kezdenek. Ez azt jelzi, hogy a kezdeti változékonyság után az értékbecslések stabilizálódnak. A tanulás későbbi szakaszában azonban mindkét algoritmus esetén növekedés figyelhető meg, ami túlillesztésre (overfitting) utal. A SAC esetében a kritikus hálózat veszteségértékei szintén növekednek, majd 60000-70000 lépés körül csökkent. A TD3 algoritmus esetén is hasonló mintákat figyelhetünk meg, de a görbe sokkal laposabb ívet mutat, ez utal az algoritmus konzervatívabb tanulására is. Összességében megállapítható, hogy a kritikus háló veszteségfüggvénye nem mutat extrém ingadozást, ami stabil tanulási folyamatot jelez. A SAC és TD3 algoritmusok esetében ez egy fontos mutató, viszont a PPO és a

TRPO esetében kisebb jelentőséggel bír, de hasonló trendet mutatnak. A kritikus hálózat veszteségfüggvényeinek alakulása alapján az algoritmusok által elért jellemző veszteségértékeket a 5.3 táblázat mutatja be.

5.3. táblázat. Critic loss jellemzők algoritmusonként

Algoritmus	**Min critic loss érték**	**Max critic loss érték**	**Általános stabilitás**
PPO	~ 0.02	~ 1.01	Jó, csökkenő trend
TRPO	~ 0.03	~ 1.02	Jó, fokozatos stabilizáció
SAC	~ 0.05	~ 1.0	Stabil, kisebb oszcillációkkal
TD3	~ 0.05	~ 1.05	Közepes stabilitás, nagyobb ingadozások

A fenti táblázat alapján az alábbi főbb megállapításokat tettem:

PPO és TRPO algoritmusoknál az értékek viszonylag alacsonyak maradtak, és a tanulás előrehaladtával csökkenő tendenciát mutattak. Ez a jó értékbecslési stabilitásra utal.

SAC algoritmusnál kissé magasabbak voltak az értékek, de változásuk fokozatos volt, és nem mutatott extrém ingadozásokat.

TD3 algoritmusnál az értékek a legmagasabbak voltak, és enyhén nagyobb ingadozás figyelhető meg.

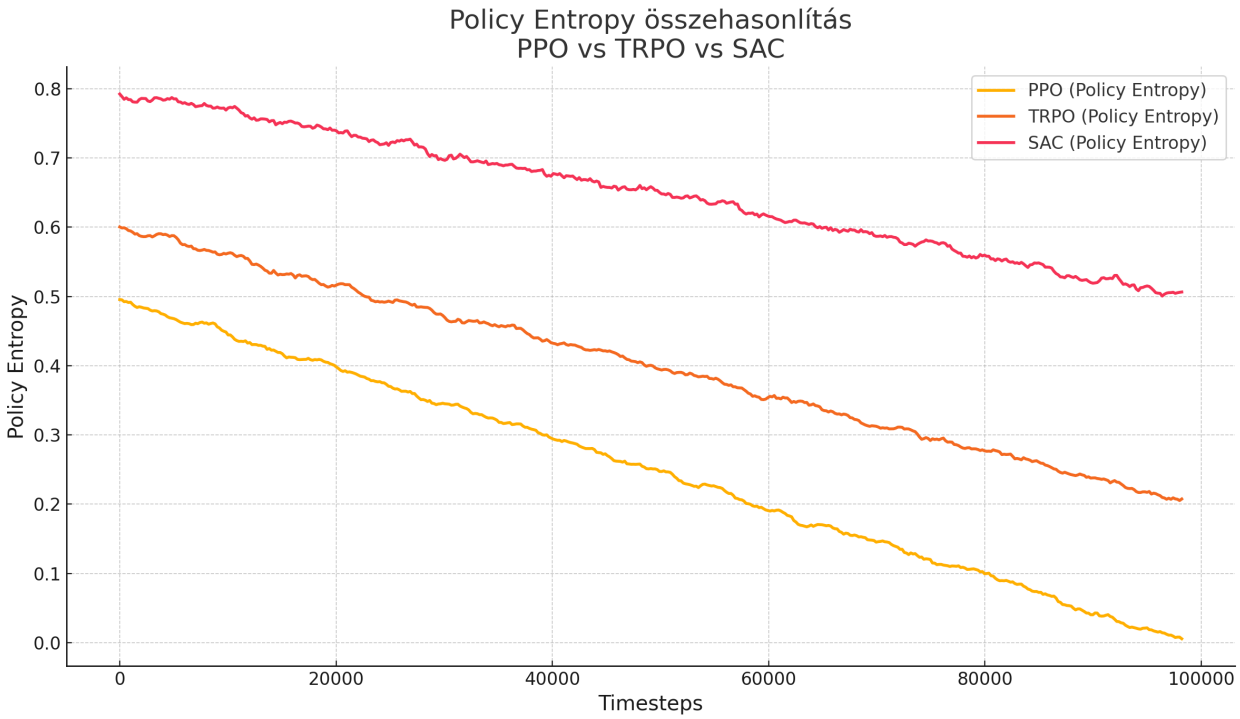
Összességében a kritikus hálózat veszteségértékei stabilan alakultak, az algoritmusok megfelelő értékbecslést tanultak a feladat során.

5.2.4. Stratégia entrópiája (Policy entropy)

A különböző algoritmusok explorációs képességeinek értékelése miatt megvizsgáltam a stratégia entrópia időbeli alakulását a tanítás során. A stratégia entrópiája azt mutatja meg, hogy az ágens mennyire bizonytalan az akcióválasztás során, magas entrópia intenzív felfedezést jelent, míg az alacsonyabb entrópia a determinisztikus viselkedést (exploitation) jelenti. A 9 ábra a PPO, TRPO és SAC algoritmusok stratégia entrópiáját mutatja be. A TD3 algoritmus determinisztikus algoritmus, ezért itt nem értelmezhető a stratégia entrópiája, ezért ez az algoritmus kimaradt az összehasonlításból.

Az ábra alapján megállapíthatjuk, hogy a SAC algoritmus kezdetben nagyon magas entrópiával rendelkezett, ami azt jelenti, hogy előtérbe helyezte a felfedezést, de az idő előrehaladtával ez egyre csökkent, de megmaradt egy közepes szint körül. A TRPO algoritmus esetén közepes értékről indult, amely szintén csökkent a tanulás előrehaladtával, majd alacsonyabb szinten stabilizálódott. A PPO algoritmus entrópiája volt a legalacsonyabb a három algoritmus közül, amely a tanulás előrehaladtával gyorsan csökkent, a végére majdnem determinisztikus viselkedést alakított ki. Ez a robotikában igazán előnyös. Összességében az entrópia változása megmutatja a különböző stratégiákat, a SAC hosszabb ideig tartja fenn az explorációt, míg a PPO nagyon hamar lecsökkentette az entrópiáját, ami hozzájárulhatott egy gyorsabb konvergenciához is.

A tanulási folyamat során rögzített stratégia entrópia jellemző adatait a 5.4 táblázat mutatja be. A táblázat alapján a következő megfigyeléseket tettem:



9. ábra. Stratégia entrópiája - simított görbék

PPO algoritmus kezdettől fogva közepes entrópiával működött, majd gyorsan csökkent, ez a gyors exploráció felhagyást jelenti.

TRPO algoritmusnál szintén fokozatos volt a csökkenés, de lassabb tempóban, mint a PPO esetén.

SAC algoritmus esetén a legmagasabb értékről indult, és onnan csökkent közepes értékig. Ez azt mutatja, hogy a SAC folyamatosan fenntart egy jelentős mértékű feltérképező viselkedést még a tanulás késői szakaszában is.

Összességében elmondható, hogy az entrópia alakulása jól tükrözte az egyes algoritmusok eltérő explorációs stratégiáit.

5.4. táblázat. Az entrópia változása a tanulás során

Algoritmus	**Kezdeti entrópia érték**	**Végső entrópia érték**	**Entrópia csökkenés jellege**
PPO	~ 0.60	~ 0.05	Gyors csökkenés, gyors stabilizáció
TRPO	~ 0.65	~ 0.10	Fokozatos, stabil csökkenés
SAC	~ 0.80	~ 0.50	Lassabb csökkenés, magasabb explorációs szint
TD3	–	–	Nem értelmezhető (deterministikus)

5.2.5. Átfogó értékelés

A tanulási folyamat értékelése során végzett vizsgálatok alapján készítettem egy átfogóbb képet az algoritmusok teljesítményéről, ezek a megfigyelések az alábbiak:

PPO algoritmus teljesített összességében a legjobban: leggyorsabb konvergenciát mutatta, magas maximális epizódjutalommal. Az entrópia csökkenése is gyors ütemben zajlott.

TRPO algoritmus szintén jó teljesítményt mutatott, kissé lassabb tanulással, stabil kritikus háló veszteségértékekkel és fokozatos entrópia csökkenéssel.

SAC algoritmus lassabban konvergált, de fenntartotta az explorációs szintet a későbbi szakaszokban is, maximális jutalomértékei elmaradtak a PPO és TRPO eredményeitől.

TD3 algoritmus a leglassabb tanulási folyamatot mutatta, legalacsonyabb maximális epizódjutalom mellett. A kritikus háló veszteségértékei nagyobb ingadozást mutattak.

Összefoglalva megállapítható, hogy a PPO algoritmus bizonyult a leghatékonyabbnak a vizsgált szimulációs feladatban, míg a SAC explorációt fenntartó tulajdonsága robusztusabbá teheti a tanulási folyamatot bonyolultabb környezetekben.

6. Következtetések és jövőbeli munkák

A dolgozat célja az volt, hogy különböző algoritmusok teljesítményét vizsgáljam szimulált objektummanipulációs környezetben. A kutatás során részletes elemzést készítettem a tanulási folyamat során elért eredményekről. Ebben a fejezetben összefoglalom a kutatás főbb tanulságait és levont következtetéseit, majd javaslatot teszek a jövőbeli munkákat illetően, melyek tovább mélyíthetik a megerősítéses tanulási megközelítések alkalmazhatóságát robotikai rendszerekben.

6.1. Következtetések

Dolgozatom kutatási céljaként azt határoztam meg, hogy az egyetemen található hardverkörnyezetet szimulációban felépítsem, majd ezen a szimulációs környezeten betanítsam a robotkart objektum felvételére különböző megerősítéses tanulási algoritmusok alkalmazásával, majd ezeket az algoritmusokat összevessem különböző szempontok alapján: átlagos epizódjutalom, tanulási konvergencia, hálózatok veszteségfüggvényeinek alakulása és explorációs viselkedés változása alapján. A különböző görbék és számszerűsített eredmények alapján a következő főbb megállapítások tehetők:

PPO algoritmus mutatta a leggyorsabb tanulási sebességet és a legmagasabb végső teljesítményt, gyors entrópiacsökkenéssel.

TRPO algoritmus szintén megbízható teljesítményt mutatott, de lassabb konvergenciával.

SAC algoritmus leghosszabb ideit tartotta az explorációs képességét, stabil tanulási folyamat mellett, de végső teljesítménye elmaradt a PPO és TRPO eredményeitől.

TD3 algoritmus esetén a tanulási folyamat a leglassabb volt, a végső teljesítmény is alacsonyabb volt a többi algoritmushoz képest.

Az eredmények alapján megállapítható, hogy a PPO alkalmazása kínálja a legjobb megoldást a gyors tanulásra, mely közben megőrzi stabilitását, egyben magas teljesítményt tud nyújtani. Ugyanakkor a SAC algoritmus által fenntartott exploráció előnyös lehet bonyolultabb környezetekben, ahol nagyon fontos az alkalmazkodóképesség. Összességében a kutatás arra is rávilágított, hogy a szimulált környezetek kiváló eszközként funkcionálnak az ágensek feltanítására, ugyanakkor a választott algoritmusok jellemzőinek és tanulási dinamikájának ismerete kulcsfontosságú a sikeres alkalmazáshoz.

6.2. Jövőbeli munkák

A kutatás során elért eredmények alapján számos lehetséges jövőbeli irány fogalmazódott meg. Ez az alfejezet ezeket az irányokat fogja bemutatni.

Valós robotra való transzfer (Sim2Real transfer): A következő logikus lépés, hogy a most feltanított stratégiát átültessük a valós robotra. Ehhez szükséges a szimulált világ, és a valós világ közötti különbségek (reality gap) minimalizálására, például fotorealisztikus megjelenítéssel, valamint domén randomizációval.

További algoritmusok vizsgálata: A kutatás során vizsgált TRPO, PPO, SAC és TD3 algoritmusokon túl érdemes lenne más, újabb fejlesztésű algoritmust is megvizsgálni, mint például a DDPG vagy akár a modern, transformer-alapú RL megközelítések.

Bonyolultabb feladatok modellezése: Jelenlegi kutatásban egy egyszerű feladaton vizsgáltam az algoritmusokat, a jövőben viszont érdemes lenne nehezebb feladatok, és komplexebb környezetek vizsgálata.

Tanulási hatékonyság és általánosítás vizsgálata: További vizsgálatok fókuszálhatnának arra, hogy az algoritmusok milyen mértékben képesek általánosítani nem látott helyzetekre, illetve hogyan lehet a tanulási minták számát csökkenteni a hatékonyság javára.

Összefoglalva a jelen kutatás egy szilárd alapot nyújt a robotmanipuláció során használható algoritmusok megismerésében, ugyanakkor számos irányba továbbfejleszthető, mind a módszertani, mind az alkalmazási aspektusokat tekintve.

7. Összefoglaló

A dolgozat célja egy UR5e robotkar kockamanipulációs feladatának szimulációs környezetben történő megoldása volt megerősítéses tanulás (Reinforcement Learning, RL) alkalmazásával. A kutatási munka során felépítettem egy valós hardverkörnyezetet modellező szimulációt Isaac Sim segítségével, majd különböző RL algoritmusok – TRPO, PPO, SAC és TD3 – teljesítményét és tanulási dinamikáját vizsgáltam. A modellek azonos feltételek mellett kerültek betanításra, így lehetőség nyílt objektív összehasonlításra. Az értékelés során figyelembe vettem az elért átlagos epizódjutalmakat, a tanulási sebességet, az actor és critic hálózatok veszteségfüggvényeinek alakulását, valamint a stratégia entrópiájának változását. A vizsgálatok alapján a PPO algoritmus mutatta a leggyorsabb konvergenciát és a legjobb végső teljesítményt, míg a SAC hosszabb ideig fenntartotta az explorációs képességét. A kutatás eredményei rávilágítottak az egyes algoritmusok jellemzőire, megerősítve a szimulációs környezetek hasznosságát az RL-alapú robotmanipulációs feladatok fejlesztésében. A dolgozatban megfogalmazott jövőbeli kutatási irányok közé tartozik a valós robotra történő átvitel (sim2real) megvalósítása, további RL algoritmusok kipróbálása, valamint összetettebb manipulációs feladatok vizsgálata.

8. Köszönetnyilvánítás

Ezúton szeretném kifejezni őszinte köszönetemet Széll Károly egyetemi docens úrnak, témavezetőmnek, aki szakmai útmutatásával, támogatásával és értékes tanácsaival nagyban hozzájárult a dolgozatom elkészítéséhez.

A dokumentumban bemutatott kutatás az EKÖP pályázatomhoz kapcsolódik.

„A KULTURÁLIS ÉS INNOVÁCIÓS MINISZTÉRIUM 2024-2.1.1 KÓDSZÁMÚ EGYETEMI KUTATÓI ÖSZTÖNDÍJ PROGRAMJÁNAK A NEMZETI KUTATÁSI, FEJLESZTÉSI ÉS INNOVÁCIÓS ALAPBÓL FINANSZÍROZOTT SZAKMAI TÁMOGATÁSÁVAL KÉSZÜLT.”

Irodalomjegyzék

[1] M. Naeem, S. T. H. Rizvi, and A. Coronato, „A Gentle Introduction to Reinforcement Learning and its Application in Different Fields," *IEEE Access*, vol. 8, 2020, conference Name: IEEE Access. [Online]. Available: https://ieeexplore.ieee.org/document/9261348

[2] M. Q. Mohammed, K. L. Chung, and C. S. Chyi, „Review of Deep Reinforcement Learning-Based Object Grasping: Techniques, Open Challenges, and Recommendations," *IEEE Access*, vol. 8, pp. 178 450–178 481, 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9210095

[3] A. Wolf, D. Wolton, J. Trapl, J. Janda, S. Romeder-Finger, P. Galambos, and K. Széll, „Towards Robotic Laboratory Automation Plug & Play: The "LAPP" Framework," *SLAS TECHNOLOGY*, vol. 27, no. 1, pp. 18–25, 2022, number: 1 Publisher: Elsevier Inc. [Online]. Available: https://doi.org/10.1016%2Fj.slast.2021.11.003

[4] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, „Deep Reinforcement Learning: A Brief Survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017. [Online]. Available: https://ieeexplore.ieee.org/document/8103164

[5] B. J. A. Kröse, „Learning from delayed rewards," *Robotics and Autonomous Systems*, vol. 15, no. 4, p. 233, 1995. [Online]. Available: https://www.academia.edu/3294050/Learning_from_delayed_rewards

[6] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, „Proximal Policy Optimization Algorithms," Aug. 2017, arXiv:1707.06347 [cs]. [Online]. Available: http://arxiv.org/abs/1707.06347

[7] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, „Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor," Aug. 2018, arXiv:1801.01290 [cs]. [Online]. Available: http://arxiv.org/abs/1801.01290

[8] S. Fujimoto, H. v. Hoof, and D. Meger, „Addressing Function Approximation Error in Actor-Critic Methods," Oct. 2018, arXiv:1802.09477 [cs]. [Online]. Available: http://arxiv.org/abs/1802.09477

[9] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, „Trust Region Policy Optimization," Apr. 2017, arXiv:1502.05477 [cs]. [Online]. Available: http://arxiv.org/abs/1502.05477

[10] A. Goyal, V. Blukis, J. Xu, Y. Guo, Y.-W. Chao, and D. Fox, „RVT-2: Learning Precise Manipulation from Few Demonstrations," Jun. 2024, arXiv:2406.08545 [cs]. [Online]. Available: http://arxiv.org/abs/2406.08545

[11] A. Goyal, J. Xu, Y. Guo, V. Blukis, Y.-W. Chao, and D. Fox, „RVT: Robotic View Transformer for 3D Object Manipulation."

[12] OpenAI, I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, J. Schneider, N. Tezak, J. Tworek, P. Welinder,

L. Weng, Q. Yuan, W. Zaremba, and L. Zhang, „Solving Rubik's Cube with a Robot Hand," Oct. 2019, arXiv:1910.07113 [cs]. [Online]. Available: http://arxiv.org/abs/1910.07113

[13] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine, „QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation," Nov. 2018, arXiv:1806.10293 [cs]. [Online]. Available: http://arxiv.org/abs/1806.10293

[14] D. Horváth, K. Bocsi, G. Erdős, and Z. Istenes, „Sim2Real Grasp Pose Estimation for Adaptive Robotic Applications," *IFAC-PapersOnLine*, vol. 56, no. 2, Jan. 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896323004676

[15] D. Horváth, G. Erdős, Z. Istenes, T. Horváth, and S. Földi, „Object Detection Using Sim2Real Domain Randomization for Robotic Applications," *IEEE Transactions on Robotics*, vol. 39, no. 2, pp. 1225–1243, 2023. [Online]. Available: https://ieeexplore.ieee.org/document/9916581

[16] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, „OpenAI Gym," Jun. 2016, arXiv:1606.01540 [cs]. [Online]. Available: http://arxiv.org/abs/1606.01540

[17] S. James, Z. Ma, D. R. Arrojo, and A. J. Davison, „RLBench: The Robot Learning Benchmark & Learning Environment," Sep. 2019, arXiv:1909.12271 [cs]. [Online]. Available: http://arxiv.org/abs/1909.12271

[18] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, and G. State, „Isaac Gym: High Performance GPU-Based Physics Simulation For Robot Learning," Aug. 2021, arXiv:2108.10470 [cs]. [Online]. Available: http://arxiv.org/abs/2108.10470

[19] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, „Sim-to-Real Transfer of Robotic Control with Dynamics Randomization," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, May 2018, pp. 3803–3810, arXiv:1710.06537 [cs]. [Online]. Available: http://arxiv.org/abs/1710.06537

[20] S. Levine, P. Pastor, A. Krizhevsky, and D. Quillen, „Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection," Aug. 2016, arXiv:1603.02199 [cs]. [Online]. Available: http://arxiv.org/abs/1603.02199

Ábrák jegyzéke

Táblázatok jegyzéke