VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ

FACULTY OF MECHANICAL ENGINEERING

ÚSTAV MECHANIKY TĚLES, MECHATRONIKY A BIOMECHANIKY

INSTITUTE OF SOLID MECHANICS, MECHATRONICS AND BIOMECHANICS

NÁVRH A REALIZACE SIMULAČNIHO MODELU DVOUNOHÉHO ROBOTU

DESIGN AND IMPLEMENTATION OF A BIPEDAL ROBOT MODEL

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — Bc. Marek Meško

AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. Stanislav Věchet, Ph.D.

SUPERVISOR

BRNO 2025

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání diplomové práce

Ústav:	Ústav mechaniky těles, mechatroniky a biomechaniky
Student:	**Bc. Marek Meško**
Studijní program:	Mechatronika
Studijní obor:	bez specializace
Vedoucí práce:	**doc. Ing. Stanislav Věchet, Ph.D.**
Akademický rok:	2024/25

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

Návrh a realizace simulačniho modelu dvounohého robotu

Stručná charakteristika problematiky úkolu:

V rámci práce realizujte model dvounohého robotu v dynamickém prostředí PyBullet. Vytvořený model musí být navržený tak aby podporoval spolupráci s trénovacími frameworky pro umělou inteligenci jako jsou Gymnasium nebo StableBaselines3.

Cíle diplomové práce:

1. Seznamte se s prostredím Python Bullet (PyBullet)

2. Prostudujte konstrukci dvounohého robota PAVO

3. Vytvořte adekvátní model v prostředí PyBullet

4. Vytvořte programové rozhranní pro využití modelu v prostředí Gymnasium

Model ve vybraném prostředí otestujte

Seznam doporučené literatury:

[1] Thrun, S., Burgard, W., Fox, D. (2005). Probabilistic robotics. Cambridge: MIT Press.

[2] ROS.org. ROS.org | Powering the world's robots. [online]. 2.11.2016 [cit. 2016-11-02]. Dostupné z: http://www.ros.org/.

[3] SOLEM, J. E., Programming Computer Vision with Python: Tools And Algorithms For Analyzing Images, 1st edition, 2012.

[4] LIGHT, R. A., Mosquitto: server and client implementation of the MQTT protocol, The Journal of Open Source Software, vol. 2, no. 13, May 2017, DOI: 10.21105/joss.00265.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2024/25

V Brně, dne

L. S.

prof. Ing. Jindřich Petruška, CSc. ředitel ústavu	doc. Ing. Jiří Hlinka, Ph.D. děkan fakulty

Abstrakt

Táto diplomová práca sa zaoberá návrhom a realizáciou simulačného modelu dvojnohého robota s využitím metód učenia posilňovaním (RL), ktorého cieľom je schopnosť pohybovať sa vpred. V teoretickej časti je spracovaná rešerš metód učenia posilňovaním so zameraním na algoritmus Proximal Policy Optimization (PPO), prehľad dostupných simulačných prostredí a analýza vybraných humanoidných robotov. V praktickej časti je podrobne analyzovaný proces tvorby modelu robota v Unified Robot Description Format (URDF) a v simulačnom prostredí MuJoCo, pričom sú tieto prístupy aj vzájomne porovnané. Následne sú prezentované experimenty vykonané v prostredí Walker2D, ktoré slúžia ako základ pre ďalšiu aplikáciu na vlastný model robota PAWO. Na záver sú získané poznatky prenesené do simulačného prostredia PAWO s cieľom dosiahnuť stabilnú a plynulú chôdzu v simulačnom prostredí PyBullet.

Abstract

This diploma thesis deals with the design and implementation of a simulation model of a bipedal robot using reinforcement learning (RL) methods, the goal of which is the ability to move forward. The theoretical part includes a search for reinforcement learning methods with a focus on the Proximal Policy Optimization (PPO) algorithm, an overview of available simulation environments and an analysis of selected humanoid robots. The practical part analyses in detail the process of creating a robot model in Unified Robot Description Format (URDF) and in the MuJoCo simulation environment, these approaches are also compared with each other. Subsequently, experiments performed in the Walker2D environment are presented, which serve as a basis for further application to the own PAWO robot model. Finally, the acquired knowledge is transferred to the PAWO simulation environment in order to achieve stable and smooth walking.

Kľúčové slová

Učenie posilňovaním, PPO, PyBullet, URDF, MuJoCo, Dvojnohý robot, Walker2D, PAWO, Simulačné prostredie, Odmenová funkcia

Keywords

Reinforcement learning, PPO, PyBullet, URDF, MuJoCo, Bipedal robot, Walker2D, PAWO, Simulation environment, Reward function

Bibliografická citácia

MEŠKO, M. *Návrh a realizace simulačního modelu dvounohého robotu*. Brno, 2025. Dostupné tiež z: https://www.vut.cz/studenti/zav-prace/detail/165867. Diplomová práca. Vysoké učení technické v Brně, Fakulta strojního inženýrství, 65 s., Vedúci práce: doc. Ing. Stanislav Věchet, Ph.D..

Čestné prehlásenie

Prehlasujem, že som diplomovú prácu na tému *Návrh a realizace simulačniho modelu dvounohého robotu* spracoval samostatne s pomocou môjho vedúceho, odbornej literatúry a zdrojov, ktoré sú všetky uvedené v zozname použitej literatúry na konci práce.

Marek Meško

Brno 23.5.2025 . . . . . .

Poďakovanie

Ďakujem vedúcemu diplomovej práce doc. Ing. Stanislavu Věchetovi, Ph.D. za jeho vedenie, cenné rady a pripomienky počas tvorby tejto práce.

Ďalej by som rád poďakoval mojej rodine a blízkym, ktorí mi počas štúdia poskytovali neustálu podporu, porozumenie a povzbudenie.

Marek Meško

Obsah

1 Úvod

Výskum v oblasti umelej inteligencie a robotiky sa v posledných rokoch čoraz viac zameriava na integráciu pokročilých algoritmov učenia s fyzikálnymi a simulačnými modelmi. Jednou z najperspektívnejších oblastí v tomto smere je RL, ktoré umožňuje agentovi zlepšovať svoje správanie na základe skúseností získaných interakciou s prostredím. Na rozdiel od klasických metód riadenia, ktoré sú založené na presných matematických modeloch a analytickej optimalizácii, RL umožňuje zvládať komplexné úlohy, ako napríklad chôdzu dvojnohého robota.

Táto práca sa zaoberá návrhom a trénovaním simulačného modelu robota pomocou algoritmu PPO, ktorý je v dnešnej dobe veľmi populárny, poskytuje spoľahlivé výsledky aj v náročnejších prostrediach a osvedčil sa pri riadení kĺbov dvojnohých robotov.

Na trénovanie a testovanie modelu bol použitý simulačný nástroj PyBullet, ktorý poskytuje realistickú fyzikálnu simuláciu, jednoduché rozhranie na integráciu RL algoritmov a je široko využívaný v akademickej aj výskumnej sfére. V práci sa okrem PyBulletu model robota vytvára aj v MuJoCo, oba prístupy sa následne porovnávajú.

Cieľom práce je pochopenie princípov RL a ich aplikácia v štandardizovanom prostredí Walker2D. Následne získané poznatky preniesť do praktického modelu dvojnohého robota PAWO, ktorý bol vytvorený ako výskumná a výučbová platforma. Model PAWO je navrhnutý ako jednoduchá, ale dostatočne realistická konštrukcia, na ktorej je možné testovať rôzne architektúry riadenia.

Experimentálna časť práce sa najskôr sústreďuje na tvorbu modelu robota PAWO v URDF súbore, ďalej na učenie chôdze v štandardizovanom prostredí Walker2D, v ktorom sa testovali rôzne konfigurácie odmenovej funkcie s cieľom identifikovať najvhodnejší prístup. Na základe týchto poznatkov prebiehal tréning modelu robota PAWO, kde sa odmenová funkcia nastavuje tak, aby sa dosiahlo pohybu vpred do vzdialenosti 2 metre.

2 Teoretické a technologické základy učenia posilňovaním

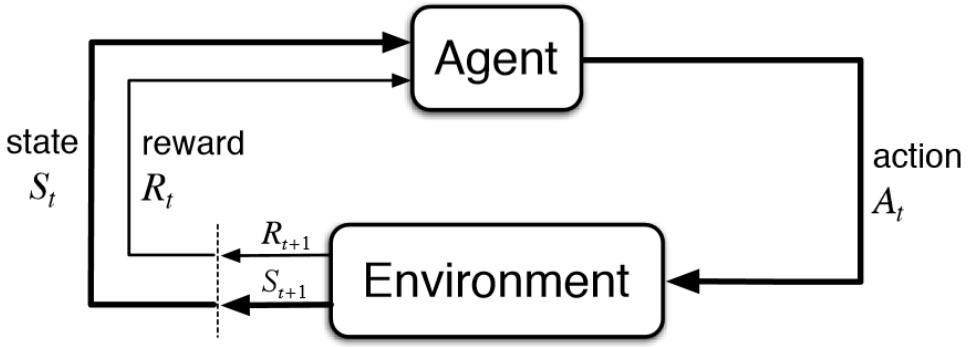
Táto kapitola poskytuje teoretický základ potrebný na pochopenie princípov učenia posilňovaním, ako aj algoritmov, ktoré sú využité v praktickej časti práce. Kapitola sa zameriava predovšetkým na algoritmy, ktoré počas učenia využívajú tú istú stratégiu rozhodovania, akú sa snažia optimalizovať (on-policy algoritmy). Ďalej je popísané hĺbkové učenie posilňovaním, ktoré využíva neurónové siete na aproximáciu stratégie aj hodnotovej funkcie. Podrobne sa rozoberajú metódy založené na optimalizácii rozhodovacích stratégií pomocou gradientných metód a taktiež sa podrobne analyzuje algoritmus PPO, ktorý patrí medzi najefektívnejšie súčasné RL metódy a zároveň sa používa aj v praktickej časti tejto práce.

Ďalšia časť kapitoly sa venuje popisu simulačných a tréningových nástrojov, ktoré boli využité na realizáciu experimentov s učením posilňovaním. Sú predstavené najpoužívanejšie fyzikálne simulačné prostredia v oblasti robotiky, ako PyBullet, MuJoCo a Gazebo, s dôrazom na ich vlastnosti, výhody a rozdiely.

V poslednej časti sa kapitola zameriava na knižnice Stable-Baselines3 (SB3) a Gymnasium, ktoré poskytujú funkcionalitu potrebnú na implementáciu a trénovanie agentov. Pozornosť sa venuje aj tvorbe vlastných prostredí a ich integrácii so simulačnými nástrojmi.

2.1. Učenie posilňovaním

RL umožňuje naučiť sa funkciu stratégie, teda spôsob, akým agent vyberá akcie na základe aktuálneho stavu[1]. Tento proces je znázornený na obrázku 2.1, kde agent iteratívne interaguje s prostredím.



Obrázok 2.1: Schéma interakcie medzi agentom a prostredím v RL[3]

Existuje mnoho rôznych prístupov k RL. Táto práca sa zameriava na on-policy a actor-critic algoritmus, ktorý je využívaný na trénovanie agenta pre požadované správanie.

On-policy učenie znamená, že počas procesu učenia agent používa tú istú stratégiu, ktorú sa snaží naučiť[2]. To znamená, že učenie začína s náhodne inicializovanou stratégiou a v každej ďalšej iterácii využíva aktualizovanú stratégiu z predchádzajúceho kroku.

Algoritmus actor-critic zahŕňa aktéra, ktorý rozhoduje, akú akciu vykonať, čo je iný názov pre funkciu stratégie a kritika, ktorý agentovi hlási, aké dobré bolo jeho rozhodnutie[2].

Hodnotiaca zložka, nazývaná aj kritik, využíva vlastnú funkciu, ktorá slúži na odhadovanie hodnoty aktuálneho stavu. Táto funkcia vyjadruje očakávaný súčet budúcich odmien, ktoré agent môže získať, ak sa bude zo súčasného stavu riadiť svojou aktuálnou stratégiou. Inak povedané, hodnotová funkcia vyjadruje, ako výhodné je nachádzať sa v danom stave z pohľadu dlhodobých prínosov.

Pri výpočte tejto hodnoty sa zohľadňujú nielen okamžité odmeny, ale aj všetky budúce odmeny, ktoré môže agent dosiahnuť, pričom vzdialenejšie odmeny majú menší význam ako tie, ktoré sú bližšie v čase. Tento princíp sa označuje ako diskontovanie budúcich odmien[2]. Hlavným cieľom učenia posilňovaním je tak nastaviť rozhodovaciu stratégiu agenta, aby maximalizoval súčet týchto diskontovaných odmien v dlhodobom horizonte.

Jednou z podoblastí RL, ktorá rieši optimalizačný problém, je hĺbkové učenie posilňovaním (DRL)[4]. Táto podoblasť predstavuje pokročilý prístup k učeniu posilňovaním, ktorý umožňuje efektívne aproximovať funkciu stratégie aj hodnotovú funkciu pomocou neurónových sietí. Tento prístup sa využíva na riešenie zložitých úloh, kde tradičné metódy už nie sú dostatočné. Aproximačná stratégia v DRL je definovaná svojimi parametrami, ktoré predstavujú váhy neurónovej siete. Tieto váhy slúžia na modelovanie rozhodovacích stratégií agenta, pričom sa iteratívne aktualizujú počas procesu učenia tak, aby agent optimalizoval svoje správanie na základe spätnej väzby z prostredia. Výsledkom je schopnosť neurónovej siete generovať stratégiu, ktorá umožňuje agentovi prijímať optimálne rozhodnutia aj v komplexných a dynamických prostrediach. Jednou z metód na trénovanie stratégie neurónovej siete sú metódy policy gradient.

2.2. Algoritmus Proximal Policy Optimization

Metódy policy gradient nám umožňujú učiť sa parametre stratégie alebo v terminológii neurónových sietí váhy pomocou algoritmu gradientového zostupu[5]. Proces učenia je iteratívny. Pri prvej iterácii sú váhy neurónovej siete inicializované náhodne. Následne používame sieť ako stratégiu v interakčnej slučke medzi agentom a prostredím na zhromaždenie nových dát. Tento prístup sa nazýva on-policy učenie, na rozdiel od off-policy učenia, kde akcie nie sú vyberané stratégiou, ktorá sa trénuje, ale inou nezávislou funkciou. Po každej epizóde alebo skôr po dávke epizód, sa váhy neurónovej siete aktualizujú pomocou algoritmu spätného šírenia chyby. Na základe

diskontovaného súčtu odmien na epizódu vypočítavame gradienty, ktoré zvyšujú pravdepodobnosť akcií vedúcich k pozitívnym odmenám a znižujú pravdepodobnosť akcií vedúcich k negatívnym odmenám.

Tento základný prístup funguje dobre v jednoduchších úlohách. V praxi však môže viesť k nestabilnému učeniu, pretože ak sa stratégia medzi iteráciami príliš rýchlo mení, agent môže stratiť užitočné stratégie, ktoré predtým objavil. Veľké aktualizácie parametrov môžu dokonca spôsobiť prudké zhoršenie výkonu. Aby sa tento problém zmiernil, bol vyvinutý algoritmus PPO[5]. Tento algoritmus zavádza mechanizmus na obmedzenie zmien stratégie v jednotlivých aktualizáciách, čím zabezpečuje stabilnejší a efektívnejší proces učenia.

PPO je jeden z algoritmov RL, ktorý využíva metódy policy gradientu na trénovanie agentov, čiže ide o policy-based učenie. Tento algoritmus rieši problém veľkých aktualizácií stratégie, ktoré často spôsobujú nestabilitu pri tradičných prístupoch policy gradientu. PPO navrhuje novú náhradnú cieľovú funkciu, ktorá obmedzuje rozsah zmien stratégie a zlepšuje stabilitu učenia[5]. Cieľová funkcia s ohraničením zmien stratégie v PPO je definovaná rovnicou 2.1.

$$L_t^{CLIP}(\theta) = \mathbb{E}_t\left[\min(\frac{\pi_\theta(a_t|o_t)}{\pi_{\theta old}(a_t|o_t)}A_t, clip(\frac{\pi_\theta(a_t|o_t)}{\pi_{\theta old}(a_t|o_t)}, 1-\epsilon, 1+\epsilon)A_t)\right] \quad (2.1)$$

Kde $\pi_\theta(a_t|o_t)$ je aktuálna stratégia (pravdepodobnosť výberu akcie a_t v stave o_t), $\pi_{\theta old}(a_t|o_t)$ je stratégia z predchádzajúcej iterácie, A_t je funkcia výhody, ktorá hodnotí kvalitu vykonanej akcie v porovnaní s očakávaním, ϵ je hyperparameter, ktorý určuje, ako veľmi sa môže nová stratégia odchýliť od starej a $\mathbb{E}_t$ je očakávanie (priemer) nad všetkými možnými pozorovaniami a akciami v čase t. Táto funkcia zavádza mechanizmus, ktorý obmedzuje rozsah zmien v pomere stratégie $\frac{\pi_\theta}{\pi_{\theta old}}$. Mechanizmus výberu minima zabezpečuje, že algoritmus pri aktualizácii stratégie vždy uprednostní tú hodnotu, ktorá vedie k menšej zmene očakávaného zisku[5]. V praxi to znamená, že ak by nová stratégia navrhovala výrazné zlepšenie, ale zároveň predstavovala riziko destabilizácie správania, algoritmus túto zmenu automaticky obmedzí.

Celková optimalizačná funkcia definovaná rovnicou 2.2 v algoritme PPO kombinuje funkciu s ohraničením zmien stratégie, chybu hodnotovej funkcie a entropický bonus[5].

$$L_t^{PPO}(\theta) = \mathbb{E}_t[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)] \quad (2.2)$$

Kde $L_t^{CLIP}(\theta)$ je funkcia s ohraničením zmien stratégie, ktorá stabilizuje zmeny stratégie, $c_1 L_t^{VF}(\theta)$ je chyba hodnotovej funkcie vypočítaná ako štvorcová chyba medzi predikovanou a skutočnou hodnotou, $S[\pi_\theta](s_t)$ je entropický bonus, ktorý podporuje prieskum prostredia tým, že udržuje stratégiu dostatočne stochastickú, koeficienty c_1 a c_2 vyvažujú príspevky jednotlivých častí.

2.3. Porovnanie simulačných prostredí

Simulačné prostredia zohrávajú kľúčovú úlohu pri vývoji robotických systémov a implementácii aplikácií RL, pretože umožňujú testovanie algoritmov a modelov v bezpečnom a kontrolovanom prostredí pred ich nasadením do reálneho sveta. Medzi najpoužívanejšie prostredia na tento účel patria MuJoCo, PyBullet a Gazebo, ktoré si získali široké uplatnenie v akademickej aj priemyselnej praxi. Každý z nich má jedinečné výhody a obmedzenia, ktoré ich predurčujú na rôzne typy aplikácií.

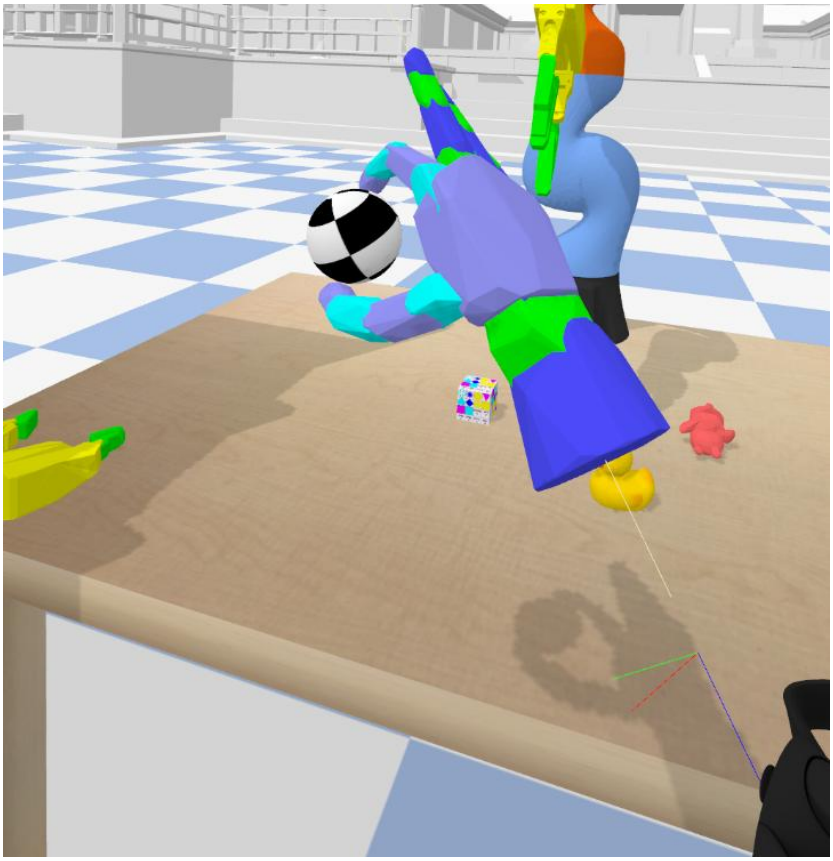
2.3.1. PyBullet

PyBullet je open-source simulačné prostredie založené na známej knižnici Bullet Physics Engine[6]. Pôvodne bol Bullet Physics Engine vyvinutý pre herný priemysel, kde sa používal na realistickú simuláciu fyzikálnych interakcií. Postupom času však našiel uplatnenie vo vedeckom výskume, inžinierstve a predovšetkým v robotike. PyBullet, ako rozšírenie tohto enginu, ponúka jednoduché a intuitívne rozhranie v jazyku Python, ktoré je ideálne pre vývojárov, výskumníkov a študentov. Umožňuje simuláciu dynamiky tuhých a mäkkých telies, čo ho robí neoceniteľným nástrojom pri návrhu a testovaní robotických systémov, ako je aj dvojnohý robot PAWO.

Jednou z hlavných výhod PyBulletu je jeho flexibilita a schopnosť integrovať sa s rôznymi nástrojmi a formátmi. Podporuje import robotických modelov vo formátoch, ako napríklad URDF, Simulation Description Format (SDF) alebo v MuJoCo formáte (MJCF) [6]. Vďaka tomu je možné jednoducho preniesť existujúce robotické modely do simulačného prostredia a overiť ich funkčnosť. Ďalšou výhodou je, že tvary a viacnásobné modely telies môžu byť definované nielen v externých rozšíriteľných značkovacích jazykoch (XML), ale aj priamo pomocou funkcií PyBullet[6]. PyBullet taktiež umožňuje simuláciu doprednej dynamiky, inverzný výpočet dynamiky, doprednú a inverznú kinematiku, výpočty kolízií, trenia a pružnosti, čo umožňuje detailné testovanie správania robotov v rôznych podmienkach[6].

Dôležitou súčasťou PyBulletu je jeho vstavaná podpora 3D vizualizácie, ktorá umožňuje sledovať simulácie v reálnom čase. Táto funkcia využíva OpenGL alebo TinyRendere, čo zabezpečuje kvalitné grafické zobrazenie prostredia a objektov. Používateľ tak môže vizuálne kontrolovať pohyb robotov, kolízie a ich interakciu s prostredím. Ďalej ponúka aj debugovacie nástroje, ako napríklad zobrazovanie textu v scéne a čiar, ktoré sa využívajú pri aplikácii externých síl pôsobiacich na modely[6].

PyBullet ako fyzikálny simulátor ponúka viacero režimov fungovania. Prvým z nich je režim GUI, v ktorom sa zobrazuje celá scéna simulácie. Tento režim je ideálny na vizuálnu kontrolu a analýzu pohybu alebo správania modelov, tento režim je zobrazený na obrázku 2.2[6]. Ďalším režimom je DIRECT, v ktorom sa scéna nevykresľuje, čo umožňuje simulácii prebiehať rýchlejšie[6]. Tento režim je obzvlášť vhodný pre aplikácie RL, kde nie je potrebné vizuálne znázornenie scény, ale zameriavame sa na samotné spracovanie a výsledky simulácie. Okrem týchto režimov existujú aj ďalšie, napríklad režimy, kde sú simulácia a vykresľovanie oddelené a prebiehajú na samostatných procesoch.



Obrázok 2.2: Simulačné prostredie PyBullet[6]

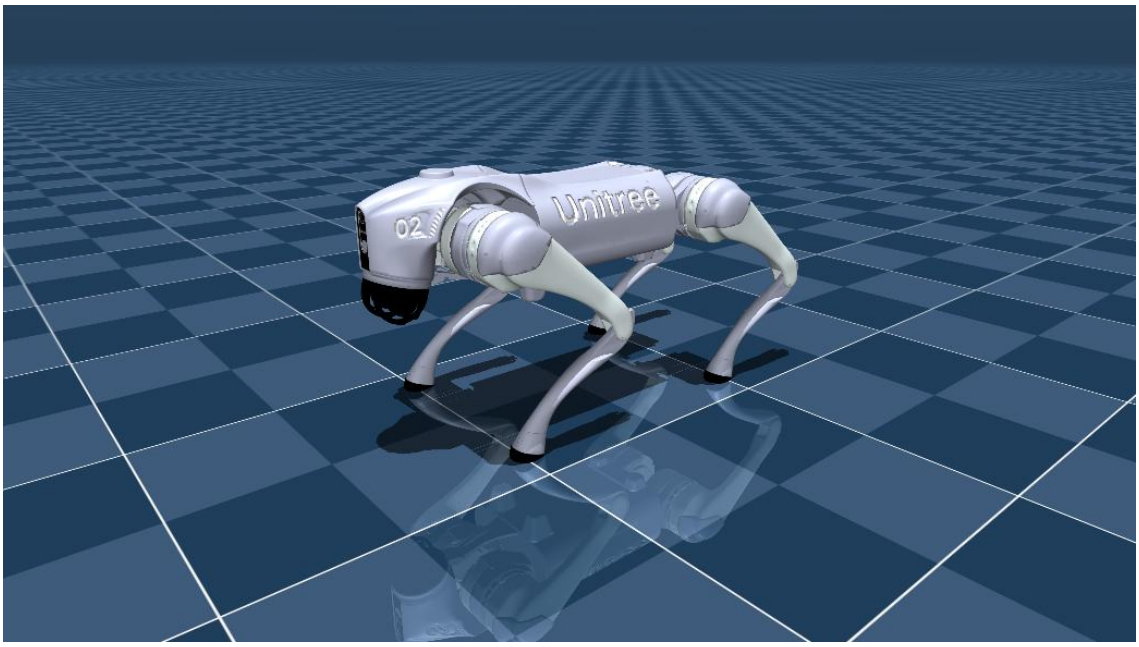
2.3.2. MuJoCo

MuJoCo (Multi-Joint dynamics with Contact) je open-source simulačné prostredie špecializovaný na robotiku, biomechaniku, animáciu a strojové učenie, vlastnený a spravovaný spoločnosťou Google DeepMind[7]. Jeho silnou stránkou je simulácia dynamiky tuhých telies, podpora rôznych typov kĺbov, detekcia kolízií a pohybové ovládanie. Vývojári MuJoCo preukázali, že toto prostredie dosahuje najlepšie výsledky v simuláciách zahŕňajúcich objekty s mnohými kĺbmi alebo prepojenými prvkami[7]. MuJoCo vyniká vysokým pomerom simulácie k reálnemu času, čo ho robí výkonným nástrojom pre rozsiahle simulácie. Tento výkon je dosiahnutý za cenu mierne nižšej presnosti, ktorá však nie je kritická pre mnohé RL aplikácie.

MuJoCo má interaktívne grafické rozhranie (GUI), ktoré umožňuje vizualizáciu simulácií, ale jeho možnosti pokročilého renderovania, ako je komplexné osvetlenie, sú obmedzené[7]. Vizualizácia prostredia je zobrazená na obrázku 2.3.

Obsah modelov a prostredí je možné definovať a upravovať pomocou XML súborov. Pre náročnejšie modely môže byť potrebné použiť externé nástroje na 3D modelovanie, ktoré umožňujú exportovať MJCF alebo URDF modely[7].

MuJoCo umožňuje užívateľom rozhodovať, ktoré časti výpočtového procesu simulácie budú spustené[7]. To znamená, že je možné izolovať konkrétne výpočty, ak je potrebné len simulovať dynamiku kĺbov bez výpočtu kolízií alebo simulácie pohybu.

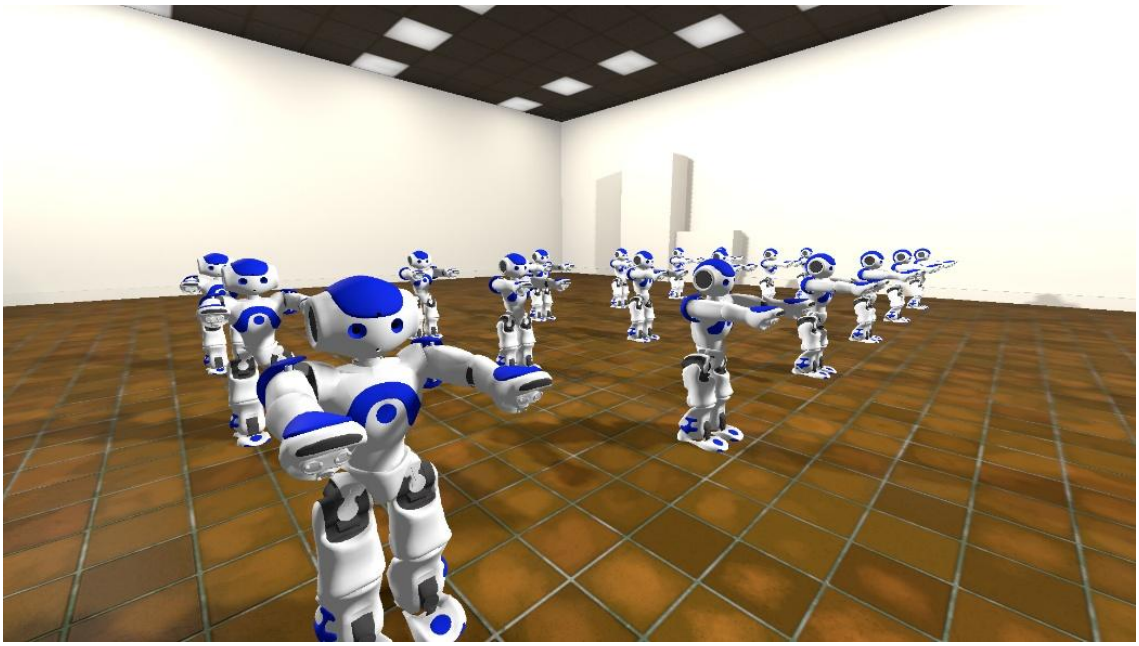


Obrázok 2.3: Simulačné prostredie MuJoCo[8]

2.3.3. Gazebo

Gazebo je pokročilé simulačné prostredie určené na testovanie a vývoj robotických systémov v realistických fyzikálnych a vizuálnych podmienkach. Je široko používané v akademickom aj priemyselnom prostredí, najmä v kombinácii s robotickým operačným systémom (ROS), s ktorým je úzko integrované[9]. Gazebo umožňuje simulovať komplexné interakcie medzi robotmi a prostredím, pričom podporuje pokročilú fyziku, vizualizáciu, systémy s viacerými robotmi a široké spektrum senzorov, ako kamery, lidar, IMU (Inertial Measurement Unit) či GPS (Global Position System)[9]. Simulácia prostredia s viacerými robotmi je zobrazená na obrázku 2.4.

Na rozdiel od PyBullet a MuJoCo poskytuje Gazebo natívny editor modelov, ktorý umožňuje používateľom vytvárať a upravovať jednoduché modely priamo v grafickom používateľskom rozhraní GUI bez potreby ručnej editácie XML alebo URDF súborov[9]. Tento editor podporuje intuitívne vkladanie, presúvanie a prepojenie objektov, ako aj definovanie kĺbov, kolíznych prvkov či vizuálnych vlastností modelu[9]. Vďaka tejto funkcionalite je Gazebo obzvlášť vhodné pre rýchle prototypovanie a testovanie mechanických štruktúr, čo výrazne zjednodušuje proces návrhu robotických systémov najmä pre používateľov bez pokročilých znalostí programovania.



Obrázok 2.4: Simulačné prostredie Gazebo[10]

Každé z prostredí má svoje silné a slabé stránky. Mujoco vyniká vysokým výkonom, presnosťou a stabilitou, a je ideálny pre simulácie pohybu a robotiky. PyBullet je vhodný pre menej náročné úlohy, kde vyniká svojou jednoduchosťou a je medzi simulačnými nástrojmi najrozšírenejší, a má okolo seba širokú komunitu. Gazebo ponúka rozsiahlu podporu senzorov a silnú integráciu s ROS, čo z neho robí ideálny nástroj pre simulácie komplexných robotických systémov, no je náročnejší na výpočtový výkon a jeho použitie môže byť komplikovanejšie pre začiatočníkov.

2.4. Stable-Baselines3

SB3 je open-source knižnica v jazyku Python určená na implementáciu a tréning RL algoritmov. Táto knižnica, postavená na frameworku PyTorch, poskytuje stabilnú, flexibilnú a výkonnú platformu pre vývoj algoritmov umelej inteligencie[11]. SB3 je pokračovaním projektu Stable-Baselines, pričom jeho modernizovaná architektúra zjednodušuje ladenie a rozširovanie algoritmov[11]. Je široko používaná vo výskume, pri vývoji robotických systémov, autonómnych aplikáciách alebo pri hrách.

SB3 ponúka implementácie rôznych RL algoritmov, vrátane PPO, Deep Q-Networks (DQN) alebo Soft Actor-Critic (SAC)[11]. Knižnica je navrhnutá tak, aby bola ľahko použiteľná aj pre začiatočníkov, pričom umožňuje jednoduchú integráciu s platformami ako Gymnasium alebo PyBullet. Medzi jej hlavné výhody patrí podpora paralelného tréningu, modulárny dizajn a optimalizácia pre škálovateľnosť, čo z nej robí robustný nástroj pre rýchle experimentovanie s rôznymi RL modelmi[11].

Porovnanie SB3 s inými RL knižnicami ukazuje, že SB3 vyniká svojou jednoduchou použiteľnosťou, kvalitnou dokumentáciou a spoľahlivosťou. Hoci SB3 priorizuje stabilitu nad rýchlym pridávaním nových funkcií, poskytuje robustné prostredie pre výskum a vývoj v oblasti RL.

2.5. Gymnasium

OpenAI Gym je nástroj, ktorý poskytuje rozhranie pre tvorbu nových úloh alebo využitie už existujúcich úloh implementovaných v rámci Gym[12]. Tento nástroj vyvinula organizácia OpenAI, ktorá sa špecializuje na vývoj umelej inteligencie. Gym ponúka širokú škálu predpripravených prostredí, na ktoré je možné navrhnúť algoritmy na riešenie konkrétnych úloh[12].

Gymnasium vzniklo na základe OpenAI Gym, ktoré sa prestalo vyvíjať kvôli zameraní spoločností na iné projekty[12]. Gymnasium vzniklo ako komunitná iniciatíva na podporu aktívneho vývoja, aby zabezpečilo opravy chýb, kompatibilitu s novými verziami Pythonu a knižníc, ako napríklad SB3[12]. Je kompatibilný s Gym API a je navrhnutý tak, aby sa dal ľahko integrovať do existujúcich projektov založených na OpenAI Gym. Pridáva viac prostredí a lepšiu podporu pre simulácie, zavádza flexibilnejšie API, ktoré umožňuje presnejšie kontrolovať správanie prostredí, rozširuje metriky na sledovanie a analýzu výkonu algoritmov.

Gymnasium umožňuje okrem iného aj tvorbu vlastných prostredí[12]. Každé prostredie pozostáva zo sady neurčitých úkonov, ktoré je potrebné definovať. Jednou z týchto funkcií je inicializačná funkcia, kde prebieha inicializácia veľkosti akcií a samotného prostredia. Súčasťou tejto inicializácie môže byť aj nastavenie fyzikálneho modelu alebo iných parametrov. Funkcia kroku sa používa počas trénovania na posun prostredia, pričom ako parameter prijíma akciu, ktorá má byť vykonaná. Resetovacia funkcia sa volá pred spustením trénovania a po skončení každej epizódy, aby trénovanie vždy začínalo vo východiskovom stave. Okrem týchto základných funkcií je možné využívať aj voliteľné funkcie, ako napríklad funkciu, ktorá poskytuje vizuálnu reprezentáciu aktuálneho stavu prostredia.

3 Prehľad vybraných dvojnohých robotov

Táto kapitola sa venuje prehľadu vybraných chodiacich humanoidných robotov, ktoré zohrávajú významnú úlohu v oblasti výskumu, vývoja a praktických aplikácií modernej robotiky. Cieľom je predstaviť rôzne typy robotických platforiem, od jednoduchších experimentálnych modelov vytvorených na akademickej pôde až po technologicky vyspelé a komerčne dostupné roboty známych svetových spoločností. V texte sú jednotlivé platformy analyzované z viacerých hľadísk, vrátane ich mechanickej konštrukcie, maximálnych dosiahnuteľných limít motorov, senzorického a výpočtového vybavenia, ako aj použitých riadiacich algoritmov. Osobitná pozornosť sa venuje aj oblastiam praktického využitia každého z robotov, ako sú priemyselné úlohy, záchranné misie či interakcia s ľuďmi. Porovnanie týchto robotických platforiem poskytuje dôležitý kontext pre pochopenie ich výhod, nevýhod a technických obmedzení, ktoré je potrebné zohľadniť pri návrhu vlastného chodiaceho robota v praktickej časti tejto práce.

3.1. PAWO

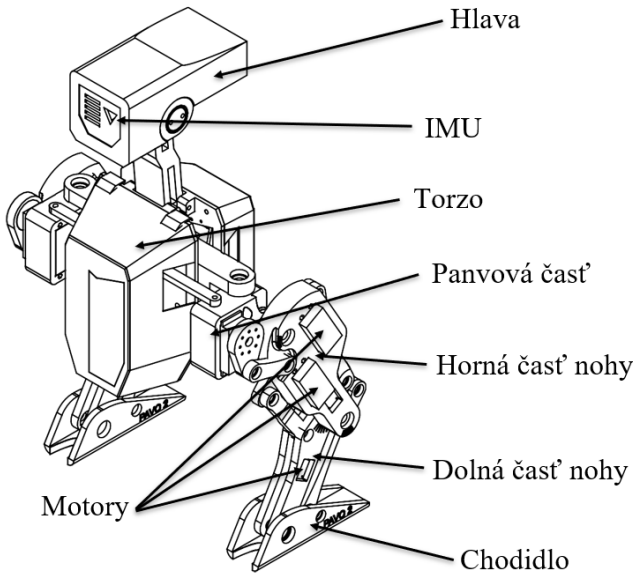
PAWO je malý, no výkonný humanoidný robot, navrhnutý predovšetkým pre účely výskumu v oblasti riadenia chôdze, rovnováhy a strojového učenia. Vďaka svojim kompaktným rozmerom a jednoduchej, no efektívnej mechanickej štruktúre sa stal vhodnou platformou pre experimentovanie s pokročilými algoritmami riadenia chôdze. Reálny robot PAWO je zobrazený na obrázku 3.1, kde sú zreteľne viditeľné hlavné časti tela robota, ktoré sú detailne popísané na obrázkoch 3.2 a 3.3.

Obrázok 3.1: Robot PAWO

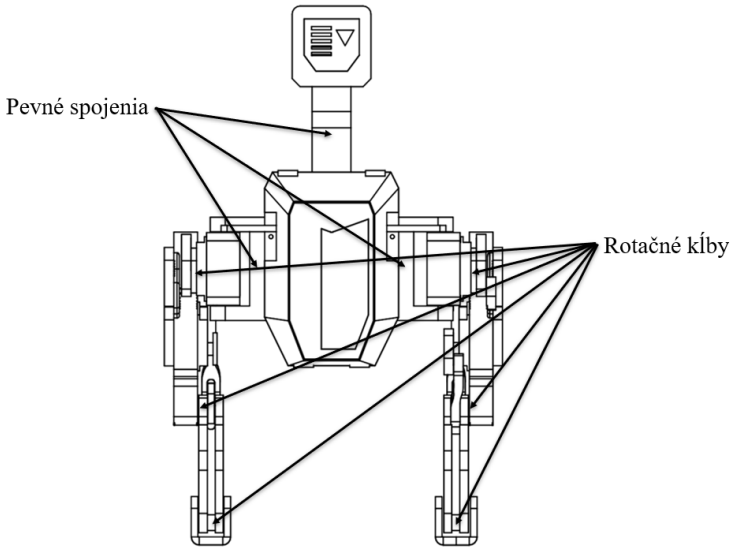
3 Prehľad vybraných dvojnohých robotov

PAWO má výšku 65 cm v úplne vystretom postoji a 55 cm v skrčenom postoji, pričom jeho hmotnosť je len 3,4 kg, čo výrazne uľahčuje manipuláciu a testovanie v laboratórnych podmienkach. Robot je vybavený šiestimi rotačnými kĺbmi, pričom každý z nich sa pohybuje len v jednej osi, čím je systém jednoducho riaditeľný a výborne modelovateľný. Všetky kĺby sú poháňané elektrickými pohonmi s maximálnym krútiacim momentom 4,9 Nm a uhlovou rýchlosťou 6,16 rad/s.

PAWO implementuje metódu PPO na učenie chôdze. Tento prístup umožňuje robotovi učiť sa chodiť autonómne v simulovanom prostredí a následne prenášať naučené správanie do reálneho sveta. Tréningový proces je založený na spätnej väzbe zo senzorov uložených v hlave robota, ktoré poskytujú informácie o orientácii, pohybe a prípadných odchýlkach od požadovanej trajektórie.



Obrázok 3.2: Popis častí tela robota PAWO



Obrázok 3.3: Popis kĺbov robota PAWO

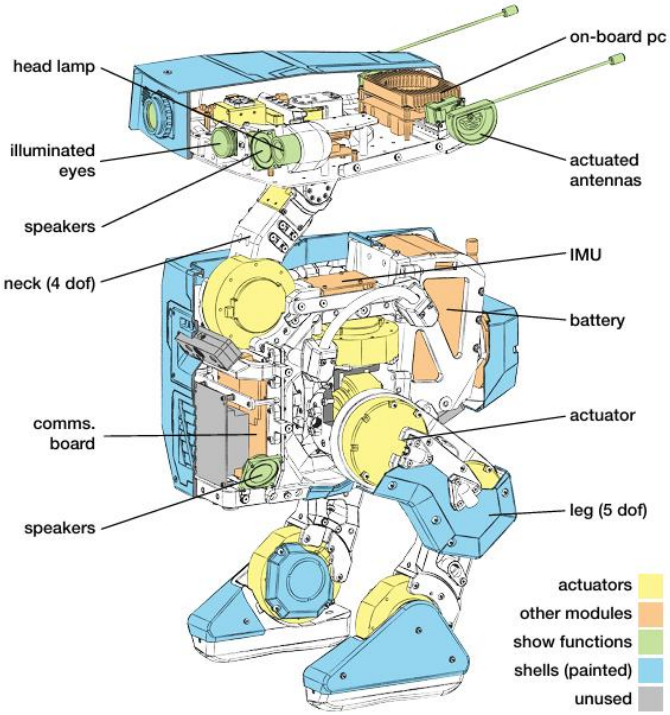
3.2. BD-1

BD-1 je pokročilý chodiaci robot, vyvinutý spoločnosťou Disney Research a Walt Disney Imagineering R&D, ktorý kombinuje umeleckú animáciu s pokročilými technológiami v robotike[13]. Pôvodne sa objavil ako fiktívny droid v hre od Star Wars, no Disney vytvorilo aj jeho fyzickú verziu, určenú na interaktívnu šou v zábavných parkoch. Tento robot je špecifický svojím dizajnom orientovaným na charakterové správanie, čím sa odlišuje od tradičných humanoidných robotov vyvíjaných na priemyselné či výskumné účely. Jeho podoba je vyobrazená na obrázku 3.4 a popis častí tela na obrázku 3.5.

Obrázok 3.4: Robot BD-1[13]

Podľa [13] má robot celkovú hmotnosť 15,4 kg, pričom torzo váži 5,8 kg, krk a hlava 2,4 kg a každá noha 3,6 kg. Je vysoký 66 cm bez započítania antén a nohy majú nominálnu dĺžku 28 cm, pričom v plnom natiahnutí dosahujú 34 cm. Aktuátory bedrového kĺbu a kolena patria medzi najsilnejšie, s maximálnym krútiacim momentom 34 Nm a maximálnou uhlovou rýchlosťou 20 rad/s. Rotácia bedra, členku a dolného krčného kĺbu disponujú maximálnym momentom 24 Nm a rýchlosťou až 30 rad/s. Tieto dve skupiny sú vybavené kvázi-priamymi pohonmi, ktoré umožňujú vysokopásmové riadenie momentu v otvorenej slučke, čo je vhodné pre dynamický pohyb. Tri aktuátory umiestnené v hlave majú vysoký prevodový pomer, pričom dosahujú maximálny krútiaci moment 4,8 Nm a uhlovú rýchlosť 6,3 rad/s[13].

BD-1 využíva RL metódu PPO na tréning pohybu[13]. Tréningové prostredie zahŕňa imitáciu umeleckých animácií cez optimalizované stratégie, ktoré sú schopné robustne reprodukovať pohyby aj za prítomnosti vonkajších porúch. Rozdelenie pohybov na perzistentné (plynulé udržiavanie rovnováhy a postojov), periodické (chôdza s fázovým signálom) a epizodické (krátke animácie, ako kývanie alebo tancovanie) umožňuje dynamické prepínanie medzi pohybmi podľa potreby scénického výstupu[13]. Odmenová funkcia pozostáva z presnosti imitácie pohybu, stability a rovnováhy, energickej efektivity a plynulosti pohybov.



Obrázok 3.5: Popis častí tela robota BD-1[13]

Počas prevádzky interaguje s okolím pomocou svetelných efektov očí a zvukového výstupu, čo z neho robí atraktívnu postavu pre publikum. Taktiež využíva IMU na meranie zrýchlenia, rotácie a náklonu[13]. Na rozdiel od väčšiny chodiacich robotov, ktoré sú optimalizované na efektivitu chôdze, BD-1 využíva umelecky prispôsobené trajektórie pohybu, ktoré sú výsledkom simulácií v softvérových nástrojoch na animáciu.

3.3. Atlas

Atlas, vyvinutý spoločnosťou Boston Dynamics, patrí medzi najpokročilejšie humanoidné roboty súčasnosti, je navrhnutý pre dynamické a vysoko náročné pohybové úlohy, ako je parkúr, beh, skákanie či balansovanie na jednej nohe, pričom využíva najnovšie poznatky v oblasti robotiky, biomechaniky a umelej inteligencie[14]. Robot Atlas je zobrazený na obrázku 3.6.

Podľa [14] s výškou približne 1,5 m a hmotnosťou 89 kg má Atlas proporcie podobné človeku a disponuje 28 stupňami voľnosti, ktoré mu umožňujú vysoko flexibilný

a adaptívny pohyb. Pohon robota zabezpečuje plne hydraulický systém, ktorý umožňuje extrémne výkonné, rýchle a silné pohyby. Najvýkonnejšie kĺby, ako bedrá a kolená, sú schopné vyvinúť krútiace momenty takmer 200 Nm pri uhlových rýchlostiach nad 12 rad/s v kolenných kĺboch. Vďaka tomu dokáže Atlas vykonávať explozívne pohyby, ako napríklad salto vzad, ktoré sú zatiaľ v humanoidnej robotike bezprecedentné[14].

Obrázok 3.6: Robot Atlas[14]

Atlas využíva pokročilé algoritmy na riadenie pohybu, medzi ktoré patrí predovšetkým Model Predictive Control (MPC) v kombinácii s dynamickým plánovaním trajektórií a stabilizačnými algoritmami, pričom v posledných verziách je navyše integrované strojové učenie, ktoré sa využíva najmä pri plánovaní pohybov v neštruktúrovanom prostredí[14].

Atlas je vybavený sústavou senzorov, ktoré zabezpečujú priestorové vnímanie, ide o lidar a stereo kamery, ktoré umožňujú 3D mapovanie okolia a detekciu prekážok, IMU sleduje náklon, zrýchlenie a rotáciu robota, kĺbové snímače a tlakové senzory v nohách poskytujú spätnú väzbu o polohe, rýchlosti a kontakte s povrchom[14].

Hoci nie je určený na komerčné účely, Atlas slúži ako výskumná platforma na testovanie schopností robotov v extrémnych podmienkach. Jeho vývoj výrazne prispieva k posunu v oblasti dynamickej humanoidnej robotiky a ukazuje, čo všetko je dnes technicky možné, pokiaľ ide o rovnováhu, silu, koordináciu a adaptabilitu v pohybe.

3.4. Optimus

Optimus, známy aj ako Tesla Bot, je humanoidný robot vyvíjaný spoločnosťou Tesla, Inc. s cieľom vykonávať všeobecné fyzické úlohy v domácnostiach, továrňach a iných ľudských prostrediach[16]. Prvý prototyp, ktorý je zobrazený na obrázku 3.7, bol predstavený v roku 2022 a od tej doby prešiel viacerými technickými vylepšeniami.

Podľa [16] s výškou približne 173 cm a hmotnosťou 56 kg má proporcie podobné človeku a je navrhnutý tak, aby mohol bezpečne spolupracovať s ľuďmi v reálnom svete. Optimus používa plne elektrické pohony, každý kĺb je vybavený vlastným elektromotorom a senzorikou, čo mu umožňuje presnú plynulú manipuláciu a pohyb. Spoločnosť Tesla vyvinula vlastné aktuátory a prevodové mechanizmy, optimalizované pre efektivitu a modularitu. Napríklad, bedrové kĺby môžu generovať krútiaci moment až 300 Nm, kolená 150–200 Nm a maximálne rýchlosti kĺbov sa pohybujú medzi 2-6 rad/s, v závislosti od konkrétneho kĺbu[16].

Obrázok 3.7: Robot Optimus[15]

Robot využíva rovnakú umelú inteligenciu a neurónové siete, ktoré Tesla používa vo svojich autonómnych vozidlách. Je schopný učiť sa úlohy pomocou videozáznamov a simulácií, svoje správanie optimalizuje pomocou strojového učenia, využíva pokročilé vizuálne vnímanie, má kamery a senzory zabudované v hlave, ktoré mu umožňujú rozpoznávať objekty, orientovať sa v priestore a bezpečne sa pohybovať v meniacom sa prostredí[16].

Cieľom Optima je stať sa univerzálnym robotickým pomocníkom, ktorý zvládne opakujúce sa alebo fyzicky náročné úlohy. Hoci je stále vo fáze vývoja, predstavuje významný posun smerom k reálne využiteľným humanoidným robotom v každodennom živote.

3.5. ASIMO

ASIMO je humanoidný robot, ktorý vyvinula spoločnosť Honda a ktorý sa stal jedným z najikonickejších robotov začiatku 21. storočia[18]. Jeho vývoj sa začal už v 80. rokoch, pričom verejne bol predstavený v roku 2000. Počas svojej aktívnej služby až do roku 2022 sa ASIMO stal symbolom pokroku v oblasti humanoidnej robotiky, predovšetkým vďaka svojej schopnosti chodiť, behať, stúpať po schodoch a interagovať s ľuďmi. Robot ASIMO je zobrazený na obrázku 3.8.

Podľa [18] má ASIMO výšku približne 130 cm a hmotnosť 48 kg. Využíva plne elektrický pohon, pričom každý z jeho viac ako 30 kĺbov je ovládaný servomotormi. Kľúčové pohyblivé časti, ako kolená a bedrá dosahujú maximálne uhlové rýchlosti okolo 6-8 rad/s a môžu vyvinúť krútiace momenty do 80-100 Nm, čo postačuje na dynamické pohyby ako rýchla chôdza (až 9 km/h) či stabilné udržanie rovnováhy pri zmenách terénu[18].

Obrázok 3.8: Robot ASIMO[17]

ASIMO používa pri chôdzi Zero Moment Point (ZMP) kontrolu, ide o algoritmus, ktorý zabezpečuje dynamickú rovnováhu robota tým, že sleduje rozloženie hmotnosti a udržuje ťažisko v stabilnej zóne[18]. Okrem pohybových schopností bol ASIMO vybavený aj kamerami, mikrofónmi a senzorickým systémom, ktorý mu umožňoval rozoznať tváre, gestá a hlasové povely[18]. Vďaka tomu vedel napríklad privítať návštevníkov, reagovať na otázky alebo priniesť predmety[18].

Aj keď bol vývoj ASIMA v roku 2022 oficiálne ukončený, jeho dedičstvo pretrváva[18]. Stal sa základom pre vývoj pokročilejších robotov a položil technologické základy v oblastiach ako rovnováha, koordinácia pohybov a ľudsko-robotická interakcia.

4 Tvorba modelu robota PAWO

Táto kapitola sa venuje procesu návrhu a implementácie digitálneho modelu chodiaceho robota PAWO určeného pre simuláciu v prostredí PyBullet. Postupne sú opísané jednotlivé fázy vývoja, od 3D modelovania základných častí robota v CAD (Computer-Aided Design) softvéri, cez export do STL (Standard Triangle Language) formátu, až po vytvorenie URDF súboru, ktorý definuje geometrickú a fyzikálnu štruktúru robota. V ďalších častiach je detailne rozobraná špecifikácia telies a kĺbov v URDF súbore vrátane ich vizuálnych, kolíznych a dynamických vlastností. Taktiež sa vytváral model jednoduchého robota v MuJoCo. Záver kapitoly predstavuje samotné porovnanie vytvorených modelov rôznym spôsobom.

4.1. Modelovanie základných častí a export do STL

Pri modelovaní robota bolo nevyhnutné správne navrhnúť jednotlivé časti tak, aby zabezpečovali jeho stabilitu, pohyb a celkovú funkčnosť. Medzi časti robota patria časti tela a nôh, pričom každá z týchto častí má špecifické požiadavky na rozmery.

Na začiatku procesu sa jednotlivé časti robota navrhovali vo vhodnom CAD programe, ktorý umožňuje presné modelovanie a podporuje export do formátu STL. Každá časť robota bola vymodelovaná samostatne podľa požiadaviek na proporcie a detaily. Na obrázku 4.1 je zobrazené chodidlo robota v STL formáte.

Po úspešnom vymodelovaní základných častí boli všetky komponenty exportované do formátu STL, ktorý je štandardom pre reprezentáciu 3D modelov. Tento formát je široko používaný nielen pri simuláciách, ale aj pri 3D tlači, pretože umožňuje presné zachovanie geometrie modelu pomocou trojuholníkových polygónov, ktoré definujú povrch objektu. Export do STL súborov bol vykonaný s dôrazom na zachovanie správnych mier a proporcií každého komponentu, aby sa zabezpečila kompatibilita jednotlivých častí robota pri ich neskoršom zostavovaní a simulácii v PyBullete.

STL súbory opisujú 3D model ako sieť trojuholníkov, kde každý trojuholník je definovaný tromi vrcholmi a normálovým vektorom, ktorý udáva smer povrchu trojuholníka. Táto jednoduchá štruktúra je ideálna na spracovanie v simulačných prostrediach, ako je PyBullet, pretože umožňuje rýchle a presné výpočty kolízií a fyzikálnych interakcií. PyBullet využíva túto sieť trojuholníkov na detekciu kontaktu medzi jednotlivými časťami robota a prostredím.

Presné modelovanie a export do STL súborov vytvorili základ pre definíciu modelu robota v URDF a MuJoCo formáte, kde sa jednotlivé časti spojili pomocou kĺbov a definovali sa ich fyzikálne vlastnosti.

Obrázok 4.1: Chodidlo robota PAWO v STL formáte

4.2. Tvorba modelu robota v URDF súbore

URDF je textový formát založený na XML, ktorý slúži na popis štruktúry robotov. Používa sa na definovanie geometrie, fyzikálnych vlastností a spojení jednotlivých častí robota. URDF je široko používaný v robotike, najmä v rámci simulačných nástrojov, ako sú PyBullet alebo Gazebo**.** Prostredníctvom URDF je možné jednoducho vytvárať modely robotov rôznej zložitosti, od jednoduchých manipulačných ramien až po dvojnohé roboty.

Každý robot popísaný v URDF pozostáva z telies a kĺbov, ktoré spájajú tieto telesá. Telesá predstavuje pevnú časť robota, zatiaľ čo kĺby umožňujú jeho pohyb. Prostredníctvom URDF je možné definovať aj ďalšie vlastnosti robota, ako sú hmotnosť jednotlivých častí, momenty zotrvačnosti, trenia alebo nastavenie pohybu kĺbov.

Štruktúra URDF súboru pozostáva z viacerých základných elementov, ktoré opisujú jednotlivé časti robota. Medzi základné elementy robota patrí hlavný element `<robot>`, definícia telies a definícia kĺbov. Ďalšia veľmi dôležitá vec pri tvorbe URDF súboru je hierarchia jednotlivých častí.

Každý URDF súbor začína hlavným elementom `<robot>`, ktorý zahŕňa všetky ostatné prvky, ako sú telesá a kĺby, a taktiež sa musí vhodne pomenovať. V tomto elemente sa nachádzajú podrobné popisy jednotlivých častí súboru.

4.2.1. Definícia telies robota v URDF súbore

Definícia telesa v URDF zahŕňa štyri hlavné kroky. Jedná sa o pomenovanie telesa, definovanie jeho vizuálnych vlastností, nastavenie fyzikálnych a kolíznych vlastností. Každý z týchto krokov je nevyhnutný na to, aby robot správne fungoval v simulačnom prostredí. Každé teleso robota sa definuje pomocou elementu `<link>`, ktorý sa vhodne pomenuje, čo slúži na jedinečnú identifikáciu konkrétnej časti. Tento názov je dôležitý, pretože sa naň neskôr odkazuje pri spájaní telies prostredníctvom kĺbov. V prípade ľavého chodidla robota PAWO bol použitý názov “leftfoot“, čo jednoznačne identifikuje túto časť v rámci celkového modelu, jeho definícia sa nachádza v tabuľke 4.1.Vizuálne vlastnosti sú špecifikované pomocou elementu `<visual>`, ktorý obsahuje informácie o polohe, geometrii a vzhľade telesa. Poloha vizuálneho modelu sa nastavuje pomocou elementu

<origin>, kde sa definujú hodnoty posunu a natočenia. V prípade ľavého chodidla robota PAWO boli hodnoty posunu a natočenia nulové, čo znamená, že vizuálny model je umiestnený presne v strede telesa bez akéhokoľvek posunu alebo rotácie. Geometria telesa je definovaná pomocou elementu <geometry>, do ktorého sa vkladá externý STL súbor obsahujúci trojrozmerný model ľavého chodidla. V elemente <mesh> sa určuje názov a cestu k tomuto STL súboru. V tomto prípade sa použil súbor "SteamDuck2Foot.stl", ktorý presne popisuje tvar ľavého chodidla. Ďalším dôležitým prvkom je element <material>, ktorý slúži na definovanie vzhľadu telesa. Tento element určuje farbu telesa pomocou RGBA (Red Green Blue Alpha) formátu a taktiež priehľadnosť.

Fyzikálne vlastnosti telesa sa definujú pomocou elementu <inertial>, ktorý obsahuje informácie o hmotnosti telesa a jeho momente zotrvačnosti. Hmotnosť telesa sa definuje pomocou elementu <mass>, kde sa špecifikuje hodnotu hmotnosti v kilogramoch. V tomto prípade bola zvolená hmotnosť 0,085 kg. Ďalším kľúčovým parametrom je moment zotrvačnosti, ktorý je definovaný pomocou elementu <inertia>, definujú sa tu hlavné momenty zotrvačnosti okolo osí x, y a z a produkty zotrvačnosti, ktoré zohľadňujú rotáciu telesa okolo kombinácie týchto osí. V prípade ľavého chodidla robota PAWO boli tieto hodnoty nastavené na hodnoty 1 pre hlavné momenty a 0 pre produkty zotrvačnosti, čo zabezpečuje zjednodušený, ale dostatočne presný výpočet pohybovej dynamiky robota počas simulácie.

Kolízne vlastnosti telesa sú definované pomocou elementu <collision>, ktorý určuje, ako sa teleso bude správať pri detekcii kolízií počas simulácie. Kolízne vlastnosti ovplyvňujú správnosť detekcie kolízií, reakciu robota na vonkajšie sily a jeho schopnosť pohybovať sa vo vytvorenom prostredí. Kolízny model je podobne ako vizuálny model definovaný geometricky pomocou elementov <geometry> a <origin>, tiež sa jeho geometria vytvára pomocou rovnakého STL súboru ako v prípade vizuálnych vlastností.

Použitie podrobného STL modelu ako kolízneho objektu zabezpečuje vysokú presnosť detekcie kolízií. V prípade potreby sa však často používajú jednoduchšie geometrické tvary, ako sú kvádre, valce alebo gule, ktoré znižujú výpočtovú náročnosť simulácie. Pre robota PAWO sa presnosť kolíznych vlastností považuje za dôležitú, pretože jeho pohyb je zameraný na chôdzu, kde správna detekcia kontaktov so zemou hrá kľúčovú úlohu.

Pri definovaní chodidiel robota PAWO je potreba správneho nastavenie kontaktu pre zabezpečenie stability pri chôdzi, čo sa pri ostatných častiach robota neberie v úvahu, pretože nie sú v kontakte so zemou pri žiadúcom správaní. Kontakt medzi chodidlami a povrchom sa špecifikuje pomocou elementu <contact>, ktorý obsahuje dôležité parametre, ako sú laterálne trenie, valivé trenie, otáčavé trenie a koeficient pružnosti nárazu.

Laterálne trenie sa nastavuje prostredníctvom elementu <lateral_friction>, ktorý určuje koeficient odporu pri pohybe chodidla po povrchu v horizontálnom smere. Pre robota PAWO bola zvolená hodnota 0,5, ktorá poskytuje dostatočné trenie pre danú situáciu. Ostatné parametre kontaktu sa nastavili nulové, pretože toto nastavenie najlepšie odráža realitu.

Tabuľka 4.1: Definícia ľavého chodidla robota PAWO v URDF súbore

```
<link name="leftfoot">
      <visual>
            <geometry>
                  <mesh filename="SteamDuck2Foot.stl"/>
            </geometry>
            <material name="White">
                  <color rgba=".99 .99 .99 1"/>
            </material>
      </visual>
      <inertial>
            <mass value=".085"/>
            <inertia ixx="1" iyy="1" izz="1"/>
      </inertial>
      <collision>
            <origin rpy="0 0 0" xyz="0 0 0"/>
            <geometry>
                  <mesh filename="SteamDuck2Foot.stl"/>
            </geometry>
      </collision>
      <contact>
            <lateral_friction value="0.5"/>
            <rolling_friction value="0.0"/>
            <spinning_friction value="0.0"/>
            <restitution value="0.0"/>
      </contact>
</link>
```

4.2.2. Definícia kĺbov v URDF súbore

Kĺby sú jedným z najdôležitejších prvkov pri tvorbe robotických modelov, pretože umožňujú spojenie jednotlivých častí robota a určujú ich pohybové možnosti. Kĺby sa používajú na spájanie dvoch telies, nadriadeného (rodičovského) telesa a podriadeného (detského) telesa, pričom definujú, ako sa tieto telesá môžu voči sebe pohybovať. Každý kĺb obsahuje informáciu o tom, ktoré teleso je nadriadené `<parent>` a ktoré teleso je podriadené `<child>`. Nadriadené teleso tvorí základ, voči ktorému sa podriadené teleso pohybuje. Tento vzťah umožňuje vytvoriť hierarchickú štruktúru robota, kde pohyb jednotlivých častí priamo závisí od pohybu ich rodičovských telies.

Podľa druhu pohybu, ktorý umožňujú, sa kĺby v URDF delia na štyri základné typy. Prvým a najčastejšie používaným typom je rotačný kĺb, ktorý umožňuje rotáciu okolo

jednej osi v definovanom rozsahu uhla. Tento typ kĺbu je potrebný pri modelovaní pohybov, ako napríklad ohýbanie členkov, kolien alebo bedier.

Ďalším typom je kontinuálny kĺb, ktorý umožňuje neobmedzenú rotáciu okolo jednej osi bez definovaného rozsahu uhla. Tento kĺb sa používa najmä pri častiach robota, ktoré sa môžu nepretržite otáčať, ako sú kolieska. Na rozdiel od rotačného kĺbu nemá obmedzenia pohybu, čo znamená, že rotácia môže prebiehať voľne v oboch smeroch.

Tretím typom je prizmatický kĺb, ktorý umožňuje lineárny posun pozdĺž jednej osi. Takýto kĺb sa využíva pri mechanizmoch, ktoré vyžadujú pohyb po priamke, napríklad pri teleskopických ramenách alebo lineárnych pohonoch. Tento typ kĺbu sa často používa na zvýšenie dosahu robotických ramien alebo na nastavovanie výšky robota.

Posledným typom je pevný kĺb, ktorý spája dve telesá bez možnosti pohybu. Používa sa tam, kde je potrebné vytvoriť pevné spojenie medzi časťami robota, ktoré sa nemajú voči sebe pohybovať, ako sú statické časti tela robota. Hoci pevné kĺby neumožňujú pohyb, sú nevyhnutné na zostavenie modelu a zabezpečenie jeho správnej štruktúry.

Robot PAWO je zložený iba z rotačných a fixných kĺbov, ktoré zabezpečujú jeho pohybové a pevné spoje. Rotačné kĺby spájajú chodidlá s časťami nôh a bedrami. Fixné kĺby spájajú časti robota, ktoré sa voči sebe nepohybujú, ako je spojenie torza s hlavou a bokmi robota.

Kĺb s názvom “joint1“ je definovaný ako rotačný kĺb, čo znamená, že umožňuje rotáciu podriadeného telesa voči nadriadenému telesu okolo jednej osi v obmedzenom rozsahu uhla, jeho popis sa nachádza v tabuľke 4.2. Tento kĺb spája dve telesá, ide o ľavé chodidlo ako nadriadené teleso a ľavú nohu ako podriadené teleso. Poloha kĺbu voči nadriadenému telesu je definovaná pomocou elementu `<origin>`. Element `<axis>` definuje os rotácie kĺbu, v prípade tohto kĺbu rotácia prebieha okolo osi x, k rotácii okolo osi y a osi z nedochádza. Element `<limit>` definuje rozsah pohybu kĺbu, maximálny moment a rýchlosť, taktiež je možné definovať tlmenie a trenie v kĺbe.

Tabuľka 4.2: Definícia rotačného kĺbu v URDF súbore

```
<joint name="joint1" type="revolute">
      <parent link="leftfoot"/>
      <child link="leftleg"/>
      <origin xyz="0 0 0"/>
      <axis xyz="1 0 0"/>
      <limit lower="-0.523" upper="0.523" effort="4.9"
velocity="6.16"/>
      <dynamics damping="0.08" friction="0.02"/>
</joint>
```

Na druhej strane kĺb označený “joint4“ je definovaný ako pevný, čo znamená, že neumožňuje žiaden pohyb medzi telesami, jeho popis sa nachádza v tabuľke 4.3. Tento kĺb spája ľavý bok s torzom robota, a taktiež je potreba dodržiavať hierarchiu spojenia telies, v tomto prípade je ľavý bok teleso nadriadená a torzo je podriadené. Okrem

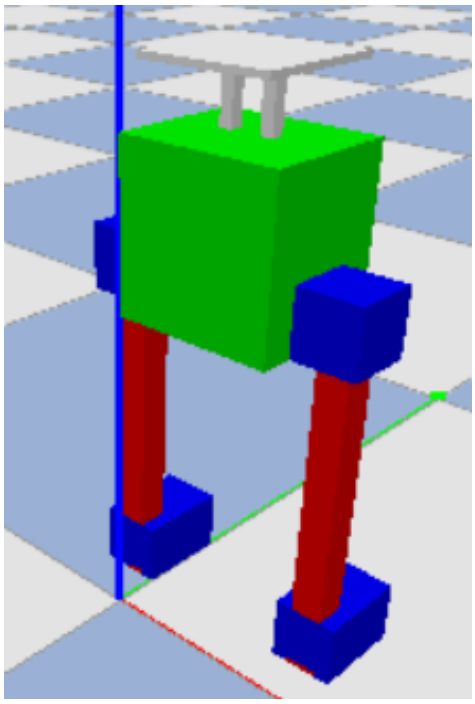
hierarchie spojenie je treba ešte definovať polohu kĺbu, ostatné parametre, ktoré sa určovali pri rotačných kĺboch, už nie je potreba definovať.

Tabuľka 4.3: Definícia pevného kĺbu v URDF súbore

```
<joint name="joint4" type="fixed">
      <parent link="lefthip"/>
      <child link="torzo"/>
      <origin xyz="-65 -12 90"/>
</joint>
```

4.3. Vytvorené modely v URDF súboroch

Pomocou postupu z predchádzajúcich kapitol sa vytvorili dva modely dvojnohých robotov, ktoré sa testovali v prostredí PyBullet. Prvý model sa vytvoril z veľmi jednoduchých kvádrových častí, na čom sa testovalo chápanie problému stavby robota v URDF súbore, model je zobrazený na obrázku 4.2. Tento model slúžil iba na jednoduché testovanie prostredia v PyBullete zo základných úkonov, ako vykopávanie nôh alebo správne spustenie a fyzikálne nastavenie prostredia.



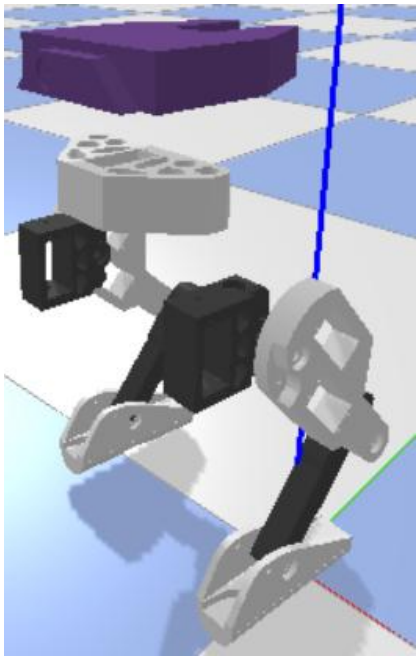
Obrázok 4.2: Model jednoduchého robota v URDF formáte

Tento model sa vytvoril aj v súbore XML, ktorý je kompatibilný so simulačným prostredím MuJoCo a taktiež sa testoval pre jednoduché úlohy.

Druhý model je robot PAWO, ktorého tvorba bola čiastočne vysvetlená v predchádzajúcich kapitolách. Tento model bol vytvorený podľa reálneho robota PAWO,

kde už sa testovali zložitejšie úkony, ako je chôdza vpred, model sa už nevytváral na testovanie v MuJoCo. Tento robot je zobrazený na obrázku 4.3.

Pri tvorbe modelu robota PAWO sa dbalo na presnosť fyzikálnych parametrov, ako sú typy a rozsahy kĺbov, výška robota, pomer medzi výškou robota a výškou nôh, rozmery chodidiel alebo hmotnosťou jednotlivých častí. Cieľom bolo vytvoriť robota, ktorý sa čo najviac fyzikálne podobá reálnemu robotovi PAWO, síce vizuálne je model od reality rozdielny, ale fyzikálne sú si veľmi blízke.



Obrázok 4.3: Model robota PAWO v PyBullete

4.4. Tvorba modelu robota v MuJoCo

MuJoCo používa vlastný formát na báze XML, pomocou ktorého sa definuje štruktúra a správanie modelu robota. Tento formát umožňuje presné zadanie geometrie, fyzikálnych vlastností, kĺbov, aktuátorov a interakcií s prostredím. Vďaka flexibilnej štruktúre XML je možné vytvárať komplexné robotické modely, ktoré pozostávajú z jednotlivých telies, navzájom prepojených pomocou kĺbov.

Základom každého MuJoCo modelu je hlavný element `<mujoco>`, v ktorom sa nachádzajú všetky ostatné komponenty modelu. Medzi kľúčové časti patrí sekcia `<worldbody>`, kde sa definuje telesná štruktúra robota, `<body>` reprezentuje jednotlivé teleso robota a určuje jeho polohu a orientáciu voči nadradenému prvku. V rámci každého telesa sa nachádza `<joint>`, ktorý definuje typ pohybu medzi telesami, vizualizácia a fyzikálne vlastnosti telesa sú definované pomocou elementu `<geom>`. Ďalej je dôležitý element **<**`actuator>`, ktorý definuje spôsob, akým sú poháňané jednotlivé kĺby a `<asset>` slúži na definovanie externých súborov, ktoré sú použité v modeli. Patria sem predovšetkým 3D modely, ako napríklad STL súbor.

Definícia telesa v MuJoCo prebieha pomocou elementu <body>, ktorý tvorí základnú stavebnú jednotku celého modelu. Každé teleso má svoj vlastný názov, definovanú pozíciu a orientáciu voči svojmu nadriadenému telesu. Vizualizácia a fyzikálne správanie telesa sa nastavujú pomocou elementu <geom>, ktorý sa nachádza v rámci každého <body>. Tento element obsahuje typ geometrie, názov importovaného STL modelu, ako aj jeho farbu a hustotu, hmotnosť sa automaticky dopočítava na základe rozmeru a hustoty. Priradenie tvaru súborov sa nerealizuje priamo v <geom>, ako je to v URDF, ale v sekcii <asset>, kde sú definované všetky externé zdroje, ako 3D modely alebo textúry. Následne sa na tieto zdroje odkazuje menom, čo zabezpečuje jednoduchšiu údržbu modelu.

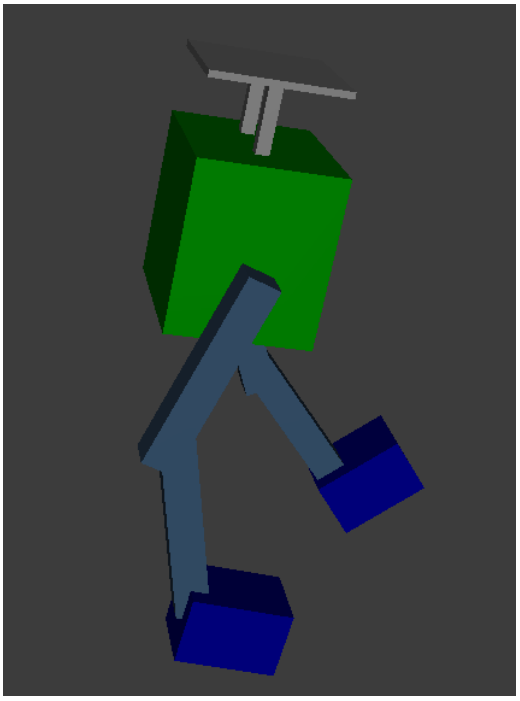
Pohybové vlastnosti sa definujú pomocou elementu <joint>, ktorý sa vkladá priamo do príslušného telesa. Umožňuje určiť typ pohybu, rozsah uhla, os rotácie a prípadne tlmenie alebo trenie. Každý element <body> môže obsahovať viacero <joint> alebo byť úplne statický, v takom prípade neobsahuje žiadne kĺby a slúži ako pevná časť konštrukcie.

Podľa postupu v tabuľke 4.4 sa vytvoril identický model robota v MuJoCo, ako aj v PyBullete. Tento model je zobrazený na obrázku 4.4. Jedná sa o veľmi jednoduchú napodobeninu robota PAWO, kde sa dbalo na podobnosť v počte kĺbov a rozmeroch, geometria sa volila zjednodušená, v Inventori sa vymodelovali kvádre, ktoré napodobňujú jednotlivé časti robota. Taktiež sa pomocou tohto robota testovalo prostredie v MuJoCo.

Tabuľka 4.4: Definícia časti modelu v MuJoCo

```
<!—Spojenie spodnej časti nohy s vrchnou časťou -->
<body name="lytko_p" pos="0.25 0 0" euler="0 0 0">
      <joint name="klb_ps_pl" type="hinge" pos="0 0 0" axis="0 0 1"
range="-0.5 0.5"/>
      <geom name="lytko" type="mesh" mesh="noha_s" rgba="0.4 0.6 0.8
1" density="1300"/>

      <!—Spojenie chodidla so spodnou časťou nohy -->
      <body name="chodidlo_p" pos="0.35 -0.01 -0.025" euler="0 0
1.57">
         <joint name="klb_pl" type="hinge" pos="0 0 0" axis="0 0 1"
range="-0.5 0.5"/>
         <geom name="chodidla_p" type="mesh" mesh="chodidlo" rgba="0 0
1 1" density="1300"/>
      </body>
</body>
```



Obrázok 4.4: Model jednoduchého robota v MuJoCo

4.5. Porovnanie tvorby modelu v URDF a MuJoCo

Formát URDF prináša výhody najmä v oblasti štandardizácie a kompatibility, predovšetkým v rámci ROS. Vyznačuje sa prehľadnou hierarchickou štruktúrou, ktorá uľahčuje tvorbu a údržbu modelov, najmä pre začiatočníkov. URDF je založený na XML, čo zaručuje čitateľnosť, jednoduchú editovateľnosť a ľahkú verzovateľnosť. Ďalšou výhodou je široká podpora zo strany rôznych simulátorov a vizualizačných nástrojov.

Prostredie MuJoCo naproti tomu ponúka vysokú mieru flexibility, presnú kontrolu nad fyzikálnymi parametrami a výkonné simulácie. Podporuje pokročilé vlastnosti ako sú pružné telesá, senzory, torzné pružiny či vlastné definície obmedzení. Pre náročné úlohy strojového učenia, najmä s dôrazom na real-time výkonnosť a presnú fyziku, predstavuje MuJoCo veľmi silný nástroj. Nevýhodou MuJoCo však môže byť zložitejšia štruktúra súborov (MJCF/XML formát), ktorá môže byť pre menej skúsených používateľov náročná na pochopenie a úpravu. Ďalšou nevýhodou je slabšia integrácia s ROS, keďže primárne vznikol ako samostatný simulátor, čím môže byť jeho využitie v komplexnejších robotických systémoch komplikovanejšie. MuJoCo takisto donedávna vyžadoval komerčnú licenciu, čo mohlo obmedzovať jeho použitie v niektorých akademických alebo open-source projektoch.

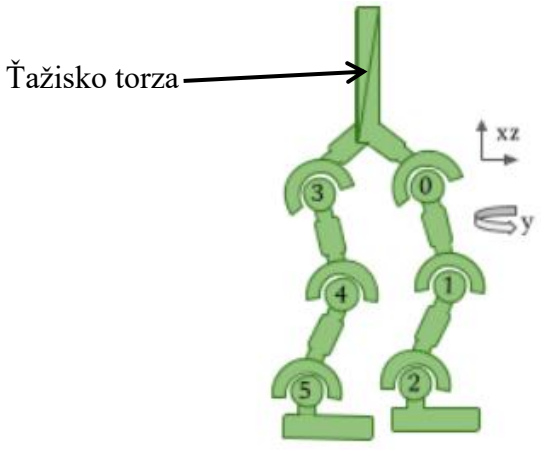
5 Walker2D

Pred aplikovaním metódy PPO na robota PAWO sa jej funkčnosť najskôr otestovala na prostredí Walker2D. Walker2D je prostredie z OpenAI Gym, ktoré simuluje dvojnohého robota pohybujúceho sa vpred v dvojrozmernom priestore. Tento agent má za úlohu naučiť sa stabilnú chôdzu, v prípade tejto práce pomocou metódy PPO. Táto kapitola najskôr rozoberie funkcionalitu prostredia, následne sa experimentálne určí odmenová funkcia, ktorá zabezpečí plynulý pohyb robota vpred.

5.1. Natavenie prostredia Walker2D

Priestor pozorovania predstavuje stavový vektor, ktorý agent využíva na rozhodovanie v rámci RL. Tento vektor obsahuje všetky relevantné informácie o aktuálnom stave robota, na základe ktorých sa učí optimalizovať svoj pohyb. Hodnoty jednotlivých premenných sa počas simulácie dynamicky menia v závislosti od interakcie agenta s prostredím. V Gymnasium je tento priestor definovaný ako spojitý. Walker2D je modelovaný ako 2D chodiaci robot so 17 parametrami priestoru pozorovania, ktoré zahŕňajú polohu a rýchlosť ťažiska, uhlové rýchlosti kĺbov a natočenie jednotlivých častí telá[19]. Pre ukážku chôdze robota v prostredí sú použité prednastavené hodnoty v intervale (-∞, ∞). V praxi sa volia rozumné limity, aby hodnoty zostali fyzicky realistické a aby sa predišlo numerickým nestabilitám pri tréningu modelu, zmena limitov sa bude praktizovať až pri zostavovaní priestoru pozorovania robota PAWO.

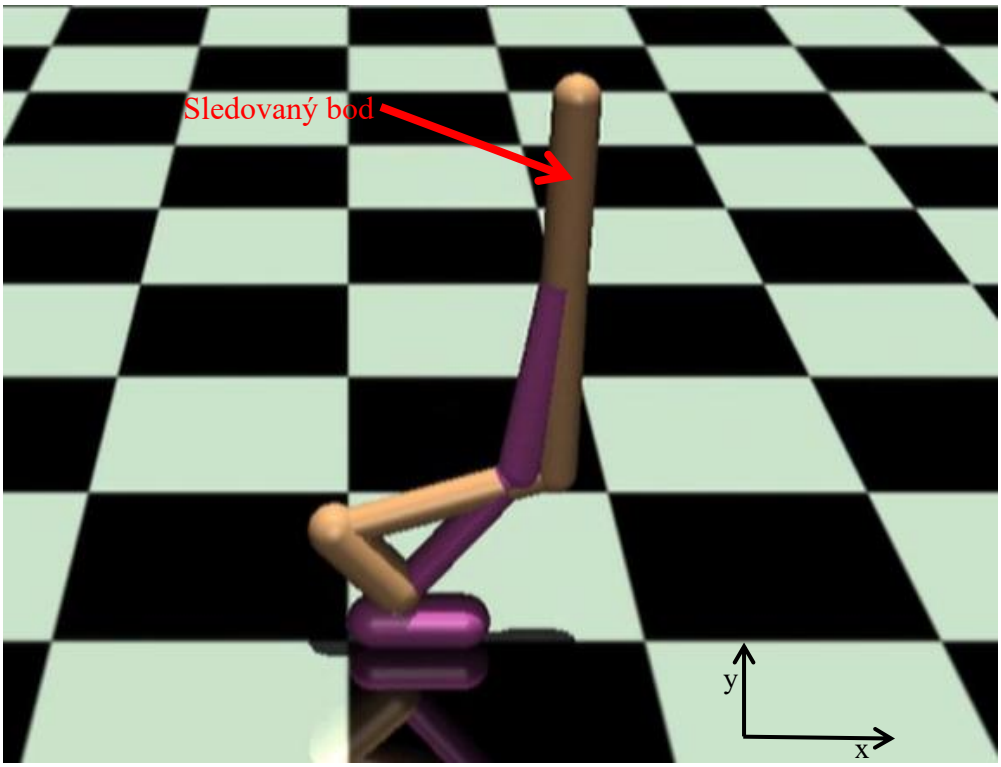
Akčný priestor v RL definuje všetky možné akcie, ktoré môže agent vykonať v danom prostredí. Na základe aktuálneho stavu agent vyberá akciu, ktorá ovplyvní budúci stav prostredia a jeho odmenu. V prostredí Walker2D je použitý prednastavený akčný priestor, ktorý je kontinuálny a pozostáva zo 6 akčných premenných, ktoré reprezentujú torzné momenty aplikované na kĺby robota. Každá z týchto premenných nadobúda hodnoty v intervale <−1,1>. Označenie jednotlivých kĺbov je zobrazené na obrázku 5.1.



Obrázok 5.1: Popis kĺbov robota z prostredia Walker2D[19]

5 Walker2D

V prostredí Walker2D je epizóda štandardne ukončená v prípade, že výška ťažiska torza robota klesne pod 0,8 m alebo presiahne 2,0 m, čo indikuje pád alebo nerealistické správanie. Zároveň je epizóda obmedzená aj maximálnym počtom krokov na 1000, čo zaisťuje efektívnosť tréningu. Prostredie Walker2D je zobrazený na obrázku 5.2.



Obrázok 5.2: Prostredie Walker2D

Jediná časť prostredia Walker2D, ktorá sa menila, je odmenová funkcia. Odmenová funkcia v prostredí Walker2D určuje, ako je agent motivovaný pri učení optimálnej stratégie pohybu vpred. Je navrhnutá tak, aby podporovala rýchly a stabilný pohyb, pričom penalizuje správanie vedúce k pádu alebo veľkým akciám v kĺboch. Celková odmena, ktorú agent získava v každom kroku simulácie pozostáva zo štyroch hlavných zložiek. Ide o súčet hodnôt rýchlosti vpred, odmeny za držanie výšky ťažiska torza v požadovanej výške, náklonu robota a súčet druhej mocniny torzných momentov aplikovaných na všetky kĺby. Všetky polohové a dynamické parametre torza sú sledované vzhľadom na jeho referenčný bod, ktorý sa nachádza v ťažisku torza.

Hyperparametre sú veľmi dôležité pre efektívny tréning modelu pri využívaní algoritmov RL. V prípade prostredia Walker2D majú veľký vplyv najmä parametre ako rýchlosť učenia, počet krokov na aktualizáciu stratégie, dĺžka epizódy či clipping. Konkrétne hodnoty hyperparametrov popisuje tabuľka 5.1. Nesprávne zvolené hodnoty môžu viesť k nestabilnému učeniu alebo k tomu, že agent uviazne v lokálnom minime. Niektoré hyperparametre sa menili a zvyšné ostali nastavené podľa pôvodného nastavenia. Parametre, ktoré sa menili, majú význam pri tréningu. Na druhej strane, parametre, ktoré nemajú výrazný vplyv na tréning, ostali nezmenené.

Tabuľka 5.1: Hyperparametre prostredia Walker2D

Paramer	Hodnota	Popis
learning_rate	0.0003	Rýchlosť učenia
n_steps	4096	Počet krokov prostredia medzi aktualizáciami siete
batch_size	64	Počet vzoriek použitých pri jednej aktualizácií stratégie
n_epochs	10	Počet prechodov (epoch) cez dáta pri aktualizácii stratégie
clip_range	0.2	Rozsah orezania zmien stratégie
ent_coef	0.01	Koeficient entropie - podpora prieskumu
gamma	0.99	Faktor budúcich odmien

5.2. Proces tréningu prostredia Walker2D

Pre určenie optimálneho nastavenia parametrov odmenovej funkcie alebo počtu krokov potrebných na dosiahnutie požadovaného správania robota bolo prostredie testované pri rôznych konfiguráciách. Agent sa opakovane trénoval pri rôznych variantoch odmenovej funkcie, kde sa postupne pridávali jednotlivé zložky odmeny. Tento postupný prístup umožnil podrobne sledovať, ako jednotlivé zložky ovplyvňujú učenie agenta a jeho výsledné správanie. Na začiatku bola odmenová funkcia navrhnutá veľmi jednoducho, pozostávala výhradne z pozitívnej odmeny za pohyb vpred. Cieľom bolo overiť, či samotná motivácia k dosiahnutiu vyššej rýchlosti stačí na osvojenie si základného pohybového vzoru.

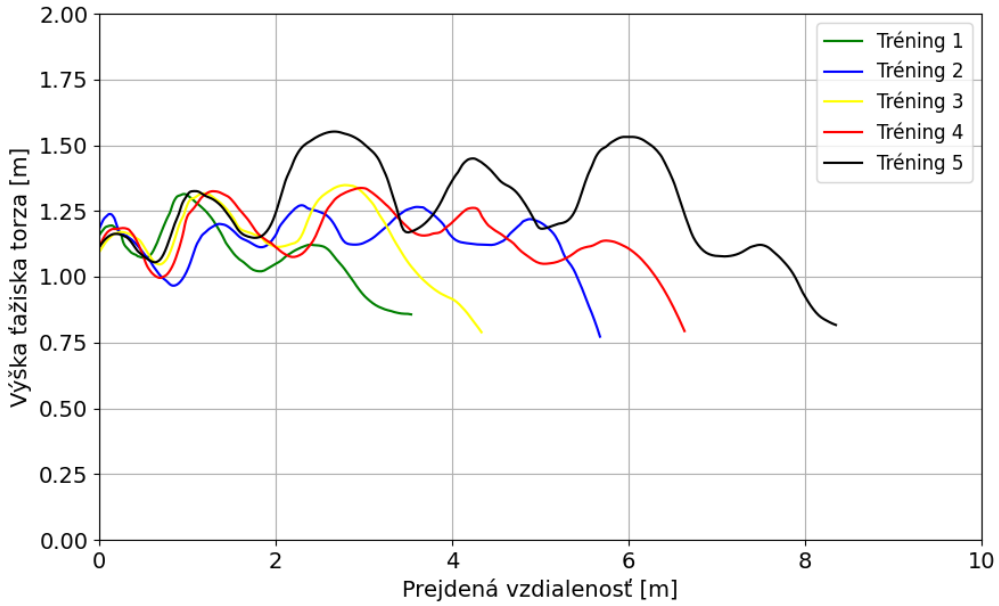
V ďalších fázach experimentov bola odmena rozšírená o komponent, ktorý zohľadňoval výšku ťažiska torza. Táto zložka mala zabezpečiť, aby si agent osvojil chôdzu s prirodzeným držaním tela a predišlo sa nežiaducej nízkej polohe torza. Následne boli do funkcie implementované penalizácie, najskôr za aplikáciu veľkých momentov do kĺbov, čím sa docielil plynulejší a realistickejší pohyb, a neskôr aj za nežiaduci náklon torza, ktorý môže viesť k strate stability a následne k pádu.

Okrem úprav samotnej odmeny sa experimentovalo aj s počtom krokov učenia. Ukázalo sa, že aj pri správne nastavenej odmenovej funkcii agent nedokázal dosiahnuť cieľ, pokiaľ bol počet tréningových krokov príliš nízky. Prostredie v týchto prípadoch často končilo epizódu predčasne kvôli splneniu podmienok ukončenia. Z tohto dôvodu bol spočiatku model trénovaný s vysokou hodnotou až 1 500 000 krokov, aby sa eliminoval vplyv predčasného ukončenia učenia. Hoci to znamenalo dlhší čas výpočtu, ale zabezpečilo to stabilné výsledky. Optimalizácia počtu krokov, s cieľom minimalizovať čas potrebný na dosiahnutie požadovaného správania, sa realizovala až v záverečnej fáze testovania, keď už boli známe optimálne hodnoty váh odmenovej funkcie.

5.2.1. Tréning s odmenou založenou výhradne na rýchlosti

Najskôr bol agent trénovaný s odmenovou funkciou, ktorá zohľadňovala výhradne rýchlosť pohybu robota vpred. Týmto spôsobom sa začala formovať základná verzia odmenovej funkcie, ktorá bola ďalej rozširovaná. Po natrénovaní agenta možno z obrázka 5.3 pozorovať, že robot sa pohybuje vpred, ale po chvíli nastane podmienka ukončenia, a epizóda končí.

Pri každom experimente sa tiež nastavili váhy jednotlivých zložiek odmenovej funkcie tak, aby sa maximalizovala plynulosť pohybu robota, ktoré sa určili experimentálne. Váhy odmenovej funkcie sú popísané v tabuľke 5.2. Cieľom bolo dosiahnuť rovnováhu medzi rýchlosťou, stabilitou a minimalizáciou nadmerných alebo trhavých akcií. Pre každé nastavenie odmenovej funkcie sa vykonalo 5 tréningov pre lepšie pochopenie správania robota v prostredí.



Obrázok 5.3: Vývoj výšky torza na prejdenej vzdialenosti pri odmene založenej na výhradne na rýchlosti vpred

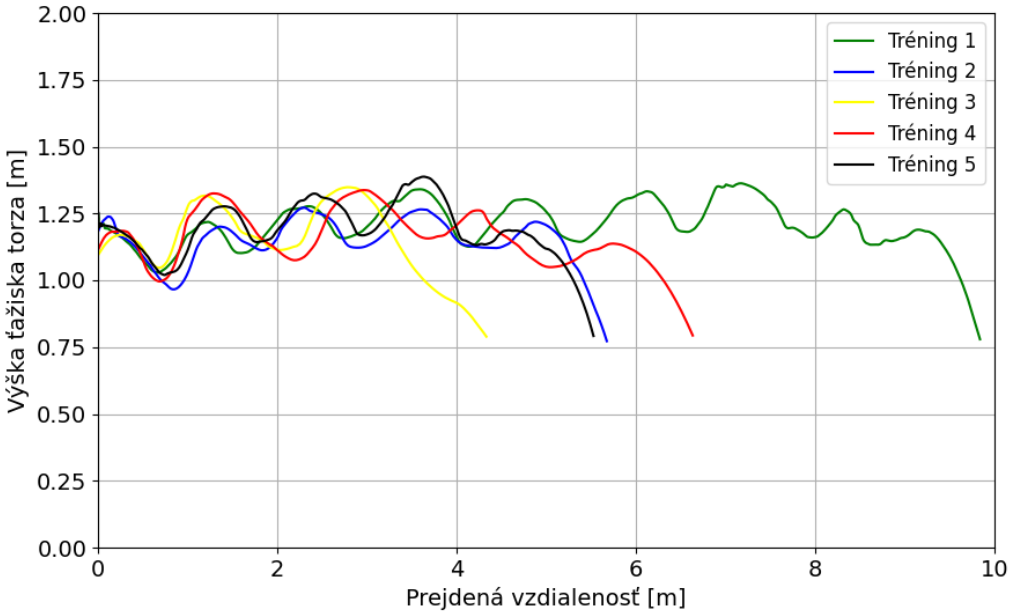
Tabuľka 5.2: Váhové parametre odmenovej funkcie – tréning založený na rýchlosti

Názov	**Zložka funkcie odmeny**	**Váha**
Rýchlosť vred	v_x	1

5.2.2. Tréning s pridaním odmeny za výšku torza

V tomto experimente bola odmenová funkcia rozšírená o podmienku, že ak sa výška ťažiska torza nachádza v požadovanom intervale, k hodnote odmeny sa pripočíta 1. Naopak, ak výška prekročí tento interval, od odmeny sa odpočíta hodnota 10. Podrobný popis jednotlivých zložiek odmenovej funkcie s váhami je uvedený v tabuľke 5.3. Toto nastavenie zabezpečuje, že sa robot nebude pohybovať v neželaných výškach, pri ktorých by jeho stabilita mohla byť narušená a pohyb by sa skončil pádom.

Maximálna povolená výška torza robota je 1,5 m a minimálna výška je 0,8 m. V rámci experimentu sa ukázalo, že výška torza nepresiahla hodnotu 1,5 m. V prípadoch, keď výška klesla pod 0,8 m, bola epizóda automaticky ukončená, čo vidno aj z obrázka 5.4.



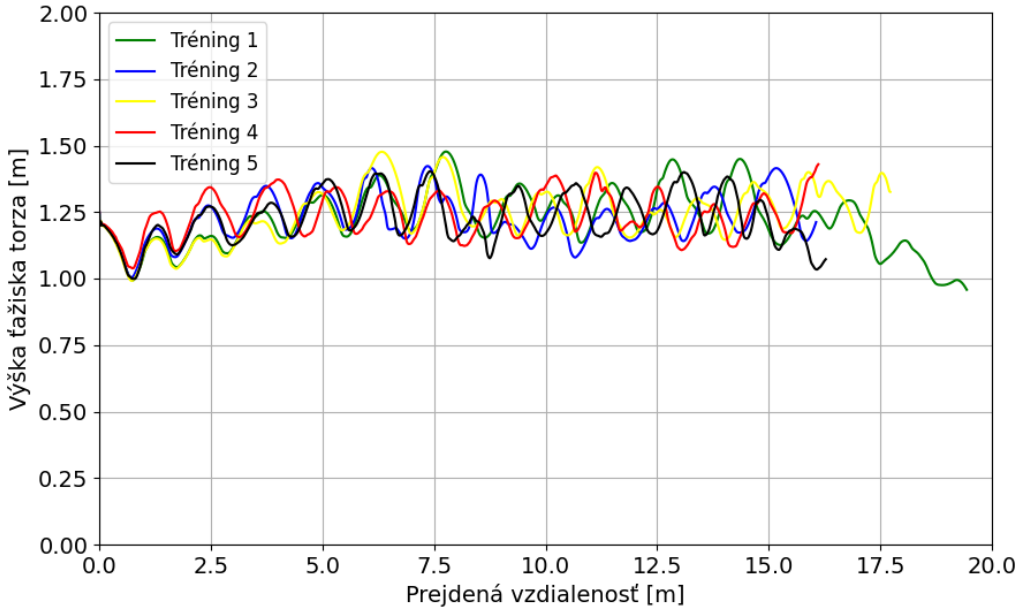
Obrázok 5.4: Vplyv odmeny za výšku torza

Tabuľka 5.3: Váhové parametre odmenovej funkcie – odmena za výšku torza

Názov	**Zložka funkcie odmeny**	**Váha**
Rýchlosť vred	v_x	1
Výška torza	$\begin{cases} 1 & 0{,}8\ \text{m} \leq h \leq 1{,}5\ \text{m} \\ -10 & h > 1{,}5\ \text{m};\ h < 0{,}8\ \text{m} \end{cases}$	5

5.2.3. Tréning s pridaním penalizácie za veľké akcie

V tomto experimente bola odmenová funkcia rozšírená o penalizáciu veľkosti vykonávaných akcií. Podrobný prehľad nastavenia odmenovej funkcie je uvedený v tabuľke 5.4, a výsledné správanie robota je vizualizované na obrázku 5.5. Agent bol teda motivovaný nielen k rýchlemu a stabilnému pohybu, ale aj k používaniu plynulejších akcií. Takto natrénovaný robot prekonal dlhšiu vzdialenosť než v predchádzajúcom prípade. Zároveň možno pozorovať, že vo väčšine prípadov sa epizódy ukončili vyčerpaním maximálneho počtu krokov, a nie pádom robota.



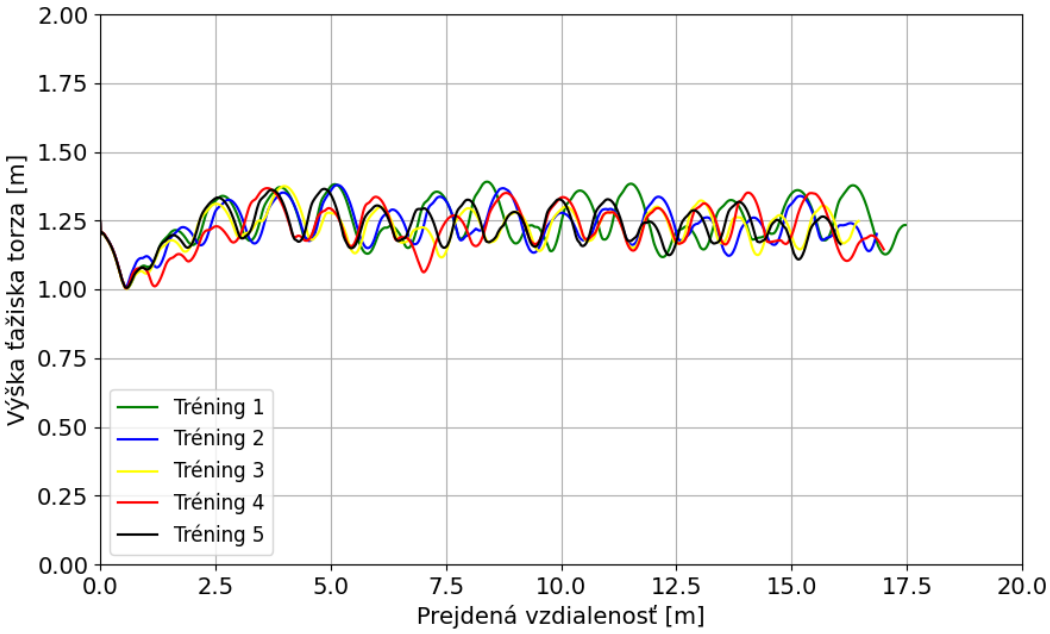
Obrázok 5.5: Zníženie veľkosti akcií prostredníctvom penalizácie

Tabuľka 5.4: Váhové parametre odmenovej funkcie – penalizácia veľkých akcií

Názov	**Zložka funkcie odmeny**	**Váha**
Rýchlosť vred	v_x	1
Výška torza	$\begin{cases} 1 & 0{,}8\ \text{m} \le h \le 1{,}5\ \text{m} \\ -10 & h > 1{,}5\ \text{m};\ h < 0{,}8\ \text{m} \end{cases}$	5
Aplikované momenty	$-\sum_{i=1}^{6} M_i^2$	0,075

5.2.4. Tréning s pridaním penalizácie za náklon torza

Pri optimálnom nastavení bola odmenová funkcia doplnená o penalizáciu za náklon torza. Agent bol tak motivovaný udržiavať vzpriamenú polohu, čo viedlo k ešte plynulejšiemu a stabilnejšiemu pohybu než v predchádzajúcich prípadoch. Penalizácia za náklon slúžila ako ochranný mechanizmus proti nadmernému kývaniu a stratám rovnováhy, ktoré môžu viesť k nechcenému správaniu, prípadne až k pádu. Konkrétne hodnoty penalizácie a nastavenia odmenovej funkcie sú uvedené v tabuľke 5.5, pričom výsledné správanie robota je znázornené na obrázku 5.6. Z výsledkov experimentu vyplýva, že všetky epizódy sa ukončili vyčerpaním maximálneho počtu krokov, nie pádom robota.



Obrázok 5.6: Optimálne nastavenie odmenovej funkcie

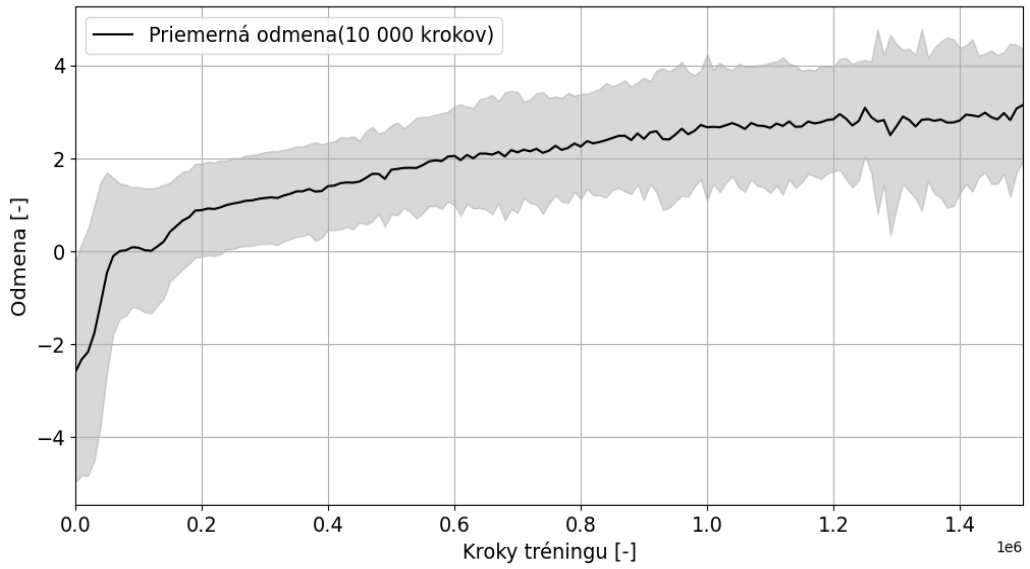
Tabuľka 5.5: Váhové parametre odmenovej funkcie – optimálne nastavenie

Názov	**Zložka funkcie odmeny**	**Váha**
Rýchlosť vred	v_x	1
Náklon torza	$-\theta_z$	6
Výška torza	$\begin{cases} 1 & 0{,}8\ \text{m} \le h \le 1{,}5\ \text{m} \\ -10 & h > 1{,}5\ \text{m};\ h < 0{,}8\ \text{m} \end{cases}$	5
Aplikované momenty	$-\sum_{i=1}^{6} M_i^2$	0,075

5.2.5. Experimentovanie s počtom krokov tréningu

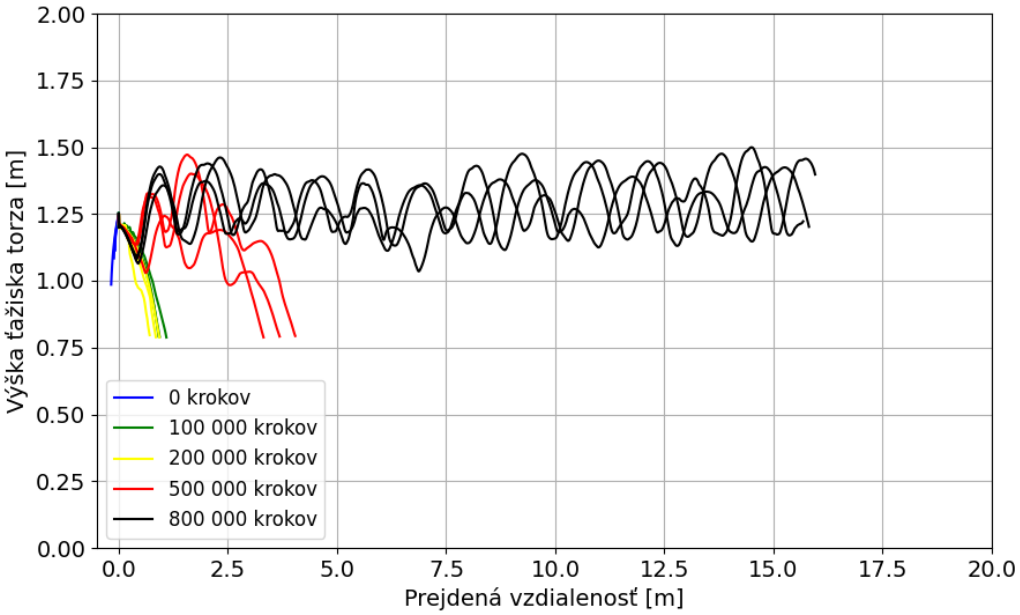
Cieľom tohto experimentu bolo identifikovať takú konfiguráciu učenia, pri ktorej agent nadobudne schopnosť stabilnej chôdze v čo najkratšom čase. Znamenalo to minimalizovať počet tréningových epizód a krokov potrebných na to, aby sa robot naučil plynulo pohybovať v prostredí. Efektívna optimalizácia procesu učenia je kľúčová nielen z hľadiska skrátenia času trénovania, ale aj z pohľadu výpočtovej náročnosti. Tieto faktory sú obzvlášť dôležité pri komplexných simulačných modeloch, kde každá epizóda predstavuje výraznú záťaž pre výpočtový systém. Zníženie počtu tréningových krokov tak prispieva k vyššej praktickej využiteľnosti algoritmov učenia posilňovaním v reálnych aplikáciách.

Na obrázku 5.7 je zobrazený priebeh odmeny, ktorá predstavuje priemernú hodnotu dosiahnutú z 10 000 krokov tréningu. Tento spôsob vizualizácie umožňuje eliminovať krátkodobé výkyvy a lepšie zachytiť dlhodobý trend zlepšovania správania agenta. Súčasne je na obrázku sivou farbou vyznačená smerodajná odchýlka, ktorá poskytuje informáciu o miere variability odmeny v danom intervale. Šedé oblasti tak vizuálne znázorňujú spoľahlivosť a stabilitu učenia — užšie pásmo odchýlky signalizuje konzistentnejšie správanie agenta, zatiaľ čo širšie pásmo poukazuje na vyššiu variabilitu výsledkov.



Obrázok 5.7: Závislosť kumulatívnej odmeny na počte krokov učenia

Výsledky experimentu potvrdili, že pri zvolenom nastavení tréningového procesu agent nadobudol základné schopnosti chôdze už po približne 800 000 krokoch učenia. Naučený pohyb bol plynulý, stabilný a vo väčšine testovacích scenárov aj konzistentne opakovateľný. Priebeh dosahovaných odmien a stabilitu učenia znázorňuje obrázok 5.8. Zároveň sa ukázalo, že epizódy vo viacerých prípadoch končili až po dosiahnutí maximálne povoleného počtu krokov, a nie v dôsledku straty rovnováhy či pádu. Tento jav predstavuje dôležitý ukazovateľ kvality naučenej stratégie a jej schopnosti udržať rovnováhu počas celej epizódy.



Obrázok 5.8: Experiment s počtom krokov učenia pre optimálne nastavenie odmeny

6 Robot PAWO

Táto kapitola predstavuje architektúru navrhnutého riešenia a organizáciu zdrojového kódu, ktorý bol vytvorený na účely simulácie, učenia a vizualizácie riadenia dvojnohého robota PAWO. Podrobne sú opísané kľúčové funkcie systému, ktoré zabezpečujú základné operácie, ako sú inicializácia prostredia, získavanie stavových údajov robota, aplikácia akcií a vykonávanie simulačných krokov. Záverečná časť kapitoly sa venuje tréningu modelu pri rôznych nastaveniach odmenovej funkcie a počte tréningových krokov, pričom výsledky sú vizualizované prostredníctvom grafov.

6.1. Architektúra riešenia a štruktúra kódu

Implementácia simulácie robota bola navrhnutá ako rozšíriteľná architektúra založená na objektovo-orientovanom programovaní. Hlavnú kostru systému tvoria dve kľúčové triedy: RobotEnv a PavoEnv, pričom PavoEnv dedí všetky základné funkcie od triedy RobotEnv. Táto štruktúra zabezpečuje opätovné použitie spoločnej logiky a zároveň umožňuje vytvárať špecifické implementácie pre konkrétny typ robota.

Trieda RobotEnv implementuje funkcie, ako `__init__()`, `reset()`, `step()` a `mld()`, ktoré zabezpečujú všeobecnú inicializáciu simulácie, riadenie priebehu epizód, aplikáciu akcií na robota, priebežnú aktualizáciu jeho stavu počas simulácie a možnosť uloženia či obnovenia predchádzajúceho stavu prostredia pre účely testovania alebo spätného ladenia. V rámci tejto triedy sú definované základné parametre simulácie, ako sú počet krokov epizódy, dĺžka aplikácie jednej akcie alebo rýchlosť simulácie.

Trieda PavoEnv predstavuje konkrétne simulačné prostredie, ktoré dedí zo základnej triedy RobotEnv a je prispôsobené na učenie a testovanie riadenia konkrétneho robotického modelu, v tomto prípade ide o robota PAWO. PavoEnv rozširuje funkcionalitu základného prostredia o špecifiká daného robota, ako je počet ovládaných kĺbov, ich rozsah pohybu, maximálne povolené momenty, ktoré sa môžu aplikovať v jednotlivých kĺboch a maximálne uhlové rýchlosti, ktorými sa kĺby môžu pohybovať. Ďalej ide o definíciu akčného priestoru, stavového priestoru, nastavenie počiatočnej polohy a správne nastavenie odmenovej funkcie. V tomto prostredí sa nastavuje aj kamera sledujúca robota, čo je potrebné pre vizualizáciu.

6.1.1. Inicializácia prostredia

Funkcia `__init__()` slúži v oboch triedach na inicializáciu prostredia simulácie, v ktorom bude agent učený. Táto funkcia zabezpečuje nastavenie základných parametrov prostredia, spustenie fyzikálneho prostredia PyBullet, načítanie modelu robota a jeho počiatočné umiestnenie v simulovanom priestore.

V základnej triede RobotEnv sa definuje cesta k URDF súboru robota, pozícia, v ktorej má byť robot umiestnený, a prepínač pre režim vizualizácie. Podľa tohto režimu sa spúšťa buď PyBullet v grafickom rozhraní GUI, alebo v režime bez vizualizácie DIRECT, čo je efektívnejšie pre tréning a používa sa pri tréningu robota PAWO. Následne

sa nastavuje fyzikálne prostredie, vytvorí sa rovina, po ktorej sa robot pohybuje, a nastaví sa gravitácia. Ďalej sa načíta samotný model robota pomocou URDF súboru a robot sa umiestni do počiatočnej pozície.

V rozšírenej triede PavoEnv, ktorá dedí od RobotEnv, sa funkcia `__init__()` špecializuje na konkrétne nastavenie robota PAWO. Okrem volania nadradenej triedy sa v tejto triede nastavuje akčný a stavový priestor.

6.1.2. Akčný a stavový priestor

Akčný priestor je definovaný ako spojitý, čo je vhodné najmä pre robotické úlohy, kde sú akcie reprezentované plynulými pohybmi kĺbov alebo motorov. V triede PavoEnv je akčný priestor implementovaný pomocou objektu `spaces.Box` z knižnice Gymnasium. Vytvára sa násobný spojitý priestor s dimenziou 6, čo znamená, že agent môže ovládať šesť stupňov voľnosti, v prípade robota PAWO šesť kĺbov. Každá hodnota v tomto priestore je ohraničená rozsahom od -1 do 1. Tieto hodnoty sa počas simulácie škálujú na rozsah $\pm\pi/6$ radiánov, čím sa zabezpečí, že skutočný rozsah pohybov zodpovedá fyzickým obmedzeniam robota. Presná definícia akčného priestoru je popísaná v tabuľke 6.1.

Tabuľka 6.1: Definícia akčného priestoru

```
self.action_space = spaces.Box(
np.array([-1] * self.n, dtype=np.float32),
np.array([1] * self.n, dtype=np.float32),
dtype=np.float32
)
```

V kontexte RL predstavuje stavový priestor množinu všetkých možných pozorovaní, ktoré agent môže získať zo svojho prostredia. V prípade robota PAWO je stavový priestor podobne ako akčný priestor definovaný v triede PavoEnv pomocou objektu `spaces.Box` z knižnice Gymnasium.

Stav zahŕňa rôzne fyzikálne veličiny, ktoré opisujú aktuálny stav robota a jeho interakciu s prostredím. Pôvodný stavový priestor obsahoval základné premenné, ako sú natočenia, uhlové rýchlosti torza okolo jednotlivých osí a poloha torza robota v trojrozmernom priestore. Tieto veličiny poskytovali základný prehľad o orientácii a stabilite robota. Ukázalo sa však, že takto obmedzený stavový priestor neobsahoval dostatok informácií pre úspešné učenie, najmä pokiaľ ide o zložitejšie úlohy, ako je pohyb vpred alebo udržanie rovnováhy pri dynamickom presune hmotnosti. Robot bez detailnejších informácií o kĺboch nebol schopný efektívne koordinovať svoje pohyby. Preto bol stavový priestor rozšírený o uhlové pozície a rýchlosti jednotlivých kĺbov, ako aj o lineárnu rýchlosť v osi y, ktorá slúži ako indikátor pohybu vpred. Týmto rozšírením sa výrazne zlepšila schopnosť agenta vnímať stav svojho tela a pohybu, čo viedlo k efektívnejšiemu učeniu. Agent mal k dispozícii presnejší obraz o svojom nastavení a dynamike, mohol sa tak naučiť stabilnejšie a koordinovanejšie pohyby.

Ďalším dôležitým aspektom pri návrhu prostredia je nastavenie hraníc stavového priestoru, teda definovanie dolných a horných limitov pre každú pozorovanú veličinu. Tieto hranice predstavujú fyzikálne a logické limity, ktoré by dané veličiny nemali prekročiť počas simulácie. Slúžia ako ochrana pre učenie v prípade, že by pozorovanie prekročilo tieto hodnoty, algoritmus by mohol byť nefunkčný. Zároveň sa tým zabezpečuje, že všetky vstupy do neurónovej siete sú normalizované do známeho rozsahu, čo výrazne pomáha stabilite a efektivite učenia. V prípade, že budú limity stavového priestoru príliš rozsiahle a budú ďaleko od reálnych hodnôt, ktoré môžu nastať, tak sa výrazne znižuje efektivita učenia a predlžuje sa čas tréningu, pretože agent sa bude musieť učiť z väčšieho množstva informácií. Stavový priestor používaný pri tréningu robota PAWO sa nachádza v tabuľke 6.2.

Tabuľka 6.2: Definícia stavového priestoru

```
self.observation_space = spaces.Box(
      # Dolné hranice
      np.array([-np.pi, -np.pi, -np.pi,  # Natočenia torza
         -10, -10,                       # Uhlové rýchlosti torza(x,y)
         -10, 0,                         # Poloha torza(y,z)
         -1,                             # Rýchlosť torza v osi y
         -np.pi/6, -6.16, -np.pi/6, -6.16,
         -np.pi/6, -6.16, -np.pi/6, -6.16,
         -np.pi/6, -6.16, -np.pi/6, -6.16],# Polohy a rýchlosti kĺbov
          dtype=np.float32),
      # Horné hranice
      np.array([np.pi, np.pi, np.pi,      # Natočenia torza
          10, 10,                         # Uhlové rýchlosti torza(x,y)
          10, 0.6,                        # Poloha torza(y,z)
          1,                              # Rýchlosť torza v osi y
          np.pi/6, 6.16, np.pi/6, 6.16,
          np.pi/6, 6.16, np.pi/6, 6.16,
          np.pi/6, 6.16, np.pi/6, 6.16],  # Polohy a rýchlosti kĺbov
          dtype=np.float32),
       dtype=np.float32)
```

6.1.3. Funkcia `get_env_state()`

Ďalšou dôležitou funkcia je `get_env_state()`, definovaná v prostredí PavoEnv, ktorá sa podieľa na získaní aktuálneho stavu prostredia, zhromažďuje všetky relevantné fyzikálne informácie o stave robota zo simulácie PyBullet a ukladá ich do vektora, ktorý následne slúži ako pozorovanie pre model.

Primárne sa sleduje torzo robota, zisťuje sa jeho pozícia a orientácia v priestore. Taktiež sa získava aj uhlová a lineárna rýchlosť torza, ktoré reprezentujú dynamiku pohybu robota, najmä v smere osi y, ktorá je rozhodujúca pre hodnotenie postupu vpred. Tieto dáta dopĺňajú základné informácie o pohybe celého tela robota, ktoré sa využívajú na určenie stability.

Dôležitou súčasťou stavu sú údaje o kĺboch robota. Pre každý z ovládaných kĺbov sa získava ich aktuálna uhlová pozícia a uhlová rýchlosť. Tieto hodnoty umožňujú agentovi sledovať konkrétne polohy jednotlivých častí nôh robota a ich zmeny v čase.

Všetky tieto hodnoty sú uložené do jedného spojitého stavového vektora, ktorý je následne v každom kroku simulácie poskytnutý ako vstup pre neurónovú sieť agenta, na základe čoho sa rozhoduje, akú akciu vykonať. Vďaka tejto funkcii tak dochádza k efektívnemu prepojeniu medzi fyzikálnou simuláciou a rozhodovacím procesom učenia.

6.1.4. Aplikácia akcií

V simulačnom prostredí je kľúčovou súčasťou procesu učenia funkcia `mdl()`, ktorá zabezpečuje aplikáciu akcií na robota a následné vrátenie jeho aktuálneho stavu. Táto funkcia je implementovaná s využitím knižnice PyBullet, kde každá akcia predstavuje požadovanú cieľovú pozíciu pre konkrétny kĺb robota. V tomto prípade sa ovláda šesť kĺbov, ktoré reprezentujú pohyblivé časti nôh robota.

Riadenie kĺbov je realizované v režime ovládania pozície, čo znamená, že simulovaný motor sa snaží dosiahnuť zadanú uhlovú polohu. Každému kĺbu sa priradí cieľová hodnota získaná zo správania agenta a zároveň sa nastavujú dve dôležité fyzikálne vlastnosti: maximálny moment a maximálna rýchlosť. Hodnota momentov bola v simulácii zvolená ako 4,9 Nm a maximálna uhlová rýchlosť bola obmedzená na 6,16 rad/s, čo sú presne hodnoty, ktoré môžu dosiahnuť reálne motory robota PAWO. Popis detailného nastavenia sa nachádza v tabuľke 6.3.

Tabuľka 6.3: Nastavenie motoru v kĺbe

```
p.setJointMotorControl2(self.robot_id,
                        jointIndex=0,              # Kĺb s indexom 0
                        # Polohové riadenie motoru
                        controlMode=p.POSITION_CONTROL,
                        targetPosition=action[0],  # Cieľová pozícia
                        force=4.9,                 # Maximálny moment

                        maxVelocity=6.16)          # Maximálná rýchlosť
```

6.1.5. Simulačný krok

Funkcia `step()` predstavuje jednu z najdôležitejších funkcií v rámci RL, pretože zabezpečuje vykonanie jedného simulačného kroku na základe akcie zvolenej agentom. Tento krok zahŕňa aplikáciu akcie na robota, aktualizáciu stavu prostredia, výpočet odmeny a kontrolu podmienok na ukončenie epizódy.

Epizóda sa ukončí vtedy, keď sa splní podmienka ukončenia, v prípade robota PAWO, ak dôjde k náklonu robota okolo osi x alebo y väčšiemu ako ±π/8. Tento prístup slúži ako ochrana pred príliš dlhými epizódami, čo skracuje zbytočne dlhé simulácie v prípadoch, keď už nie je možné dosiahnuť pozitívny výsledok a zároveň zabezpečuje, že agent je penalizovaný za veľké náklony, ktoré vedú k pádu.

Funkcia `step()` očakáva ako vstup akciu, v tomto prípade vektor s hodnotami pre šesť riadených kĺbov robota. V triede PavoEnv je táto akcia najprv škálovaná do fyzikálne zmysluplného rozsahu v intervale <- π/6 , π/6 >, aby sa hodnoty pohybu kĺbov udržali v realistických medziach.

Následne sa zavolá rodičovská funkcia `step()` zo základnej triedy RobotEnv, ktorá zabezpečí základnú správu krokov, ako sú počítadlá krokov, informácie o ukončení akcie, následne sa aplikovaná akcia realizuje v simulácii pomocou funkcie `mdl()`. Výsledkom je aktualizovaný stav robota, ktorý je následne analyzovaný na výpočet odmeny.

6.2. Proces tréningu robota PAWO

Na trénovanie modelu riadenia robota bol vytvorený vlastný skript, ktorý integruje navrhnuté simulačné prostredie s algoritmom PPO pomocou knižnice SB3. Tento skript predstavuje dôležitý prvok implementácie, keďže umožňuje opakovateľné spúšťanie učenia v rôznych konfiguráciách. Na začiatku skriptu sa inicializuje prostredie PavoEnv v režime bez vizualizácie, čím sa výrazne znižuje výpočtová náročnosť a zvyšuje rýchlosť samotného učenia. Takéto nastavenie je výhodné predovšetkým počas dlhých tréningových fáz, kde nie je potrebné sledovať správanie robota vizuálne.

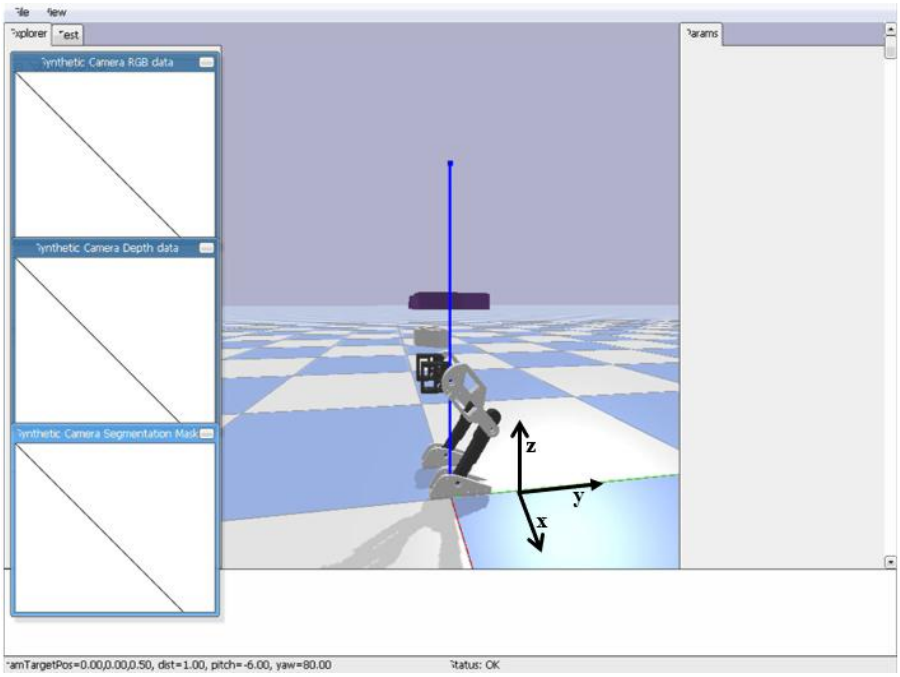
Podobne ako v prípade prostredia Walker2D, aj v tomto prípade bol model trénovaný na 1 500 000 krokov s vopred zvolenými hyperparametrami, ktoré boli experimentálne určené a sú zobrazené v tabuľke 6.4. Súčasťou procesu učenia bolo aj experimentovanie s odmenovou funkciou, ktorá sa vyvíjala podobne ako pri prostredí Walker2D. V jednotlivých spusteniach tréningu sa menili váhy jednotlivých zložiek odmeny, aby sa našla optimálna kombinácia, ktorá vedie k stabilnému a plynulému pohybu do vzdialenosti približne dvoch metrov vpred.

Po ukončení trénovania je natrénovaný model automaticky uložený do súboru vo formáte, ktorý je kompatibilný s knižnicou SB3. Tento model je následne možné jednoducho načítať vo vizualizačnom skripte a spustiť jeho vyhodnotenie v grafickom režime, čo je dôležité pre vizuálne overenie kvality naučenej stratégie.

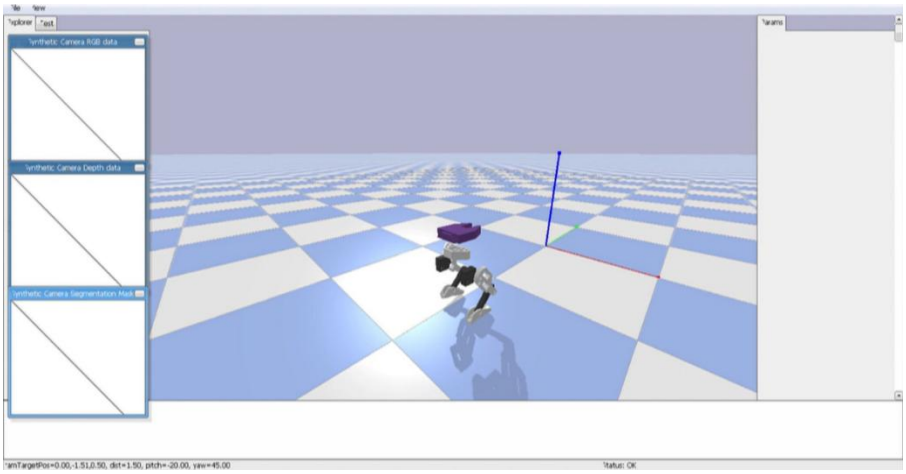
Tabuľka 6.4: Hyperparametre pri tréningu robota PAWO

Paramer	Hodnota
learning_rate	0.0003
n_steps	2048
batch_size	256
n_epochs	10
clip_range	0.2
ent_coef	0.01
gamma	0.98

Na vizualizáciu modelu bol vytvorený doplnkový skript, ktorý umožňuje spustenie trénovaného modelu v prostredí PavoEnv v režime s vizualizáciou, ktorý je zobrazený na obrázkoch 6.1 a 6.2. Po načítaní uloženého modelu sa simulácia spúšťa krok po kroku s výstupom informácií na konzolu. Skript zaznamenáva počet vykonaných akcií, dosiahnuté odmeny a stav ukončenia epizódy. Ak dôjde k ukončeniu epizódy, prostredie sa automaticky reštartuje a čítač akcií sa vynuluje. Výstupné informácie umožňujú lepšie porozumieť správaniu agenta v simulovanom prostredí a vizuálne overiť kvalitu naučenej stratégie.



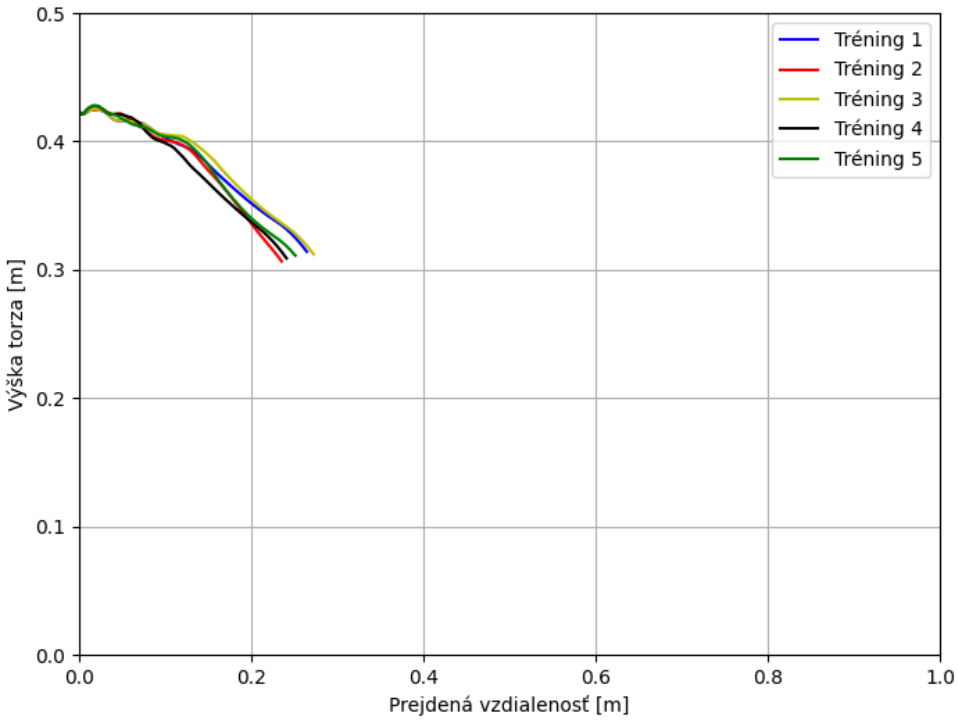
Obrázok 6.1: Robot PAWO v počiatočnej polohe v prostredí PyBullet



Obrázok 6.2: Robot PAWO v pohybe v prostredí PyBullet

6.2.1. Tréning založený výhradne na odmene za rýchlosť vpred

Podobne ako v prípade prostredia Walker2D bol agent trénovaný s odmenovou funkciou, ktorá sa postupne menila, pričom sa nastavovali aj optimálne váhy jednotlivých zložiek odmeny a pre každé nastavenie sa tréning opakoval 5-krát. Prehľad použitých zložiek odmenovej funkcie a zodpovedajúcich váh je uvedený v tabuľke 6.5. Pri tomto experimente sa zohľadňovala výhradne rýchlosť pohybu robota vpred, ide o pohyb robota proti smeru osi y. Po natrénovaní agenta je z obrázku 6.3 viditeľné, že robot sa pohybuje vpred, ale po chvíli dôjde k splneniu podmienky na ukončenie a epizóda končí.



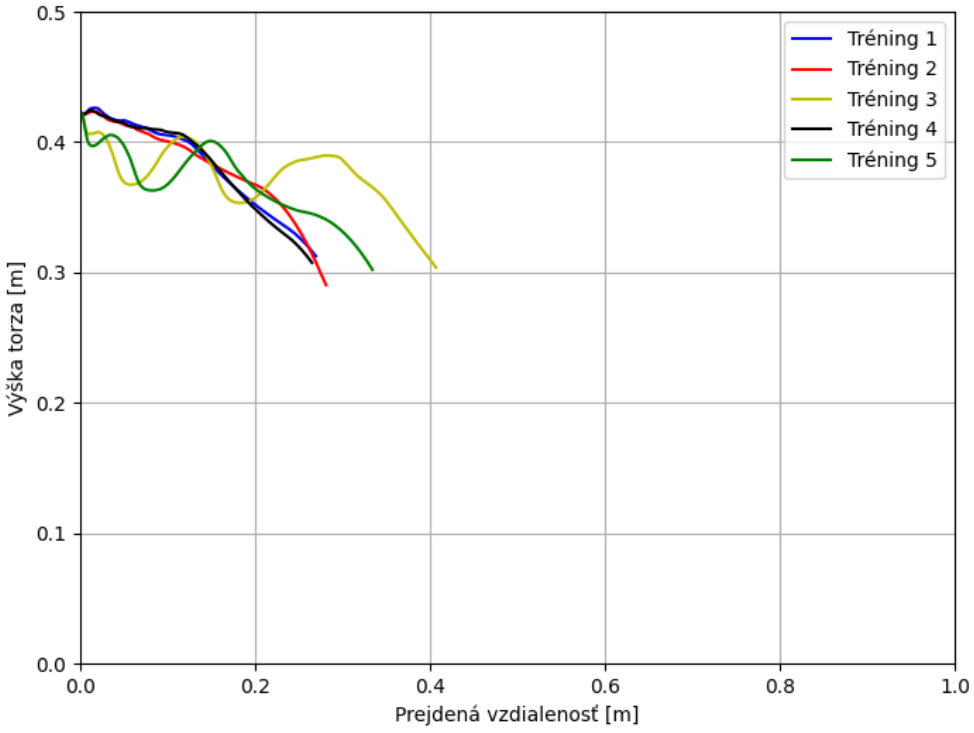
Obrázok 6.3: Vývoj výšky torza v závislosti na prejdenej vzdialenosti pri odmene založenej na výhradne na rýchlosti vpred robota PAWO

Tabuľka 6.5: Váhové parametre odmenovej funkcie robota PAWO – tréning založený na rýchlosti vpred

Názov	**Zložka funkcie odmeny**	**Váha**
Rýchlosť vred	v_y	4

6.2.2. Tréning s pridaním penalizácie za náklon okolo osi x a y

V tomto experimente sa pridala penalizácia za náklon robota okolo osi x a y, vďaka čomu sa robot dokázal posunúť ďalej oproti predchádzajúcemu nastaveniu. Napriek tomu už po krátkom čase opakovane došlo k splneniu podmienky na ukončenie epizódy, čo naznačuje, že samotné pridanie penalizácie za náklon a odmeňovanie pohybu vpred ešte nie je postačujúce na dosiahnutie stabilnej chôdze. Priebeh experimentu sa nachádza na obrázku 6.4. Tento experiment však predstavuje dôležitý krok v návrhu odmenovej funkcie. Na tomto základe bude ďalej rozširovaná odmenová funkcia o ďalšie prvky, aby sa zlepšila stabilita, plynulosť a dĺžka chôdze robota v simulácii. Rozšírená odmenová funkcia s penalizáciami za veľké náklony sa nachádza v tabuľke 6.6.



Obrázok 6.4: Stabilizácia torza robota PAWO pomocou penalizácie náklonov

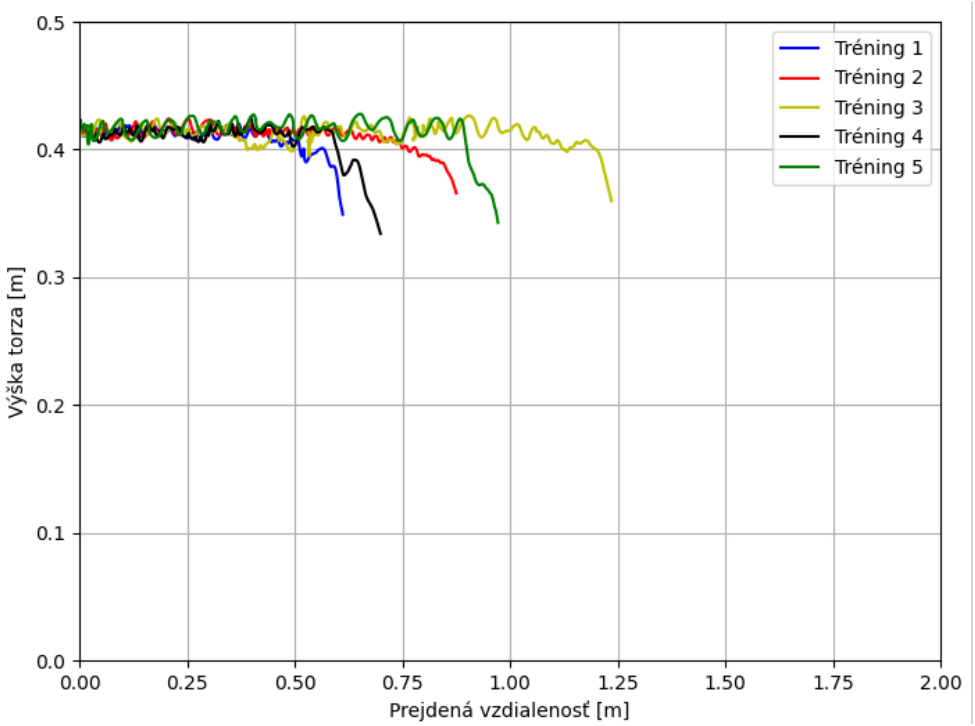
Tabuľka 6.6: Váhové parametre odmenovej funkcie robota PAWO – penalizácia náklonov

Názov	Zložka funkcie odmeny	Váha
Rýchlosť vred	v_y	4
Náklon okolo osi x	$-\theta_x$	5
Náklon okolo osi y	$-\theta_z$	5

6.2.3. Tréning s pridaním odmeny za výšku torza a penalizáciou za veľké akcie

Pridaním odmeny za výšku torza a penalizácie za veľké akcie sa pohyb robota výrazne zlepšil v porovnaní s predchádzajúcimi experimentmi. Táto úprava odmenovej funkcie umožnila agentovi lepšie udržiavať rovnováhu, keďže vyššia poloha torza zvyčajne súvisí so stabilnejším držaním tela pri pohybe vpred. Podoba odmenovej funkcie sa nachádza v tabuľke 6.7. Penalizácia veľkých akcií zároveň prinútila agenta voliť jemnejšie a plynulejšie pohyby motorov, čím sa zvýšila celková efektivita pohybu.

Pozitívnym výsledkom bola výrazné zníženie počtu náhlych strát rovnováhy a menej časté prerušenie epizód v dôsledku pádov. Robot vykazoval konzistentnejší pohyb vpred a dokázal udržať vertikálnu pozíciu torza počas väčšiny epizódy. Napriek týmto zlepšeniam však agent stále nedokázal prekonať požadovanú vzdialenosť v rámci jednej epizódy, čo je vidno z obrázku 6.5.



Obrázok 6.5: Optimalizácia výšky torza a obmedzenie akcií kĺbov robota PAWO

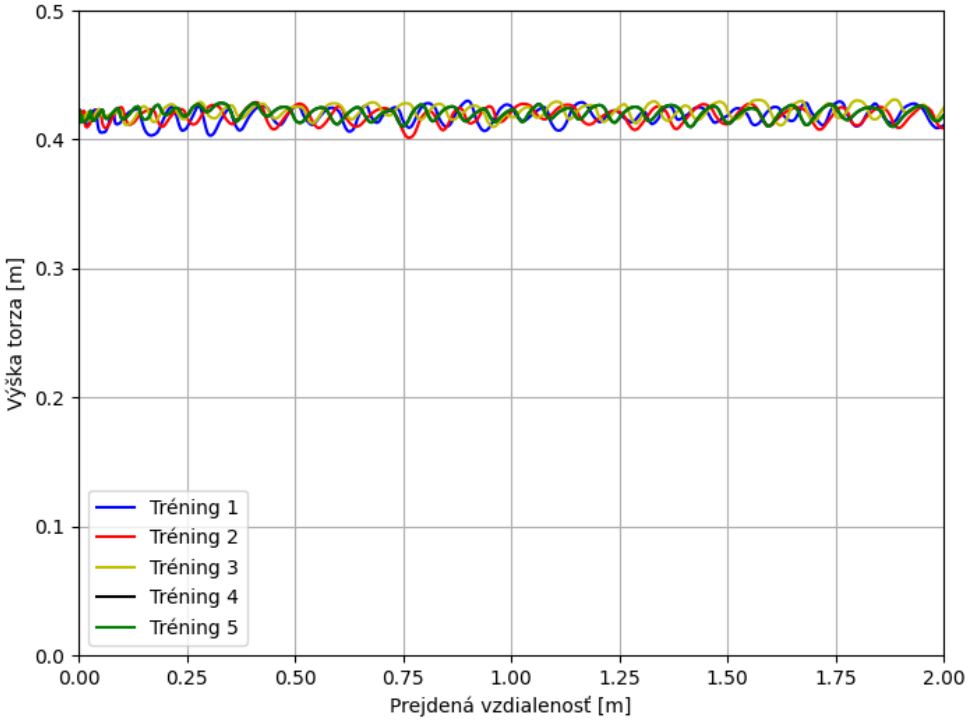
Tabuľka 6.7: Váhové parametre odmenovej funkcie robota PAWO – odmena za výšku torza, penalizácia za veľké akcie

Názov	**Zložka funkcie odmeny**	**Váha**
Rýchlosť vred	v_y	4
Náklon okolo osi x	$-\theta_x$	5
Náklon okolo osi y	$-\theta_y$	5
Výška torza	$\begin{cases} 1 & 0{,}35\ \text{m} \le h \le 0{,}45\ \text{m} \\ -10 & h > 0{,}45\ \text{m};\ h < 0{,}35\ \text{m} \end{cases}$	1
Aplikované momenty	$-\sum_{i=1}^{6} M_i^2$	3

6.2.4. Pridania penalizácie za rotáciu okolo z

V tomto experimente bola k odmenovej funkcii pridaná penalizácia za natočenie robota okolo osi z, čo znamená, že agent bol motivovaný udržiavať smer pohybu vpred bez výrazných odklonov doľava alebo doprava. Takáto úprava motivovala robota k minimalizácii bočných pohybov a rotácií, čím sa vytvorilo správanie, ktoré uprednostňovalo priamu a vyváženú chôdzu pozdĺž osi y. Týmto spôsobom sa podarilo eliminovať výkyvy z trajektórie, ktoré v predchádzajúcich experimentoch často viedli k strate rovnováhy alebo prerušeniu epizódy. Omenová funkcia je popísaná tabuľkou 6.8.

Výsledky experimentu ukázali, že vďaka tejto konfigurácii bol robot schopný dosiahnuť cieľovú vzdialenosť dvoch metrov v rámci jednej epizódy, čo predstavuje významný posun oproti predchádzajúcim nastaveniam. Pohyb robota bol plynulejší, stabilnejší a menej rozkolísaný, čo možno pripísať práve obmedzeniu nežiaducej rotácie a lepšej kontrole torza pri pohybe. Výsledky experimentov sú zobrazené na obrázku 6.6.



Obrázok 6.6: Optimálne nastavenie odmenovej funkcie robota PAWO

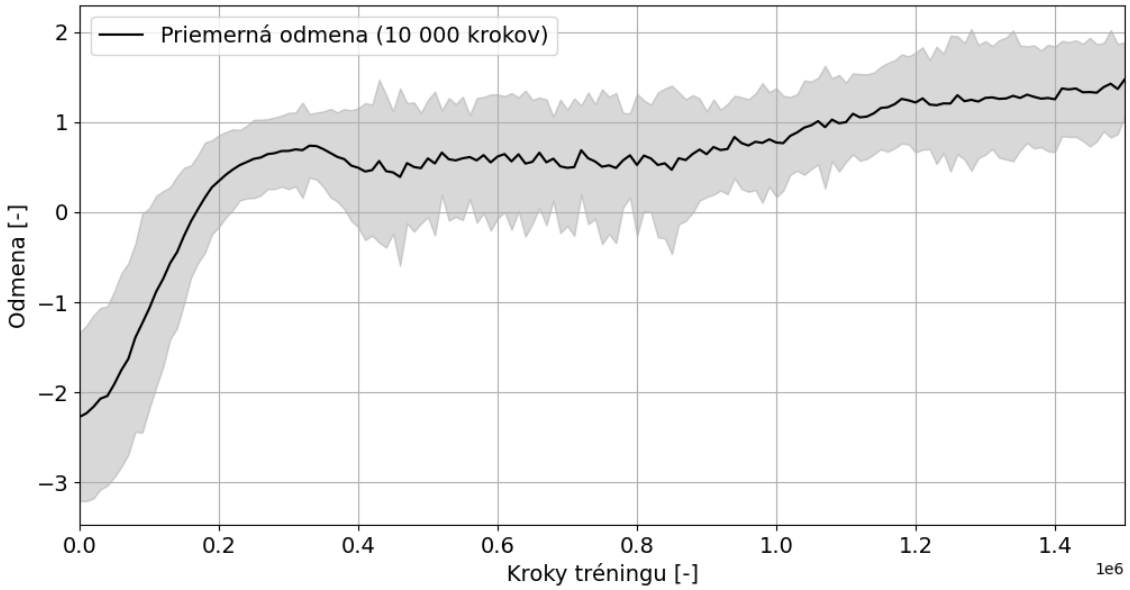
Tabuľka 6.8: Váhové parametre odmenovej funkcie robota PAWO – optimálne nastavenie

Názov	**Zložka funkcie odmeny**	**Váha**
Rýchlosť vred	v_y	4
Náklon okolo osi x	$-\theta_x$	5
Náklon okolo osi y	$-\theta_y$	5
Výška torza	$\begin{cases} 1 & 0{,}35\ \text{m} \leq h \leq 0{,}45\ \text{m} \\ -10 & h > 0{,}45\ \text{m};\ h < 0{,}35\ \text{m} \end{cases}$	1
Aplikované momenty	$-\sum_{i=1}^{6} M_i^2$	3
Natočenie okolo osi z	$-\theta_z$	3

6.2.5. Experiment na určenie počtu krokov tréningu robota PAWO

Cieľom tohto experimentu s robotom PAWO bolo dosiahnuť čo najefektívnejšie učenie chôdze, teda znížiť počet tréningových krokov potrebných na osvojenie si stabilného a plynulého pohybového správania agentom. Dôraz bol kladený na optimalizáciu celého procesu učenia s cieľom minimalizovať nielen čas potrebný na dosiahnutie požadovaného výsledku, ale aj výpočtovú záťaž simulácie. Skrátenie trvania tréningu má zásadný význam z hľadiska spotreby výpočtových zdrojov a energetickej efektivity, čo je zvlášť dôležité pri použití fyzikálne realistických simulácií s vysokou výpočtovou náročnosťou. Efektívne učenie zároveň umožňuje rýchlejšie testovanie rôznych konfigurácií modelu, čím sa výrazne urýchľuje vývoj správania robota.

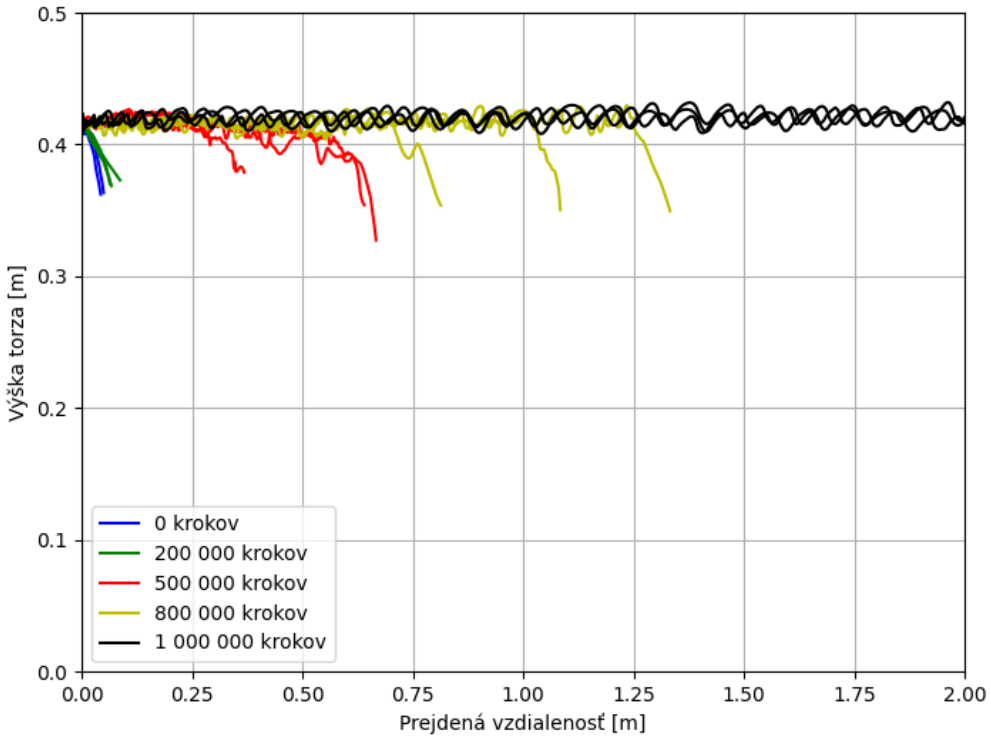
Na obrázku 6.7 je zobrazený priebeh odmeny v tréningu robota PAWO, ktorá predstavuje priemernú hodnotu dosiahnutú z 10 000 krokov tréningu. Tento spôsob vizualizácie umožňuje eliminovať krátkodobé výkyvy a lepšie zachytiť dlhodobý trend zlepšovania správania agenta. Súčasne je na obrázku sivou farbou vyznačená smerodajná odchýlka, ktorá poskytuje informáciu o miere variability odmeny v danom intervale. Šedé oblasti tak vizuálne znázorňujú spoľahlivosť a stabilitu učenia — užšie pásmo odchýlky signalizuje konzistentnejšie správanie agenta, zatiaľ čo širšie pásmo poukazuje na vyššiu variabilitu výsledkov



Obrázok 6.7: Kumulatívna odmena v závislosti od počtu krokov učenia

Výsledky experimentu ukázali, že pri optimálnom nastavení odmenovej funkcie agent nadobudol základné schopnosti stabilnej chôdze približne po 1 000 000 krokoch učenia, čo je vidno na obrázku 6.8. Približne od tohto momentu vykazoval robot PAWO plynulý, vyvážený a opakovateľný pohyb vpred do vzdialenosti aspoň dvoch metrov, ktorý bol vo väčšine testovacích scenárov dostatočne stabilný na to, aby epizódy pokračovali až do vyčerpania maximálne povoleného počtu krokov, bez pádov alebo porušenia rovnováhy.

V porovnaní s robotom BD-1, ktorý je najpodobnejší robotovi PAWO spomedzi modelov prezentovaných v tejto práci, možno konštatovať, že PAWO dosiahol základný cieľ, stabilnú a opakovateľnú chôdzu v kratšom čase a s nižšou tréningovou komplexnosťou. Treba však poznamenať, že BD-1 sleduje odlišný cieľ: namiesto čisto funkčného pohybu je navrhnutý ako expresívny robot, schopný plynule prechádzať medzi rôznymi typmi správania (napríklad státie, chôdza, naklonenie) a vyjadrovať emócie prostredníctvom pohybu[13].



Obrázok 6.8: Testovanie vplyvu počtu tréningových krokov na kumulatívnu odmenu

6.2.6 Zhrnutie tréningu robota PAWO

Proces tréningu robota PAWO prešiel sériou experimentov, ktoré postupne rozširovali odmenovú funkciu o nové zložky s cieľom dosiahnuť stabilnú chôdzu. Na začiatku bola testovaná základná konfigurácia s jednoduchou odmenou za pohyb vpred, ktorá síce umožnila základný pohyb, no bola náchylná na stratu rovnováhy a nežiaduce oscilácie.

V ďalších fázach boli do odmenovej funkcie pridávané penalizácie za náklon torza, veľkosť vykonávaných akcií, odchýlky vo výške ťažiska a náklony okolo jednotlivých osí. Tieto úpravy výrazne prispeli k zlepšeniu stability, zníženiu výskytu pádov a zvýšeniu konzistencie pohybu naprieč epizódami. Výsledky ukázali, že pridanie týchto regulačných prvkov umožnilo agentovi naučiť sa efektívnejšie pohybovať, pričom čoraz častejšie epizódy končili vyčerpaním maximálneho počtu krokov namiesto predčasného ukončenia pádom. Optimálne nastavenie odmenovej funkcie sa nachádza v tabuľke 6.8.

Záverečné experimenty sa zamerali na optimalizáciu počtu tréningových krokov a identifikáciu minimálneho potrebného množstva na dosiahnutie požadovaného správania.

Ukázalo sa, že pri vhodne zvolenej konfigurácii bolo možné dosiahnuť stabilnú chôdzu už po približne 1 000 000 krokoch. Tento výsledok poukazuje na praktickú efektivitu navrhnutej architektúry a správne nastavenej odmenovej funkcie.

Celkovo možno konštatovať, že tréning robota PAWO potvrdil účinnosť algoritmu PPO v spojení so starostlivo navrhnutou odmenovou funkciou. Experimenty zároveň ukázali, že aj pri zložitejších úlohách, ako je chôdza robota, možno v simulácii dosiahnuť spoľahlivé a realistické výsledky, ktoré predstavujú dobrý základ pre ďalšie nasadenie v reálnom svete.

7 Záver

Hlavným cieľom tejto diplomovej práce bolo oboznámiť sa so simulačným prostredím PyBullet, podrobne preskúmať konštrukciu dvojnohého robota PAWO a vytvoriť jeho funkčný model, ktorý adekvátne reprezentuje fyzikálne vlastnosti reálneho robota. Model bol implementovaný a integrovaný do prostredia PyBullet, pričom bol doplnený o programové rozhranie kompatibilné so systémom Gymnasium, čo umožnilo jeho efektívne využitie pri trénovaní algoritmov RL. Výsledný simulačný model bol dôsledne otestovaný s cieľom overiť jeho stabilitu, funkcionalitu a pripravenosť na proces učenia, ktorého cieľom bolo naučiť robota pohybovať sa vpred s využitím algoritmu PPO.

V rešeršnej časti práce bola spracovaná teoretická analýza metód RL so zameraním na algoritmus PPO. Ďalej boli porovnané tri najčastejšie používané simulačné prostredia v oblasti robotiky – PyBullet, MuJoCo a Gazebo – pričom sa hodnotili ich charakteristické vlastnosti. Súčasťou rešerše bola aj analýza vybraných humanoidných robotov, zameraná najmä na ich fyzikálne obmedzenia a spôsob interakcie s okolím.

V praktickej časti práce bolo vytvorenie presného modelu robota PAWO v simulačnom prostredí pomocou URDF súboru, ktorý sa následne implementoval do PyBulletu. Najskôr sa vytvoril jednoduchý model robota zložený z kvádrových častí pre osvojenie si konštruovania v URDF súbore, následne sa rovnaký model vytvoril aj v MuJoCo štruktúre, čím sa vytvoril základ na porovnanie oboch prístupov. Konštrukcia jednotlivých častí modelu prebiehala v Inventori, výsledné 3D komponenty boli exportované do formátu STL a integrované do URDF a MuJoCo štruktúry, kde boli detailne definované všetky telesá, kĺby a ich dynamické správanie. Ďalej sa pokračovalo iba s modelom vytvoreným v URDF súbore, pomocou ktorého sa vytvoril funkčný model robota PAWO.

Na pochopenie fungovania algoritmu PPO bol najprv použitý štandardizovaný simulačný model Walker2D. Tento model poskytol spoľahlivé testovacie prostredie na experimentovanie s rôznymi variantmi odmenovej funkcie, ktoré sa postupne rozširovali od jednoduchej odmeny za rýchlosť vpred cez výšku torza, penalizácie za veľké akcie až po náklon torza. Tieto experimenty viedli k identifikácii kombinácií parametrov, ktoré významne zlepšili stabilitu a plynulosť chôdze. Následne sa testoval minimálny počet krokov učenia a nastavovali sa hyperparametre tak, aby bol tréning čo najefektívnejší z hľadiska výpočtových nákladov. Výsledky z Walker2D poskytli dôležité poznatky, ktoré boli prenesené do učenia modelu PAWO.

Po overení vhodnosti odmenovej funkcie a hyperparametrov nasledovala aplikácia týchto poznatkov na vlastný simulačný model robota PAWO. V prostredí PyBullet bolo vytvorené kompletné programové rozhranie, definovaný akčný a stavový priestor, ako aj mechanizmus interakcie agenta s prostredím. Už v úvode tréningu sa ukázalo, že vhodné nastavenie stavového priestoru má zásadný vplyv na správanie agenta a priebeh učenia. Samotný tréning prebiehal s rôznymi kombináciami zložiek odmenovej funkcie vrátane penalizácie za náklony, odmeny za výšku torza, udržiavanie smeru, pohyb vpred a

obmedzenia veľkých akcií. Pri optimálnom nastavení robot úspešne prešiel vzdialenosť aspoň dva metre po približne 1 000 000 krokoch učenia, pričom väčšina epizód skončila až po vyčerpaní maximálneho počtu krokov, nie v dôsledku pádu alebo straty rovnováhy. Tieto výsledky potvrdzujú, že vytvorený model v kombinácii s algoritmom PPO umožňuje efektívne trénovať dvojnohého robota na stabilný pohyb, čím boli splnené hlavné ciele práce.

Hoci sa podarilo dosiahnuť stabilný pohyb robota a úspešný prechod požadovanej vzdialenosti, samotná chôdza zatiaľ nepôsobí vizuálne prirodzene. V určitých momentoch je možné pozorovať miernu nekoordinovanosť pohybov, čo poukazuje na potenciál ďalších zlepšení, napríklad úpravou odmenovej funkcie alebo jemnejším doladením parametrov učenia. Zároveň by bolo vhodné overiť robustnosť navrhnutého riešenia aj v inom simulačnom prostredí, ako je MuJoCo, ktoré ponúka presnejšiu fyzikálnu simuláciu a vyšší výpočtový výkon. To by mohlo prispieť k realistickejšiemu modelovaniu pohybu a presnejšiemu doladeniu chôdze. Uvedené odporúčania predstavujú prirodzené pokračovanie výskumu a smerujú k vytvoreniu ešte spoľahlivejšieho a dôveryhodnejšieho modelu chôdze dvojnohého robota.

Literatúra

[1] SUTTON, Richard S.; BARTO, Andrew G. *Reinforcement Learning: An Introduction* [online]. 2. vyd. 2018 [cit. 2025-04-28]. Dostupné z: https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf

[2] KAELBLING, Leslie; LITTMAN, Michael; MOORE, Andrew. *Reinforcement Learning: A Survey* [online]. 1996 [cit. 2025-04-28]. Dostupné z: https://www.jair.org/index.php/jair/article/view/10166/24110

[3] BHATT, Shweta. *Reinforcement Learning 101*. Medium [online]. 2018 [cit. 2025-04-28]. Dostupné z: https://medium.com/data-science/reinforcement-learning-101-e24b50e1d292

[4] LI, Yuxi. *Deep reinforcement learning* [online]. 2018 [cit. 2025-04-28]. Dostupné z: https://arxiv.org/pdf/1810.06339

[5] SCHULMAN, John; WOLSKI, Filip; DHARIWAL, Prafulla; RADFORD, Alec; KLIMOV, Oleg. *Proximal Policy Optimization Algorithms* [online]. 2017 [cit. 2025-04-28]. Dostupné z: https://arxiv.org/abs/1707.06347

[6] COUMANS, Erwin. *PyBullet – Python module for physics simulation for games, robotics and machine learning* [online]. 2017 [cit. 2025-04-28]. Dostupné z: https://pybullet.org

[7] TODOROV, Emanuel; EREZ, Tom; TASSA, Yuval. *MuJoCo: A physics engine for model-based control* [online]. 2012 [cit. 2025-04-28]. Dostupné z: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6386109

[8] MuJoCo Documentation. Model Gallery [online]. 2025 [cit. 2025-04-28]. Dostupné z: https://mujoco.readthedocs.io/en/stable/models.html

[9] KOENIG, Nathan; HOWARD, Andrew. *Design and use paradigms for Gazebo, an open-source multi-robot simulator* [online]. 2004 [cit. 2025-04-28]. Dostupné z: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1389727

[10] Gazebo Simulator [online]. 2025 [cit. 2025-04-28]. Dostupné z: https://gazebosim.org/home

Literatúra

[11] RAFFIN, Antonin; HILL, Ashley; GLEAVE, Adam; KANERVISTO, Anssi; ERNESTUS, Maximilian; DORMANN, Noah. *Stable-Baselines3: Reliable Reinforcement Learning Implementations* [online]. 2021 [cit. 2025-04-28]. Dostupné z: https://www.jmlr.org/papers/volume22/20-1364/20-1364.pdf

[12] TOWERS, Mark; KWIATKOWSKI, Ariel; TERRY, Jordan; BALIS, John U.; DE COLA, Gianluca; DELEU, Tristan; GOULÃO, Manuel; KALLINTERIS, Andreas; KRIMMEL, Markus; KG, Arjun; PEREZ-VICENTE, Rodrigo; PIERRÉ, Andrea; SCHULHOFF, Sander; TAI, Jun Jet; TAN, Hannah; YOUNIS, Omar G. *Gymnasium: A Standard Interface for Reinforcement Learning Environments* [online]. 2024 [cit. 2025-04-28]. Dostupné z: https://arxiv.org/abs/2407.17032

[13] GRANDIA, Ruben; KNOOP, Espen; HOPKINS, Michael A.; WIEDEBACH, Georg; BISHOP, Jared; PICKLES, Steven; MÜLLER, David; BÄCHER, Moritz. *Design and Control of a Bipedal Robotic Character* [online]. 2025 [cit. 2025-04-28]. Dostupné z: https://arxiv.org/abs/2501.05204

[14] Boston Dynamics. *Atlas* [online]. 2025 [cit. 2025-04-28]. Dostupné z: https://bostondynamics.com/atlas/

[15] ROBOTS: Your Guide to the World of Robotics. *Optimus (Tesla Bot)* [online]. 2024 [cit. 2025-04-28]. Dostupné z: https://robotsguide.com/robots/optimus

[16] MALIK, Ali Ahmad; MASOOD, Tariq; BREM, Alexander. *Intelligent humanoids in manufacturing to address worker shortage and skill gaps: Case of Tesla Optimus* [online]. 2023 [cit. 2025-04-28]. Dostupné z: https://arxiv.org/abs/2304.04949

[17] ROBOTS: Your Guide to the World of Robotics. *ASIMO* [online]. 2024 [cit. 2025-04-28]. Dostupné z: https://robotsguide.com/robots/asimo

[18] SHIGEMI, Satoshi. *ASIMO and Humanoid Robot Research at Honda* [online]. 2019 [cit. 2025-04-28]. Dostupné z: https://www.cs.columbia.edu/~allen/S19/ASIMO.pdf

[19] FARAMA FOUNDATION. *Walker2D – Gymnasium Documentation* [online]. 2024 [cit. 2025-04-28]. Dostupné z: https://gymnasium.farama.org/environments/mujoco/walker2d/

Zoznam skratiek a symbolov

RL	Reinforcement Learning
PPO	Proximal Policy Optimization
URDF	Unified Robot Description Format
DRL	Deep Reinforcement Learning
SDF	Simulation Description Format
MJCF	MuJoCo File
XML	Extensible Markup Language
GUI	Graphical User Interface
ROS	Robot Operating System
IMU	Inertial Measurement Unit
GPS	Global Positioning System
SB3	Stable-Baselines3
DQN	Deep Q-Networks
SAC	Soft Actor-Critic
MPC	Model Predictive Control
ZMP	Zero Moment Point
CAD	Computer-Aided Design
STL	Standard Triangle Language
RGBA	Red Green Blue Alpha

Zoznam obrázkov

Zoznam obrázkov

Zoznam tabuliek