Fizikalni simulatori za analizu robotskih sustava

Šabić, Filip

Undergraduate thesis / Završni rad

2023

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Mechanical Engineering and Naval Architecture / Sveučilište u Zagrebu, Fakultet strojarstva i brodogradnje**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:235:557182

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-05-11**

Repository / Repozitorij:

Repository of Faculty of Mechanical Engineering and Naval Architecture University of Zagreb

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Filip Šabić

Zagreb, 2023.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentor:
Dr.sc Petar Ćurković, dipl.ing

Student:
Filip Šabić

Zagreb, 2023.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za završne i diplomske ispite studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment, inženjerstvo materijala i mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa: **602 – 04 / 23 – 6 / 1**	
Ur.broj: **15 - 1703 - 23 -**	

ZAVRŠNI ZADATAK

Student: **Filip Šabić** JMBAG: **0035227616**

Naslov rada na hrvatskom jeziku: **Fizički simulatori za analizu robotskih sustava**

Naslov rada na engleskom jeziku: **Physical simulators for the analysis of robotic systems**

Opis zadatka:

Fizički simulatori omogućuju realistično modeliranje radnog prostora robota uključujući modeliranje gravitacije, trenja, deformacija i kolizije. Primjenom simulacijskih okolina ubrzava se razvoj robotskih aplikacija, posebno u slučaju primjena zahtjevnih računalnih modela poput strojnog učenja. Simulatore možemo podijeliti u dvije osnovne grane – komercijalne simulatore i simulatore otvorenog koda.

U ovom je radu potrebno istražiti dostupne simulatore za obje navedene kategorije. Napraviti usporednu analizu i izdvojiti prednosti odnosno ograničenja svakog od analiziranih simulatora.

U drugom dijelu rada potrebno je izraditi prostorne modele robota pokretanog kotačima te robota pokretanog nogama. Izvesti modele u prikladni format te osigurati njihov prihvat u odabranom fizičkom simulatoru. Napisati upravljačke zakone za kretanje robote u fizičkom simulatoru, uključujući praćenje linije i izbjegavanje prepreke.

U radu je potrebno navesti korištenu literaturu i eventualno dobivenu pomoć

Zadatak zadan:
30. 11. 2022.

Datum predaje rada:
1. rok: 20. 2. 2023.
2. rok (izvanredni): 10. 7. 2023.
3. rok: 18. 9. 2023.

Predviđeni datumi obrane:
1. rok: 27. 2. – 3. 3. 2023.
2. rok (izvanredni): 14. 7. 2023.
3. rok: 25. 9. – 29. 9. 2023.

Zadatak zadao:
Izv. prof. dr. sc. Petar Ćurković

Predsjednik Povjerenstva:
Prof. dr. sc. Branko Bauer

Izjavljujem da sam ovaj rad izradio samostalno koristeći znanja stečena tijekom studija te navedenu literaturu.

Zahvaljujem se mentoru, izv. prof. dr. sc. Petru Ćurkoviću, na stručnim savjetima te pruženoj pomoći tijekom izrade ovog rada.

Filip Šabić

Sadržaj

Popis slika II
Popis tablica III
Sažetak IV
Summary V
1. UVOD 1
2. FIZIKALNI SIMULATORI 2
2.1. Simulatori otvorenog koda 2
2.2. Komercijalni simulatori 3
3.KARAKTERISTIKE SIMULATORA 4
3.1. Gazebo 6
3.2 MuJoCo 8
3.3. PyBullet 10
3.4 RaiSim 12
3.5 Coppelia SIM 13
4.OPIS OSNOVNIH NAREDBI 14
5.PRIMJENA 16
5.1 Praćenje linije 16
5.2 Izbjegavanje prepreke 17
5.3. Robot pokretan dvjema nogama 18
6. ZAKLJUČAK 21
LITERATURA 22
Prilozi 24
I. Programski kod za primjer praćenja linije 25
II. Programski kod za primjer izbjegavanja prepreke 27
III. Programski kod za primjer hoda dvonožnog robota 28

Popis slika

Slika 1. Modeli u fizikalnim simulatorima [3] 2
Slika 2. Grafički prikaz učestalosti korištenja fizikalnih simulatora koje opisujemo [3] 5
Slika 3. Model robota u Gazebo sučelju [10] 7
Slika 4. Struktura Gazebo simulatora [11] 7
Slika 5. Modeli robota u MuJoCo sučelju [12] 9
Slika 6. Četveronožni robot u MuJoCo sučelju [15] 9
Slika 7. Dvonožni robot u PyBullet sučelju [18] 11
Slika 8. Vizualni prikaz nastanka fizikalne simulacije [16] 11
Slika 9. Četveronožni roboti u RaiSim sučelju [19] 12
Slika 10. Model robota prikazan u sučelju CoppeliaSIM [21] 13
Slika 11. Robot pokretan kotačima u PyBullet sučelju 16
Slika 12. Robot pokretan kotačima te prepreka u obliku kocke u PyBullet sučelju 17
Slika 13. Robot pokretan kotačima tijekom zaobilaženja prepreke 17
Slika 14. Robot pokretan nogama u PyBullet sučelju 18
Slika 15. Četveronožni robot u PyBullet sučelju [23] 18
Slika 16. Ovisnost kuta zakreta motora kukova robota o vremenu 19
Slika 17. Ovisnost kuta zakreta motora koljena o vremenu 20
Slika 18. Ovisnost kuta zakreta motora gležnjeva o vremenu 20

Popis tablica

Tablica 1. Karakteristike simulatora koje ćemo opisati 4

Sažetak

U ovom radu predstavljena je razlika između fizikalnih simulatora otvorenog koda i komercijalnih simulatora. Napravljena je usporedna analiza te su opisane prednosti i mane pojedinih fizikalnih simulatora. Nadalje, u drugom dijelu rada su izvedeni modeli robota pokretanog kotačima i robota pokretanog nogama u *PyBullet* simulatoru te su za navedene modele napisani programski kodovi za kretanje robota u fizičkom simulatoru s naglaskom na praćenje linije, izbjegavanje prepreka i na problem simulacije hoda robota pokretanog dvjema nogama.

Ključne riječi: fizikalni simulator, *PyBullet*, praćenje linije, izbjegavanje prepreka.

Summary

This thesis explains the difference between open source physics simulators and commercial simulators. A comparative analysis was made and the advantages and disadvantages of individual physical simulators were described. Furthermore, in the second part of the thesis, the models of the wheel-driven robot and the foot-driven robot were derived in the physics simulator PyBullet, and the codes for the movement of the robot in the physical simulator were written for the mentioned models with an emphasis on line tracking, obstacle avoidance and also solving the problem of simulating the walk of a two-legged robot.

Keywords: physics simulator, PyBullet, line tracking, obstacle avoidance.

1. UVOD

Simulacija je jedna od najraznovrsnijih tema s kojima se danas inženjer može susresti na radnom mjestu. Također može biti jedna od najvažnijih za tvrtke, bez obzira na industriju. Simulacija utječe na kvalitetu, sigurnost i produktivnost, bez obzira na to nastaju li problemi u uredu, u trgovini ili u skladištu. Simulacija se često koristi kao alat za povećanje proizvodnog kapaciteta. [1]

Povijest računalne simulacije seže do Drugog svjetskog rata, kada su se dva matematičara Jon von Neumann i Stanislaw Ulam bavila zagonetnim problemom ponašanja neutrona. Eksperimenti testiranja bili su preskupi, a problem previše kompliciran za analizu. Zbog toga su matematičari izmislili tehniku ruletnog pravila. [1]

Zamoran je zadatak fizički dizajnirati robota i testirati ga u svakoj fazi. Moderni simulatori robota mogu nam pomoći u izvođenju napornih fizikalnih simulacija, 3D vizualizacija, virtualnog modeliranja robota i vrhunskih istraživanja koje nam štede vrijeme i novac.

U prvom dijelu rada opisane su dvije grupe dostupnih fizikalnih simulatora: komercijalni simulatori i simulatori otvorenog koda. Drugi dio rada obuhvaća primjere rješenja problema praćenja linije i izbjegavanja prepreke, kao i problem simulacije hoda dvonožnog robota u programskom paketu *PyBullet*.

2. FIZIKALNI SIMULATORI

Fizikalni simulator sastoji se od zakona fizike, matematike i geometrije koji objedinjeni stvaraju matematički model onoga što se želi simulirati, na primjer robota. Iz dobivenog matematičkog modela potom se stvara numerički model koji služi kao podloga za programiranje u računalnom programu. Na temelju programa moguće je dobiti predodžbu o tome kako bi se neki sustav ponašao u stvarnom svijetu. [2]

[Slika 1] prikazuje simulirane modele u nekim od fizikalnih simulatora: pod slovom A je model u simulatoru *MuJoCO*, pod slovom B je model u *PyBulletu*, a pod slovom C je prikazan model u *Gazebo* simulatoru.

Slika 1. Modeli u fizikalnim simulatorima [3]

2.1. Simulatori otvorenog koda

Softver otvorenog koda javno je dostupan kod koji svatko može pregledavati, mijenjati i redistribuirati kako smatra prikladnim. Softver otvorenog koda razvija se na decentraliziran i suradnički način, temeljen na recenziji i radu zajednice.

Otvoreni kod promiče univerzalni pristup putem otvorenog koda, besplatnih licenci za projekt ili plan proizvoda i opću redistribuciju tog projekta ili plana. Otvoreni kod postao je popularan s razvojem interneta. Pokret softvera otvorenog koda osnovan je za rješavanje pitanja autorskih prava, licenciranja, domena i potrošača.

Neki od fizikalnih simulatora otvorenog koda su *Gazebo, Webots, CARLA, MuJoCo* te *PyBullet*.

2.2. Komercijalni simulatori

Komercijalni simulatori ili simulatori zatvorenog koda su intelektualno vlasništvo tvrtke koja stoji iza kreiranja istih te nije moguća slobodna distribucija softvera.

Softver zatvorenog koda je stabilniji i usmjereniji proizvod, a ukoliko je potrebna tehnička pomoć, lakše je pristupiti korisničkoj podršci. Nedostaci ovog tipa koda su visoka cijena te brzina otklanjanja potencijalnih problema - ukoliko su prisutne greške ili nedostaju značajke, potrebno je čekati da razvojni programeri tvrtke riješe navedeno. Budući da se softver zatvorenog koda koristi u kontroliranom okruženju s grupom ljudi, ima sigurnosnu prednost. Samo pružateljima softvera je dopušteno mijenjati izvorni kod. Kao rezultat toga, manje su šanse za hakiranje sustava i druge ranjivosti. [4]

Iako softver otvorenog koda dolazi s vrlo fleksibilnim izvornim kodom, još uvijek teško zadovoljava poslovne potrebe jer previše fleksibilnosti u kodu može biti problem za tvrtke. Stoga je poželjniji softver zatvorenog koda jer ažuriranje istog osiguravaju programeri tvrtke koja je vlasnik softvera, te isti potvrđuju da je to pouzdan proizvod.

Primjeri komercijalnih fizičkih simulatora su *Raisim, HAVOK, CoppeliaSIM, X-plane*, itd.

3.KARAKTERISTIKE SIMULATORA

U [Tablica 1] je prikazana dostupnost osnovnih karakteristika simulatora koji su opisani u nastavku ovog rada.

Tablica 1. Karakteristike simulatora

	ROS	Inverzna kinematika	Više fizikalnih *engine*-a	*Path planning*	Inverzna dinamika
Gazebo	+	+	+	+	+
MuJoCo	-	-	-	-	+
PyBullet	-	+	-	+	+
RaiSim	-	+	-	-	-
CoppeliaSIM	+	-	+	+	-

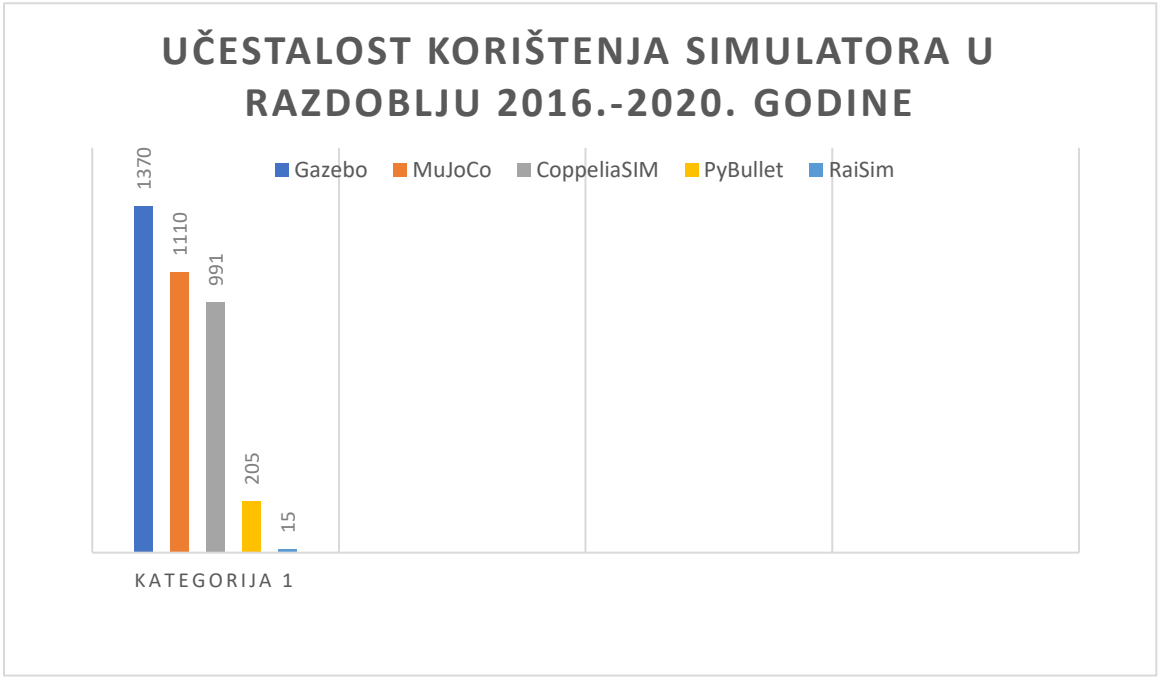
Robot Operating System (*ROS*) je skup softverskih biblioteka i alata otvorenog koda koji nam pomažu u izradi robotskih aplikacija. Sadrži niz alata, od upravljačkih programa do najsuvremenijih algoritama sa snažnim razvojnim alatima. [5]

Inverzni kinematički problem određuje varijable zglobova koje odgovaraju zadanom položaju i orijentaciji vrha manipulatora. Rješavanje ovog problema je jako kompleksno jer postoji beskonačno mnogo rješenja. [6]

Inverzni dinamički problem nam određuje gibanja zglobova iz poznatih opterećenja na zglobove. [6]

Path planning – Računski problem za pronalaženje konfiguracija za pokretanje robota do odredišne točke najbržim i/ili najjednostavnijim putem. Ulazni parametri za *Path Planning* su mapa okruženja te konačni cilj, odnosno konačna pozicija u koju želimo dovesti robota. [7]

Graf na [Slika 2] predstavlja učestalost korištenja određenih simulatora u razdoblju od 2016. do 2020. godine. [3]



Slika 2. Grafički prikaz učestalosti korištenja fizikalnih simulatora koje opisujemo [3]

3.1. Gazebo

Gazebo je fizikalni 3D simulator otvorenog koda integriran na *ODE* fizikalnom *engine*-u koji je bio komponenta Player Projecta od 2004. do 2011. godine kada je postao samostalan projekt kojeg je razvijao laboratorij *Willow Garage*. [8]

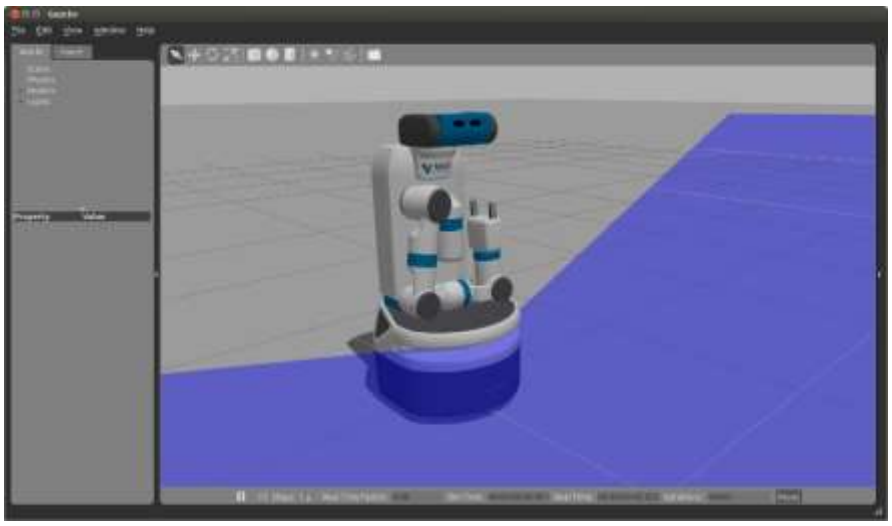
Kao predvodnik u robotskim simulacijama, sjajna stvar kod Gazebo-a je što je moguće stvoriti dinamičku simulaciju koristeći više fizikalnih *engine*-a visokih performansi kao što su *ODE, Simbody, Bullet i DART*. [8]

DART je *Gazebo Physics*-ov zadani *engine* za fiziku i nudi točnost koja nadilazi čak i softvere namijenjene za kreiranje igrica. Prednost ovog simulatora je da ima jako bogatu knjižnicu punu modela i sučelja koji su dostupni za korištenje te postoji mogućnost integracije i testiranja višestrukih senzora za koje nam softver pruža alate, kao i alate za dizajniranje robota kako bismo iz njih dobili maksimum. Zbog svoje homogene veze s *ROS*-om, može se koristiti isto kontrolno sučelje za simulaciju i stvarne fizičke sustave. [8]

Budući da je *Gazebo* softver otvorenog koda, postoje i mnogi dodaci i rješenja trećih strana koji pomažu riješiti specifične probleme na koje bismo mogli naići te ubrzavaju naš tijek rada.

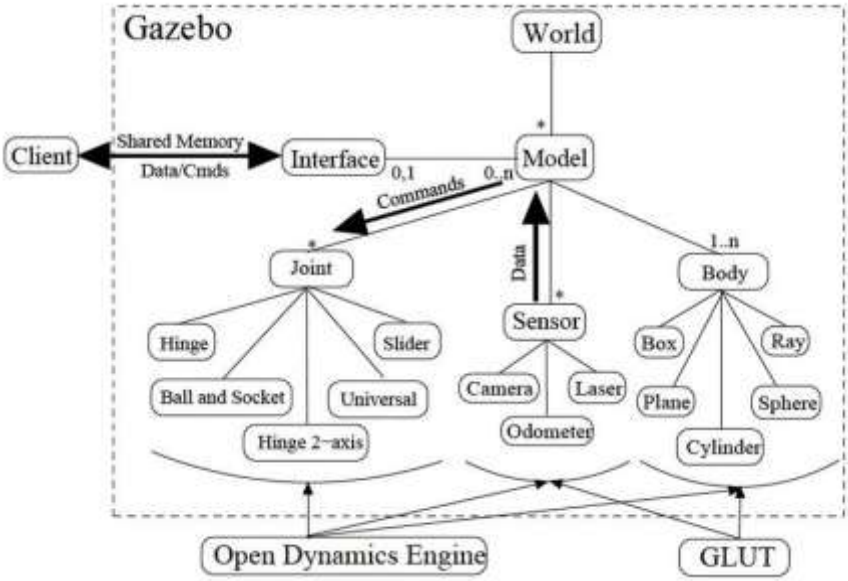
Problemi koji bi se mogli izdvojiti kod ovog softvera je što je jako teško uvesti 3D modele u softver, pripremiti datoteke za softver te je također jako teško instalirati softver na *Windows* budući da ukupna instalacija ima 18 koraka. Također, u softveru ne postoji mogućnost dodavanja deformabilnih objekata. [9]

Na [Slika 3] je prikazan model robota unutar *Gazebo* sučelja.



Slika 3. Model robota u *Gazebo* sučelju [10]

Na [Slika 4] je shematski prikaz strukture *Gazebo* simulatora.



Slika 4. Struktura *Gazebo* simulatora [11]

3.2 MuJoCo

MuJoCo (*Multi-Joint dynamics with Contact*) je besplatni fizikalni simulator otvorenog koda dizajniran za olakšavanje istraživanja i razvoja u robotici, biomehanici, grafici i animaciji i drugim područjima gdje je potrebna brza i precizna simulacija. Razvio ga je *Robots LLC* i bio je dostupan kao komercijalni simulator od 2015. do 2021. godine kada je postao simulator otvorenog koda dostupan svima. [12] Programski jezik koji se koristi je C ili C++. Korisnik definira modele deskriptivnim jezikom u formatu koji je dizajniran na način da bude čitljiv i prilagođen uređivanju.

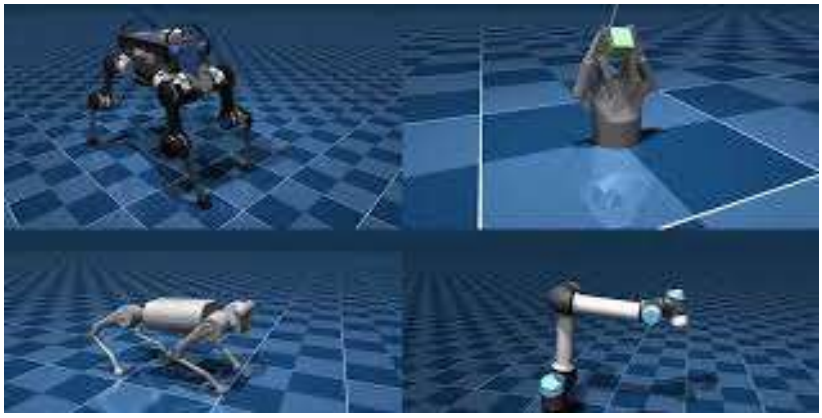
MuJoCo ima mnoštvo funkcionalnosti od kojih su najznačajnije korištenje generaliziranih koordinata u kombinaciji s modernom kontaktnom dinamikom, modeli su u *.mjcf* formatu koji zauzimaju jako malo prostora te su optimirani za proračune, mogućnost automatskog generiranja deformabilnih objekata te razdvojenost modela i podataka za simulaciju. [12]

Kao i drugi simulatori krutog tijela, *MuJoCo* izbjegava sitne detaljne deformacije na mjestu kontakta i radi mnogo brže nego u stvarnom vremenu. Za razliku od drugih simulatora, rješava kontaktne sile korištenjem konveksnog Gaussovog principa, koji osigurava jedinstvena rješenja i dobro definiranu inverznu dinamiku. *MuJoCo* je fleksibilan i pruža više parametara koji se mogu prilagoditi širokom rasponu fenomena kontakta te pruža izbor između *Eulerove* i *Runge-Kutta* metode za numeričku analizu. [13]

U brojnim eksperimentima *MuJoCo* se pokazao iznimno kvalitetnim, na razini *Gazebo* simulatora. Lošije rezultate je pokazao samo u simulaciji sfernih modela. [14]

Unutar simulatora ne postoji mogućnost *Path planning*-a: ako prepustimo modelu potpunu autonomnost i zadamo mu samo konačni položaj, on ne može odrediti najkraći put ili put s najmanje prepreka. Također, za razliku od *Gazebo*-a, *MuJoCo* nema podršku za *ROS* te koristi samo jedan fizikalni *engine* koji je njegov vlastiti. [3]

Na [Slika 5] prikazani su modeli robota u *MuJoCo* sučelju. [Slika 6] prikazuje robota pokretanog četirima nogama te pripadajući kod unutar *MuJoCo* sučelja.



Slika 5. Modeli robota u *MuJoCo* sučelju [12]

Slika 6. Četveronožni robot u *MuJoCo* sučelju [15]

3.3. PyBullet

Pybullet je Python modul otvorenog koda kojeg je razvio Erwin Coumans za fizikalne simulacije, robotiku, igrice, vizualne efekte i strojno učenje, a temeljen je na *Bullet Physics SDK*. Karakterizira ga velika zajednica ljudi koja razvija simulacijska okruženja kao projekt otvorenog koda te nudi kvalitetnu podršku za početnike. [16]

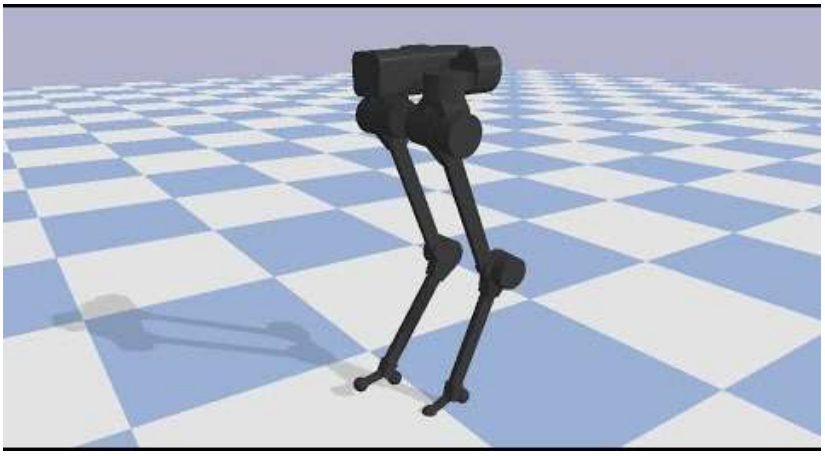
PyBulletom možemo učitati modele iz *.urdf, .sdf, .mjcf* i drugih formata datoteka. *PyBullet* pruža simulaciju direktne dinamike, izračunavanje inverzne dinamike, direktnu i inverznu kinematiku te detekciju sudara. Koristi *Featherstone* zglobna kruta tijela (*btMultiBody*) prema zadanim postavkama, što je prikladnije za zadatke povezane s robotikom. [16] U odnosu na ostale simulatore ima jako jednostavnu instalaciju - moguće ju je pokrenuti jednom linijom koda **py -m pip install pybullet** u Command prompt.

Pip (preferred installer program) potraži paket u *PyPI*-ju te isti instalira u trenutno Python okruženje te tako osigura besprijekoran rad navedenog.

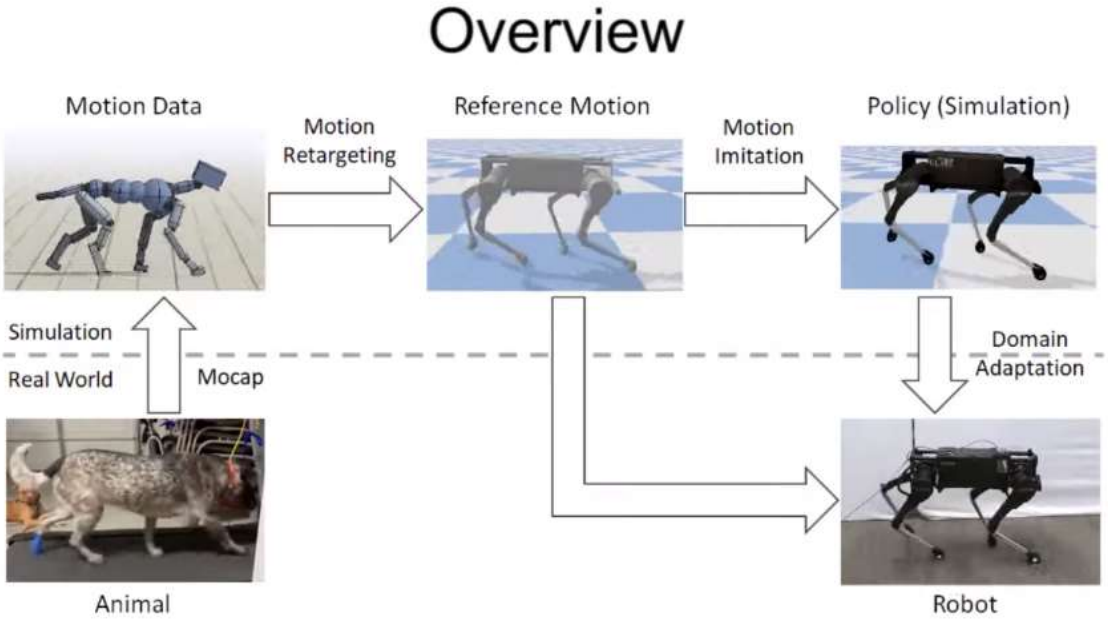
U *PyBullet* je moguće implementirati module za strojno učenje poput *PyBullet Gymperium* koji je implementacija *OpenAI Gym-a* te koristi okruženja *MuJoCo* simulatora ili *PyTorch.* [16]

Budući da se razvoj *PyBulleta* većinom oslanja na zajednicu ljudi, korisnička dokumentacija je i dalje poprilično štura, a primjeri nisu detaljno opisani kao kod drugih simulatora. Također, potrebno je puno detaljnije kalibrirati model u odnosu na druge simulatore zbog prevelike razlike u preciznosti u odnosu na stvarni model. Prikazivanje modela je lošije u odnosu na druge simulatore te *PyBullet*, kao ni *MuJoCo*, nema podršku za *ROS*. [17]

Na [Slika 7] je prikazan model dvonožnog robota u *PyBullet* sučelju, a na [Slika 8] je prikazan vizualiziani model nastanka fizikalne simulacije u PyBullet sučelju.



Slika 7. Dvonožni robot u *PyBullet* sučelju [18]



Slika 8. Vizualni prikaz nastanka fizikalne simulacije [16]

3.4 RaiSim

RaiSim je višeplatformski fizikalni simulator za robotiku i umjetnu inteligenciju razvijen od strane *RaiSim Tech*. U potpunosti podržava *Linux, Mac OS* i *Windows*. *RaiSim* je softver zatvorenog koda i distribuira se s različitim vrstama licenci. Dizajniran je da pruži i točnost i brzinu za simulaciju robotskih sustava. Međutim, to je i generički simulator krutog tijela i može vrlo učinkovito simulirati bilo koje kruto tijelo. Temelji se na C++ jeziku. [19]

Raisim dopušta uvođenje neravnih terena u simulacije koristeći visinske mape, nije potpuno opremljen poput drugih simulatora koje smo opisali već dizajniran za pružanje dinamičkih obrazaca kontakta visoke vjernosti koji su potrebni kako bismo robota prebacili iz simulacije u stvarnost. [3]

Ima svoju knjižnicu modela i okruženja koju su provjerili njihovi programeri te ima dostupnu besplatnu akademsku licencu. [19]

Senzori unutar simulatora su za sada samo u eksperimentalnoj fazi, kao i svaki komercijalni softver potrebna je licenca da bi se isti mogao u potpunosti koristiti, licenca košta 1000 dolara godišnje za jedno računalo. Također, nema podršku za *ROS,* koristi samo jedan fizikalni *engine* poput *MuJoCo*-a te je moguće strojno učenje napraviti samo na kontrolnoj razini te nema mogućnosti za *Path planning*.[19]

Na [Slika 9] prikazani su modeli četveronožnih robota u *RaiSim* sučelju.

Slika 9. Četveronožni roboti u *RaiSim* sučelju [19]

3.5 Coppelia SIM

CoppeliaSim, ranije poznat kao *V-REP*, je fizikalni simulator koji se koristi u industriji, obrazovanju i istraživanju. Izvorno je razvijen unutar *Toshiba R&D*-a, a trenutno ga aktivno razvija i održava *Coppelia Robotics AG*, mala tvrtka sa sjedištem u Zürichu u Švicarskoj. [20]

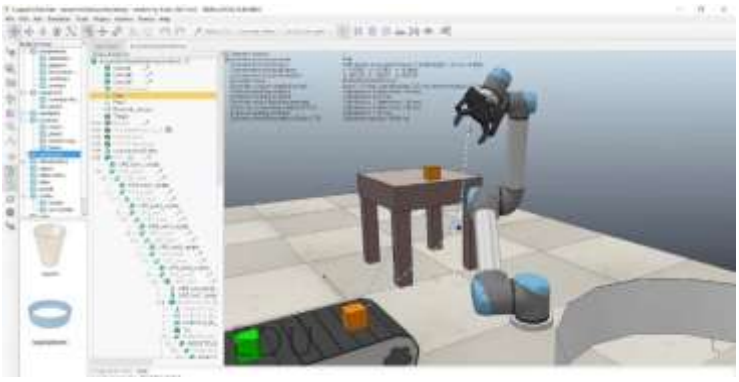
Ima integrirano razvojno okruženje temeljeno na distribuiranoj upravljačkoj arhitekturi: svakim objektom/modelom može se upravljati zasebno putem integrirane skripte, dodatka, *ROS* čvora, udaljenog API klijenta ili prilagođenog rješenja. Upravo to čini *CoppeliaSim* vrlo svestranim i idealnim za aplikacije s više robota. Kontroleri mogu biti napisani u C/C++, *Python, Java, Lua, Matlab* ili *Octave* programskim jezicima. [20]

CoppeliaSim koristi se za brzi razvoj algoritama, simulaciju tvorničke automatizacije, brzu izradu prototipova i validaciju, obrazovanje o robotici, daljinski nadzor, dvostruke sigurnosne provjere i kreiranje digitalnih blizanaca. [3]

Simulator koristi čak 5 različitih fizikalnih *engine*-a: *MuJoCo, Bullet, Vortex, Newton i ODE.* Kao i *PyBullet,* ima mogućnost proračunavanja i direktne i inverzne kinematike te ima mogućnost brzog određivanja minimalnih udaljenost između dvaju točaka modela. *Path planning* podržan je na vrlo fleksibilan način putem OMPL biblioteke umotane u dodatak za *CoppeliaSim*. Ima besplatnu edukacijsku licencu za učenike, učitelje ili hobiste. [20]

U simulatoru nije dostupno izračunavanje inverzne dinamike. Okruženje se sprema u poseban oblik datoteke te se sva uređivanja navedenog moraju raditi kroz sučelje CoppeliaSIM-a. [3]

Na [Slika 10] je prikazan model robota prikazan u sučelju *CoppeliaSIM*.



Slika 10. Model robota prikazan u sučelju *CoppeliaSIM* [21]

4.OPIS OSNOVNIH NAREDBI

PyBullet je dizajniran oko API-ja vođenog klijentom i poslužiteljem, pri čemu klijent šalje naredbe, a fizički poslužitelj vraća status. PyBullet ima neke ugrađene poslužitelje za fiziku: DIRECT i GUI. I GUI i DIRECT veze izvršit će simulaciju fizike i renderiranje u istom procesu kao i PyBullet. U načinu rada u DIRECT ne možemo pristupiti hardverskim značajkama OpenGL i VR, ali nam omogućuje renderiranje slika. [22]

connect(p.GUI)- naredba za povezivanje s poslužiteljem, ulazne varijable su pybullet.GUI te pybullet.DIRECT. GUI stvara grafičko sučelje, dok DIRECT naredbe šalje izravno u fizikalni *engine*.

loadURDF() – naredba koja nam omogućuje učitavanje fizikalnih modela iz .urdf datoteke koja opisuje i definira geometriju robota i ostalih modela.

Dodatni neobavezni argumenti naredbe su: pozicija baze, orijentacija baze, korištenje fiksne baze, primjena faktora za skaliranje modela.

Orijentacija baze stvara bazu objekta orijentiranu prema vektoru koordinata [x, y, z].

setGravity() - naredba uz pomoć koje postavljamo simulaciju gravitacijskog polja unutar simulacije, sile su u Newtonima te su usmjerene u osima x, y i z.

pybullet_data.getDataPath - vraća podatkovnu stazu po kojoj PyBullet traži modele.

getNumJoints() - naredba koja nam daje broj zglobova učitanog modela, ulazni parametar je naziv modela za koji želimo saznati broj zglobova.

getJointInfo() - naredba koja nam za pojedini zglob daje informacije o nazivu zgloba, vrsti zgloba, trenju, gornjoj i donjoj granici slobode gibanja, maksimalnoj sili i maksimalnom ubrzanju, nazivu članka i ostalo. Ulazni parametri su model i oznaka zglob za koji nas zanima navedeno.

p.setJointMotorControl2() - naredba koja se koristi za upravljanje robotom tako da se postavi način upravljanja pojedinog zgloba. Tijekom simulacije fizikalni *engine* simulira motore kako bi se postigla zadana ciljna vrijednost koja se može postići unutar maksimalnih motornih sila i drugih ograničenja. Postoje 3 vrste upravljanja: pozicijom, brzinom i momentom.

getBasePositionAndOrientation()- naredba koja nam daje trenutni položaj i orijentaciju baze tijela u Kartezijevim svjetskim koordinatama. Orijentacija je kvaternion u formatu [x,y,z,w]. Ulazni parametar je model za koji želimo znati navedene informacije.

resetBasePositionAndOrientation() – naredba koja resetira orijentaciju i poziciju baze željenog modela. Najbolje je resetirati navedeno na početku, a ne tijekom simulacije jer će naredba nadjačati učinak svih fizikalnih simulacija. Ulazni paramtar je model za koji želimo resetirati orijentaciju i poziciju baze.

stepSimulation() – naredba koja će izvršiti radnje u jednom koraku simulacije dinamike, kao što je detektiranje sudara, rješavanje ograničenja i integracija. Nisu potrebni nikakvi ulazni parametri, te funkcija nema izlaznih parametara.

setRealTimeSimulation() – naredba koja omogućava fizikalnom *engine*-u da pokreće simulaciju prema svom realnom vremenu, neće postupno pokretati simulaciju osim ako ne pošaljemo naredbu stepSimulation(). Ako je omogućena naredba serRealTimeSimulation(), stepSimulation() nije potreban. Naredba ne radi u DIRECT modu rada.

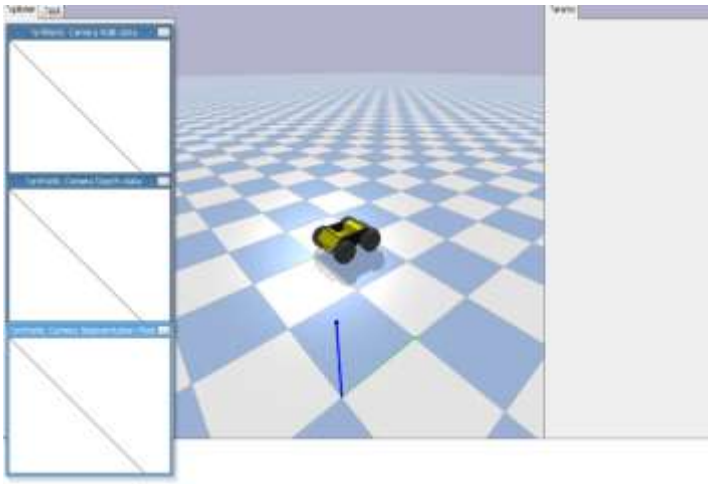
createConstraint() – naredba koja nam stvara ograničenja u kretanju određenog modela, vrste ograničenja su FIXED, POINT2POINT i PRISMATIC. Obvezni ulazni parametri su oznaka roditeljskog tijela, oznaka dijela roditelja koji želimo spojiti, oznaka tijela potomka, oznaka dijela modela potomka koju želimo spojiti s roditeljskim modelom, vrsta ograničenja koji su u cjelobrojnom obliku te os oko kojeg će se isto kreirati, pozicija roditeljskog okvira te pozicija okvira potomka koji su u obliku vektora sa 3 decimalna broja.

5.PRIMJENA

5.1 Praćenje linije

Kao primjer rješavanja problema praćenja linije odabran je robot pokretan kotačima *husky* koji je učitan u PyBullet okruženje iz *.urdf* datoteke u kojoj je izmodeliran. Pomoću programskog koda omogućeno je kretanje robota koji prati imaginarnu liniju u obliku kvadrata. Programski kod ovog primjera je dan u prilogu na kraju rada.

Na [Slika 11] je prikazan model robota pokretanog kotačima u *PyBullet* sučelju tijekom praćenja linije.

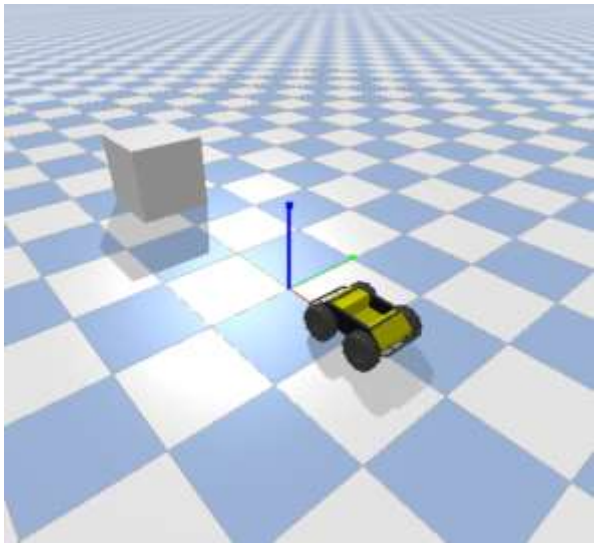


Slika 11. Robot pokretan kotačima u *PyBullet* sučelju

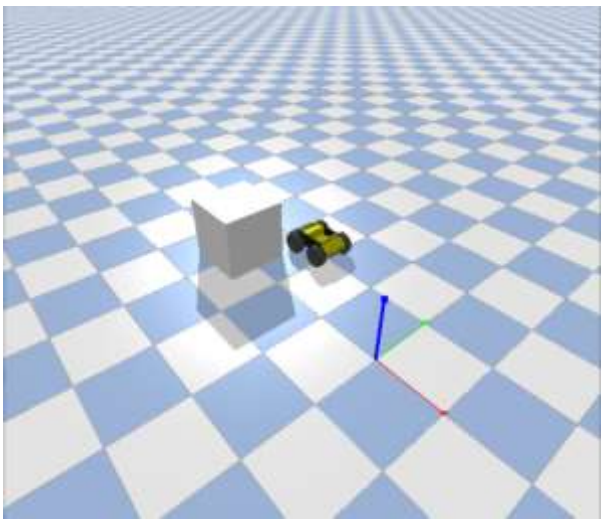
5.2 Izbjegavanje prepreke

Za primjer rješavanja problema izbjegavanja prepreke odabran je isti robot pokretan kotačima kao i u prethodnom primjeru. Učitan je model kocke u *.urdf* obliku koji predstavlja prepreku koju robot zaobilazi . Programski kod je dan u prilogu na kraju zadatka.

[Slika 12] prikazuje robota pokretanog kotačima u *PyBullet* sučelju te prepreku u obliku kocke, dok [Slika 13] prikazuje robota u trenutku zaobilaženja prepreke.



Slika 12. Robot pokretan kotačima te prepreka u obliku kocke u *PyBullet* sučelju

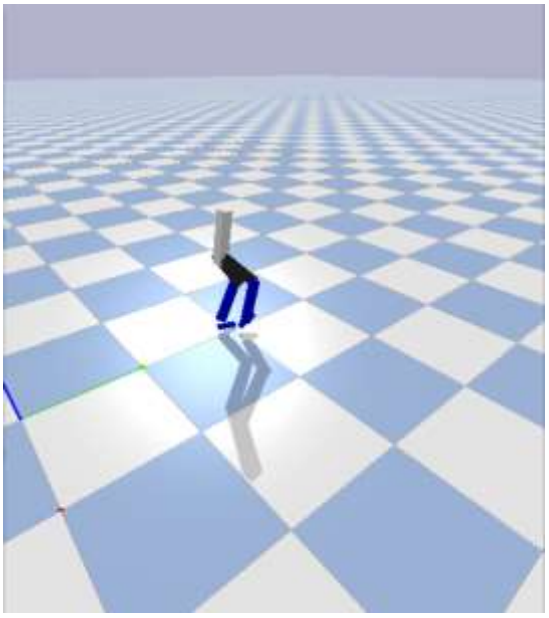


Slika 13. Robot pokretan kotačima tijekom zaobilaženja prepreke

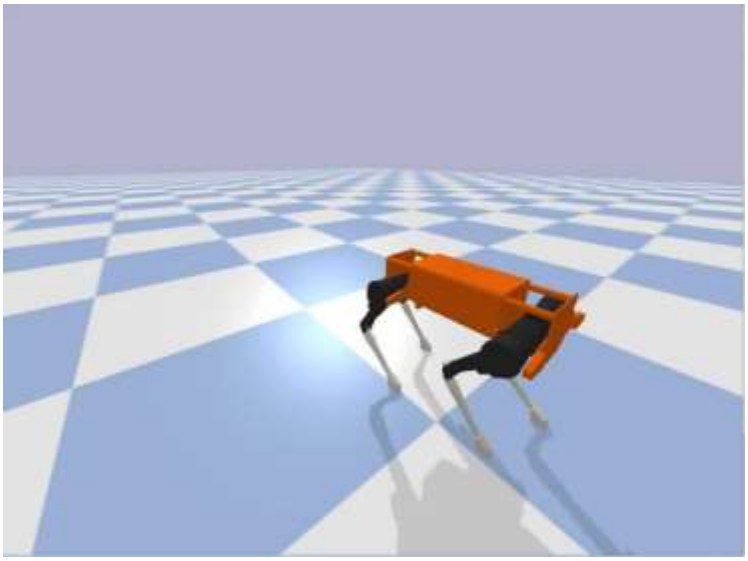
5.3. Robot pokretan dvjema nogama

Rješavanje hoda dvonožnog robota predstavljeno je na primjeru dvonožnog modela u *.urdf* obliku Simulacija se sastoji od 4 međukoraka koji se izmjenjuju tako da robot održava ravnotežu i kreće se poput čovjeka. Mehanizam je jako sličan onom kod četveronožnog robota prikazanog na [Slika 15] uz dodatak stabilizacije trupa u jednom od koraka. Kod za opisani primjer je dan u prilogu na kraju zadatka.

Na [Slika 14] je prikazan model robota pokretanog dvjema nogama unutar *PyBullet* sučelja.

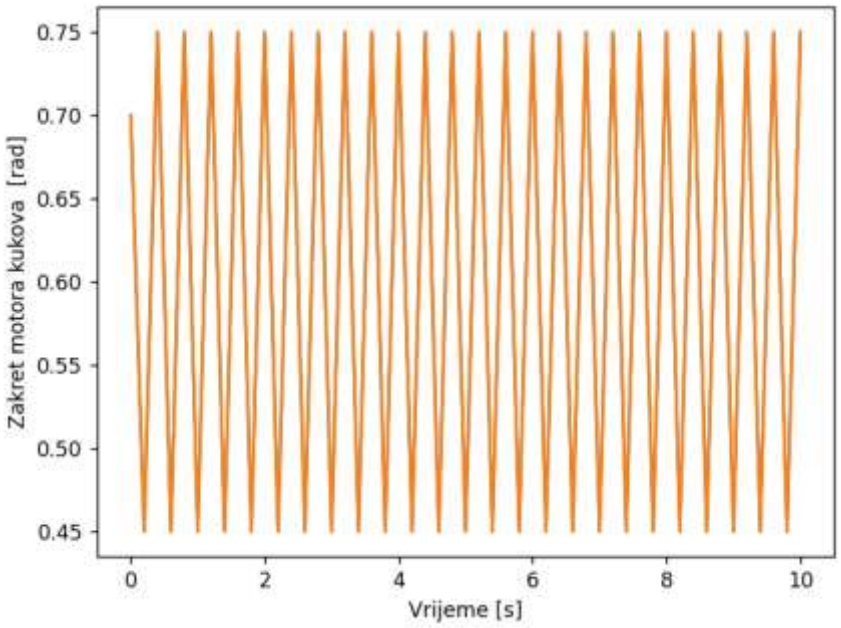


Slika 14. Robot pokretan nogama u *PyBullet* sučelju

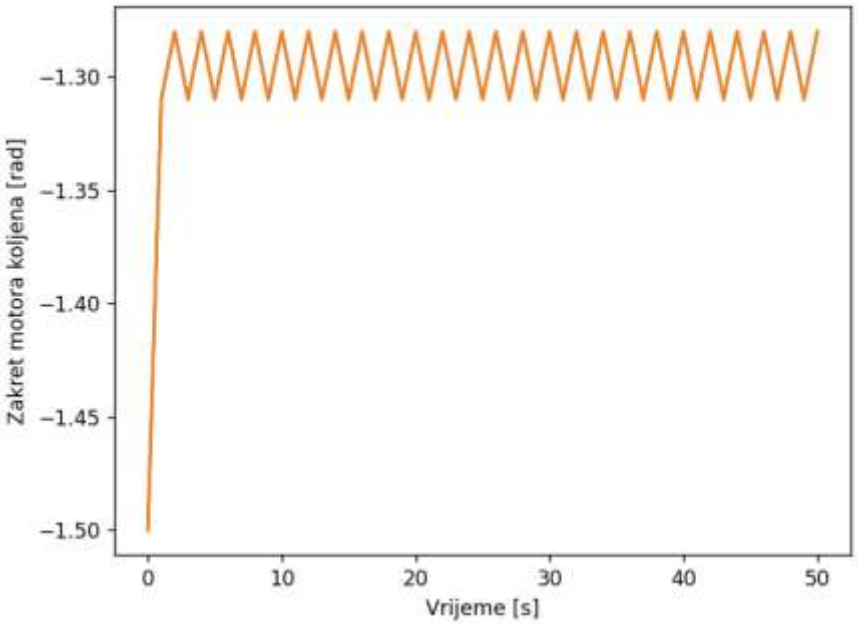


Slika 15. Četveronožni robot u *PyBullet* sučelju [23]

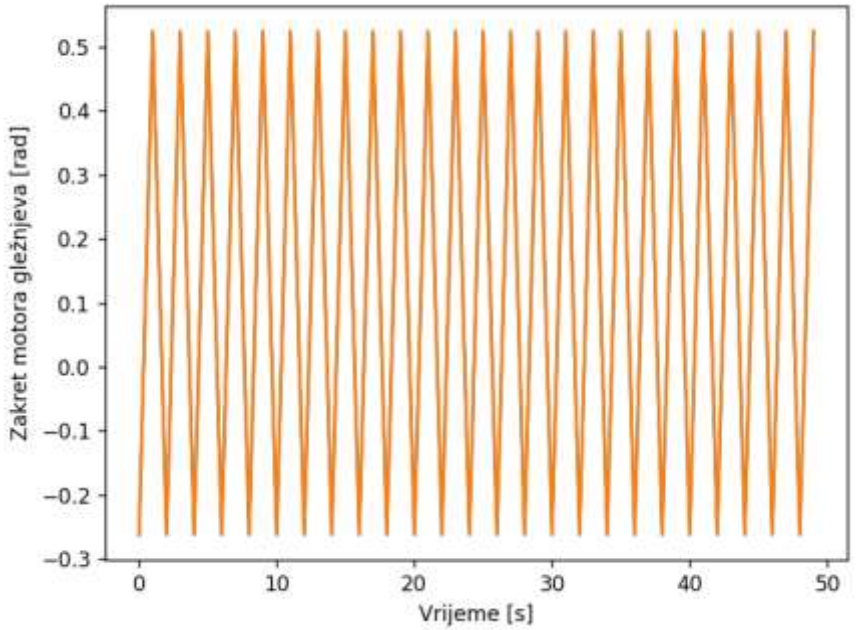
Na [Slika 16],[Slika 17], [Slika 18] je prikazan graf ovisnosti zakreta motora zglobova na modelu robota pokretanog nogama: na [Slika 16] je prikazana ovisnost kuta zakreta motora kukova robota u ovisnosti o vremenu, na [Slika 17] je prikazana ovisnost kuta zakreta motora koljena o vremenu, a na [Slika 18] je prikazana ovisnost kuta zakreta motora gležnjeva o vremenu.



Slika 16. Ovisnost kuta zakreta motora kukova robota o vremenu



Slika 17. Ovisnost kuta zakreta motora koljena o vremenu



Slika 18. Ovisnost kuta zakreta motora gležnjeva o vremenu

6. ZAKLJUČAK

Današnja je industrija nezamisliva bez upotrebe simulatora koji pružaju mogućnost rekonstruiranja elemenata, događaja i pojava iz stvarnosti poput vjetra, kiše, potresa kao i mogućnost poboljšanja već postojećih postrojenja ili pogona složenim proračunima. Tijekom prošlog desetljeća došlo je do unaprjeđenja ovog područja do te razine da danas postoji obilje simulacijskih metoda koje rješavaju mnoge probleme. U ovom je radu predstavljeno samo nekoliko računalnih simulatora s najširom primjenom u industriji, kao i mane i prednosti svakog od njih. Njihovim kolaborativnim korištenjem svakodnevno se postiže napredak na području robotike. Kao primjer su u radu prikazane kretanja robota te implementacija robota složenijeg kretanja, robota pokretanog kotačima i robota pokretanog nogama u fizikalnom simulatoru *PyBullet*. Efikasnost i primjena mobilnih robota svakodnevno se dramatično povećava s razvojem algoritama tako da sva 3 primjera mogu poslužiti kao osnova za daljnji razvoj fizikalnih simulacija kretanja u *PyBulletu.* [24]

LITERATURA

[1] https://uh.edu/~lcr3600/simulation/historical.html (pristup 4.2.2022)

[2] Kenny Erleben, Jon Sporring, Knud Henriksen, and Henrik Dohlmann: Physics-Based Animation, Charles River Media, INC., Massachusetts, 2005.

[3] Jack Collins, Shelvin Chand, Anthony Vanderkop, David Howard: A Review of Physics Simulatorsfor Robotic Applications, IEEE, 9/2021.

[4] Mishal Roomi: Hitchtechwizz, 5/2021.

[5] https://www.ros.org/ (pristup 6.2.2022)

[6] Andrej Jokić: Predavanja iz kolegija Industrijski i mobilni roboti, 2022.

[7] https://www.mathworks.com/discovery/path-planning.html (pristup 6.2.2022)

[8] https://gazebosim.org/features (pristup 4.2.2022)

[9] https://roboticsimulationservices.com/ros-gazebo-everything-you-need-to-know/

(pristup 4.2.2022)

[10] https://docs.fetchrobotics.com/ (pristup 6.2.2022)

[11] https://www.researchgate.net/figure/Architecture-of-Gazebo (pristup 6.2.2022)

[12] https://mujoco.org/ (pristup 4.2.2022)

[13] https://mujoco.readthedocs.io/en/stable/overview.html (pristup 4.2.2022)

[14] Shraddha Goled: DeepMind's Decision To Buy MuJoCo; Good Or Bad?, AIM, 10/2021

[15] https://www.youtube.com/watch?v=0ORsj_E17B0 (pristup 6.2.2022)

[16] https://pybullet.org/ (pristup 4.2.2022)

[17] Andrew Lobbezoo, Hyock-Ju Kwon: Simulated and Real Robotic Reach, Grasp, and Pick-and-Place Using Combined Reinforcement Learning and Traditional Controls, Robotics, 1/2023.

[18] https://www.youtube.com/watch?v=Amru0AXeQRQ (pristup 6.2.2022)

[19] https://raisim.com/ (pristup 6.2.2022)

[20] https://www.coppeliarobotics.com/ (pristup 6.2.2022)

[21] https://www.youtube.com/watch?v=6gAYBNwImoY (pristup 6.2.2022)

[22] https://github.com/bulletphysics/bullet3/blob/master/docs/pybullet_quickstart_guide (pristup 6.2.2022)

[23] J. Kozumplik, "Fizikalni simulator PyBullet", Završni rad, Sveučilište u Zagrebu, Fakultet strojarstva i brodogradnje, Zagreb, 2022. Dostupno na: https://urn.nsk.hr/urn:nbn:hr:235:167190

[24] Ćurković, Petar; Mišković, Luka; Šarančić, David, Legged 3D Printed Mobile Robot // 29th DAAAM International Symposium on Intelligent Manufacturing and Automation. Vienna: DAAAM International, 2018. str. 394-399 doi:10.2507/29th.daaam.proceedings.057

Prilozi

I. Programski kod za primjer praćenja linije

II. Programski kod za primjer izbjegavanja prepreke

III. Programski kod za primjer hoda dvonožnog robota

I. Programski kod za primjer praćenja linije

```
import pybullet as p
import pybullet_data
import time
import math

p.connect(p.GUI) #povezivanje s poslužiteljem
p.setAdditionalSearchPath(pybullet_data.getDataPath())
p.loadURDF("plane.urdf") #učitavanje ravnine
p.setGravity(0, 0, -9.81) #gravitacija djeluje negativno u smjeru z osi
print(pybullet_data.getDataPath) #kod koji nam služi za provjeru odakle
učitavamo .urdf fileove
huskypos=[2, 0, 0.1]
husky = p.loadURDF("husky/husky.urdf",huskypos[0], huskypos[1], huskypos[2])
#.urdf robota pokretanog kotačima

numJoints = p.getNumJoints(husky)
print(numJoints)

for joint in range(numJoints):
    print(p.getJointInfo(husky, joint))

targetPos=-90 #rad
maxForce=10 # N
targetVel= -10 #rad/s

for joint in range(numJoints):

    p.setJointMotorControl(husky, joint, p.VELOCITY_CONTROL, targetVel,
maxForce)

for step in range(600):
    p.stepSimulation()
    time.sleep(1./720.)

for joint in range(numJoints):
    position, orientation = p.getBasePositionAndOrientation(husky)
    p.resetBasePositionAndOrientation(husky, position, [0,0,-1,1])
    p.setJointMotorControl(husky, joint, p.VELOCITY_CONTROL, targetVel,
maxForce)

for step in range(575):
    p.stepSimulation()
    time.sleep(1./720.)

for joint in range(numJoints):
    position, orientation = p.getBasePositionAndOrientation(husky)
```

```
    p.setJointMotorControl(husky, joint, p.VELOCITY_CONTROL, -1*targetVel,
maxForce)
    p.resetBasePositionAndOrientation(husky, position, [0,0,0,1])

for step in range(600):
    p.stepSimulation()
    time.sleep(1./720.)

for joint in range(numJoints):
    position, orientation = p.getBasePositionAndOrientation(husky)
    p.setJointMotorControl(husky, joint, p.VELOCITY_CONTROL, targetVel,
maxForce)
    p.resetBasePositionAndOrientation(husky, position, [0,0,1,1])

for step in range(610):
    p.stepSimulation()
    time.sleep(1./720.)

for joint in range(numJoints):
    position, orientation = p.getBasePositionAndOrientation(husky)
    p.resetBasePositionAndOrientation(husky, position, [0,0,0,1])
    p.stepSimulation()
    time.sleep(1./720.)
```

II. Programski kod za primjer izbjegavanja prepreke

```
import pybullet as p
import pybullet_data
import time
import math


p.connect(p.GUI)
p.setAdditionalSearchPath(pybullet_data.getDataPath())
p.loadURDF("plane.urdf")
p.setGravity(0, 0, -9.81)

huskypos=[2, 0, 0.1]
husky = p.loadURDF("husky/husky.urdf",huskypos[0], huskypos[1], huskypos[2])


cube = p.loadURDF("cube.urdf",-3,0,1)
constraint_cube= p.createConstraint(cube, -1, -1, -1, p.JOINT_POINT2POINT,
[0,0,0], [0,0,0], [-3,0,0])
numJoints = p.getNumJoints(husky)
print(numJoints)


targetPos=-90
maxForce=100
targetVel= -10

for joint in range(numJoints):

    p.setJointMotorControl(husky, joint, p.VELOCITY_CONTROL, targetVel,
maxForce)

for step in range(370):
    p.stepSimulation()
    time.sleep(1./480.)

for joint in range(numJoints):
    position, orientation = p.getBasePositionAndOrientation(husky)
    p.setJointMotorControl(husky, joint, p.VELOCITY_CONTROL, targetVel,
maxForce)
    p.resetBasePositionAndOrientation(husky, position, [0,0,-0.33,1])

for step in range(250):
    p.stepSimulation()
    time.sleep(1./480.)
```

III. Programski kod za primjer hoda dvonožnog robota

```
import pybullet as p
import pybullet_data
import numpy as np
import time
import matplotlib.pyplot as plt
import math
import numpy as np
from time import sleep
print(pybullet_data.getDataPath())

p.connect(p.GUI)
p.setAdditionalSearchPath(pybullet_data.getDataPath())
p.loadURDF("plane.urdf")
p.setGravity(0, 0, -9.81)

biped= p.loadURDF("biped/biped2d_pybullet.urdf",0, 0, 0)
numJoints = p.getNumJoints(biped)

print(numJoints)

for joint in range(numJoints):
  print(p.getJointInfo(biped, joint))

p.setRealTimeSimulation(1)
maxForce1=5


def pocetna_poz():
  sleep(0.2)
  zglob3, zglob4, zglob6, zglob7 = 0.7, -1.5, 0.7, -1.5
  p.setJointMotorControl2(biped, 3, p.POSITION_CONTROL, zglob3, maxForce1)
  p.setJointMotorControl2(biped, 4, p.POSITION_CONTROL, zglob4, maxForce1)
  p.setJointMotorControl2(biped, 6, p.POSITION_CONTROL, zglob6, maxForce1)
  p.setJointMotorControl2(biped, 7, p.POSITION_CONTROL, zglob7, maxForce1)
  return zglob3, zglob4, zglob6, zglob7

def desna_noga_nazad():
  sleep(0.2)
  zglob3, zglob4, zglob5 = 0.75, -1.28, math.pi/6
  p.setJointMotorControl2(biped, 3, p.POSITION_CONTROL, zglob3, maxForce1)
  p.setJointMotorControl2(biped, 4, p.POSITION_CONTROL, zglob4, maxForce1)
  p.setJointMotorControl2(biped, 5, p.POSITION_CONTROL, zglob5, maxForce1)
  return zglob3, zglob4, zglob5
```

```
def desna_noga_naprijed():
  sleep(0.2)
  zglob3, zglob4, zglob5 = 0.45, -1.31, -math.pi/12
  p.setJointMotorControl2(biped, 2, p.POSITION_CONTROL, -math.pi/8, maxForce1)
  p.setJointMotorControl2(biped, 3, p.POSITION_CONTROL, zglob3, maxForce1)
  p.setJointMotorControl2(biped, 4, p.POSITION_CONTROL, zglob4, maxForce1)
  p.setJointMotorControl2(biped, 5, p.POSITION_CONTROL, zglob5,maxForce1)
  return zglob3, zglob4, zglob5

def lijeva_noga_nazad():
  sleep(0.2)
  zglob6, zglob7, zglob8 = 0.75, -1.28, math.pi/6

  p.setJointMotorControl2(biped, 6, p.POSITION_CONTROL, zglob6, maxForce1)
  p.setJointMotorControl2(biped, 7, p.POSITION_CONTROL, zglob7, maxForce1)
  p.setJointMotorControl2(biped, 8, p.POSITION_CONTROL, zglob8, maxForce1)
  return zglob6, zglob7, zglob8

def lijeva_noga_naprijed():
  sleep(0.2)
  zglob6, zglob7, zglob8 = 0.45, -1.31, -math.pi/12

  p.setJointMotorControl2(biped, 6, p.POSITION_CONTROL, zglob6, maxForce1)
  p.setJointMotorControl2(biped, 7, p.POSITION_CONTROL, zglob7, maxForce1)
  p.setJointMotorControl2(biped, 8, p.POSITION_CONTROL, zglob8, maxForce1)
  return zglob6, zglob7, zglob8


SAVING_CRITERIUM = 100
zglobovi = { 3: [], 4: [], 5: [], 6: [], 7: [], 8: []}

counter = 0
zglob3, zglob4, zglob6, zglob7 = pocetna_poz()

zglobovi.get(3).append(zglob3)
zglobovi.get(4).append(zglob4)
zglobovi.get(6).append(zglob6)
zglobovi.get(7).append(zglob7)

def draw(zglobovi):
  plt.ylabel("Zakret motora kukova  [rad]")
  plt.xlabel("Vrijeme [s]")
  space = [i * 0.2 for i in range(0, len(zglobovi.get(3)))]
  #print(space)
  plt.plot(space, zglobovi.get(3))
  plt.plot(space, zglobovi.get(6))
  plt.savefig("3_6.png")
  plt.show()
```

```
  plt.ylabel("Zakret motora koljena [rad]")
  plt.xlabel("Vrijeme [s]")
  plt.plot(range(0, len(zglobovi.get(4))), zglobovi.get(4))
  plt.plot(range(0, len(zglobovi.get(7))), zglobovi.get(7))
  plt.savefig("4_7.png")
  plt.show()
  plt.ylabel("Zakret motora gležnjeva [rad]")
  plt.xlabel("Vrijeme [s]")
  plt.plot(range(0, len(zglobovi.get(5))), zglobovi.get(5))
  plt.plot(range(0, len(zglobovi.get(8))), zglobovi.get(8))
  plt.savefig("5_8.png")
  plt.show()


drawn = False

while(True):
  if drawn == False and counter >= SAVING_CRITERIUM:
      draw(zglobovi)
      drawn = True

  zglob6A, zglob7A, zglob8A = lijeva_noga_naprijed()
  zglobovi.get(6).append(zglob6A)
  zglobovi.get(7).append(zglob7A)
  zglobovi.get(8).append(zglob8A)
  counter += 1
  zglob6B, zglob7B, zglob8B = lijeva_noga_nazad()
  zglobovi.get(6).append(zglob6B)
  zglobovi.get(7).append(zglob7B)
  zglobovi.get(8).append(zglob8B)
  counter += 1
  zglob3A, zglob4A, zglob5A = desna_noga_naprijed()
  zglobovi.get(3).append(zglob3A)
  zglobovi.get(4).append(zglob4A)
  zglobovi.get(5).append(zglob5A)
  counter += 1
  zglob3B, zglob4B, zglob5B = desna_noga_nazad()
  zglobovi.get(3).append(zglob3B)
  zglobovi.get(4).append(zglob4B)
  zglobovi.get(5).append(zglob5B)
  counter += 1
```