Fizikalni simulator PyBullet

Kozumplik, Josip

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Josip Kozumplik

Zagreb, 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentor: Student:

Izv. prof. dr. sc. Petar Ćurković, dipl. ing. Josip Kozumplik

Zagreb, 2022.

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

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

Zahvaljujem se svim kolegama koji su mi na bilo koji način pomogli tijekom preddiplomskog studija.

Posebnu zahvalnost upućujem svojim roditeljima, bratu i djedu na bezuvjetnoj podršci, razumijevanju i vjeri u mene.

Josip Kozumplik

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 / 22 – 6 / 1**	
Ur.broj: **15 - 1703 - 22 -**	

ZAVRŠNI ZADATAK

Student: **Josip Kozumplik** JMBAG: **0035210867**

Naslov rada na hrvatskom jeziku: **Fizikalni simulator PyBullet**

Naslov rada na engleskom jeziku: **PyBullet physical simulator**

Opis zadatka:

Fizikalni simulatori danas su neizostavni alati u razvoju kompleksnih proizvoda poput robota. Omogućuju pristupačno, efikasno ali i realistično analiziranje rada robota u virtualnom okružju. U slučaju fizikalnih simulatora otvorenoga kôda, jedan od najbrže rastućih je PyBullet. To je fizikalni simulator temeljen na Python programskom jeziku. Omogućuje analizu kolizija, dinamiku krutih i mekih tijela što ga čini primjenjivim u velikom broju scenarija. Sljedeća karakteristika ovog simulatora je da je vrlo jednostavno integrirati ga s okolinama za računalnu inteligenciju temeljenim na programskom jeziku Python, poput okoline PyTorch.

U ovom je radu potrebno napraviti sljedeće:

- Upoznati se s okolinom PyBullet
- Detaljno opisati postupak instalacije i dostupne inačice
- Opisati i dokumentirati osnovne knjižnice i naredbe
- Implementirati primjer mobilnog robota pokretanog kotačima te primjer mobilnog robota pokretanog nogama

U radu je potrebno navesti korištenu literaturu i eventualno dobivenu pomoć.

Zadatak zadan:

30. 11. 2021.

Zadatak zadao:

Doc. dr. sc. Petar Ćurković

Datum predaje rada:

1. rok: 24. 2. 2022.
2. rok (izvanredni): 6. 7. 2022.
3. rok: 22. 9. 2022.

Predviđeni datumi obrane:

1. rok: 28. 2. – 4. 3. 2022.
2. rok (izvanredni): 8. 7. 2022.
3. rok: 26. 9. – 30. 9. 2022.

Predsjednik Povjerenstva:

Prof. dr. sc. Branko Bauer

SADRŽAJ

POPIS SLIKA

SAŽETAK

U ovom radu objašnjen je pojam fizikalnog simulatora, neizostavnog alata pri razvoju kompleksnih proizvoda poput robota. Nadalje, prikazani su primjeri primjene fizikalnih simulatora u različitim područjima robotike te su navedeni neki od dostupnih fizikalnih simulatora. Glavna tema ovog rada je fizikalni simulator PyBullet, temeljen na Python programskom jeziku, koji omogućuje analizu kolizija, simulaciju dinamike krutih i mekih tijela, rad s direktnom i inverznom kinematikom i dinamikom te ima podršku za rad s opremom za virtualnu stvarnost. Detaljno je opisan postupak instalacije PyBullet-a na Windows 10 operativnom sustavu te su dokumentirane i opisane osnovne naredbe komunikacije, postavljanja parametara simulacije, naredbe potrebne za upravljanje robotom te naredbe korištenja inverzne kinematike i dinamike. Na kraju, prikazana je implementacija primjera mobilnog robota R2-D2 pokretanog kotačima te primjer mobilnog robota Laikago pokretanog nogama u PyBullet okruženju. Programski kodovi primjera mobilnih robota dani su u prilogu rada.

Ključne riječi: fizikalni simulator, simulacija, PyBullet, Python, instalacija, naredbe, mobilni roboti, R2-D2 robot, četveronožni robot, Laikago

SUMMARY

In this thesis, concept of a physics simulator, an indispensable tool in development of complex products such as robots, is explained. Furthermore, examples of application of physics simulators in different fields of robotics are presented and some of available physics simulators are listed. The main topic of this thesis is a physics simulator PyBullet, based on Python programming language, which enables collision analysis, simulation of dynamics of rigid and soft bodies, working with direct and inverse kinematics and dynamics, and has support for working with virtual reality equipment. Installation procedure of PyBullet on Windows 10 operating system is described in details and basic commands for communication, setting simulation parameters, commands for using inverse kinematics and dynamics are documented and described. In the end, implementation of an example of mobile robot R2-D2 driven by wheels and an example of mobile robot Laikago driven by legs in PyBullet environment is shown. The program codes of examples of mobile robots are given in the attachment of the thesis.

Key words: physics simulator, simulation, PyBullet, Python, installation, commands, mobile robots, R2-D2 robot, quadruped robot, Laikago

1. UVOD

Fizikalni simulatori postali su neizostavan alat u znanosti i inženjerstvu zahvaljujući ubrzanom razvoju računala zadnjih trideset godina. U robotici, prilikom razvoja robota, upotreba fizikalnog simulatora inženjerima omogućuje, u najranijim fazama razvoja, ispitivanje, razumijevanje i analizu rada robota u virtualnom okruženju bez potrebe za izradom fizičkih prototipova. Zahvaljujući tome javljaju se prednosti u pogledu učinkovitosti, sigurnosti, ubrzavanja razvoja složenog proizvoda, tj. robota te naposljetku upotreba fizikalnog simulatora znači značajno smanjenje troškova.

Posljednjih godina javlja se veliko zanimanje za robote s umjetnom inteligencijom koji se snalaze u nestrukturiranim okruženjima ili sudjeluju u interakciji s ljudima. Da bi ti roboti imali mogućnost donošenja odluka putem umjetne inteligencije potrebna je velika količina podataka. Prikupljanje podataka eksperimentima u stvarnom svijetu može biti izazovno i opasno, a u nekim slučajevima nemoguće je dobiti potrebne podatke. Stvaranjem virtualnog okruženja, pomoću fizikalnog simulatora, u kojem se robot nalazi doprinosi kreiranju velike količine podataka potrebnih za učenje za različite slučajeve bez da se ljudi ili roboti dovode u opasnost. [1]

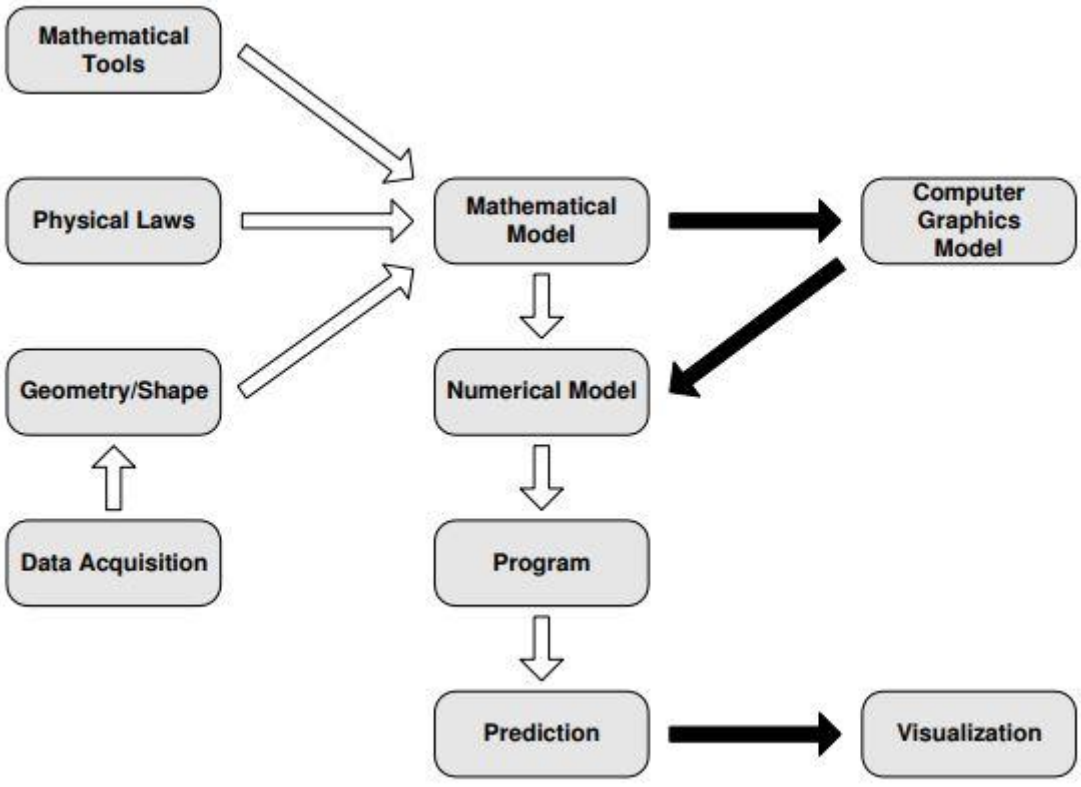
U ovom radu težište je na primjeni fizikalnih simulatora u robotici. Rad je podijeljen u sedam poglavlja. Drugo poglavlje sadrži objašnjenje pojma fizikalnog simulatora te su dani primjeri primjene u robotici. U trećem poglavlju opisan je fizikalni simulator *PyBullet* koji je glavna tema ovog rada te su prikazani istraživački radovi u kojima se koristi. Četvrto poglavlje sadrži detaljan opis postupka instalacije *PyBullet*-a, a u petom poglavlju opisane su osnovne naredbe potrebne za rad. U šestom poglavlju prikazani su implementirani primjeri mobilnih robota pokretani nogama i kotačima. Zadnje poglavlje sadrži zaključak te su u prilogu rada dani programski kodovi implementiranih primjera.

2. FIZIKALNI SIMULATOR

Računalna simulacija je postupak procjene ponašanja nekog dinamičkog sustava pomoću računala. Kad su u računalnu simulaciju uključeni zakoni fizike za imitaciju ponašanja dinamičkih sustava, onda se kaže da je to fizikalna simulacija. U robotici, fizikalni zakoni poput zakona očuvanja mase i momenta najviše se koriste. [1]

Fizikalni simulator sastoji se od zakona fizike, matematike i geometrije koji kad su zajedno 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]

Na [Slika 1] dan je shematski prikaz područja od kojih se sastoji fizikalni simulator.



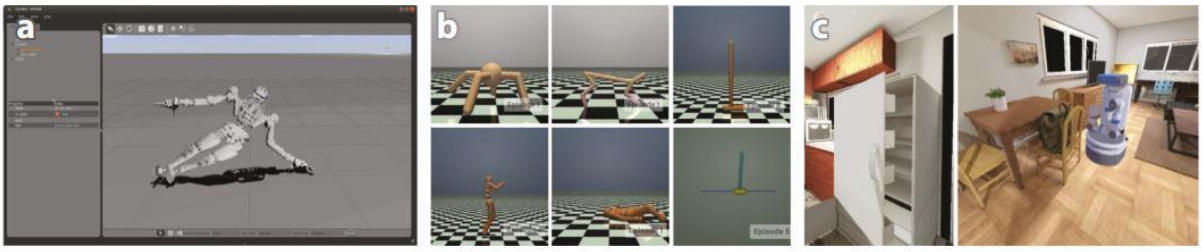
Slika 1. Shematski prikaz područja fizikalnog simulatora [2]

Računalni *software* putem kojeg je omogućeno rekreiranje pojava iz stvarnog svijeta zove se fizikalni *engine*. Putem fizikalnog *engine*-a moguće je u fizikalnim simulatorima simulirati dinamiku krutih i mekih tijela, detektirati kolizije, simulirati dinamiku fluida, aerodinamiku i termodinamiku. Područja primjene fizikalnih simulatora su u znanosti, filmskoj industriji te u izradi video igrica. [3,4]

2.1. Fizikalni simulatori u robotici

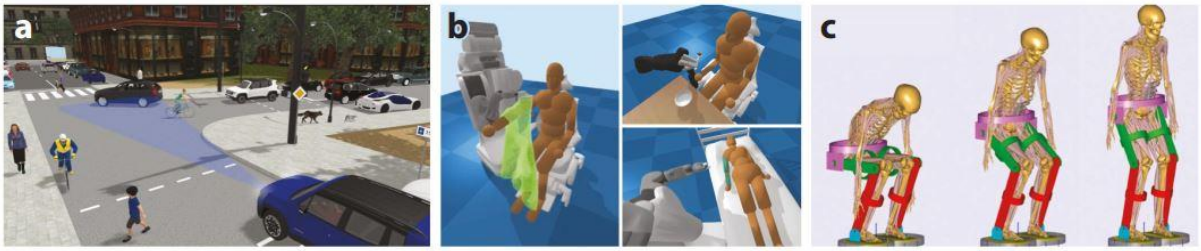
Fizikalni simulatori koji imaju mogućnost rekonstruirati elemente fizike kao što su dinamika krutih i deformabilnih (mekih) tijela, dinamiku fluida te imaju mogućnost prikazivanja različitih vrsta podloga omogućuju simulaciju velikog broja slučajeva koji su važni prilikom ispitivanja i razvoja robotskog sustava. Također, fizikalnim simulatorima moguće je simulirati rad senzora i aktuatora te interakciju između čovjeka i robota. [1]

Primjeri korištenja fizikalnih simulatora prikazani su na [Slika 2]. Primjer označen slovom a prikazuje primjenu fizikalnog simulatora za razvoj dvonožnog robota. Primjer označen sa b prikazuje primjenu fizikalnog simulatora pri razvoju algoritama za učenje simuliranih agenata. Primjer označen sa c prikazuje virtualno okruženje prostorije nastale oblikovanjem stvarne prostorije koja se koristi kako bi se simuliralo kretanje mobilnog robota. [1]



Slika 2. Primjeri korištenja fizikalnih simulatora u robotici [1]

Primjeri simulacije interakcije čovjeka i robota prikazani su na [Slika 3]. Primjer pod a prikazuje način na koji autonomno vozilo, na siguran način za pješake, prikuplja podatke u virtualnom okruženju potrebne za razvoj algoritama umjetne inteligencije. Primjeri označeni sa b i c prikazuju virtualno okruženje fizikalnog simulatora u kojem se na siguran način može ispitati ispravan rad robota u zdravstvu. [1]



Slika 3. Primjeri simulacije interakcije čovjeka i robota [1]

2.2. Primjena fizikalnih simulatora u robotici

Glavna područja robotike u kojima se primjenjuju fizikalni simulatori su:

- mobilna robotika
- manipulatori
- medicinska robotika
- pomorska robotika
- zrakoplovna robotika
- meka robotika
- područje učenja u robotici (*Deep/Reinforcement Learning)*

Područje mobilne robotike, u kojem se istražuju autonomna vozila, tj. roboti pogonjeni kotačima ili nogama, područje je koje se najviše istražuje u robotici. Ovo područje uključuje u razmatranje navigaciju, prepoznavanje, upravljanje, percepciju, kretanje, mapiranje i drugo.

U području robotike u koje pripadaju manipulatori fizikalnim simulatorima istražuju se i ispituju konstrukcije robotskih ruku i hvataljki te se razvijaju algoritmi potrebni za upravljanje te planiranje prostornog gibanja.

Operacije te terapije rehabilitacije uz pomoć robota domene su medicinske robotike koje se razvijaju pomoću fizikalnih simulatora.

Prilikom razvoja i ispitivanja pomorske robotike ključne stavke za simulaciju su navođenje, slijeđenje unaprijed određenih točaka, stvaranje mape morskog dna te upravljanje temeljeno na podacima dobivenih sa različitih senzora.

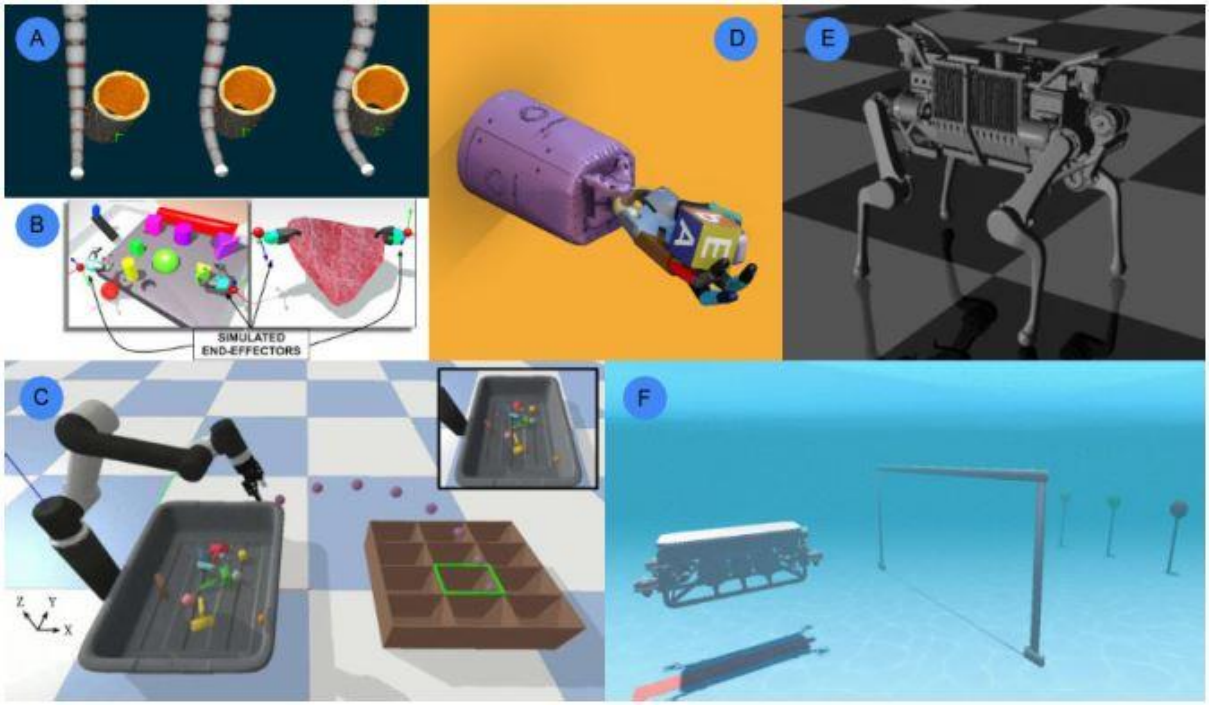
Zrakoplovna robotika područje je robotike u kojem fizikalni simulatori mogu doprinijeti u istraživanjima i testiranjima u pogledu rekreiranja složenih pojava iz stvarnog svijeta kao što su turbulencija, gustoća zraka, utjecaj vjetra te modeliranje oblaka i padalina.

Prilikom razvoja i ispitivanja ispravnosti rada mekih robota fizikalni simulatori koriste se pri simulaciji različitih, neuobičajenih načina aktivacije članaka mekih robota kao što su aktivacija pomoću tetiva te pneumatska ili aktivacija putem prijenosa topline. Također, simulira se i ostvarivanje dodira između mekih robota i krutih ili deformabilnih materijala.

Učenju u robotici pridodaje se velika pažanja posljednjih deset godina. Kako bi se izbjegle neželjene štete na robotima koje se mogu dogoditi zbog nedostataka algoritama dubokog učenja, najprije se provode ispitivanja u virtualnom okruženju fizikalnih simulatora. Duboko učenje primjenjuje se u zadacima kao što su manipuliranje i hvatanje objekata, planiranje putanji kretanja mobilnih robota neravnim terenima i drugo. [5]

Ključna svojstva i mogućnosti koje fizikalni simulatori moraju imati kako bi zadovoljili potrebe primjene u robotici su: stvaranje realističnih fizikalnih pojava putem fizikalnog *engine*-a, otkrivanje nastanka kolizija, modeliranje trenja, postojanje grafičkog sučelja, sučelje za programiranje koje je prilagođeno programiranju programskim jezicima često korištenim u robotici kao što su *Python* ili C++, mogućnost rada sa modelima raznih zglobova, senzora i aktuatora te mogućnost rada s direktnom i inverznom kinematikom i dinamikom. [5]

Na [Slika 4] prikazani su primjeri primjene fizikalnih simulatora iz područja meke, medicinske, pomorske i mobilne robotike te primjena pri manipuliranju predmetima.



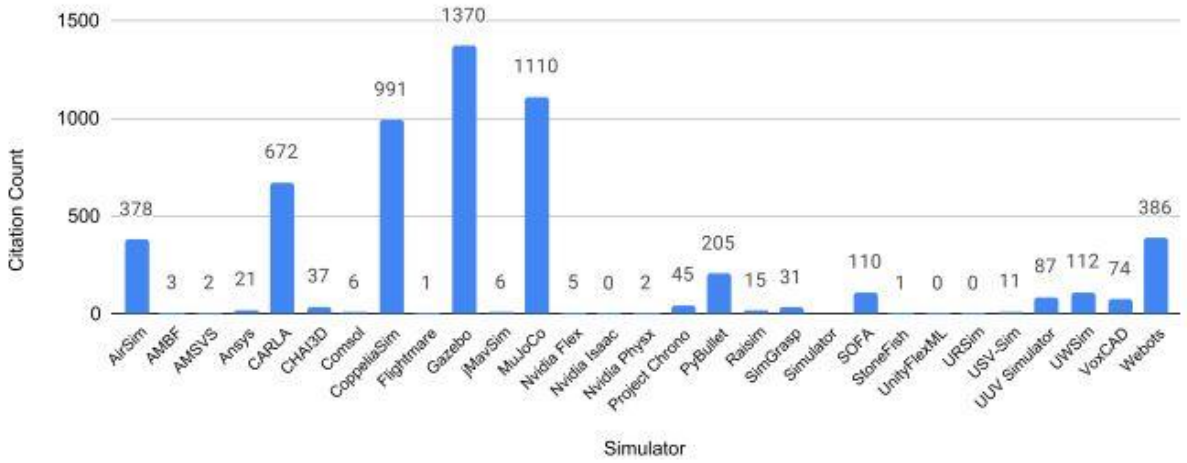
Slika 4. Primjeri primjene fizikalnih simulatora u različitim područjima robotike [5]

2.3. Dostupni fizikalni simulatori

Danas su dostupni brojni fizikalni simulatori, kako otvorenog koda tako i oni za čiju je uporabu potrebno platiti licencu. Neki od fizikalnih simulatora koji se ističu u robotici prema broju korisnika koji ih koriste navedeni su abecednim redom kako slijedi: [5]

- *Airsim*
- *CARLA*
- *CoppeliaSim*
- *Gazebo*
- *MuJoCo*
- *PyBullet*
- *SOFA*
- *UWSim*
- *Chrono*
- *Webots*

Učestalost korištenja pojedinih fizikalnih simulatora u robotici u istraživanjima, znanstvenim i ostalim radovima u razdoblju od 2016. do 2020. godine prikazano je stupčastim grafikonom na [Slika 5].



Slika 5. Prikaz učestalosti korištenja pojedinih fizikalnih simulatora u robotici [5]

U nastavku ovog rada detaljnije je obrađen fizikalni simulator *PyBullet*.

3. FIZIKALNI SIMULATOR PYBULLET

PyBullet je programski modul otvorenog koda, temeljen na *Python* programskom jeziku, koji se koristi kao fizikalni simulator u robotici te za razvoj robotskog učenja s naglaskom na prijenos dobivenih podataka simulacijom u stvarnost. *PyBullet* je razvijen od strane razvojnog programera Erwin-a Coumans-a na *Bullet* fizikalnom *engine*-u koji se koristi pri simulaciji dinamike krutih i mekih tijela te simulaciji kolizija u video igrama i stvaranju vizualnih efekata u filmovima.

Fizikalni simulator *PyBullet* omogućuje analizu kolizija, simulaciju dinamike krutih i mekih tijela, rad s direktnom i inverznom kinematikom i dinamikom te ima podršku za rad s opremom za virtualnu stvarnost poput *Oculus Rift* ili *HTC Vive*.

Modele robota i ostalih elemenata potrebnih za simulaciju moguće je učitati u .urdf (eng. *Unified Robot Description Format*), .sdf (eng. *Structure-Data File*) i drugim formatima datoteka koje se po želji mogu oblikovati dostupnim računalnim alatima za 3D modeliranje.

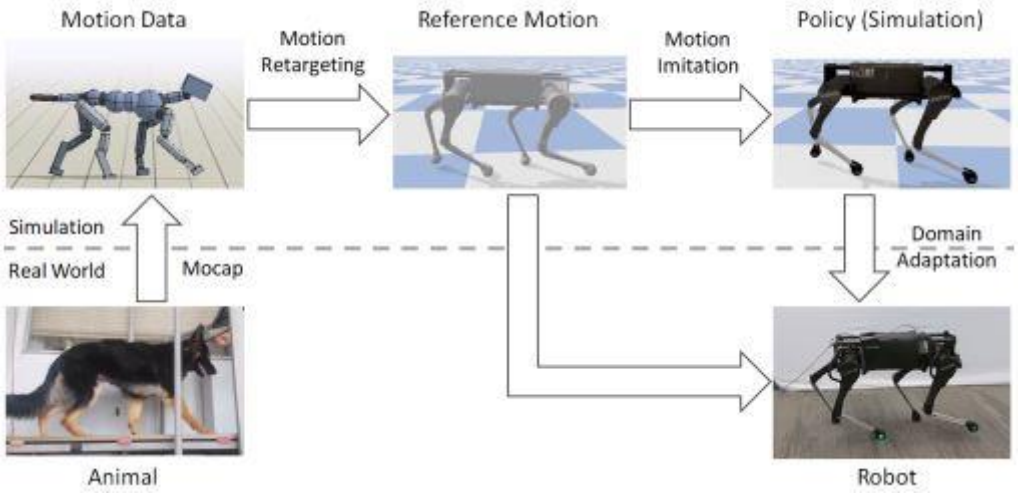
PyBullet je moguće lako integrirati s modulima za strojno učenje poput *TensorFlow*-a ili *PyTorch*-a. [6]

Navedene značajke čine *PyBullet* primjenjivim u velikom broju slučajeva u robotici, na primjer poput analize samosklapajućih origami robota. [7]

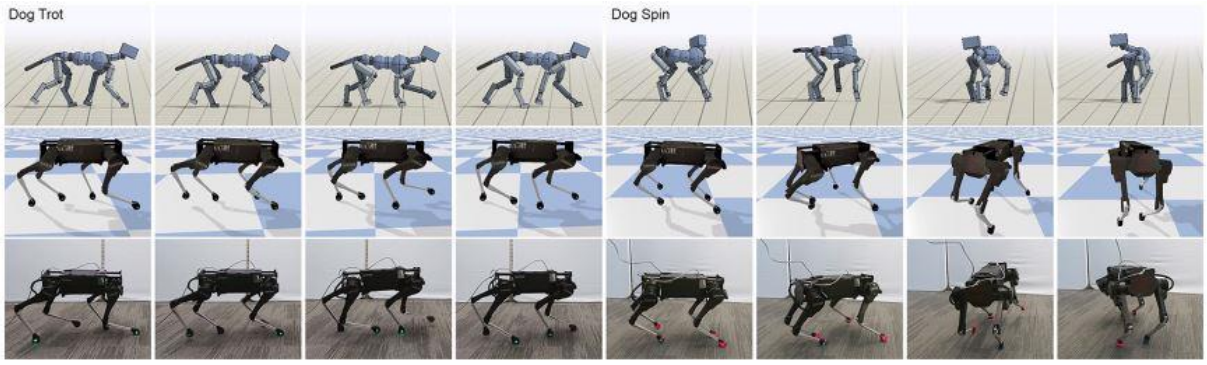
3.1. Primjena fizikalnog simulatora *PyBullet*

Fizikalni simulator *PyBullet* primjenjuje se u brojnim znanstvenim istraživanjima. Neki od primjera primjene su:

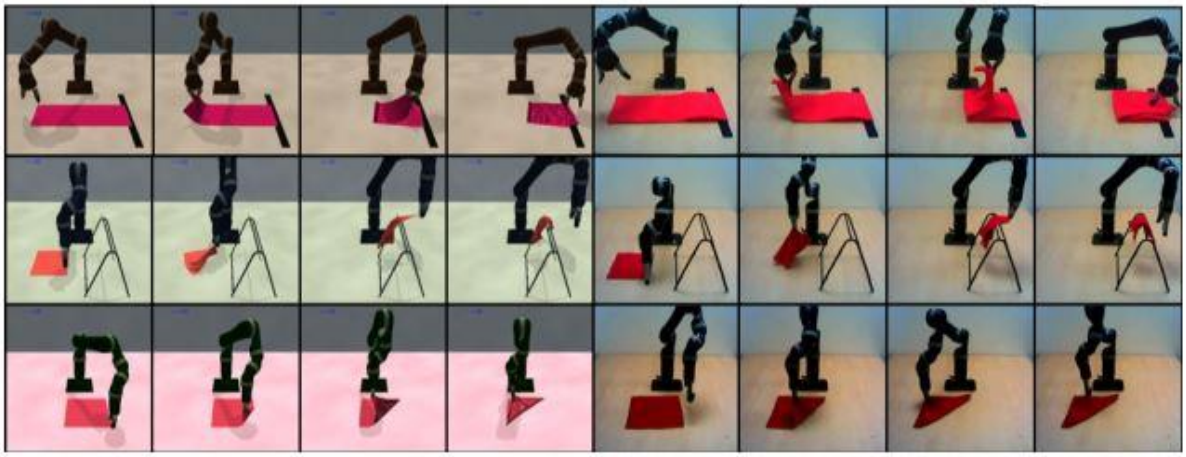
- Učenje agilnih robotskih vještina kretanja oponašanjem životinja [8], prikazano na [Slika 6] i [Slika 7]
- Korištenje strojnog učenja za manipulaciju deformabilnim objektima [9], prikazano na [Slika 8]
- Učenje bacanja proizvoljnih objekata [10], prikazano na [Slika 9]
- Učenje agilnih kretnji za četveronožne robote [11], prikazano na [Slika 10]
- Učenje pravila za skupljanje smeća pomoću hvataljke robota simulacijom robusnih nizova hvatanja [12]



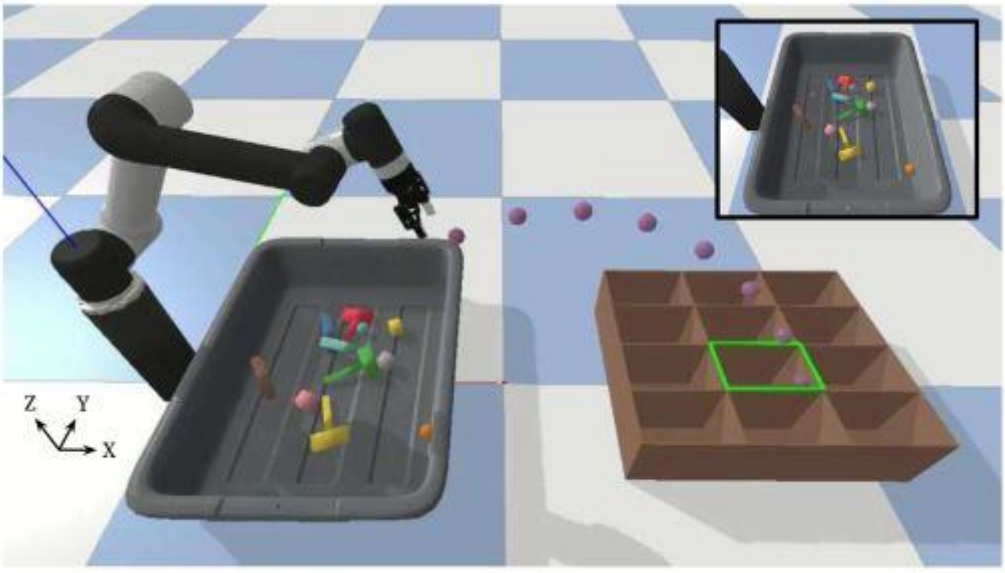
Slika 6. Prikaz koraka pri učenju robotskih vještina oponašanjem pokreta psa [8]



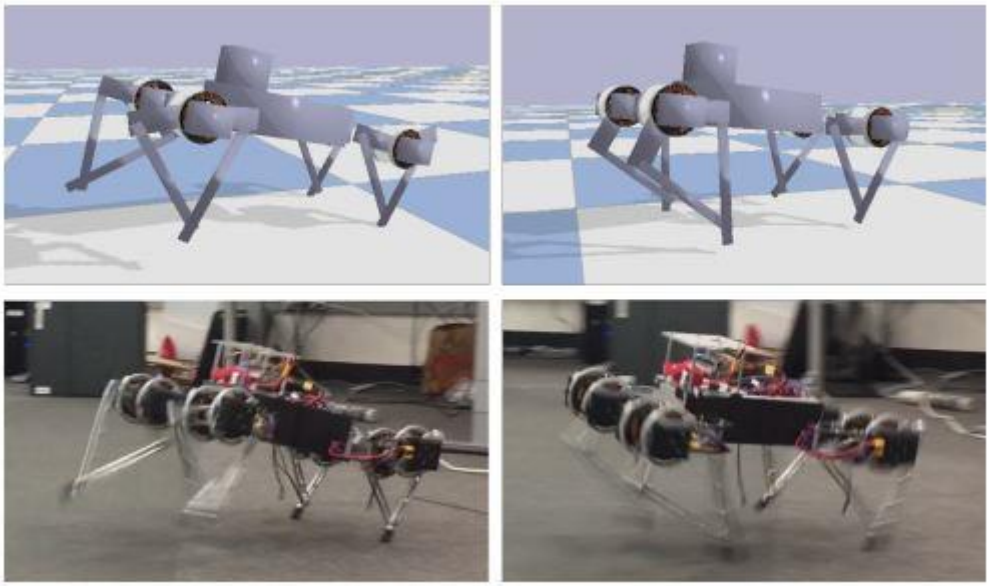
Slika 7. Prikaz snimljenih, simuliranih te implementiranih pokreta [8]



Slika 8. Učenje pravila slaganja deformabilnog objekta u simulaciji te ispitivanje u stvarnosti [9]



Slika 9. Bacanje proizvoljnih objekata u kutiju [10]



Slika 10. Učenje galopiranja dubokim učenjem u simulaciji potom primjena na fizičkom robotu [11]

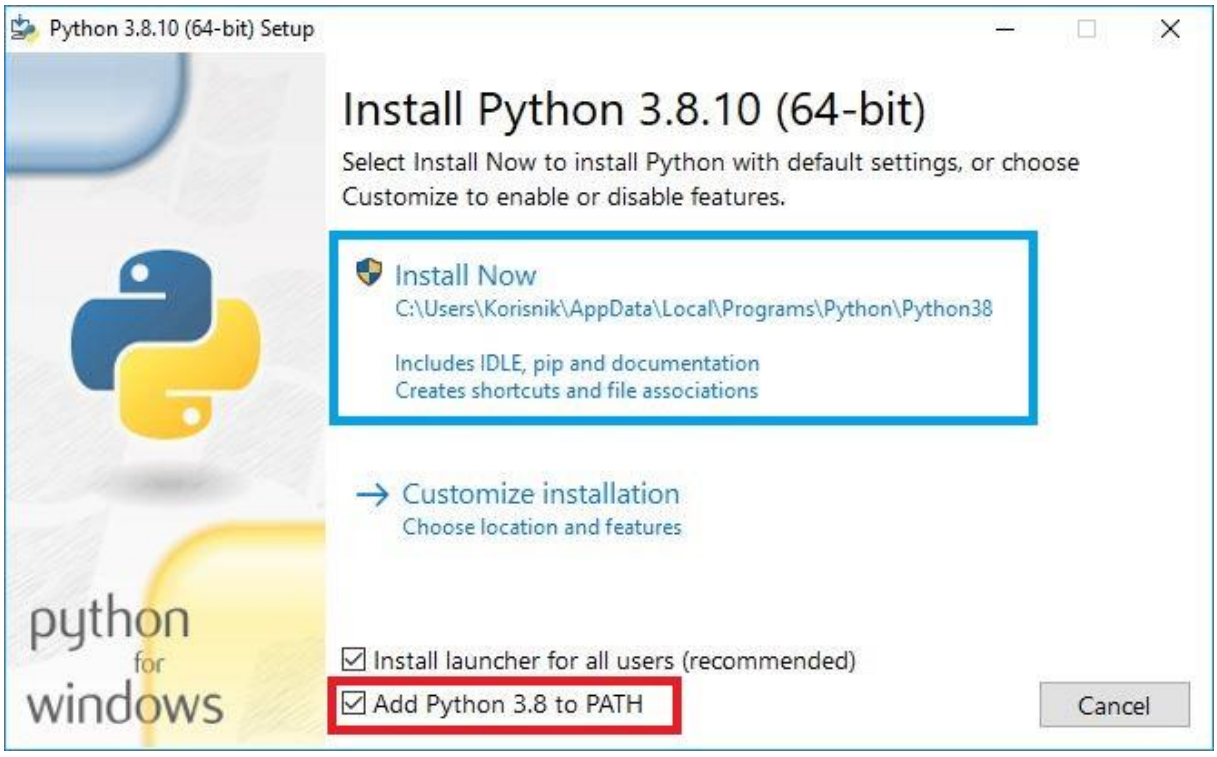
4. POSTUPAK INSTALACIJE FIZIKALNOG SIMULATORA PYBULLET

Fizikalni simulator *PyBullet* moguće je instalirati na *Windows, macOS* te *Linux* operativnim sustavima. Postupak instalacije, u ovom radu, prikazan je na *Windows* 10 operativnom sustavu. Instalacija se sastoji od nekoliko koraka, a to su:

1. Instalacija programskog paketa *Python*
2. Instalacija *Visaul Studio Build Tools*-a
3. Ažuriranje upravitelja *Python* modula – *pip*-a (eng. *Package Installer for Python*)
4. Instalacija modula *PyBullet*

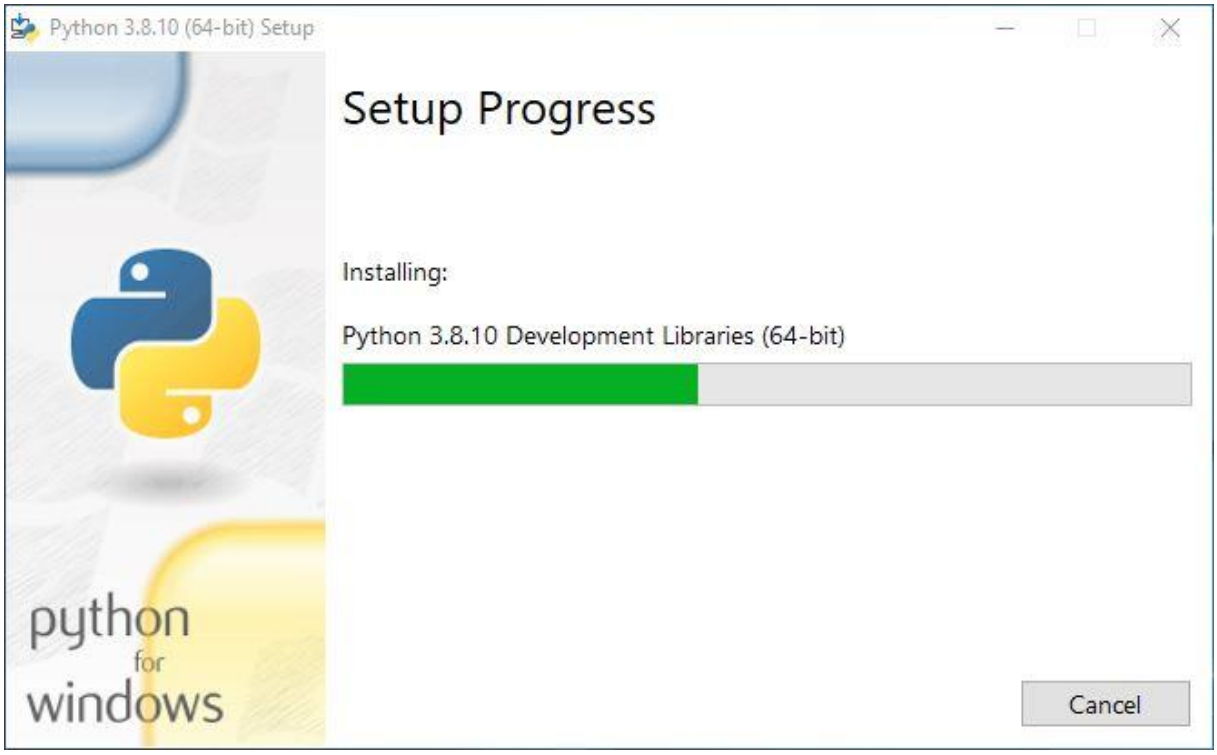
4.1. Instalacija programskog paketa *Python*

Programski paket *Python* potrebno je preuzeti sa internet stranice: https://www.python.org/downloads/. U ovom radu korištena je verzija *Python* 3.8.10. Nakon preuzimanja instalacijske datoteke i otvaranja iste otvara se prozor za početak instalacije, prikazano na [Slika 11].



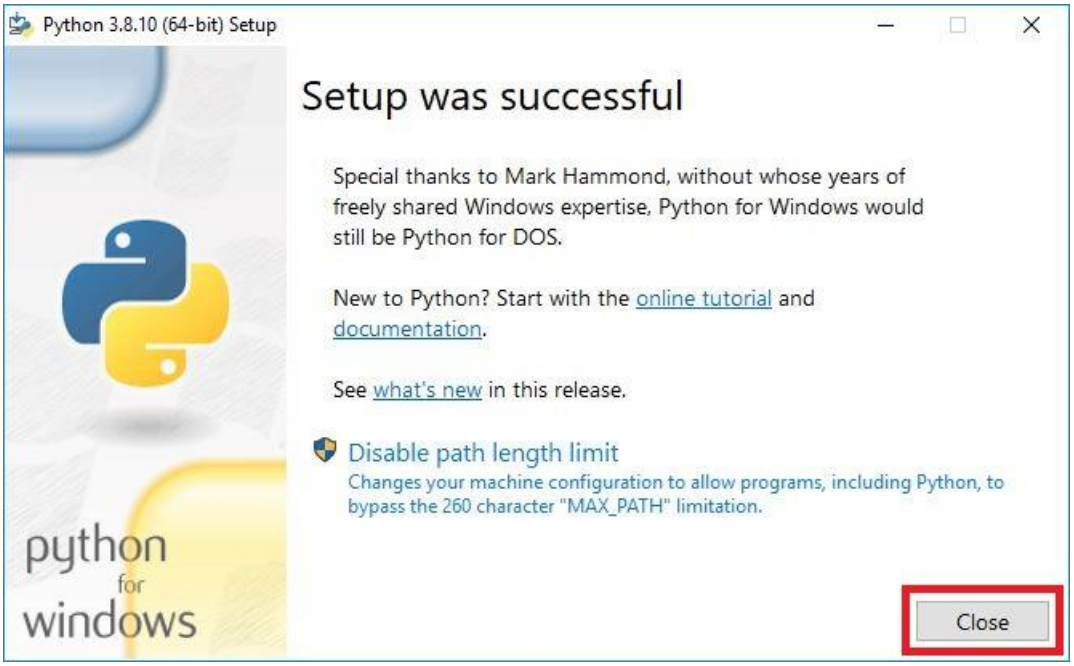
Slika 11. Priprema instalacije *Python* 3.8.10

U instalacijskom prozoru potrebno je odabrati opciju *Add Python 3.8 to PATH*, označeno crvenim pravokutnikom na [Slika 11]. Nakon toga pritiskom na *Install Now*, označeno plavim pravokutnikom na [Slika 11], započinje instalacija, kao što je prikazano na [Slika 12].



Slika 12. Instalacija *Python* 3.8.10 u tijeku

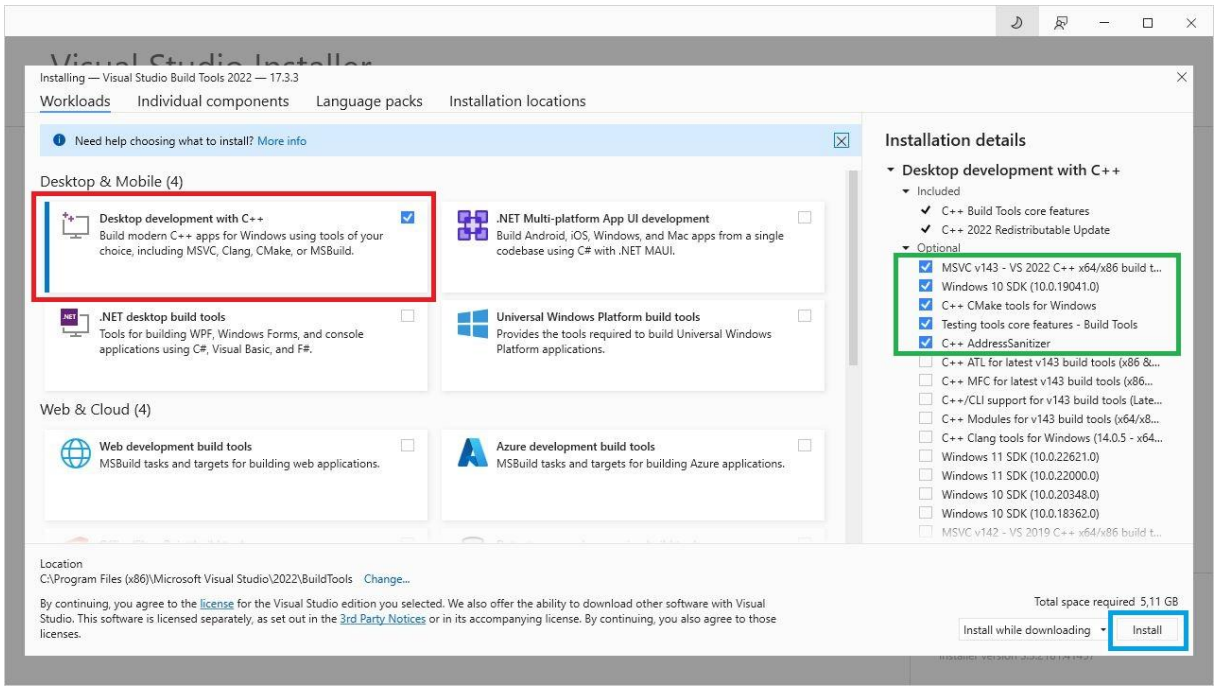
Završetkom postupka instalacije programskog paketa *Python* prikazuje se obavijest o uspješno izvršenoj instalaciji, kao što je prikazano na [Slika 13]. Pritiskom na *Close*, označeno crvenim pravokutnikom, završava postupak instalacije *Python* 3.8.10. Sljedeći korak je instalacija *Visual Studio Build Tools*-a.



Slika 13. Završetak instalacije *Python* 3.8.10

4.2. Instalacija *Visual Studio Build Tools*-a

Kako bi se moglo instalirati *Visual Studio Build Tools* potrebno je na internet stranici: https://visualstudio.microsoft.com/visual-cpp-build-tools/ preuzeti instalacijsku datoteku. Otvaranjem preuzete datoteke otvara se instalacijski prozor prikazan na [Slika 14].

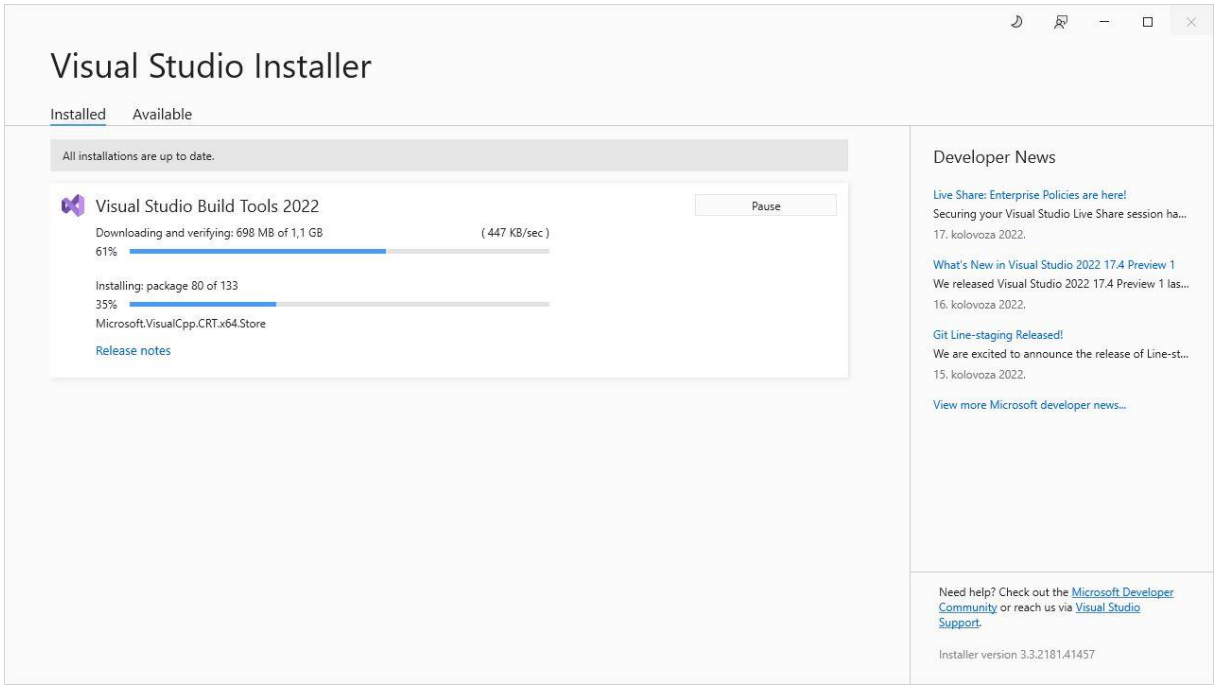


Slika 14. Instalacijski prozor *Visual Studio Build Tools*

Na [Slika 14] crvenim pravokutnikom označena je opcija, *Desktop development with C++*, koju je potrebno odabrati. Zelenim pravokutnikom prikazane su stavke koje pripadaju odabranoj opciji. Pritiskom na *Install*, označeno plavim pravokutnikom, počinje preuzimanje dodatnih datoteka, kao što je prikazano na [Slika 15], koje su odabrane opcijom.

Završetkom instalacije prikazuje se obavijest o uspješno provedenom postupku.

U sljedećem koraku prikazan je postupak ažuriranja upravitelja *Python* modula.



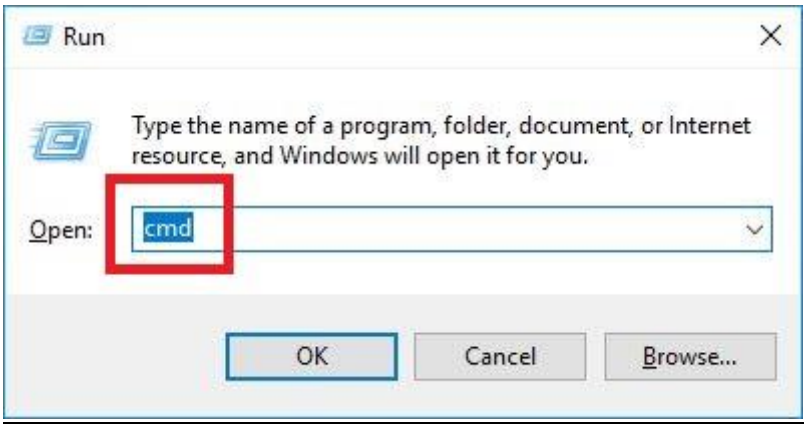
Slika 15. Preuzimanje i instalacija *Visual Studio Build Tools*-a u tijeku

4.3. Ažuriranje upravitelja *Python* modula - pip

Za uspješno instaliranje fizikalnog simulatora *PyBullet*, potrebno je najprije ažurirati upravitelja *Python* modula, tj. pip (eng. *Package Installer for Python*). Ažuriranje se provodi tako da se u *Command Prompt* upiše naredba:

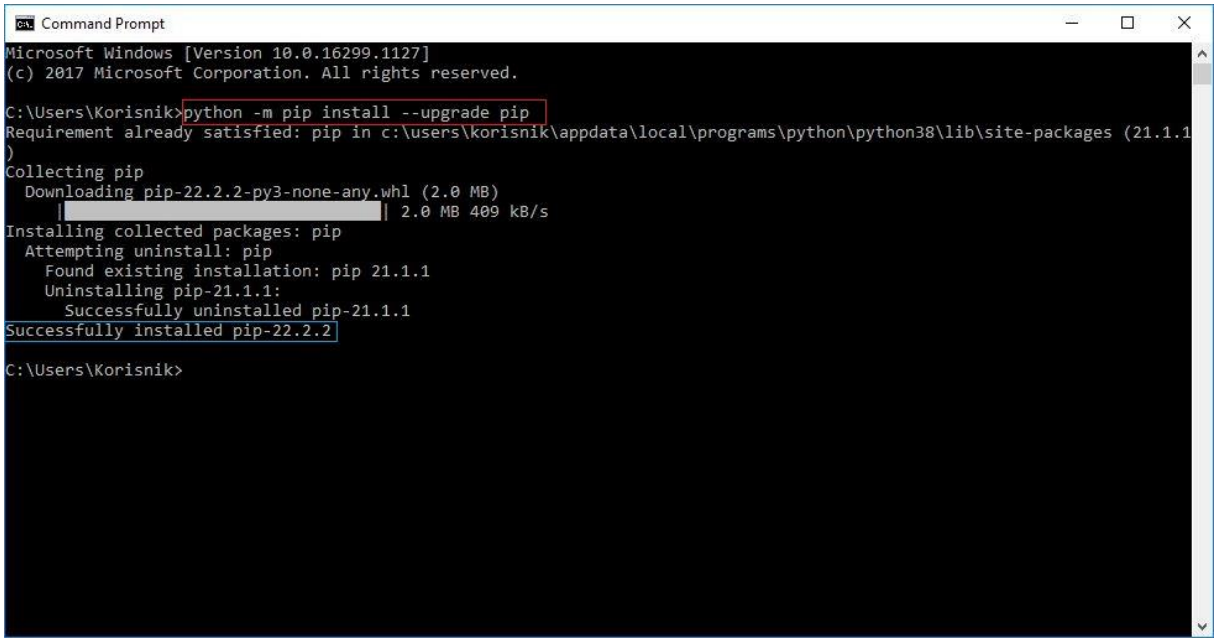
```
python -m install --upgrade pip
```

Command Prompt pokreće se pritiskom kombinacije tipki: *Windows* tipka + R . Pritiskom navedenih tipki otvara se prozor *Run*, kao što je prikazano na [Slika 16], u koji je potrebno upisati: cmd, označeno crvenim pravokutnikom na [Slika 16] . Pritiskom na *OK* otvara se *Command Prompt*.



Slika 16. Prozor *Run*

Upisivanjem prethodno navedene naredbe, označeno crvenim pravokutnikom na [Slika 17], te pritiskom tipke *Enter* počinje preuzimanje i instalacija najnovije inačice *pip*-a. Proces je prikazan na [Slika 17]. Završetkom instalacije dana je obavijest o uspješnoj instalaciji, označeno plavim pravokutnikom.



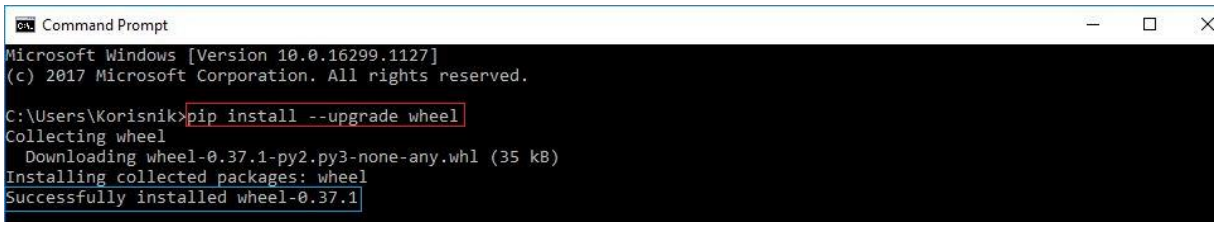
Slika 17. Ažuriranje *pip*-a

Također, za uspješnu instalaciju modula *PyBullet*, potrebno je ažurirati i knjižnice *wheel* i *setuptools* koje omogućuju nesmetanu instalaciju programskih paketa. Ažuriranje navedenih knjižnica provodi se na isti način kao što je ažuriranje *pip*-a.

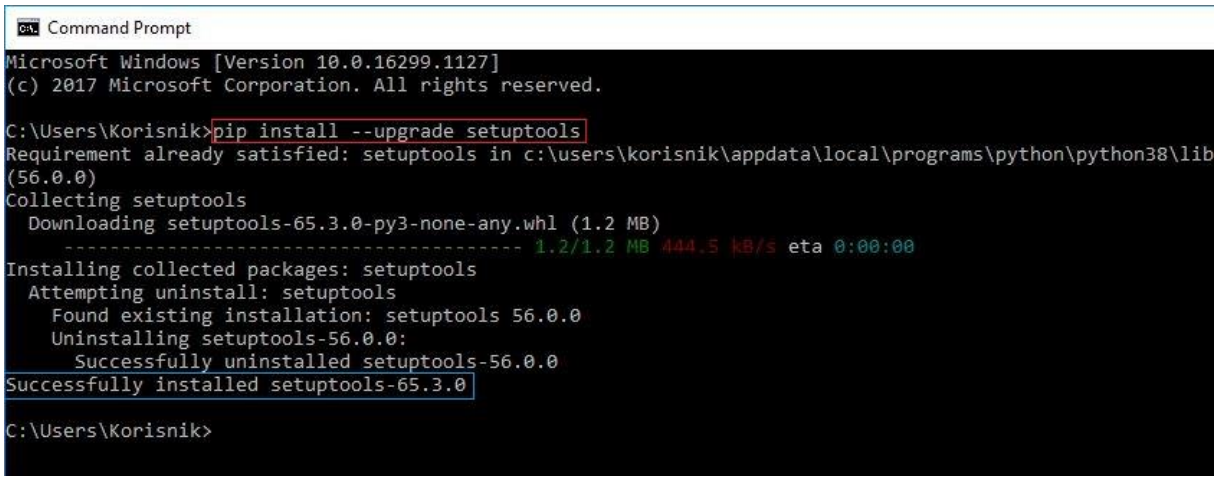
Upisivanjem naredbi u *Command Prompt*:

```
pip install --upgrade wheel
pip install --upgrade setuptools
```

Postupak je prikazan na [Slika 18] i [Slika 19]. Crvenim pravokutnicima označene su upisane naredbe, a plavim obavijest o uspješno provedenom ažuriranju knjižnica.



Slika 18. Ažuriranje *wheel*-a



Slika 19. Ažuriranje *setuptools*-a

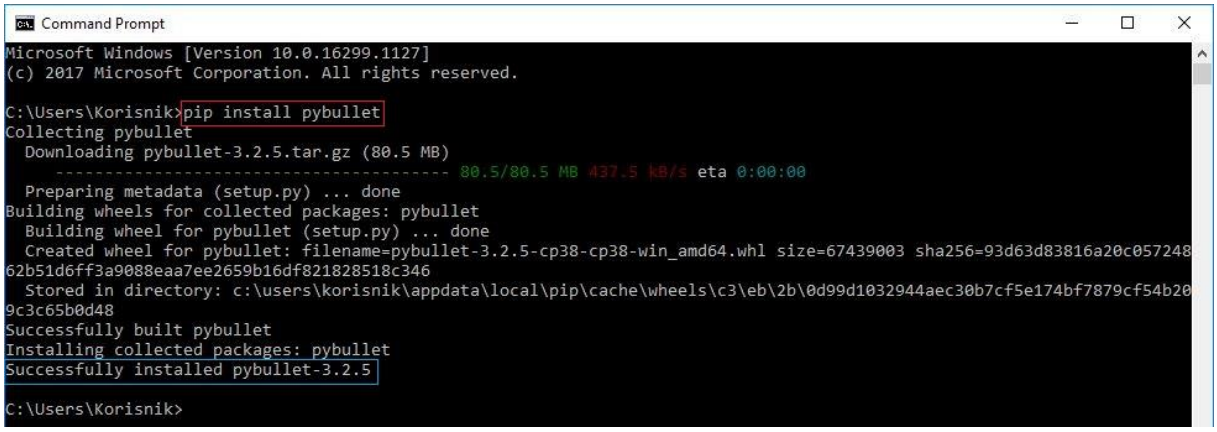
Instalacija modula *PyBullet* prikazana je u nastavku.

4.4. Instalacija modula *PyBullet*

Nakon što su prethodna tri koraka uspješno provedena, zadnji korak je instalacija modula *PyBullet*. Za instalaciju modula potrebno je u *Command Prompt* upisati naredbu:

```
pip install pybullet
```

kao što je prikazano na [Slika 20] unutar crvenog pravokutnika. Pritiskom na tipku *Enter* počinje preuzimanje podataka potrebnih za instalaciju. Nakon što su podaci preuzeti započinje instalacija modula. Obavijest o uspješnosti instalacije označava završetak postupka, što je označeno na [Slika 20] plavim pravokutnikom.



Slika 20. Instalacija modula *PyBullet*

Instalirana inačica modula *PyBulet* je 3.2.5. Datum izlaska inačice je 20. svibnja 2022. godine.

4.5. Greške kod instaliranja *PyBullet*-a

Prilikom procesa instalacije i pokretanja fizikalnog simulatora *PyBullet* mogu se javiti greške zbog:

- neažuriranog upravitelja *Python* modula *pip*-a te knjižnica *setuptools i wheel*
- neinstaliranog *Visual Studio Build Tools* (zahtjev je *Visual Studio C++ 14.0* ili više)
- grafičke kartice koja ne podržava *OpenGL* 3 [6]

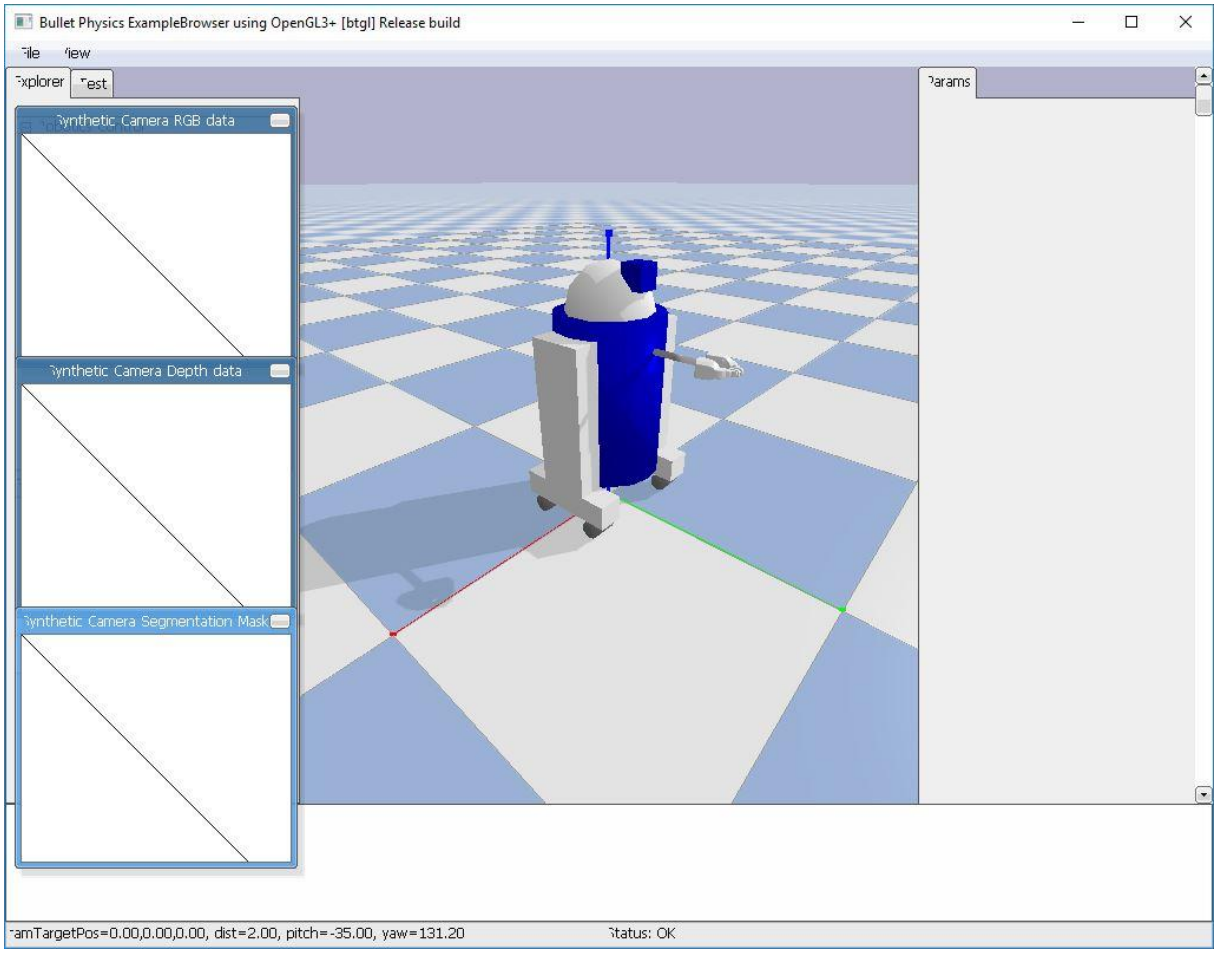
4.6. Provjera ispravnosti rada fizikalnog simulatora *PyBullet*

Za provjeru ispravnosti rada upisan je programski kod putem kojeg se stvara ravnina i model robota. Programski kod upisan je u uređivač koda *IDLE* (eng. *Integrated Development and Learning Environment*) koji je instaliran prilikom instalacije programskog paketa *Python*.

Programski kod glasi:

```
import pybullet as p
from time import sleep
import pybullet_data


physicsClient = p.connect(p.GUI)

p.setAdditionalSearchPath(pybullet_data.getDataPath())

p.setGravity(0, 0, -9.81)
planeId = p.loadURDF("plane.urdf")
robotStartPos = [0, 0, 1]
robotStartOrientation = p.getQuaternionFromEuler([0, 0, 0])
robotId = p.loadURDF("r2d2.urdf", robotStartPos, robotStartOrientation)
robotPos, robotOrn = p.getBasePositionAndOrientation(robotId)

useRealTimeSimulation = 0

if (useRealTimeSimulation):
  p.setRealTimeSimulation(1)

while 1:
  if (useRealTimeSimulation):
    p.setGravity(0, 0, -9.81)
    sleep(0.01)
  else:
    p.stepSimulation()
```

Pokretanjem programskog koda otvara se sučelje u kojem je stvorena ravnina i model mobilnog robota, prikazano na [Slika 21]. Ovime je provjerena ispravnost instalacije fizikalnog simulatora *PyBullet*.



Slika 21. Sučelje fizikalnog simulatora *PyBullet*

5. OPIS I DOKUMENTIRANJE OSNOVNIH NAREDBI [6]

5.1. Naredbe komunikacije sa poslužiteljem, postavljanje parametara simulacije i drugo

Programsko sučelje *PyBullet*-a, API (eng. *Application Programming Interface*), konstruirano je na temelju veze klijent-poslužitelj na način da klijent šalje naredbe, a fizikalni poslužitelj, tj. *server*, vraća podatke o trenutnom stanju. Fizikalni poslužitelji ugrađeni u modulu *PyBullet* su 'DIRECT' i 'GUI' (eng. *Graphical User Interface*) odnosno grafičko korisničko sučelje. Korištenjem 'DIRECT' načina rada ne mogu se koristiti značajke *OpenGL*-a te koristiti oprema za virtualnu stvarnost.

U nastavku opisane su osnovne naredbe potrebne za komunikaciju sa poslužiteljem te naredbe za postavljenje parametara simulacije. Masno otisnuta slova označavaju naziv naredbe.

connect( ) – naredba za povezivanje sa fizikalnim poslužiteljem koristeći 'DIRECT' ili 'GUI'

Povezivanje putem 'GUI' poslužitelja stvara uz pomoć 3D *OpenGL* renderiranja grafičko korisničko sučelje dok se pri povezivanju na 'DIRECT' poslužitelja naredbe šalju izravno u fizikalni *engine*, bez grafičkog prikaza, te se podatci o stanju odmah vraćaju nakon izvršavanja naredbi.

Primjeri korištenja naredbe:

```
pybullet.connect(pybullet.GUI)
pybullet.connect(pybullet.DIRECT)
```

disconnect( ) – naredba koja služi sa prekid veze sa fizikalnim poslužiteljem

U slučaju povezivanja putem 'GUI' ili 'DIRECT' poslužitelja naredba *disconnect( )* će ih ugasiti.

Primjer:

```
pybullet.disconnect()
```

setGravity( ) – naredba koja omogućuje postavljanje korisnički definirane gravitacijske sile za sve objekte koji se nalaze u simulacijskom prostoru. Prema početnim postavkama gravitacijska sila nije omogućena stoga je potrebno definirati istu. Ulazni parametri naredbe su željene vrijednosti gravitacijske sile u smjerovima koordinatnih osi x, y i z.

Primjer:

```
pybullet.setGravity(0,0,-9.81)
```

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

Potrebni argument naredbe je naziv datoteke koji se piše u obliku putanje koja vodi do mjesta pohrane URDF datoteke, bilo na računalu ili fizikalnom poslužitelju.

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

Opcija pozicija baze stvara bazu objekta na poziciji određenom vektorom koordinata [x, y, z].

Opcija orijentacija baze stvara bazu objekta orijentiranu prema koordinatama definiranim u vektoru koordinata [x, y, z, w].

Primjer:

```
pybullet.loadURDF("r2d2.urdf",[0,0,1],[0,0,0,1])
```

saveState( ) – naredba koja omogućuje pohranu svih važnih informacija stanja, pohrana se vrši u memoriji modula *PyBullet*

saveBullet( ) – naredba omogućuje pohranu informacija stanja na disku u .bullet datoteci

restoreState( ) – naredba omogućuje povrat informacija dobivenih naredbama *saveState( )* i *saveBullet( )*

removeState( ) – naredba koja služi za brisanje iz memorije prethodno pohranjenih informacija dobivenih naredbama *saveState( )* i *saveBullet( )*

saveWorld( ) – naredba koja služi za pohranu informacija u obliku Python datoteke o trenutnom okruženju, tj. svijetu, koji se koristi pri simulaciji. Spremljeni položaj robota, određene početne pozicije zglobova, postavljeni objekti i okruženje koje je potrebno pri simulaciji moguće je ponovo učitati otvaranjem spremljene datoteke.

Potrebni argument naredbe je naziv datoteke u koju se želi pohraniti stanje okruženja.

stepSimulation( ) – naredba koja omogućuje izvršavanje svih radnji u jednom simulacijskom koraku, zadan vremenski korak je 1/240 sekunde

setRealTimeSimulation( ) – naredba koja omogućuje pokretanje simulacije u stvarnom vremenu na način da fizikalni poslužitelj automatski dodjeljuje korak simulacije prema satu stvarnog vremena RTC (eng. *Real-Time Clock*). Pri korištenju ove naredbe ne treba koristiti naredbu *stepSimulation( ).*

Potrebni ulazni parametar naredbe je broj 0 ili 1 kako bi se onemogućila ili omogućila simulacija u stvarnom vremenu.

Primjer:

```
pybullet.setRealTimeSimulation(1)
```

resetSimulation( ) – naredba koja omogućuje brisanje svih objekata koji se nalaze u simulacijskom prostoru te ponovno postavljanje istih prema zadanim početnim uvjetima

getBasePositionAndOrientation( ) – naredba se koristi za dobivanje izvješća o trenutnoj poziciji i orijentaciji baze željenog objekta u Kartezijevom koordinatnim sustavu. Potrebni ulazni parametar je identifikacijska oznaka koja je dodijeljena objektu.

Primjer:

```
pybullet.getBasePositionAndOrientation(boxId)
```

resetBasePositionAndOrientation( ) – naredba koja omogućuje ponovno postavljanje pozicije i orijentacije baze željenog objekta. Potrebni ulazni argumenti naredbe su: identifikacijska oznaka objekta, pozicija i orijentacija objekta, zapisano u obliku vektora, na koju se želi ponovo postaviti baza objekta.

getQuaternionFromEuler( ) - naredba koja se koristi za transformaciju zapisa orijentacije iz Euler-ovog zapisa u zapis u obliku kvaterniona [x, y, z, w]. Potrebni ulazni argument je vektor koji sadrži tri podatka o tri Euler-ova kuta rotacije u radijanima.

getEulerFromQuaternion( ) – naredba koja se koristi za transformaciju zapisa orijentacije iz zapisa u obliku kvaterniona u Euler-ov zapis [x, y, z], gdje su redom dobivene vrijednosti kuta rotacije oko x, y i z koordinatne osi. Potrebni ulazni argument su vrijednosti kvaterniona.

5.2. Naredbe potrebne za upravljanje robotom

Modeli robota koji se koriste pri simulaciji, a opisani su u .urdf datoteci, imaju bazu te članke koji su povezani zglobovima. Svaki od zglobova povezuje jedan nadređeni članak s podređenim člankom. Bazu robota moguće je prikazati kao potpuno nepomičnom, bez stupnjeva slobode gibanja ili potpuno slobodnom tako da ima šest stupnjeva slobode gibanja.

getNumJoints( ) – naredba koja za unesenu identifikacijsku oznaku robota kao ulazni parametar daje broj zglobova

getJointInfo( ) – naredba koja za pojedini zglob daje informacije o nazivu zgloba, vrsti, trenju, gornjoj i donjoj granici slobode gibanja, maksimalnoj sili, maksimalnom ubrzanju, nazivu članka i drugo.. Ulazni parametri naredbe su identifikacijska oznaka robota i oznaka zgloba.

setJointMotorControl2( ) – naredba koja se koristi za upravljanje robotom tako da se postavi način upravljanja jednog zgloba. Prema početnim postavkama, svaki revolutni i prizmatični zglob imaju podešen način upravljanja brzinom motora. Ostali načini upravljanja motora koji se nalaze u zglobovima robota su upravljanje položajem i upravljanje momentom motora. Potrebni ulazni argumenti naredbe su identifikacijska oznaka robota, oznaka zgloba, te način upravljanja. Dodatni neobavezni argumenti su željena pozicija zgloba, željena brzina, maksimalna sila kojom se želi postići pozicija i brzina.

Primjer:

```
pybullet.setJointMotorControl2(bodyUniqueId=objid, jointIndex=0,
                               controlMode=pybullet.VELOCITY_CONTROL,
                               targetVelocity= targetVel, force = 500)
```

setJointMotorControlArray( ) – naredba koja se koristi za upravljanje robotom na način da se definiraju načini upravljanja za sve zglobove. Naredba se koristi da bi se izbjeglo pisanje naredbe upravljanja za svaki zglob robota zasebno. Ulazni parametri naredbe su isti kao i kod naredbe *setJointMotorControl2* samo što se u ovoj naredbi upisuju oznake zglobova u nizu.

getJointState( ) – naredba koja služi za dobivanje informacije o stanju zgloba kao što je pozicija, brzina, primijenjeni moment te iznos reakcijskih sila unutar zgloba. Ulazni parametri su identifikacijska oznaka objekta te oznaka zgloba.

getJointStates( ) – naredba koja je ista kao i *getJointState( )* samo što se prilikom korištenja ove naredbe oznake zglobova pišu u nizu te se time dobivaju informacije stanja za više zglobova odjednom

resetJointState( ) – naredba koja služi za ponovno postavljanje stanja zgloba. Potrebno ju je koristiti na početku, prije pokretanja simulacije. Potrebni ulazni parametri naredbe su identifikacijska oznaka objekta, oznaka zgloba, željena vrijednost pozicije ili kuta te dodatna opcija željena vrijednost brzine

getLinkState( ) – naredba koja se koristi za dobivanje podataka o položaju i orijentaciji centra mase članaka robota. Potrebni ulazni parametri naredbe su identifikacijska oznaka objekta, oznaka članka, a dodatni neobavezni ulazni parametri su omogućivanje izračuna brzine članka te omogućivanje izračuna pozicije i orijentacije članka putem direktne kinematike.

getLinkStates( ) – naredba koja omogućuje dobivanje informacija o stanju više članaka odjednom

getBaseVelocity( ) – naredba pomoću koje je moguće dobiti informaciju o linearnoj i kutnoj brzini baze objekta. Potrebni ulazni parametar naredbe je identifikacijska oznaka objekta

resetBaseVelocity( ) – naredba koja služi za ponovno postavljanje linearne ili kutne ili obje brzine baze objekta. Potrebni ulazni parametar naredbe je identifikacijska oznaka objekta dok su dodatne neobavezne mogućnosti postavljanje željene linearne i kutne brzine.

applyExternalForce( ) / **applyExternalTorque( )** – naredbe pomoću kojih je moguće primijeniti vanjsku silu ili moment na objekt. Naredba ne može primijeniti prilikom simulacije u stvarnom vremenu, tj. prilikom korištenja naredbe *setRealTimeSimulation( )*. Potrebni ulazni parametri naredbe su identifikacijska oznaka objekta, oznaka članka na koji se želi primijeniti

sila ili moment, vektor koji sadrži iznos sile ili momenta u tri smjera koordinatnih osi, vektor pozicije na koju se želi primijeniti sila ili moment te je potrebno odrediti koji koordinatni sustav se koristi, globalni ili lokalni.

changeDynamics( ) – naredba koja omogućuje promjenu svojstava objekta kao što su masa, koeficijenti trenja, koeficijenti prigušenja, postavljanja gornjih i donjih granica gibanja zglobova, ograničavanje maksimalne sile i brzine zgloba

5.3. Naredbe korištenja inverzne kinematike i dinamike

calculateInverseKinematics( ) – naredba koja omogućuje izračun kuta zakreta zglobova koji su potrebni kako bi vrh hvataljke robota zauzeo željenu poziciju u prostoru. Potrebni ulazni parametri naredbe su identifikacijska oznaka objekta, broj kojim je označena hvataljka te željena pozicija vrha hvataljke. Dodatne neobavezni ulazni parametri su gornja i donja granica gibanja zglobova, metoda po kojom se izračunavaju vrijednosti kuteva, maksimalni broj iteracija.

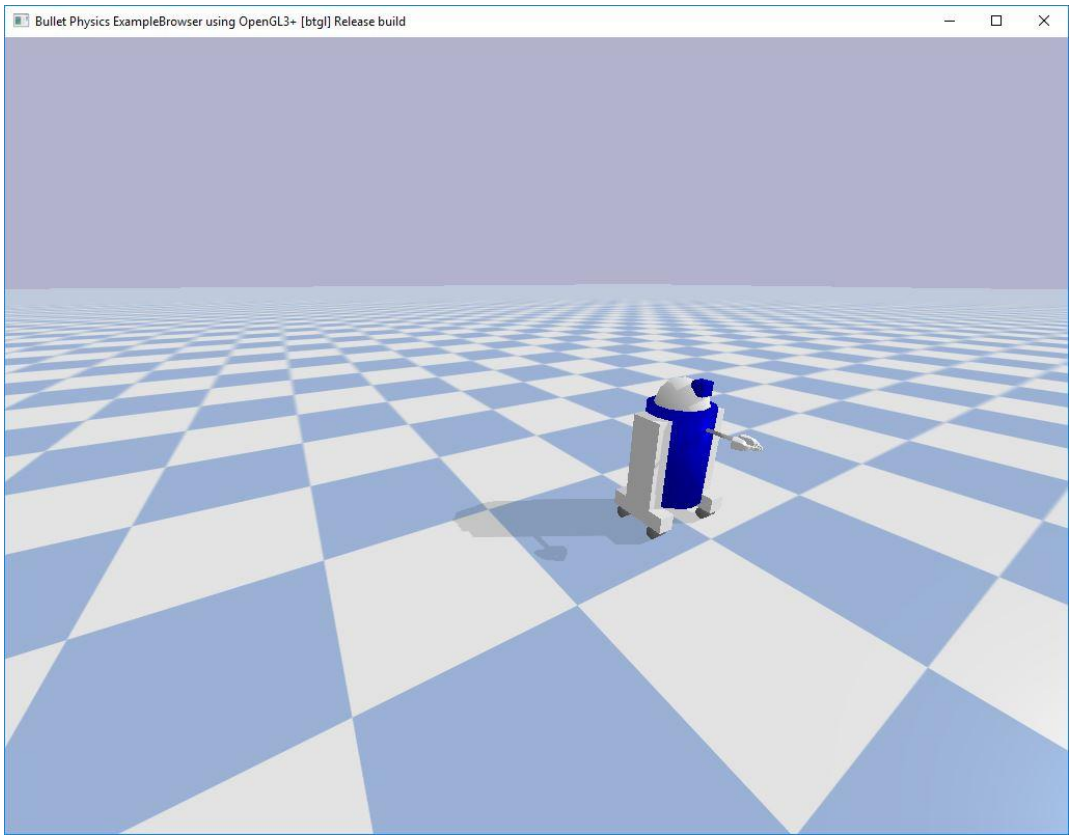
calculateInverseDynamics( ) – naredba koja se koristi za izračun vrijednosti sila koje su potrebne za ostvarenje zadanih ubrzanja zglobova na temelju pozicije i brzine zgloba. Potrebni ulazni parametri naredbe su identifikacijska oznaka objekta, pozicije zglobova i brzine zglobova za svaki stupanj slobode gibanja te željno ubrzanje zglobova.

6. IMPLEMENTACIJA PRIMJERA MOBILNIH ROBOTA U PYBULLET FIZIKALNI SIMULATOR

6.1. Mobilni robot pokretan kotačima

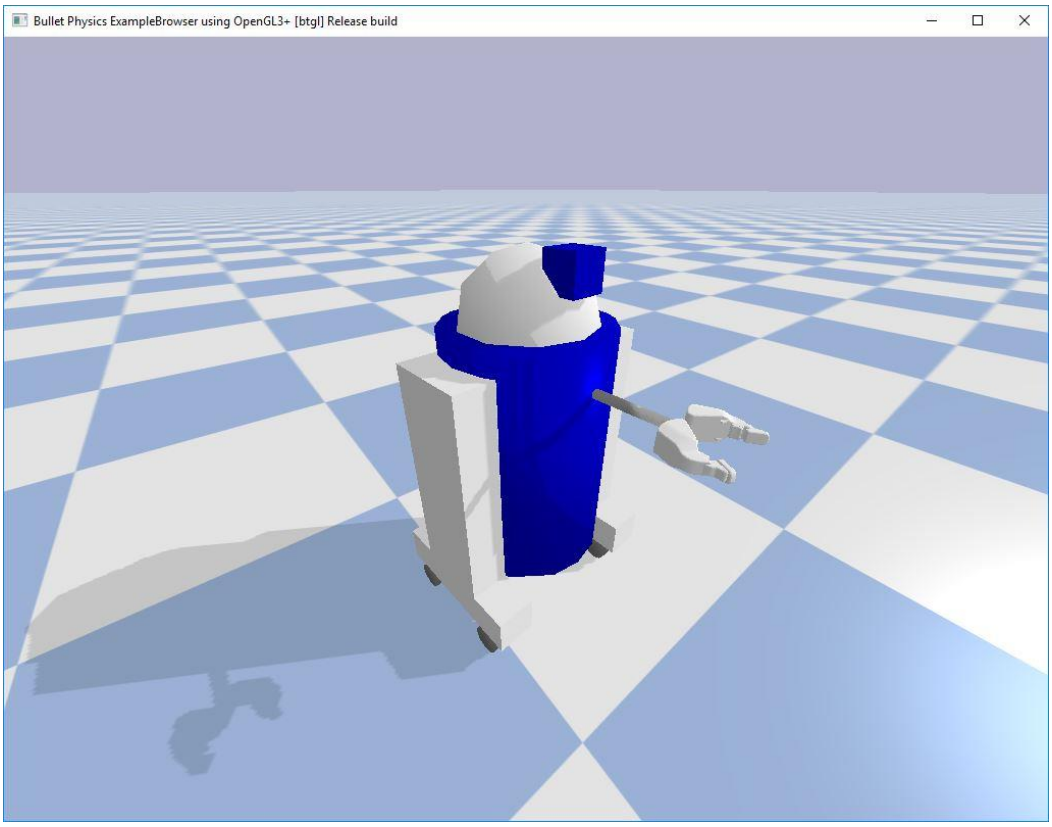
Za primjer mobilnog robota pokretanog kotačima odabran je model robota R2-D2 koji je izmišljeni lik mobilnog robota iz filmske franšize Zvjezdani ratovi (eng. *Star Wars)*. Model robota učitan je u *PyBullet* okruženje iz .urdf datoteke u kojoj je modeliran tako da ima 15 zglobova. U ovom primjeru napisan je programski kod pomoću kojeg se mobilni robot kreće prema naprijed, natrag, otvara, zatvara, uvlači te izvlači hvataljku, okreće se oko svoje osi te okreće glavom. Programski kod ovog primjera dan je u prilogu na kraju rada.

Na [Slika 22] prikazan je mobilni robot u gibanju prema naprijed.

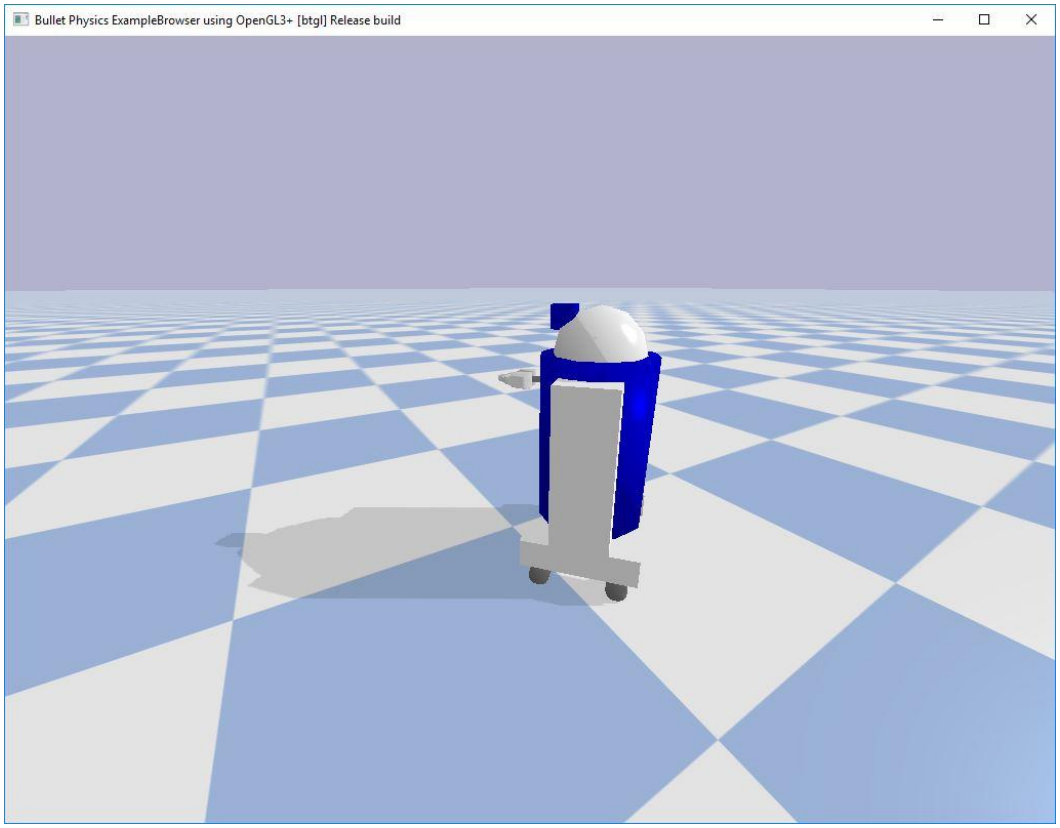


Slika 22. Mobilni robot pokretan kotačima u gibanju

Otvorena hvataljka mobilnog robota prikazana je na [Slika 23], a uvučena i zatvorena hvataljka te okret mobilnog robota prikazani su na [Slika 24].



Slika 23. Otvorena hvataljka mobilnog robota



Slika 24. Okret mobilnog robota s uvučenom i zatvorenom hvataljkom

6.2. Mobilni robot pokretan nogama

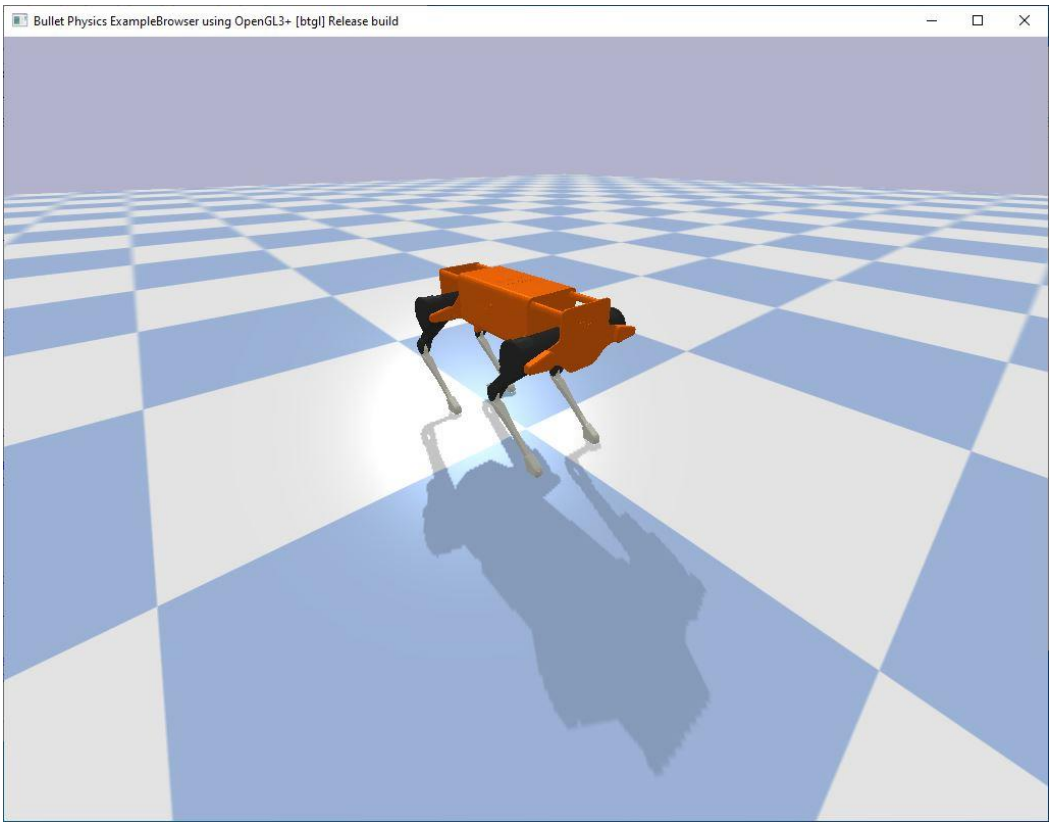
Za primjer mobilnog robota pokretanog nogama odabran je model robota Laikago. To je četveronožni (eng. *quadruped*) robot razvijen kao platforma za istraživanja. Razvijen je u kineskoj tvrtki *Unitree Robotics*. Može hodati naprijed, natrag ili u stranu po neravnim terenima, penjati se uzbrdicama te ostati stabilan u slučajevima u kojima mu se narušava ravnoteža. Ime robota Laikago inspirirano je imenom psa Laika koji je prva životinja koja je lansirana u orbitu Zemlje. [13]

Model robota učitan je u *PyBullet* okruženje iz .urdf datoteke u kojoj je modeliran tako da ima 12 stupnjeva slobode kojima se može upravljati. Svaka noga robota ima po tri zgloba, to su kuk (eng. *hip*), zglob za pomicanje natkoljenice, tj. bedra (eng. *thigh*) te zglob za pomicanje potkoljenice, tj. lista (eng. *calf*). U ovom primjeru napisan je programski kod koji omogućuje hodanje četveronožnog robota prema naprijed. Način hoda robota programiran je tako da robot hoda sporim kasom (eng. *slow trot*), što bi odgovaralo načinu hoda psa ili konja. Spori kas je dvotaktni hod u kojem se noge na jednoj dijagonali istovremeno pomiču prema naprijed potom prema natrag. Pri pomaku nogu prema natrag noge na drugoj dijagonali pomiču se prema naprijed. U hodu je četveronožni robot uvijek oslonjen na dvije dijagonalne noge.

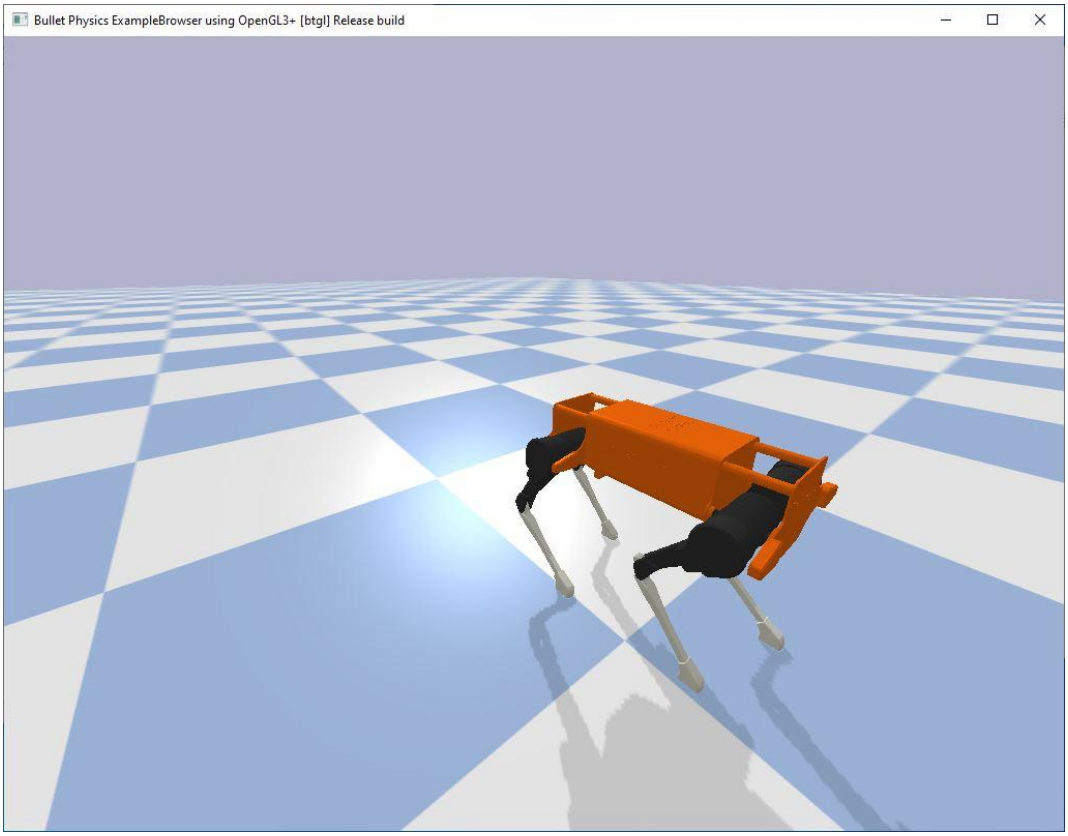
Za razvoj programskog koda koji omogućuje stabilno i zadovoljavajuće hodanje četveronožnog robota bilo je potrebno nešto više od osam sati rada.

Programski kod ovog primjera dan je u prilogu na kraju rada.

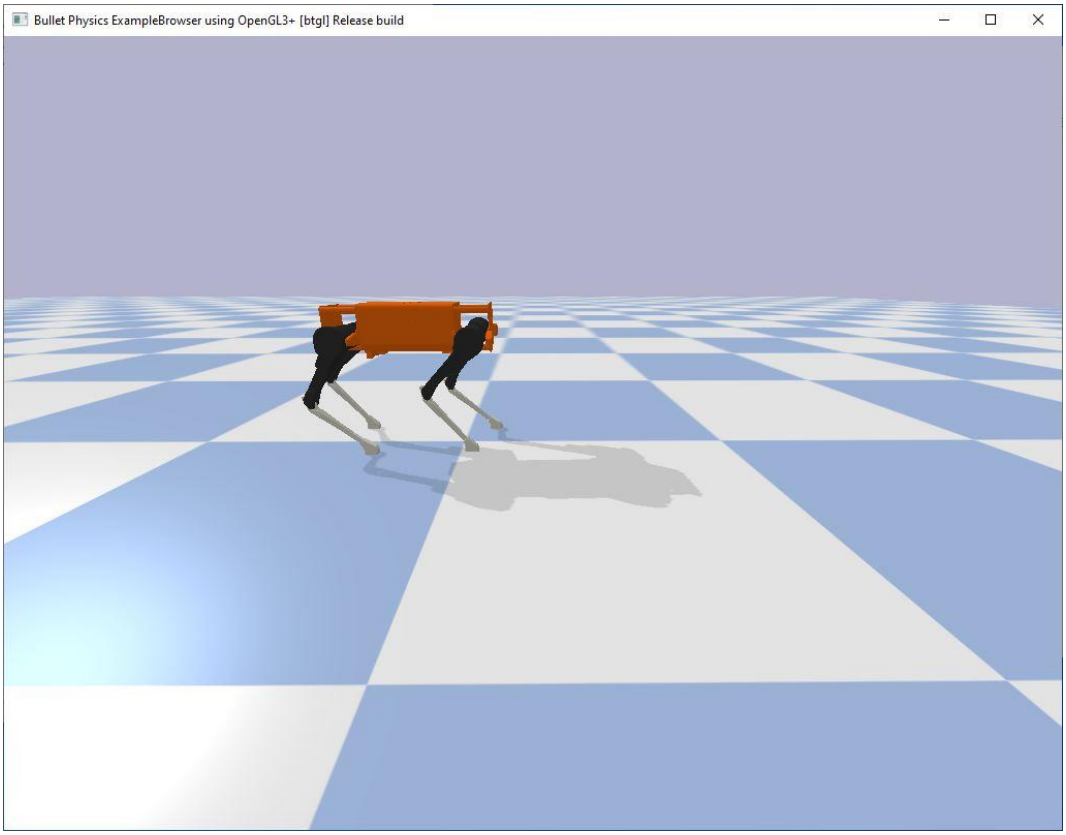
Početna pozicija robota u *PyBullet* okruženju prikazana je na [Slika 25], dok je robot u hodu prikazan na [Slika 26] i [Slika 27].



Slika 25. Četveronožni robot u početnoj poziciji



Slika 26. Četveronožni robot u hodu



Slika 27. Četveronožni robot u hodu – pogled s boka

7. ZAKLJUČAK

Upotreba fizikalnih simulatora u robotici omogućuje ispitivanje, razumijevanje i analizu rada robota u virtualnom okruženju što dovodi do povećane učinkovitosti, sigurnosti te smanjenja troškova. Ovakvi simulatori komplementaran su alat realnim robotskim sustavima. Fizikalni simulator korišten u ovom radu je *PyBullet,* temeljen na *Python* programskom jeziku, koji omogućuje analizu kolizija, simulaciju dinamike krutih i mekih tijela, rad s direktnom i inverznom kinematikom i dinamikom te ima podršku za rad s opremom za virtualnu stvarnost. Moguće su implementacije realističnih robotskih scenarija te je moguća integracija i povezivost sa stvarnim robotima. Budući da je temeljen na *Python*-u, lako ga je integrirati s modulima za strojno učenje poput *TensorFlow*-a ili *PyTorch*-a. U ovom radu napravljena je implementacija dva primjera karakterističnih robotskih konfiguracija. Robot pokretan kotačima kao relativno dostupna struktura te robot pokretan nogama kao složenija i zahtjevnija struktura. Primjeri koji su implementirani dokazali su primjenjivost *PyBullet*-a za oba scenarija te mogu poslužiti kao osnova za daljnji razvoj.

LITERATURA

[1] Liu, C.K., Negrut, D.: The Role of Physics-Based Simulators in Robotics, *Annu. Rev. Control Robot. Auton. Syst.*, 4 (2021.) 35-58

[2] Erleben, K., Sporring, J., Henriksen, K., Dohlmann, H.: Physics-Based Animation, Charles River Media, INC., Massachusetts, 2005.

[3] Wikipedia, Physics engine, https://en.wikipedia.org/wiki/Physics_engine (pristup 25.08.2022.)

[4] Big Data E-Book, Introduction to Physics Simulation, https://sites.psu.edu/bigdataebook/chapter4/04-02/ (pristup 25.08.2022.)

[5] Collins, J., Chand, S., Vanderkop, A., Howard, D.: A Review of Physics Simulators for Robotic Applications, *IEEE*, 9 (2021.) 51416-51431

[6] Coumans, E., Bai, Y.: PyBullet Quickstart Guide, 2022.

[7] Roban, J.: Samosklapajući origami roboti, Diplomski rad, Fakultet strojarstva i brodogradnje, Zagreb, 2021.

[8] Peng, X.B., Coumans, E., Zhang, T., Lee, T.W.E., Tan, J., Levine, S.: Learning Agile Robotic Locomotion Skills by Imitating Animals, 2020.

[9] Matas, J., James, S., Davison, A.J.: Sim-to-Real Reinforcement Learning for Deformable Object Manipulation, 2018.

[10] Zeng, A., Song, S., Lee, J., Rodriguez, A., Funkhouser, T.: Tossing Bot: Learning to Throw Arbitrary Objects With Residual Physics, *IEEE Transactions on Robotics*, 36 (2020.) 1307-1319

[11] Tan, J., Zhang, T., Coumans, E., Iscen, A., Bai, Y., Hafner, D., Bohez, S., Vanhoucke, V.: Sim-to-Real: Learning Agile Locomotion For Quadruped Robots, 2018.

[12] Mahler, J., Goldberg, K.: Learning Deep Policies for Robot Bin Picking by Simulating Robust Grasping Sequences, *CoRL*, 2017.

[13] Laikago – ROBOTS, https://robots.ieee.org/robots/laikago/ (pristup 12.09.2022.)

PRILOZI

I. Programski kod primjera mobilnog robota pokretanog kotačima

II. Programski kod primjera mobilnog robota pokretanog nogama

I. Programski kod primjera mobilnog robota pokretanog kotačima

```
import pybullet as p   # ucitavanje modula pybullet pod skracenim nazivom p
from time import sleep # ucitavanje naredbe sleep iz modula time
import pybullet_data   # ucitavanje biblioteke pybullet_data

# povezivanje s fizikalnim posluziteljem putem posluzitelja 'GUI',
# tj. prikaz grafickog korisnickog sucelja
physicsClient = p.connect(p.GUI)

# naredba koja omogucuje ucitavanje datoteka iz biblioteke pybullet_data
p.setAdditionalSearchPath(pybullet_data.getDataPath())

planeId = p.loadURDF("plane.urdf")  # ucitavanje ravnine iz .urdf datoteke

# postavljanje pocetne pozicije robota u Kartezijevom koordinatnom sustavu
robotStartPos = [0, 0, 1]

# Postavljanje pocetne orijentacije robota. U PyBullet programskom sucelju,
# tj. API-u, koristi se kvaternion za prikaz orijentacija. Kvaternion je
# algebarsko prosirenje kompleksnih brojeva te sadrzi tri imaginarne
# jedinice i, j i k. Kvaternion nije lako za predociti stoga se pomocu
# naredbe p.getQuaternionFromEuler() upisuju vrijednosti rotacije u
# Euler-ovim kutevima rotacije te se pretvaraju u zapis kvaterniona.
robotStartOrientation = p.getQuaternionFromEuler([0, 0, 0])

# ucitavanje mobilnog robota iz .urdf datoteke sa zadanom pozicijom i
# orijentacijom
robotId = p.loadURDF("r2d2.urdf", robotStartPos, robotStartOrientation)

# dobivanje informacije o broju zglobova robota
a = p.getNumJoints(robotId)
print('Broj zglobova =',a)

# ispis informacija o pojedinom zglobu robota
for j in range (p.getNumJoints(robotId)):
    print(p.getJointInfo(robotId,j))

# pokretanje simulacije u stvarnom vremenu
p.setRealTimeSimulation(1)
sleep(1)

# Oznake zglobova i raspon vrijednosti pojedinih zglobova ocitani su iz
# ispisa koji je dobiven naredbom p.getJointInfo() te su odabrane
# vrijednosti koje zadovoljavaju pojedinu funkciju.

p_DK = 2        # oznaka zgloba prvog desnog kotaca
z_DK = 3        # oznaka zgloba zadnjeg desnog kotaca
p_LK = 6        # oznaka zgloba prvog lijevog kotaca
z_LK = 7        # oznaka zgloba zadnjeg lijevog kotaca
l_zgl_H = 9     # oznaka lijevog zgloba hvataljke
d_zgl_H = 11    # oznaka desnog zgloba hvataljke
glv = 13        # oznaka zgloba glave robota
lnk = 8         # oznaka zgloba clanka na kojem je hvataljka


Sila_stp =  1   # sila koja se koristi pri zaustavljanju kotaca robota
Sila_okr = 10   # sila koja se koristi pri okretanju robota
```

```
# sila pri okretanju glave robota i otvaranju i zatvaranju hvataljke
Sila_h = 50
Sila_nprd = 10  # sila pri pokretanju kotaca robota prema naprijed


# Funkcija naprijed() pokrece kotace robota tako da se robot krece prema
# naprijed. Motori u kotacima upravljani su brzinom te silom kojom se
# postize ta brzina.
def naprijed():
# cekanje od jedne sekunde da bi se izbjeglo prevrtanje prilikom stvaranja
# robota u PyBullet okruzenju
    sleep(1)
    p.setJointMotorControl2(robotId,p_DK,p.VELOCITY_CONTROL,
                            targetVelocity=- 50,force=Sila_nprd)

    p.setJointMotorControl2(robotId,z_DK,p.VELOCITY_CONTROL,
                            targetVelocity=-50,force=Sila_nprd)

    p.setJointMotorControl2(robotId,p_LK,p.VELOCITY_CONTROL,
                            targetVelocity=-50,force=Sila_nprd)

    p.setJointMotorControl2(robotId,z_LK,p.VELOCITY_CONTROL,
                            targetVelocity=-50,force=Sila_nprd)
    sleep(3)    # robot se krece prema naprijed tri sekunde


# Funkcija stani() zaustavlja gibanje robota na nacin da se brzina svih
# kotaca postavi na vrijednost nula.
def stani():
    p.setJointMotorControl2(robotId,p_DK,p.VELOCITY_CONTROL,
                            targetVelocity=0,force=Sila_stp)

    p.setJointMotorControl2(robotId,z_DK,p.VELOCITY_CONTROL,
                            targetVelocity=0,force=Sila_stp)

    p.setJointMotorControl2(robotId,p_LK,p.VELOCITY_CONTROL,
                            targetVelocity=0,force=Sila_stp)

    p.setJointMotorControl2(robotId,z_LK,p.VELOCITY_CONTROL,
                            targetVelocity=0,force=Sila_stp)
    sleep(1)    # cekanje jednu sekundu

# Funkcija okreni_glavu() sluzi za okretanje glave robota. Motor koji se
# nalazi u zglobu glave robota upravljan je pozicijom.
def okreni_glavu():
    p.setJointMotorControl2(robotId,glv,p.POSITION_CONTROL,
                            targetPosition=-0.38,force=Sila_h)
    sleep(1)
    p.setJointMotorControl2(robotId,glv,p.POSITION_CONTROL,
                            targetPosition=0,force=Sila_h)
    sleep(1)

# Funkcija otvori_hvataljku() sluzi za otvaranje hvataljke robota
def otvori_hvataljku():
    p.setJointMotorControl2(robotId,l_zgl_H,p.POSITION_CONTROL,
                            targetPosition=0.548,force=Sila_h)

    p.setJointMotorControl2(robotId,d_zgl_H,p.POSITION_CONTROL,
                            targetPosition=0.548,force=Sila_h)
    sleep(1)
```

```
# Funkcija zatvori_hvataljku() sluzi za zatvaranje hvataljke robota
def zatvori_hvataljku():
    p.setJointMotorControl2(robotId,l_zgl_H,p.POSITION_CONTROL,
                            targetPosition=0,force=Sila_h)

    p.setJointMotorControl2(robotId,d_zgl_H,p.POSITION_CONTROL,
                            targetPosition=0,force=Sila_h)
    sleep(1)


# Funkcija uvuci_hvataljku() sluzi za uvlacenje hvataljke robota
def uvuci_hvataljku():
    p.setJointMotorControl2(robotId,lnk,p.POSITION_CONTROL,
                            targetPosition=-0.15,force=5)
    sleep(1)


# Funkcija izvuci_hvataljku sluzi za izvlacenje hvataljke robota
def izvuci_hvataljku():
    p.setJointMotorControl2(robotId,lnk,p.POSITION_CONTROL,
                            targetPosition=0,force=2)
    sleep(1)

# Funkcija pomak_naprijed() sluzi za mali pomak robota prema naprijed
def pomak_naprijed():
    p.setJointMotorControl2(robotId,p_DK,p.VELOCITY_CONTROL,
                            targetVelocity=-3,force=Sila_nprd)

    p.setJointMotorControl2(robotId,z_DK,p.VELOCITY_CONTROL,
                            targetVelocity=-3,force=Sila_nprd)

    p.setJointMotorControl2(robotId,p_LK,p.VELOCITY_CONTROL,
                            targetVelocity=-3,force=Sila_nprd)

    p.setJointMotorControl2(robotId,z_LK,p.VELOCITY_CONTROL,
                            targetVelocity=-3,force=Sila_nprd)
    sleep(3)

# Funkcija pomak_nazad() sluzi za mali pomak robota prema nazad
def pomak_nazad():
    p.setJointMotorControl2(robotId,p_DK,p.VELOCITY_CONTROL,
                            targetVelocity=5,force=Sila_nprd)

    p.setJointMotorControl2(robotId,z_DK,p.VELOCITY_CONTROL,
                            targetVelocity=5,force=Sila_nprd)

    p.setJointMotorControl2(robotId,p_LK,p.VELOCITY_CONTROL,
                            targetVelocity=5,force=Sila_nprd)

    p.setJointMotorControl2(robotId,z_LK,p.VELOCITY_CONTROL,
                            targetVelocity=5,force=Sila_nprd)
    sleep(3)


# Funkcija okreni_se() sluzi za okretanje robota. Robot se okrece 2.58
# sekundi te potom staje. Robot se okrece na nacin da se desni kotaci
# okrecu prema nazad, a lijevi kotaci prema naprijed.
def okreni_se():
    p.setJointMotorControl2(robotId,p_DK,p.VELOCITY_CONTROL,
                            targetVelocity=20,force=Sila_okr)
```

```
    p.setJointMotorControl2(robotId,z_DK,p.VELOCITY_CONTROL,
                            targetVelocity=20,force=Sila_okr)

    p.setJointMotorControl2(robotId,p_LK,p.VELOCITY_CONTROL,
                            targetVelocity=-20,force=Sila_okr)

    p.setJointMotorControl2(robotId,z_LK,p.VELOCITY_CONTROL,
                            targetVelocity=-20,force=Sila_okr)
    sleep(2.58)
    p.setJointMotorControl2(robotId,p_DK,p.VELOCITY_CONTROL,
                            targetVelocity=0,force=Sila_stp)

    p.setJointMotorControl2(robotId,z_DK,p.VELOCITY_CONTROL,
                            targetVelocity=0,force=Sila_stp)

    p.setJointMotorControl2(robotId,p_LK,p.VELOCITY_CONTROL,
                            targetVelocity=0,force=Sila_stp)

    p.setJointMotorControl2(robotId,z_LK,p.VELOCITY_CONTROL,
                            targetVelocity=0,force=Sila_stp)
    sleep(1)

while(True):
    p.setGravity(0, 0, -9.81)     # postavljanje gravitacijske sile
    sleep(0.01)

# Petlja u kojoj se pozivaju prethodno definirane funkcije te se ispisuje
# korak koji se trenutno izvrsava.
    for i in range(4):
        if (i==0):
            print('Prilaz mjestu izuzimanja i uzimanje predmeta')
            naprijed()
            stani()
            okreni_glavu()
            otvori_hvataljku()
            pomak_naprijed()
            stani()
            zatvori_hvataljku()
            uvuci_hvataljku()
            pomak_nazad()

        if (i==1):
            print('Okret')
            okreni_se()

        if (i==2):
            print('Prilaz mjestu odlaganja i odlaganje predmeta')
            naprijed()
            stani()
            izvuci_hvataljku()
            pomak_naprijed()
            stani()
            otvori_hvataljku()
            pomak_nazad()
            stani()
            zatvori_hvataljku()

        if (i==3):
            print('Okret')
            okreni_se()
```

II. Programski kod primjera mobilnog robota pokretanog nogama

```
import pybullet as p   # ucitavanje modula pybullet pod skracenim nazivom p
from time import sleep # ucitavanje naredbe sleep iz modula time

# povezivanje s fizikalnim posluziteljem putem posluzitelja 'GUI',
# tj.prikaz grafickog korisnickog sucelja
p.connect(p.GUI)


planeId = p.loadURDF("plane.urdf")   # ucitavanje ravnine iz .urdf datoteke
p.setGravity(0,0,-9.81)    # postavljanje gravitacijske sile

laikagoStartPos = [0, 0, 0.48]    #definiranje pocetne pozicije robota

# definiranje pocetne orijentacije robota pomocu zapisa kvaterniona, zapis
#  [0, 0, 0, 1] oznacava da nema rotacije
laikagoStartOrientation = [0,0,0,1]

# ucitavanje mobilnog robota iz .urdf datoteke sa zadanom pozicijom i
# orijentacijom te onemogucivanje fiksne baze robota
laikagoId = p.loadURDF("laikago/urdf/laikago.urdf", laikagoStartPos,
laikagoStartOrientation, useFixedBase=False)

# dobivanje informacije o broju zglobova robota
a = p.getNumJoints(laikagoId)
print('Broj zglobova =',a)

# ispis informacija o pojedinom zglobu robota
for j in range (p.getNumJoints(laikagoId)):
        print(p.getJointInfo(laikagoId,j))

# pokretanje simulacije u stvarnom vremenu
p.setRealTimeSimulation(1)
sleep(1)

# Oznake zglobova, nazivi te raspon vrijednosti pojedinih zglobova ocitani
# su iz ispisa koji je dobiven naredbom p.getJointInfo() te su odabrane
# vrijednosti koje zadovoljavaju pojedinu funkciju.

# PREDNJA DESNA NOGA
PD_kuk = 0    # oznaka zgloba kuka (eng. hip) prednje desne noge

# oznaka zgloba za pomicanje bedra (eng. thigh), tj. natkoljenice prednje
# desne noge
PD_bedro = 1

# oznaka zgloba za pomicanje lista (eng. calf), tj. potkoljenice prednje
# desne noge
PD_list = 2

# PREDNJA LIJEVA NOGA
PL_kuk = 3    # oznaka zgloba kuka (eng. hip) prednje lijeve noge

# oznaka zgloba za pomicanje bedra (eng. thigh), tj. natkoljenice prednje
# lijeve noge
PL_bedro = 4
```

```
# oznaka zgloba za pomicanje lista (eng. calf), tj. potkoljenice prednje
# lijeve noge
PL_list = 5

# ZADNJA DESNA NOGA
ZD_kuk = 6    # oznaka zgloba kuka (eng. hip) zadnje desne noge

# oznaka zgloba za pomicanje bedra (eng. thigh), tj. natkoljenice zadnje
# desne noge
ZD_bedro = 7

# oznaka zgloba za pomicanje lista (eng. calf), tj. potkoljenice zadnje
# desne noge
ZD_list = 8

# ZADNJA LIJEVA NOGA
ZL_kuk = 9    # oznaka zgloba kuka (eng. hip) zadnje lijeve noge

# oznaka zgloba za pomicanje bedra (eng. thigh), tj. natkoljenice zadnje
# lijeve noge
ZL_bedro = 10

# oznaka zgloba za pomicanje lista (eng. calf), tj. potkoljenice zadnje
# lijeve noge
ZL_list = 11

Sila = 20  # sila koja se koristi pri pokretanju motora u zglobovima robota

bedro_poc = 0.7    # pocetna vrijednost zgloba bedra
list_poc = -1.5    # pocetna vrijednost zgloba lista


# vrijednost zgloba bedra koja se postavlja pri pomaku bedra prema nazad
bedro_naz = 0.7

# vrijednost zgloba bedra koja se postavlja pri pomaku bedra prema naprijed
bedro_napr = 0.45

# vrijednost zgloba lista koja se postavlja pri pomaku lista prema nazad
list_naz = -1.28

# vrijednost zgloba lista koja se postavlja pri pomaku lista prema naprijed
list_napr = -1.31

slp = 0.04    # iznos vremena cekanja izmedu izvrsavanja naredbi

# Funkcija pocetna_poz() postavlja robota u pocetnu poziciju. Prilikom
# ucitavanja modela robota iz .urdf datoteke robot stoji s ispruzenim
# nogama tako da su okomite na ravninu na kojoj stoji sto ga cini
# nestabilnim pri hodanju, zbog toga potrebno ga je spustiti u pocetnu
# poziciju. Postavljanje robota u pocetnu poziciju vrsi se postavljanjem
# pocetnih vrijednosti pozicije motora u svim zglobovima za pokretanje
# bedra zatim lista.
def pocetna_poz():
    sleep(2)    # cekanje dvije sekunde pri ucitavanju robota
    p.setJointMotorControl2(laikagoId,ZD_bedro,p.POSITION_CONTROL,
                            targetPosition=bedro_poc,force=Sila)

    p.setJointMotorControl2(laikagoId,ZL_bedro,p.POSITION_CONTROL,
                            targetPosition=bedro_poc,force=Sila)
```

```
    p.setJointMotorControl2(laikagoId,PL_bedro,p.POSITION_CONTROL,
                            targetPosition=bedro_poc,force=Sila)

    p.setJointMotorControl2(laikagoId,PD_bedro,p.POSITION_CONTROL,
                            targetPosition=bedro_poc,force=Sila)
# cekanje 0.01 sekunde izmedu pomaka bedra i lista da bi se izbjeglo
# rusenje robota
    sleep(0.01)

    p.setJointMotorControl2(laikagoId,ZD_list,p.POSITION_CONTROL,
                            targetPosition=list_poc,force=Sila)

    p.setJointMotorControl2(laikagoId,ZL_list,p.POSITION_CONTROL,
                            targetPosition=list_poc,force=Sila)

    p.setJointMotorControl2(laikagoId,PL_list,p.POSITION_CONTROL,
                            targetPosition=list_poc,force=Sila)

    p.setJointMotorControl2(laikagoId,PD_list,p.POSITION_CONTROL,
                            targetPosition=list_poc,force=Sila)
    sleep(1.5)    # cekanje 1.5 sekunde za izvrsavanje sljedece naredbe


# Funkcija PD_ZL_nazad() pokrece prednju desnu nogu i zadnju lijevu nogu
# robota prema nazad na nacin da se najprije spusti potkoljenica robota te
# se nakon 0.04 sekunde pomakne natkoljenica prema nazad za zadane
# vrijednosti.
def PD_ZL_nazad():
    p.setJointMotorControl2(laikagoId,PD_list,p.POSITION_CONTROL,
                            targetPosition=list_naz,force=Sila)

    p.setJointMotorControl2(laikagoId,ZL_list,p.POSITION_CONTROL,
                            targetPosition=list_naz,force=Sila)

# cekanje od 0.04 sekunde izmedu pomaka potkoljenice i natkoljenice robota
    sleep(slp)

    p.setJointMotorControl2(laikagoId,PD_bedro,p.POSITION_CONTROL,
                            targetPosition=bedro_naz,force=Sila)

    p.setJointMotorControl2(laikagoId,ZL_bedro,p.POSITION_CONTROL,
                            targetPosition=bedro_naz,force=Sila)

# Funkcija PD_ZL_naprijed() pokrece prednju desnu nogu i zadnju lijevu nogu
# robota prema naprijed na nacin da se najprije podigne potkoljenica robota
# te se nakon 0.04 sekunde pomakne natkoljenica robota prema naprijed za
# zadane vrijednosti.
def PD_ZL_naprijed():
    p.setJointMotorControl2(laikagoId,PD_list,p.POSITION_CONTROL,
                            targetPosition=list_napr,force=Sila)

    p.setJointMotorControl2(laikagoId,ZL_list,p.POSITION_CONTROL,
                            targetPosition=list_napr,force=Sila)
    sleep(slp)
    p.setJointMotorControl2(laikagoId,PD_bedro,p.POSITION_CONTROL,
                            targetPosition=bedro_napr,force=Sila)

    p.setJointMotorControl2(laikagoId,ZL_bedro,p.POSITION_CONTROL,
                            targetPosition=bedro_napr,force=Sila)
    sleep(slp)     # cekanje od 0.04 sekunde za izvrsavanje sljedece naredbe
```

```
# Funkcija PL_ZD_nazad() pokrece prednju lijevu nogu i zadnju desnu nogu
# robota prema nazad na nacin da se najprije spusti potkoljenica robota te
# se nakon 0.04 sekunde pomakne natkoljenica prema nazad za zadane
# vrijednosti.
def PL_ZD_nazad():
    p.setJointMotorControl2(laikagoId,PL_list,p.POSITION_CONTROL,
                            targetPosition=list_naz,force=Sila)

    p.setJointMotorControl2(laikagoId,ZD_list,p.POSITION_CONTROL,
                            targetPosition=list_naz,force=Sila)
    sleep(slp)
    p.setJointMotorControl2(laikagoId,PL_bedro,p.POSITION_CONTROL,
                            targetPosition=bedro_naz,force=Sila)

    p.setJointMotorControl2(laikagoId,ZD_bedro,p.POSITION_CONTROL,
                            targetPosition=bedro_naz,force=Sila)


# Funkcija PL_ZD_naprijed() pokrece prednju lijevu nogu i zadnju desnu nogu
# robota prema naprijed na nacin da se najprije podigne potkoljenica robota
# te se nakon 0.04 sekunde pomakne natkoljenica robota prema naprijed za
# zadane vrijednosti.
def PL_ZD_naprijed():
    p.setJointMotorControl2(laikagoId,PL_list,p.POSITION_CONTROL,
                            targetPosition=list_napr,force=Sila)

    p.setJointMotorControl2(laikagoId,ZD_list,p.POSITION_CONTROL,
                            targetPosition=list_napr,force=Sila)
    sleep(slp)
    p.setJointMotorControl2(laikagoId,PL_bedro,p.POSITION_CONTROL,
                            targetPosition=bedro_napr,force=Sila)

    p.setJointMotorControl2(laikagoId,ZD_bedro,p.POSITION_CONTROL,
                            targetPosition=bedro_napr,force=Sila)
    sleep(slp)


pocetna_poz()    # poziv funkcije koja robota postavlja u pocetnu poziciju

# beskonacna petlja u kojoj se naizmjenicno pozivaju funkcije koje
# omogucuju hodanje robota
while(True):
    PD_ZL_naprijed()
    PD_ZL_nazad()
    PL_ZD_naprijed()
    PL_ZD_nazad()
```