Primjena simulacijskog okruženja za oblikovanje radnog zadatka robota

Knežević, Mario

Undergraduate thesis / Završni rad

2022

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:235:473744

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mario Knežević

Zagreb, 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentor:

Doc. dr. sc. Marko Švaco, mag. ing.

Student:

Mario Knežević

Zagreb, 2022.

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

Mario Knežević

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: **Mario Knežević** JMBAG: **0035211794**

Naslov rada na hrvatskom jeziku: **Primjena simulacijskog okruženja za oblikovanje radnog zadatka robota**

Naslov rada na engleskom jeziku: **Application of a simulation environment to design a robot task**

Opis zadatka:

Simulacija rada robota bitan je alat u razvoju te kasnijoj implementaciji robotskih rješenja. Dobro oblikovan simulacijski proces omogućuje brzo testiranje algoritama i oblikovanje radnog zadatka robota koristeći realistične scenarije. Prednosti korištenja alata za simulaciju su u tome što štede vrijeme u oblikovanju robotskih aplikacija, a također mogu povećati razinu sigurnosti povezane s robotskom opremom budući da se razni scenariji mogu validirati u simulacijskom okruženju.

U završnom radu potrebno je korištenjem dostupnog simulacijskog softvera napraviti vlastito virtualno okruženje za simulaciju aplikacije robotskog rukovanja. Na temelju prostornih i fizikalnih ograničenja radne okoline potrebno je oblikovati i verificirati robotske programe za manipulaciju predmetima rada. Zadatak uključuje i implementaciju simulacijski planiranih gibanja robota na postojećem robotu u Laboratoriju za autonomne sustave.

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

Zadatak zadan:
30. 11. 2021.

Zadatak zadao:
Doc. dr. sc. Marko Švaco

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

Tema ovog rada je opisati aktualne simulacijske softvere i objasniti njihovu ulogu u današnjoj industriji te izraditi funkcionalno virtualno okruženje u simulatoru *NVIDIA Isaac Sim*. Osim toga, podrobno je opisana struktura samog simulatora te način rada u njemu. U sklopu izrade virtualnog okruženja, objašnjen je proces dodavanja objekata, definiranja fizikalnih zakona, sklapanja robota i dodavanja industrijskih robota. Naposljetku je opisana izrada ekstenzija te primjena simulacija za manipulaciju radnih objekata.

Ključne riječi: Simulacija, NVIDIA Isaac Sim, Virtualno okruženje, Ekstenzije

SUMMARY

The topic of this paper is to describe different simulation softwares and their role in today's industry, and to create a functional virtual environment in the NVIDIA Isaac Sim simulator. The structure of the simulator itself and how it works were explained in the first half of the paper. While the process of creating a virtual environment, the process of adding objects, defining physical laws, assembling robots and adding robots were explained in the second part of the paper. Creation of extensions and the application of simulations for the manipulation of work objects were explained at the end.

Keywords: Simulation, NVIDIA Isaac Sim, Virtual Environment, Extensions

1. UVOD

Simulacija je postala temelj održavanja kvalitete proizvodnje. Samim time se razvijaju potrebni alati sa svrhom pojednostavljenja već dovoljno kompleksnih sustava. Brz razvitak industrije uvjetuje precizne i što realnije simulacijske softvere koji bi isporučili pouzdane rezultate.

U prvom dijelu rada se postavio teorijski okvir o simulacijskim alatima uključujući vrste simulacijskih softvera i njihove značajke. Osim toga, podrobnije se opisao *NVIDIA Isaac Sim* simulator, što on predstavlja i njegovu ulogu u razvijanju simulacijskih alata.

U drugom dijelu rada se objasnio princip postavljanja i upravljanja virtualnog okruženja u *Isaac Sim* simulatoru, te izrada ekstenzija, odnosno aplikacija za realizaciju robotskih zadataka. Virtualno okruženje u tom kontekstu je virtualna scena s definiranim fizikalnim zakonima, proizvoljnim značajkama i raznim objektima kojima se upravlja različitim industrijskim robotima.

2. RAZVOJ SIMULACIJSKIH ALATA

Simulacija je definirana kao imitacija procesa u stvarnom svijetu kroz određeno vrijeme, drugim riječima simulacija je pokušaj preslikavanja mehanizama i sustava onako kako bi se oni odvijali u stvarnosti pomoću matematičkih i logičkih zaključaka [1].

Potreba za konzistentnom kvalitetom završnog proizvoda u različitim industrijama je dovela do neophodnog razvoja širokog spektra simulacijskih platformi i alata. Simulacija kao takva je značajna u razvoju proizvoda i nadogradnji istih kako bi se smanjilo vrijeme razvoja i uštedilo na iscrpnim resursima. Ideja simulacije je da se predvide smetnje i problemi onda kada to ne utječe na cjelokupnu infrastrukturu na način da se ideje testiraju u nižem sloju razvoja, u sigurnoj okolini na računalu. Danas, pogotovo u strojarstvu, ali i šire, simulacija djeluje kao važan faktor u povećanju kvalitete, ne samo unutar razvoja proizvoda, nego i van, u vidu konteksta napretka tehnologije i tehnoloških rješenja [2].

Stoga razvojem simulacijskih alata amortizira se sve veća kompleksnost tehnologije, proizvodnje i automatizacije. Također, simulacija pridonosi zaštiti na radu i omogućava jednostavniju prilagodbu tehnološkim problemima. Simulacija zbog gore navedenih razloga je neizdvojiv element složene mreže interdisciplinarnosti koja je nužna za budući iskorak tehnologije [2].

Prije gotovo sedamdeset godina izumljena je jedna od prvih konkretnih metodologija za simulaciju pod nazivom *Discrete Event Simulation*, definirana kao model u kojoj se stanje varijabli mijenja samo na diskretnim točkama u vremenu kad se događaj pojavi [1]. Koristi se u svrhu simuliranja sustava koji se žele podrobnije razumjeti. U nastavku se radilo na novim metodama za novu generaciju simulacijskih alata, jedna od kojih je *Agent Based Simulation* – kolekcija heterogenih, inteligentnih i interaktivnih agenata koji operiraju i postoje u okolišu koji je zauzvrat također sastavljen od agenata. Agenti u tom kontekstu predstavljaju objekte sa svojstvom izvođenja zadataka i komunikacijom s ostalim objektima [3]. U paru sa *Object-Oriented* programiranjem ova metoda stvara potrebne alate za autonomne sustave [2].

Činjenica da su prethodne metodologije bile razvijene u vrijeme kada je računalna tehnologija bila značajno ograničena rezultirala je time da su svi pokušaji u razvijanju alata za simulaciju bili krajnje pojednostavljeni. Filozofija iza korištenih metoda je ostala nepromijenjena do razvitka modernijih računalnih alata. Pojavom istih, krajem 90-ih godina, zanimacija za

razvijanje simulacijskih metodologija je drastično porasla, te su napisani mnogi znanstveni radovi koji su pojašnjavali problematiku i potrebu proizvođača za simulacijskim alatima [4].

Neki od čestih izazova s kojima se industrija suprotstavljala su: reduciranje ciklusa rješavanja problema, razvoj sposobnosti rješavanja problema temeljene na simulaciji u stvarnom vremenu, uključi-i-pokreni (eng. *plug-and-play*) interoperabilnost između simulacije i drugih softverskih alata. Razvoj simulacijskih alata najviše je pogodovao automobilskoj industriji i industriji poluvodiča, zbog čega su tadašnji softveri najviše bili prilagođeni okolini koja je imala svojstva sistematičke proizvodnje – bolja vizualizacija uključujući virtualnu stvarnost i biblioteke sa povuci-i-pusti (eng. *drag-and-drop*) komponentama [4].

Iako je potreba za simulacijom u industriji bila odavno važna, jasno je bilo da razvoj samih alata nije na zadovoljavajućoj razini koja bi opravdala ulaganje. Problem se javljao u ekstrapolaciji podataka unutar razvoja. Unatoč njihovoj obilatosti, sama ekstrapolacija podataka je složena i skupa, te nije garantirana njihova točnost. Nadalje, očito je da se promjena proizvodnje događa brzo sa praktički neograničenim priljevom podataka u sastavu sa umjetnom inteligencijom u stvarnom vremenu. Slojevitost razvoja tehnologije ostaje ista ili postaje još kompleksnija, težnja za edukacijom je sve zahtjevnija, te rješenja koja bi karakterizirala budućnost četvrte razine industrije postaju sve apstraktnija. Razumijevanje kompleksnih sustava se nastavlja biti zahtjev u implementaciji kvalitetnih modela [4].

3. SIMULACIJSKI SOFTVERI

Kao što je već navedeno, simulatori igraju važeću ulogu u istraživanju robotike pogotovo u vidu testiranja robusnosti, sigurnosti i efikasnosti sustava i algoritama. Osobito je ključno testiranje u scenarijima koji zahtijevaju robote da budu u neposrednoj blizini ili interakciji s ljudima. U tom kontekstu su se dizajnirali razni simulacijski programski paketi. U nastavku će se usporediti neki od softvera koji su osmišljeni za ispitivanje na industrijskim i mobilnim robotima i u tom području najbolje djeluju, no sami paketi mogu poslužiti i u ostalim okolinama [5].

Postoje komercijalni simulacijski softveri i softveri otvorenog-izvora (eng. *open-source*). Komercijalni softveri zahtijevaju plaćene licence čiji pristup izvornom kodu (eng. *source code*) imaju samo programeri, timovi ili organizacije koji su taj kod napravili, te su blisko u suradnji s pojedinim industrijama u cilju prikupljanja podataka, dok softveri otvorenog-izvora imaju najčešće benefit besplatne instalacije koji pružaju korisnicima da imaju otvoren pristup izvornom kodu da ga mijenjaju, nadograđuju, proučavaju i dijele, a prikupljanje podataka većinom dobavljaju od korisnika. Odluka hoće li pojedini softver biti komercijalan ili otvorenog-izvora stoji isključivo iza marketinške strategije koju određena firma odabere. *Dassault Systemes Delmia V5, RoboDK, Simulink* su jedni od najpopularnijih komercijalnih softvera dok su *Gazebo* i *Webots* otvorenog-izvora [5]. Ovi programski paketi su čvrsto uspostavljeni na tržištu i već dugi niz godina usavršavaju implementaciju alata za simulacijske sustave, međutim u 2018. se počeo razvijati *Nvidia Isaac Simulator*, još jedan simulacijski sustav otvorenog-izvora čiji pristup izgradnje je ponešto drugačiji od prijašnje navedenih i koji nagovještava vrstu dizajna potrebnog za daljnje usavršavanje budućih simulatora.

3.1. VRSTE SIMULATORA I NJIHOVE ZNAČAJKE

Većina programskih paketa dijele osnovne karakteristike za manipuliranje i simuliranje okoline i robota. Osnovne značajke koje simulator treba posjedovati su 3D simulacija okoliša, prilagodba atributa poput tekstura, mase, trenja i oblika, simuliranje širokog spektra senzora, alatne trake prilagođene korisniku i grafička sučelja koja služe za interakciju s objektima, unos CAD modela, korištenje različitih programskih jezika za pisanje kontrolnih programa.

Delmia V5 je komercijalni programski paket *Dassault Systemes*-a koji je u širokoj upotrebi u automobilskoj industriji, uglavnom za planiranje operacija točkastog zavarivanja. Ono što izdvaja ovaj softver su napredni i efikasni alati za planiranje postrojenja proizvodnje koji sadržavaju u sebi kataloge za pokretne trake, stolove, police, spremnike, modele robota širokog spektra proizvođača, te alate za dizajniranje rasporeda postrojenja i toka proizvodnje [6]. U sve to ulazi također i projektiranje stanica u tvornici, organiziranje prostora i izbjegavanje kolizija [7].

RoboDK je univerzalni simulator za programiranje izvan mreže (eng. *offline programming*). Glavna svrha programiranja izvan mreže je testiranje kontrolnih algoritama kako bi se osiguralo da funkcioniraju prije nego ih se primjeni na prave robote. Ova metoda je posebno korisna ako se roboti moraju često konfigurirati. Programiranje izvan mreže se ne kosi sa samom proizvodnjom i dopušta da se robot programira koristeći virtualnu maketu, samim time omogućuje inicijalno testiranje ideje prije nego se algoritam prenese na stvarni robot. Pritom se reducira vrijeme zastoja i poboljšava učinkovitost proizvodnje [8]. *RoboDK* je napravljen iz potrebe za povoljnim simulatorom za industrijske robote i intuitivnim načinom za programiranje istih. Biblioteka *RoboDK*-a sadrži preko 300 robota od 30 različitih proizvođača. Ističu se iz razloga što sa *RoboDK*-om se može programirati bilo koji robot koristeći vlastiti *RoboDK* API (eng. *Application Programming Interface*) uz *Python* programski jezik [9].

Webots 3D je simulacijska okolina za modeliranje, simuliranje i programiranje mobilnih robota koja podržava *C, C++, Java, Python* i *Matlab* programske jezike za stvaranje kontrolnih programa [5]. *Webots* koristi *Open Dynamics Engine* (ODE) za simuliranje dinamike krutih tijela [5]. *Open Dynamics Engi*ne je besplatna, industrijska biblioteka za simuliranje artikulirane dinamike krutih tijela, dizajnirana da se koristi u interaktivnoj simulaciji koja se odvija u stvarnom vremenu. Pogotovo je kvalitetna za simuliranje pokretnih objekata u promjenjivom okolišu u virtualnoj stvarnosti. Često se koristi zbog svoje robusnosti, brzine i stabilnosti [10].

ODE također koristi i *Gazebo* kojemu omogućava da precizno reproducira dinamičke 3D okoline u kojem bi se robot mogao naći – pridruživanje različitih atributa objektima koji ih čine da se ponašaju realistično kada se manipulira s njima (povlačenje, guranje, bacanje, nošenje). Simulirani roboti se mogu montirati korištenjem raznih oblika između kojih se nalaze zglobovi. Ovi alati pomažu u kreaciji širokog raspona robotskih platformi [5].

Robot Operating System je operativni sustav otvorenog izvora razvijen s ciljem unapređenja znanja u robotici. To je sustav koji posjeduje širok spektar alata za razvitak simulacijskih aplikacija. Velika prednost ROS-a je upravo njegova infrastruktura. ROS je jedan od najvećih i najpopularnijih platformi u edukaciji robotike zbog sposobnosti dijeljenja komponenata aplikacija. Zbog te sposobnosti i činjenice da je modularne prirode je rezultiralo u mnoštvu suradnje unutar robotske zajednice u cilju dizajniranja najprestižnijih robotskih biblioteka za unaprjeđenje simulacije [8].

Benefit programskih paketa otvorenog izvora je fleksibilnost i prilagodba izvornog koda koje rezultira sudjelovanjem korisnika u razvoju samih alata pritom mijenjajući organizacijske odluke proizvoda. Ovakva razmjena znanja i podataka između zajednice korisnika i proizvođača softvera čini prikladnu opciju za edukativne institucije. Drugim riječima, visokoobrazovnim institucijama više odgovara korištenje softvera otvorenog izvora zbog svoje isplativosti, režima suradnje i povratnih informacija [11]. Ovo je posebno važno uzimajući u obzir kompleksnosti današnje/buduće proizvodnje, informacijske tehnologije i zahtjeve znanja u informatici kako bi se manipuliralo sa simulacijskim alatima [4]. Obrazovne institucije su u prijekoj potrebi za efikasnim sustavima na kojima će se specijalizirati buduće generacije ljudi. *Robot Operating System* (ROS) je primjer takve vrste sustava koji iskorištava *C++* i *Python* programski jezik u robotskom programiranju u sklopu sa simulacijskim softverima koji je postao standard u edukaciji ljudi u robotici [12].

3.2. NVIDIA ISAAC SIMULATOR

NVIDIA Isaac Sim je aplikacija (eng. *extension*) za simuliranje robota za *NVIDIA Omniverse* platformu. *NVIDIA Omniverse* je platforma za suradnju i izgradnju različitih aplikacija u 3D dizajnu i virtualnoj stvarnosti [13]. Kao i ostali simulatori *Isaac Sim* posjeduje esencijalne alate za izgradnju virtualnih robotskih okruženja i eksperimenata. *Isaac Sim* podržava navigacijske i manipulacijske aplikacije kroz *ROS/ROS2*, te je u mogućnosti simuliranja senzora kao što su *RGB-D, Lidar i IMU* [14].

3.2.1. KARAKTERISTIKE ISAAC SIMULATORA

Omniverse sadrži komplet alata za razvijanje ekstenzija koji koristi *Python* aplikacijsko programsko sučelje (eng. *Application Programming Interface*), pod nazivom *Omniverse Kit*. API je mehanizam koji omogućava da dvije komponente nekog softvera komuniciraju jedna s drugom koristeći set protokola i definicija [15]. Pomoću tog kompleta alata je izrađen *Isaac* simulator i svi njegovi mehanizmi za izradu virtualne okoline i manipulacije iste. *Omniverse Kit* je dan korisnicima na raspolaganje u službi korištenja za razvijanje novih aplikacija i ekstenzija. U najjednostavnijem obliku ekstenzija unutar *Omniverse* konteksta je obična datoteka s konfiguracijskom datotekom (eng. *configuration file*) u kojoj su definirani svi aspekti te iste ekstenzije odnosno njezin oblik i operacije. *Omniverse*-ova baza podataka i mehanizam za suradnju (eng. *collaboration engine*) se zove *Nucleus* s pomoću koje *Isaac Sim* ima pristup sadržaju kao što su datoteke za izradu okoliša i robota. *Omniverse Nucleus* dopušta raznolikim aplikacijama da dijele i modificiraju razne reprezentacije virtualnih svjetova unutar *Isaac Sim*-a. Od digitalnih medija *Nucleus* sadrži geometrije, osvjetljenje, materijal, teksture i ostale podatke koji čine virtualnu stvarnost i njezinu evoluciju kroz vrijeme. Sve modifikacije medija se prenose u stvarnom vremenu između spojenih aplikacija. Najčešći format digitalnog sadržaja za prikaz virtualnih scena je USD (eng. *Universal Scene Description*). USD je format otvorenog-izvora za opis 3D scena razvijen od strane *Pixar*-a za stvaranje sadržaja. USD se široko usvaja radi svoje snage i svestranosti, ne samo u zajednici vizualnih efekata, nego i u arhitekturi, dizajnu, robotici, proizvodnji, i ostalim disciplinama [14].

3.2.2. SPECIFIKACIJE ISAAC SIMULATORA

Glavne prednosti *Isaac Sim*-a je visoka razina realnosti simulacije. *Isaac Sim* koristi sposobnost *Omniverse* platforme da koristi tehnologiju snažnih grafičkih kartica koje imaju kapacitet foto realizma s praćenjem svjetlosnih zraka (eng. *ray tracing*) u stvarnom vremenu. Simulacija fizike je popraćena *Physx 5* fizičkim pokretačem (eng. *physics engine*) čiju funkciju omogućava radna memorija hardvera [16]. Prednost *Omniverse* platforme je ujedno i njezin veliki nedostatak. Kako bih *Isaac Sim* mogao stabilno raditi, instalacija *Omniverse* platforme zahtjeva hardver koji je vrlo skup. Minimalne specifikacije koje bi računalo trebalo imati su: *Intel Core i7* (7. generacija) s 4 jezgre, 32 GB RAM memorije, 50 GB SSD memorije, *Nvidia GeForce*

RTX 2070 grafičku karticu, 8 GB VRAM memorije s napomenom da se ne može garantirati stabilni rad s minimalnim zahtjevima. Idealni slučaj bi zahtijevao: *Intel Core i9* (X serija ili više) sa 16 jezgri, 64 GB RAM memorije, 1 TB *NVMe SSD, Nvidia RTX A6000* grafička kartica i 48 GB VRAM memorije. *Omniverse* aplikacije najbolje rade na *Linux* operativnim sustavima dok je moguća instalacija i na *Windows 10*, ali sa napomenom ograničenog rada [14]. Računala s minimalnim specifikacijama u prosjeku koštaju oko 10 000 kn, dok računalo s idealnim slučajem košta od 50 000 kn pa na dalje [17]. Odabir kvalitete virtualne stvarnosti nauštrb pristupačnosti je odabrani dizajn *Isaac Sim*-a. Zbog sve veće kompleksnosti proizvodnje ovakav pristup je pozitivan korak u smjeru usavršavanja simulacije stvarnosti.

Još jedan od benefita *Isaac Sim*-a je njegova modularna arhitektura. Naime, *Isaac Sim* aplikacija (i ostale aplikacije unutar *Omniverse* paketa) nije dizajnirana kao tradicionalne aplikacije gdje različiti dijelovi potječu iz istog osnovnog koda, već je sklopljena od skupa ekstenzija i alata. Svaki od ekstenzija i alata posjeduje svoje vlastite izvorne kodove, te su njihovi dijelovi potpuno modularni i stoje na raspolaganju korisnicima da modificiraju i nadograđuju iste. *Isaac Sim* čini već sklopljen niz različitih alata koji imaju svoj individualni identitet kako bi zajedno tvorili sve potrebne instrumente za simuliranje i manipuliranje robotskih scenarija [18]. Ovakva arhitektura vidljiva na slici 1 se postiže sudjelovanjem zasebnih slojeva koji surađuju jedni s drugim. Najniži sloj koristi snagu hardvera, koja se upotrebljava kroz grafičke, procesne i AI tehnologije kako bi se pomoću alata *Omniverse* platforme, kao što su *Nucleus* i *Kit,* dizajnirale ekstenzije i aplikacije. Naposljetku kulminacija ovog procesa kao i zbirka raznih ekstenzija čini službene *Omniverse* aplikacije.

Slika 1 Arhitektura Omniverse aplikacija, [18]

Upravo ta vrsta dizajna je ono što izdvaja *Isaac Sim* od ostalih suvremenih simulatora za robotiku – potpuna sloboda u oblikovanju vlastitih aplikacija.

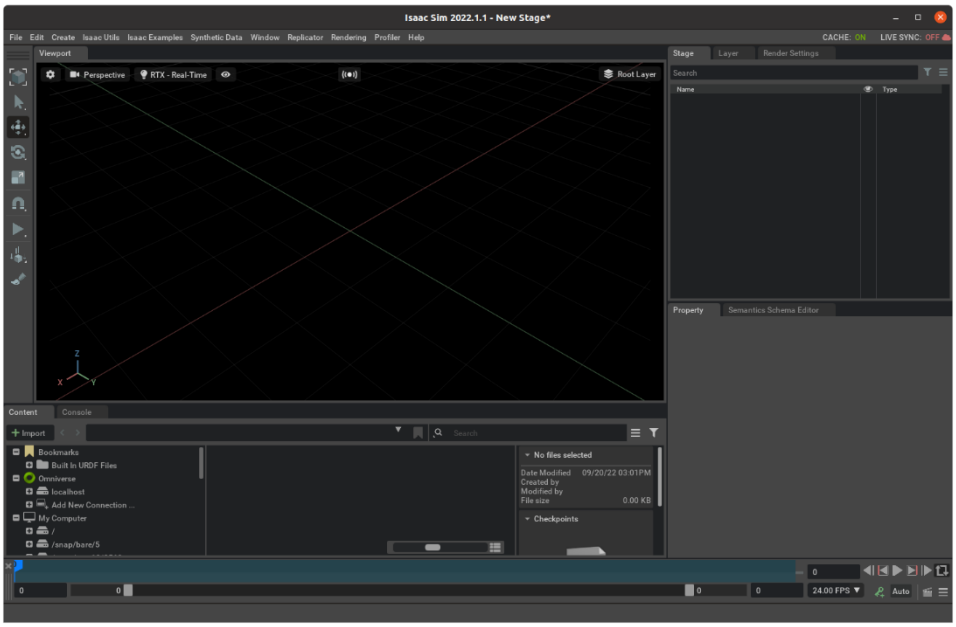
Ideja iza *Isaac Sim*-a je da se pokrije što više robotskih slučajeva upotrebe uključujući manipulaciju, navigaciju, generaciju sintetičkih podataka za treniranje podataka (eng. *training data*) i da omogućava jednostavnu povezanost, interoperabilnost, dijeljenje i kolaborativno građenje pomoću *Nucleusa* i ostalih konektora [16].

Dokumentacija *Isaac Sim*-a je dostupna na službenim stranicama zajedno s potrebnim uputama za instalaciju i korištenje osnovnih alata. Daljnja izobrazba se tereti na individualnom radu i eksperimentiranju, a komunikacija s programerima se izvršava preko službenog foruma gdje se raspravljaju zasebne teme i dokumentiraju razni problemi.

3.2.3. OSNOVNO GRAFIČKO SUČELJE

Osnovno grafičko sučelje korisnika (eng. *Graphical user interface*) *Isaac Sim*-a prikazuje elemente koji prenose informacije i predstavlja operacije nad kojima korisnik može obavljat

radnju. Ono se sastoji od sljedećih ekstenzija: prozora za prikaz (eng. *viewport*), sadržaja (eng. *content*), konzole (eng. *console*), pozornice (eng. *stage*), svojstva (eng. *property*), sloja (eng. *layer*), postavke prikaza (eng. *render settings*), alata pozicioniranja i glavnog izbornika (slika 2).



Slika 2 Prikaz osnovnih ekstenzija Isaac Sim-a Izvor: Autor

Viewport je ekstenzija u kojoj se prikazuje grafika simulacije. Svi uneseni objekti (eng. *prims*), njihovi 3D modeli i pozicije u prostoru se predočavaju u prozoru za prikaz. Ova ekstenzija je ključna za stvaranje virtualne stvarnosti jer pomoću nje korisnik može manipulirati s objektima i entitetima (označavanje, pomicanje, skaliranje, rotiranje) i manevrirati kroz virtualnu okolinu.

Content je ekstenzija koja dopušta korisniku da pristupa i pretražuje po lokalnim datotekama i datotekama na oblaku (*Nucleus*).

Console je ekstenzija koja objavljuje izlazni zapis podataka u stvarnom vremenu. Također dopušta korisniku da unosi direktno naredbe.

Stage je ekstenzija koja omogućuje pregled svih objekata unutar prikazane USD scene. Objekti su sortirani po hijerarhijskom redoslijedu. Kategorizirani su po tipu i imenu, te se mogu jednostavno pretraživati pomoću različitih filtera ili direktnim unosom naziva.

Property ekstenzija omogućuje konfiguriranje svojstva željenih objekata. Neki od osnovnih svojstava su: *Transform* – pozicioniranje i skaliranje objekata, *Materials* – primjena raznih materijala na objekt, *Visual* – modificiranje vidljivosti objekata. Ovom se ekstenzijom također

mogu dodati zasebna svojstva poput prilagođenih atributa, fizičkih parametara i dodatnih željenih prikaza.

Layers obavlja funkciju organiziranja i upravljanja simulacijske scene. Objekti su posloženi na način da onaj pri vrhu ima najveći prioritet. S ovim alatom je moguće efikasno mijenjanje različitih konfiguracija scene bez uništavanja i brisanja objekata.

Render Settings ekstenzija omogućava prilagodbu parametara prikaza za optimizaciju kvalitete, izvođenja i korištenje memorije.

Prethodne ekstenzije kao i ostatak postojećih ekstenzija se odabiru preko izbornika *Exstensions*. U izborniku su navedene sve informacije za pojedinu ekstenziju, te se preko njega može pristupiti izvornom kodu odabrane ekstenzije [19].

3.2.4. SIMULACIJA FIZIKALNIH ZAKONA

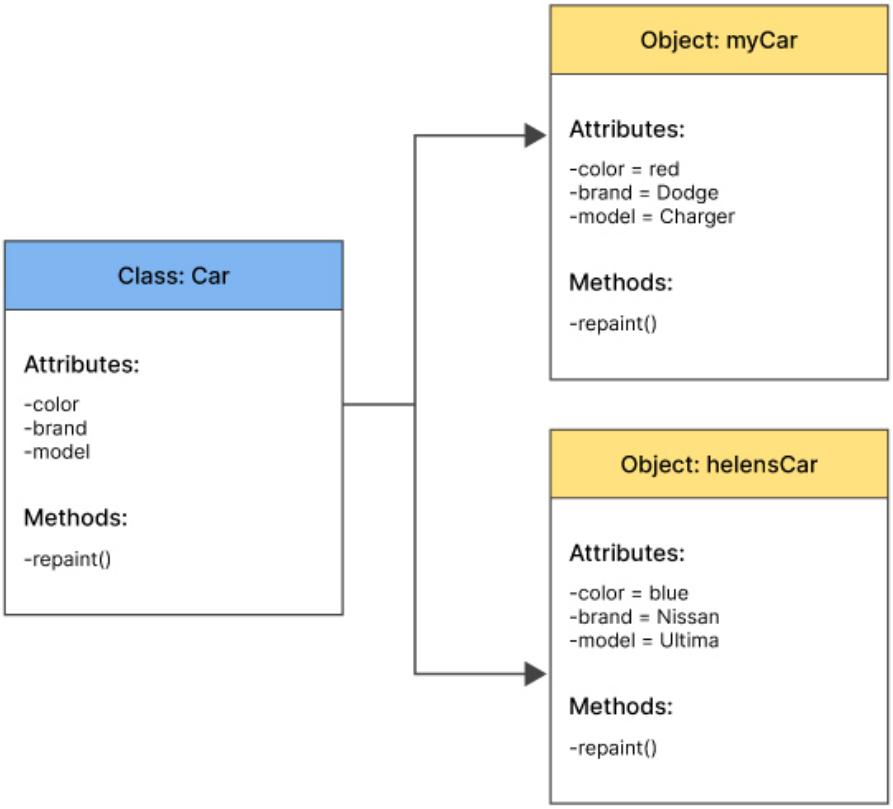
Glavni pokretač koji stoji iza Omniverse simulacije fizičkog okruženja je *Physx*. *Physx* je fizički pokretač na kojem se počelo raditi još 2011. godine od strane *NVIDIA*-e. Kroz godine, *Physx* je prošao kroz više iteracija, gdje je u početku bio namijenjen kao fizički pokretač za igre dok je u zadnje vrijeme prenamijenjen za precizno pokretanje fizičkih sustava u robotici. Najnovija verzija *Physx* pokretača podržava i koristi dinamiku krutih tijela, dinamiku mekih tijela, implementaciju zglobova, dinamiku vozila, kinematske kontrolore likova, simulacija oblika i geometrija, simulaciju realističnih materijala, simulaciju čestica, simulaciju fluida, simulaciju tkanine itd. [20]. *Physx* zajedno s USD tehnologijom uspijeva praktično realizirati virtualnu stvarnost unutar *Isaac Sim*-a. Parametri 3D grafike poput prikaza geometrija i materijala modela, animacije, virtualne kamere i rasporeda scene se definiraju preko USD API.

3.2.5. PYTHON BIBLIOTEKE I OBJEKTNO ORIJENTIRANO PROGRAMIRANJE

Pošto je svaki aspekt *Omniverse* platforme i *Isaac Sim*-a definiran u *Python* programskom jeziku, za očekivati je da su jednako potrebna znanja u programiranju i implementaciji *Python* biblioteka. Potrebno je spomenuti da *Omniverse* API podržava i *C++* programski jezik, no fokus je usmjeren prema *Python*-u, te se aplikacije grade u skladu s tim. Dokumentacija za *Isaac Sim* i *Omniverse Python* Biblioteke je dana na službenim stranicama *Omniverse*-a.

Omniverse API dokumentacija je usredotočena na razvijanje ekstenzija uz pomoć *Omniverse Kit*-a, dok je *Isaac Sim* API dokumentacija orijentirana na razvijanje robotskih aplikacija.

Isaac Sim koristi objektno orijentirano programiranje (eng. *object-oriented programming*) za izgradnju svojih aplikacija. Objektno orijentirano programiranje je jedan od mogućih načina pisanja koda koji se bazira na stvaranju podatkovnih struktura. Strukture se grade s komadima koda koji se još nazivaju klasama, pomoću kojih se definiranju objekti koji nose instance atributa i metoda koji obavljaju nekakvu funkciju. Primjerice, ako klasa predstavlja automobil sa zadanim atributima (boja, marka, model) i metodama (prebojiti, upaliti motor) onda se specificira koje atribute i metode će nositi njezini objekti. Kada se stvaraju objekti koristeći ovu klasu oni će posjedovati navedene atribute i metode, ali se mogu i razlikovati po njima (slika 3) [21].



Slika 3 Povezanost klase i objekta, [21]

Drugim riječima, klase su apstraktni nacrti koji služe za izgradnju specifičnih i konkretnih algoritama. Klase također imaju svojstvo nasljeđivanja obilježja drugih klasa. Ako postoje više klasa koje dijele atribute i metode nasljeđivanjem se može skratiti vrijeme pisanja koda. S umrežavanjem klasa se stvara podatkovna struktura s kojom se mogu obavljati kompleksni

zadaci. Ovakav pristup programiranja građenja blokova ili komada koda je posebno povoljan jer se klase mogu efikasno reciklirati i uvoziti [22].

Glavne biblioteke *Isaac Sim*-a se sastoje od svih klasa i naredbi pomoću kojih su izgrađene postojeće ekstenzije. One u sebi sadržavaju potrebne blokove programskog znanja za definiranje artikulacija robota, implementiranje robotskih kontrolera, definiranje konteksta scene (materijali, objekti, fizički parametri, itd.), definiranje kretnji robota, definiranje parametara virtualnog svijeta, dohvata podataka o virtualnom svijetu, implementiranje robotskih zadataka. Dokumentacija sadrži i detaljnije klase za specifične industrijske robote poput UR (*Universal Robots*), *Franka, DofBots* i mobilnih robota.

Navedene *Python* biblioteke su esencijalne za korištenje, učenje i razvijanje robotskih aplikacija, one služe kao osnovni alat za izgradnju virtualnih scenarija, te su namijenjene za nadogradnju i moduliranje istih. *Isaac Sim* paket podržava *Visual Studio Code* koji pruža okolinu za *Python* programiranje i otklanjanje nepravilnosti (eng. *debugging*) [14]. Također je uređivač izvornog koda (eng. *code editor*) koji se koristi za razvojne operacije poput otklanjanje nepravilnosti, isticanja sintakse, inteligentno dovršavanje koda, izvođenje zadataka (eng. *task running*) i pisanja koda [23].

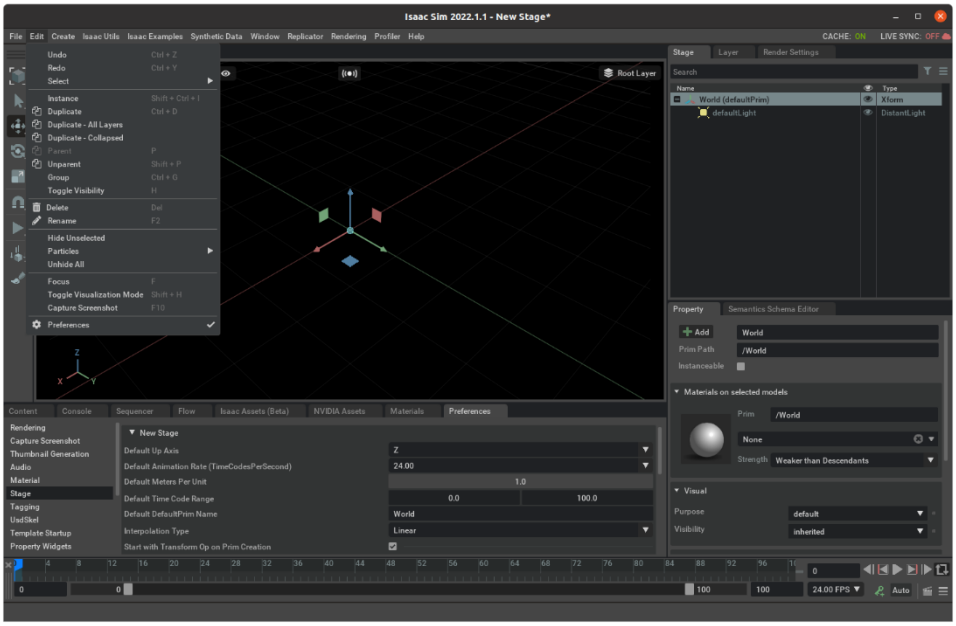
4. IZRADA VIRTUALNOG OKRUŽENJA

U praktičnom dijelu rada opisale su se funkcije *Isaac Sim*-a. Demonstrirali su se načini strukturiranja virtualnog okruženja i dizajniranje zasebnih ekstenzija. Unutar praktičnog dijela detaljno će se okarakterizirati važni alati koji dolaze sa *Isaac Sim* softverom, kako ih koristiti i čemu služe. Prokomentirati će se osnovni pristup pri programiranju simulacije manipulacije, te glavne principe korištenja *Isaac Sim* programske dokumentacije.

Postavljanje virtualne scene unutar *Isaac Sim*-a je temelj za izgradnju precizne simulacije. Sofisticirana grafika koju omogućuje *Physx 5* stvara foto-realistično okruženje po kojem je vrlo jednostavno manevrirati. Za izradu okoline *Isaac Sim* pruža gotove alate – ekstenzije koje omogućuju da korisnik intuitivno stvara osobne scenarije. *Isaac Sim* paket ostvaruje da korisnik može istodobno kontrolirati parametre okruženja i dobivati vizualne informacije u stvarnom vremenu. Objasnit će se osnovni koraci za izgradnju svojstvene virtualne scene, te na što treba obratiti pozornost i kako manipulirati s entitetima unutar virtualne scene.

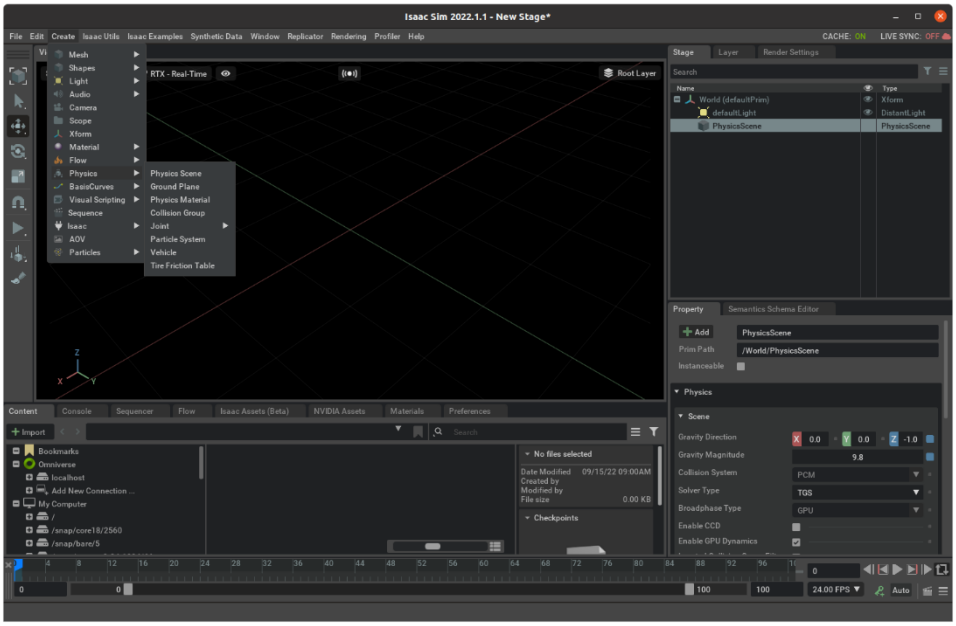
4.1. DEFINIRANJE SCENE

Isaac Sim u sebi sadržava standardne postavke mjerenja sustava. Prije nego što se počne sa postavljanjem elemenata u virtualno okruženje potrebno je provjeriti i po nuždi postaviti standarde mjerenja sustava. Otvaranjem ekstenzije *Preferences* dobije se izbornik u kojem su navedene standardne postavke za skupinu ekstenzija koje su nam bitne (slika 4). U kreiranju svoje scene korisnik se odnosi na postavljenu mjernu jedinicu duljine. Unutar *Isaac Sim*-a standard je metar po jedinici, no korisnik može mijenjati mjerni sustav po svojoj želji. Sljedeća postavka na koju je potrebno obratit pozornost je postavljanje pozitivnog smjera osi Z. Naime, ekstenzija *Stage* u sebi ima definiran koordinatni sustav po kojem se korisnik može referirati u prostoru. Pod opcijom *Default Up Axis* postavlja se od koje osi će pozitivni smjer gledati prema gore. Zadana vrijednost je pozitivni smjer osi Z što je u većini slučajeva najintuitivniji odabir. Ista stavka se može mijenjati od strane korisnika po njegovoj želji.



Slika 4 Prikaz ekstenzije Preferences Izvor: Autor

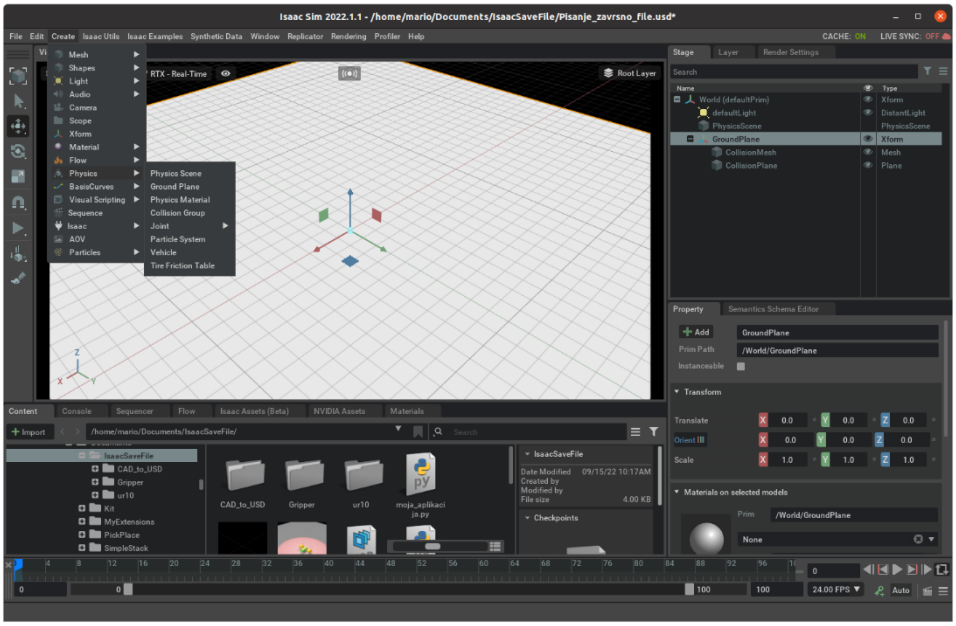
Unutar ekstenzije *Stage* se pojavljuju svi objekti (eng. *Prims)* koji se dodaju u scenu. Pomoću ekstenzije *Stage* se organiziraju i označuju željeni objekti dok se preko ekstenzije *Viewport* dobiva vizualna informacija o označenom objektu. Objekti unutar *Stage* ekstenzije se mogu jednostavno preimenovati. Prilikom izrade nove scene unutar *Stage* javlja se objekt tipa *XForm*. *XForm* tip je najjednostavniji oblik objekta, te služi kao prazna datoteka za lociranje ostalih objekata. Kao takav je vrlo važan za organizaciju scene. Kako bi se definirao fizički sustav koji djeluje na cijeli *XForm* potrebno je unijeti objekt zadužen za manipuliranje fizičkih parametara pod nazivom *Physics Scene* (slika 5).



Slika 5 Unos Physics Scene objekta Izvor: Autor

Svaki od dodanih objekata ima svoju vlastitu *Property* karticu. *Property* je zasebna ekstenzija koja daje detaljan opis svih promjenjivih parametara svakog objekta. U ovom slučaju *Physics Scene* u svojoj *Property* kartici sadrži svoje fizičke parametre s kojima korisnik može manipulirati. Za početak je potrebno definirati gravitaciju, njezin smjer i akceleraciju. Pošto je mjerni sustav postavljen u metrima po jedinici, akceleracija gravitacije će u ovom slučaju biti 9,8 m/s^2, a smjer u kojem djeluje sila gravitacije će biti u negativnom smjeru osi Z određeno jediničnim vektorom $[\vec{\imath}, \vec{\jmath}, -\vec{k}]$

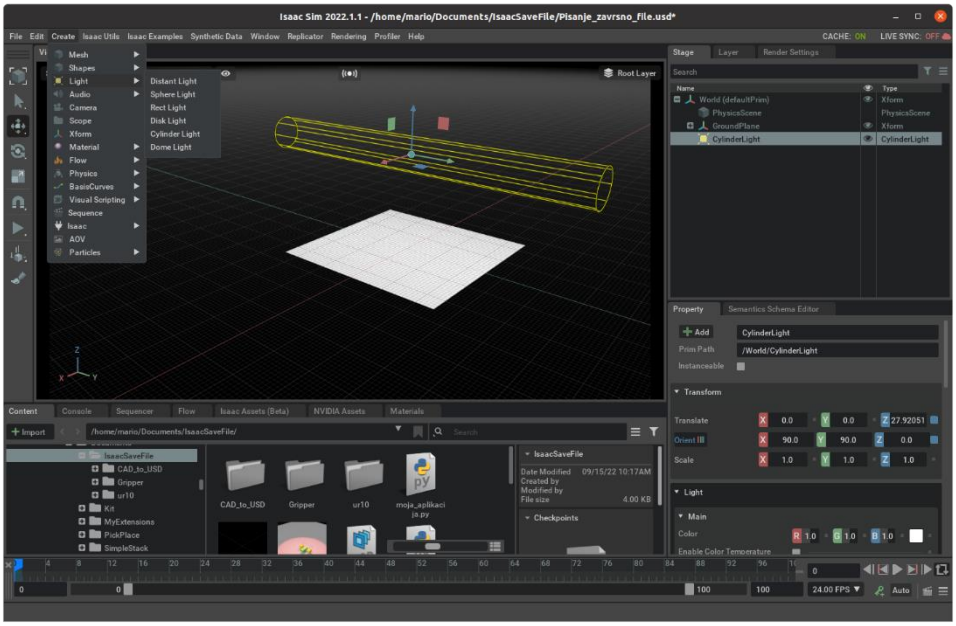
Kako bi se mogli postavljati razni 3D modeli u scenu potrebno je definirati podlogu ili podložnu ravninu (eng. *Ground plane* (slika 6). Podložna ravnina se standardno smješta u x-y ravninu koordinatnog sustava. Ona ograničava prostor virtualnog okruženja na takav način da sprječava slobodan pad objekata pod utjecajem gravitacije. *Ground plane* je statičan element i on služi kao virtualni temelj simulacije.



Slika 6 Unos podložne ravnine Izvor: Autor

Ground plane je *XForm* tipa koji se sastoji od *Mesh* i *Plane* objekta. *Mesh* objekt je struktura modela sastavljen od mreže poligona koja definira oblik objekta, dok *Plane* objekt postavlja poziciju kolizije podloge. *Plane* i *Mesh* koordinatni sustavi moraju biti u jednakoj poziciji kako bi se kolizija pravilno preslikavala na model podloge. Kolizijska ravnina je produžena u beskonačnost scene, no sam *Mesh* je ograničen radi smanjenja troška procesne snage. *Mesh* se može proširiti i suziti po želji korisnika.

Stvaranjem nove scene unutar *Isaac Sim*-a u *Stage* ekstenziji se automatski dodaje standardno osvjetljenje kako bi scena bila vidljiva od začetka. Međutim *Isaac Sim* podržava razne verzije osvjetljenja koje se mogu implementirati poput sferno osvjetljenja, cilindričnog osvjetljenja i slično (slika 7). Ovisno o vrsti osvjetljenja, objekt se postavlja unutar *Stage* prozora, te se u *Viewportu* prikazuje njegov oblik i utjecaj djelovanja.



Slika 7 Unos osvjetljenja Izvor: Autor

Svi parametri poput boje, pozicije, veličine, temperature svijetla, intenziteta, ekspozicije i ostalih vrijednosti se prilagođavaju unutar *Property* kartice.

Naime, kako *Property* ekstenzija služi za izmjenu parametara pojedinog objekta, kontrole pozicioniranja i skaliranja objekta se jednostavnije izvodi pomoću alatne trake namijenjene za translaciju, rotaciju, skaliranje, simuliranje i označavanje objekata. Svakoj od operacija je dodijeljen pripadajući prečac na tipkovnici kojim se vrlo jednostavno i intuitivno može manipulirati. Same operacije za pozicioniranje se također izvode pomoću *Viewport* ekstenzije. Pošto je svakom objektu dodijeljen i koordinatni sustav, taj isti koordinatni sustav se može manipulirat s kombinacijom alatne trake za pozicioniranje. Osi koordinatnih sustava su interaktivne te mogu biti upravljane pomoću miša i tipkovnice brzo i efikasno.

4.2. DODAVANJE 3D OBJEKATA I DEFINIRANJE FIZIKALNIH PARAMETARA

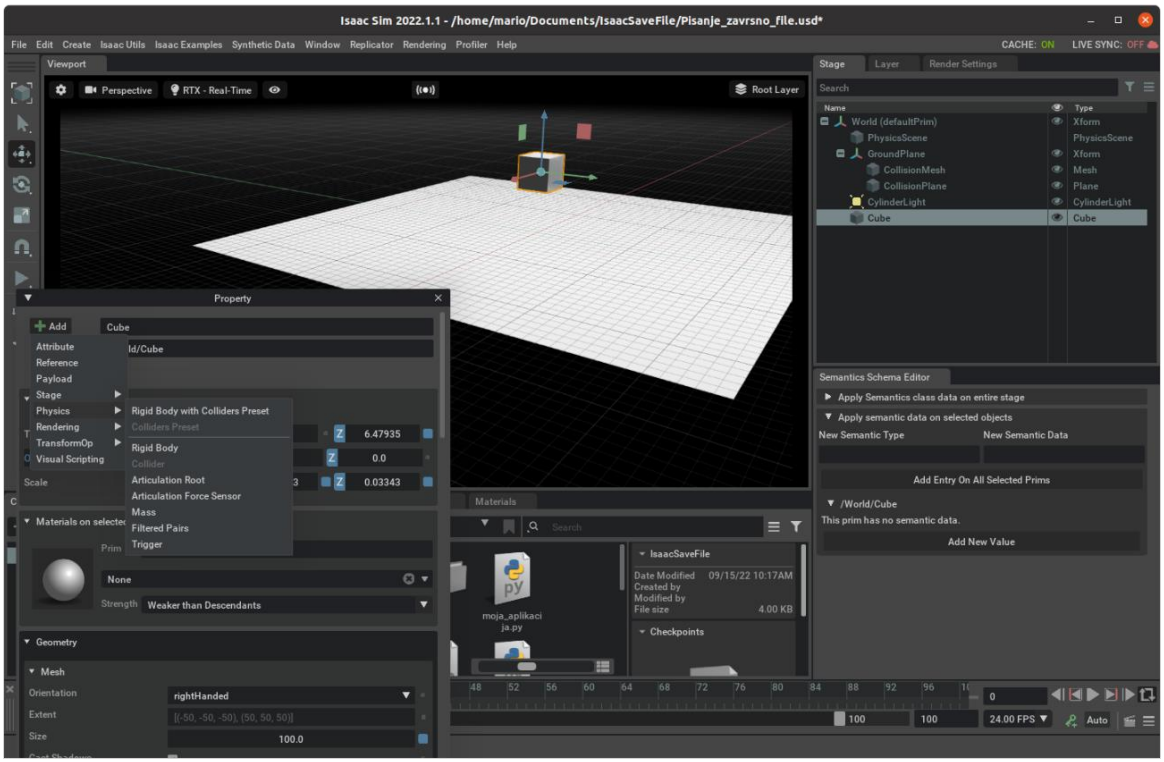
Za početak *Isaac Sim* ima za opciju pristupa primitivnim geometrijskim oblicima. Svaki od primitivnih oblika ima svoj zaseban *Mesh* koji definira izgled 3D modela. Osnovni oblici su: kocka, sfera, cilindar, kapsula i stožac (slika 8). To su savršeno glatka geometrijska tijela na koja se mogu primijeniti različiti parametri, materijali i teksture. Oni služe radi lakšeg

snalaženja s alatima *Isaac Sim*-a, ali isto tako ih se može koristit i u oblikovanju svojstvenih i kompliciranijih scenarija. U ovom primjeru će se koristiti u cilju predstavljanja funkcija *Isaac Sim* alata.

Slika 8 Unos primitivnih oblika Izvor: Autor

Pomoću *Property* kartice ili koordinatnog sustava, skaliranjem se mogu mijenjati dimenzije oblika. S *Property* karticom se mogu precizno upisati mjere koje se žele postići dok s alatnom trakom i koordinatnim sustavom se to postiže aproksimativno.

Takvi 3D objekti koji su jedino opisani s *Mesh* strukturom su za početak samo vizualni zapis unutar *Viewport* ekstenzije. Kako bi se primijenili fizički parametri sustava na pojedine 3D objekte, korisnik mora ručno primijeniti fiziku definiranom sa *PhysicsScene* na pojedini 3D objekt (slika 9). Nekad je pogodno imati modele koji su neovisni o samoj fizici sustava radi pozicioniranja i upravljanja istim bez smetnji, što je razlog zašto se fizika automatski ne primjenjuje na 3D objekte. Ručnom primjenom fizike na objekt automatski se postavlja fizika krutih tijela i fizika kolizija na *Mesh* koji definira taj 3D objekt. U slučaju pokretanja simulacije, gravitacija definirana *Physics Scene* objektom će se primijeniti na kocku i kocka će početi padati dok se ne sudari s podložnom ravninom.



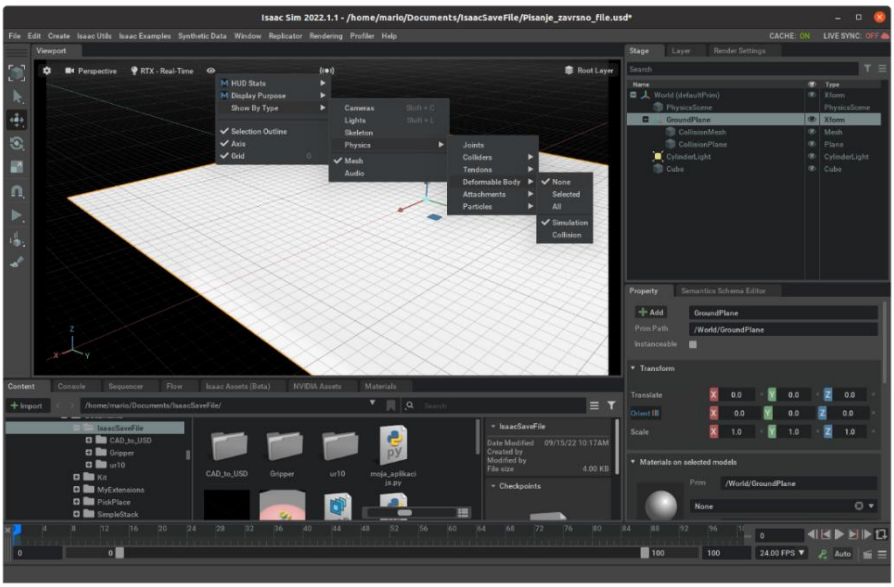
Slika 9 Primjena fizike na objekt Izvor: Autor

Nakon što se primjenila fizika na objekt unutar *Property* kartice se pojavljuju *Rigid Body* i *Collider* ekstenzije (slika 10) koje služe za detaljnije rukovanje parametrima fizike krutih tijela i kolizija. Svi parametri koji su određeni unutar postavki će početi djelovati tek kad se pokrene simulacija virtualnog okruženja.

Slika 10 Prikaz Property ekstenzije Izvor: Autor

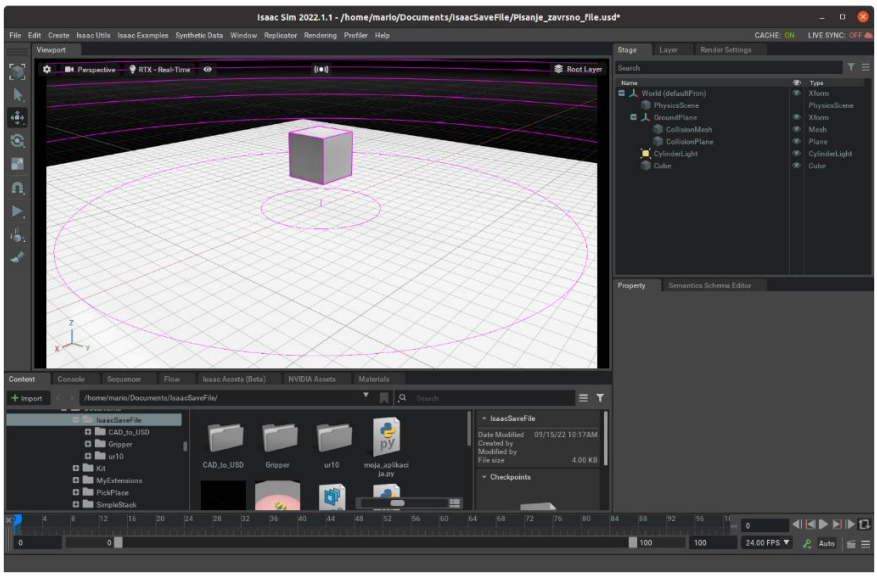
Simulacija se pokreće s klikom na ikonu za simuliranje na alatnoj traci za pozicioniranje. Pokretanjem simulacije sve radnje unutar virtualnog okruženja se odvijaju u vremenu. Tako i sama gravitacija kao i ostala fizika ovisna o vremenu može tek utjecati na objekte kad se ima vrijeme kao varijablu. Simulacija se može zaustaviti u bilo kojem vremenu, potom svi objekti dolaze u početnu poziciju onako kako su definirani u *Property* ekstenziji.

Kako bi se moglo detaljnije analizirati primijenjena fizika, *Viewport* ekstenzija omogućava opciju vizualnog prikaza implementacije fizike krutih tijela i kolizije (slika 11).



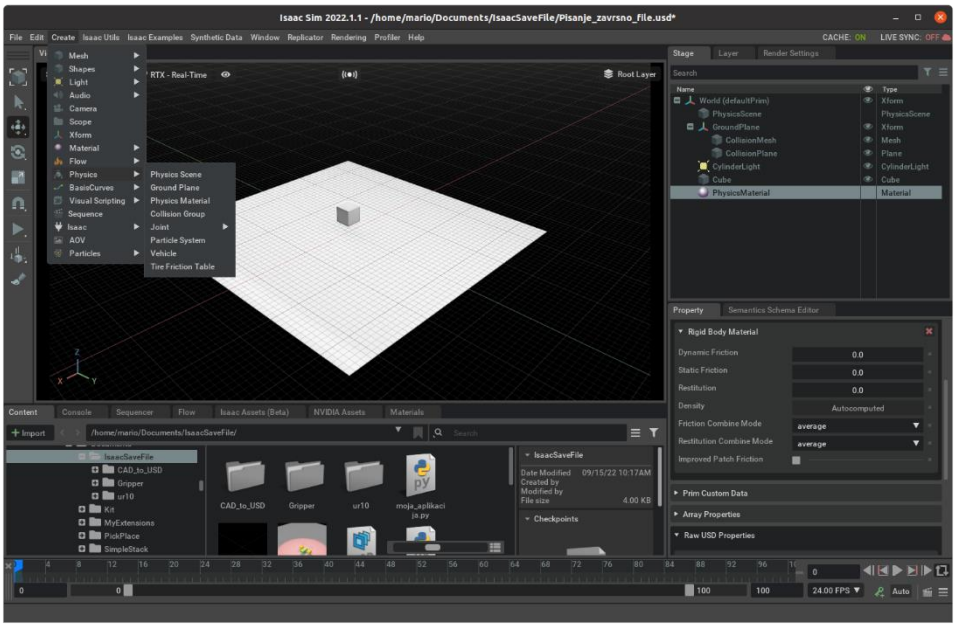
Slika 11 Destinacija vizualnog prikaza fizikalnih parametara Izvor: Autor

Ovim načinom se istodobno dobiva vizualna informacija o tome gdje djeluje fizika virtualnog okruženja. (slika 12).



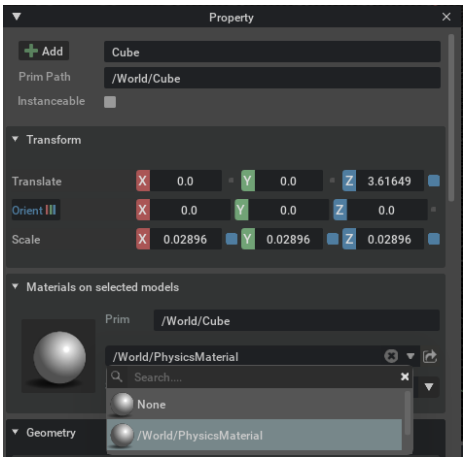
Slika 12 Vizualni prikaz fizikalnih parametara Izvor: Autor

Dodavanje fizičkih parametara materijala, poput koeficijenta trenja i restitucije, se postiže na način da se doda novi objekt u *Stage* koji služi kao šablona u kojoj je opisana fizika materijala. (slika 13). Ona je neovisan element unutar *Stage* ekstenzije i utječe samo na objekte na koje se primjeni.



Slika 13 Dodavanje šablone fizičkih parametara materijala Izvor: Autor

Kako bi se fizika materijala unutar šablone primijenila, u *Property* kartici pojedinog 3D objekta se odabere željena šablona sa svojstvenim parametrima te se dana fizika materijala aktivira na odabran objekt (slika 14).



Slika 14 Prikaz primjene šablone Izvor: Autor

Na isti se način može primijeniti i boja na objekt pomoću šablone. Kreirana scena se sprema kao USD datoteka.

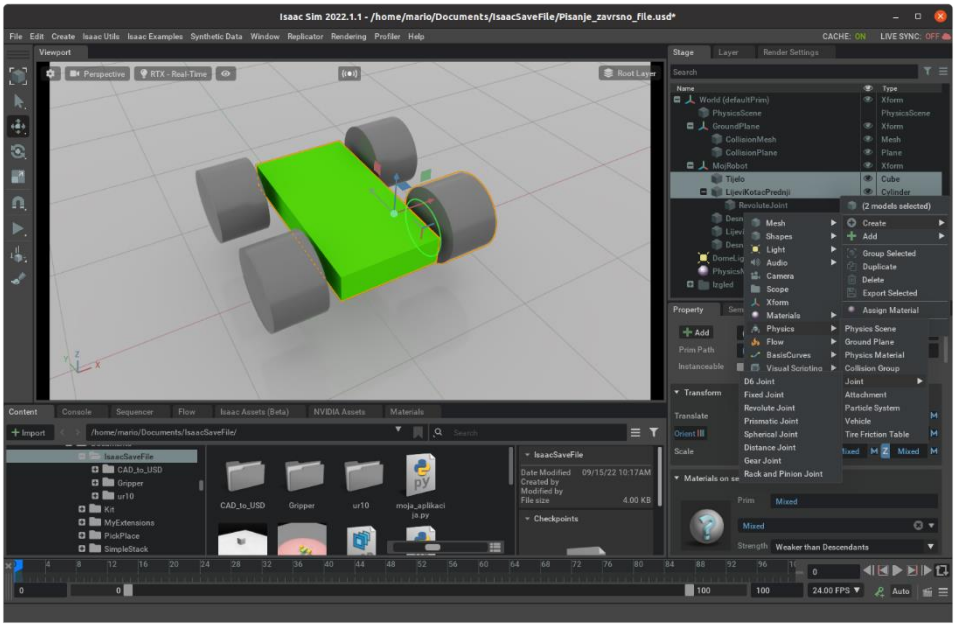
4.3. SKLAPANJE ROBOTA I DEFINIRANJE ARTIKULACIJE ZGLOBOVA

Isaac Sim sadržava alate za sastavljanje vlastitog robota. Glavni aspekt sastavljanja robota je stvaranje artikulacije zglobova (eng. *joint*) između elemenata. S *Isaac Sim* paketom dolazi i ekstenzija za uvoz (eng. *importer*) *URDF*, *STEP* i *OnShape* modela. *URDF*, *STEP* i *OnShape* su različiti formati 3D modela koji sa sobom nose različite karakteristike. *OnShape* modeli proizlaze iz *OnShape* softvera koji je kombinacija PDM i CAD struktura, dok je *STEP* format standardiziran ISO normama, te ga podržava većina CAD paketa [24].

URDF format modela se razlikuje od *STEP* i *OnShape* na način da je artikulacija zglobova između elemenata već definirana, te je potrebno samo uvesti model preko ekstenzije za uvoz, dok se za preostala dva formata moraju definirati zglobovi i njihova hijerarhija.

Artikulacija zglobova robota se definira na sljedeći način. U primjeru je kombiniran primitivan sklop tijela s četiri kotača kojima je već pridodijeljena fizika krutih tijela i kolizija. Prilikom izgradnje mobilnog robota unutar *Isaac Sim*-a potrebno je definirati revolutne zglobove između kotača i tijela. One čine spoj takav da se kotač slobodno može vrtjeti oko svoje osi dok je u aksijalnom i radijalnom smjeru čvrsto spojen za tijelo. Označavanjem elemenata kojima se želi dodijeliti zglob potrebno je paziti na hijerarhiju tijela, na način da se označavaju prvo primarni

elementi , a zatim elementi koji se spajaju na primaran element. U ovom primjeru roditelj (eng. *parent*) element čini tijelo dok dijete (eng. *Child*) element čini kotač.

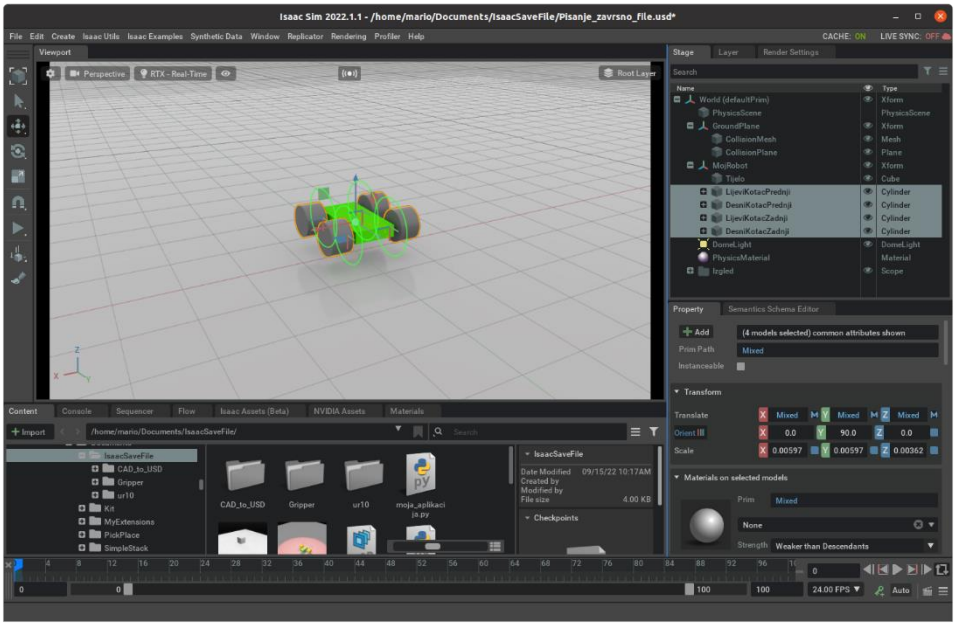


Slika 15 Prikaz implementacije zglobova Izvor: Autor

Na ovaj način *Isaac Sim* podržava više vrsta zglobova kao na slici 15, a to su: fiksni zglob (eng. *Fixed Joint*), revolutni zglob (eng. *Revolute Joint*), prizmatični zglob (eng. *Prismatic Joint*), D6 zglob (eng. *D6 Joint*), sferni zglob (eng. *Spherical Joint*), zupčasti zglob (eng. *Gear Joint*), duljinski zglob (eng. *Distance Joint*) i zglob s letvom i zupčanikom (eng. *Rack and Pinion Joint*).

Nakon dodavanja revolutnog zgloba, unutar *Stage* se može uočiti objekt *PhysicsRevoluteJoint.* Kao i ostali objekti tako i zglobovi imaju svoju vlastitu *Property* ekstenziju. Unutar *Property* ekstenzije se mogu naći parametri koji definiraju funkciju zgloba poput osi rotacije, dimenzijskog ograničenja, brzinskog ograničenja i pozicije zgloba u prostoru.

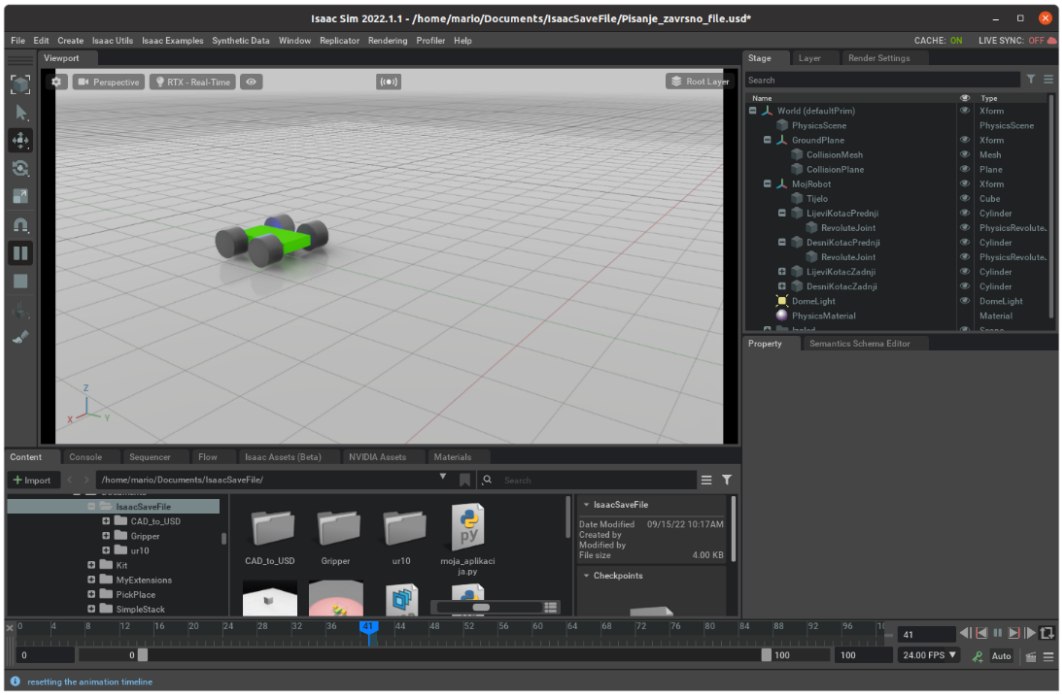
Nakon unošenja zgloba, mora se osigurati njegova pozicija u odnosu na elemente koje povezuje (slika 16).



Slika 16 Pozicioniranje zglobova u odnosu na elemente Izvor: Autor

Dodavanjem zglobova osigurava se kruta povezanost između elemenata. Iako kotači nisu fizički povezani s tijelom, zglobovi nam omogućuju da djeluju kao da jesu. Djelovanjem sile na neki od elemenata robota, dijelovi će se ponašati kao jedan sklop.

Netom poslije dodavanja zglobova, oni djeluju samo kao mehanička povezanost između dva elementa. Kako bi se kontroliralo i pokretalo zglobove, potrebno je implementirati adekvatni pogon. Pogon (eng. *Drive*) omogućuje definiranje pogonskih parametara poput sile, brzine, prigušivanja i krutosti zgloba. Razlikuju se dvije vrsta zglobova: zglobovi kontrolirani položajem i zglobova kontrolirani brzinom. Kod zglobova kontroliranih položajem se definira visoka krutost zgloba dok je prigušenje nisko ili jednako nuli. Kod zglobova kontroliranih brzinom definira se visoko prigušenje dok je krutost zgloba jednaka nuli. U ovom primjeru su zglobovi koji su kontrolirani brzinom pošto se radi o primitivnom mobilnom robotu. Pogon je definiran na prednja dva kotača, te se stavlja visoko prigušenje, krutost zgloba jednaka nuli i srednja brzina pogona. Pomoću ovih parametara se simulira stvarno ponašanje zgloba. Kada se pokrene simulacija scene, mobilni robot pod utjecajem gravitacije prvo pada na podložnu ravninu, zatim pogon na prva dva kotača vuče ostatak sklopa za sobom i pokreće mobilni robot prema naprijed (slika 17).

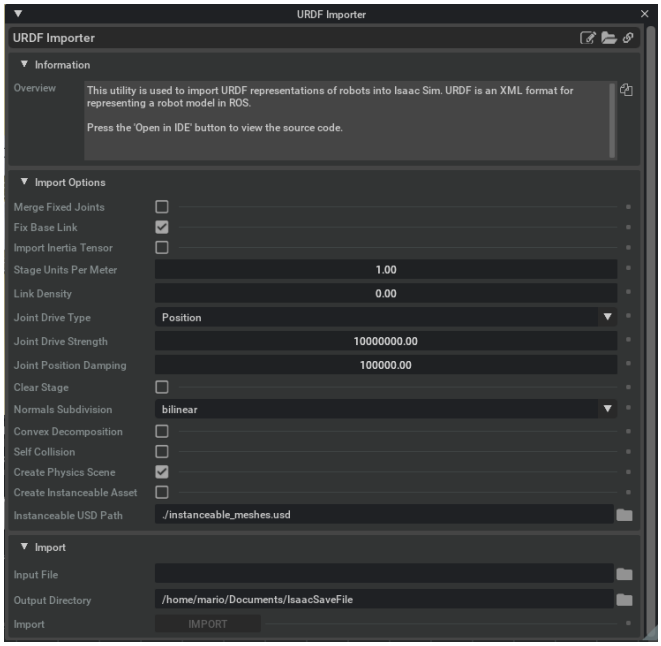


Slika 17 Prikaz pokretanja simulacije i djelovanja pogona zglobova Izvor: Autor

Kod kompleksnijih sklopova, gdje je mnogo povezanih elemenata, te gdje puno zglobova djeluje istodobno za obavljanje različitih radnji, dodavanje zglobova nije dovoljno. Mora se stvoriti pravilna struktura artikulacije zglobova. Artikulacija zglobova omogućava organizirano djelovanje unutar *Physx* API sustava kako bi se uštedilo na procesnoj snazi, povećala kvaliteta vjernosti simulacije i smanjile greške djelovanja zglobova. Pravilno organiziranje artikulacije zglobova se bazira na strukturi stabla (eng. *Tree structure*) slično kao i kod označavanja elemenata uslijed dodavanja zgloba, mora se odrediti roditelj (korijen) hijerarhije povezanih zglobova. Pri određivanju korijena artikulacije potrebno je identificirati vrstu artikulacije. Dvije su vrste artikulacije: artikulacija fiksne baze i artikulacija lebdeće baze. Razlika između fiksne baze i lebdeće baze je da se lebdeća baza može slobodno kretati u prostoru dok fiksna baza ne može. Kod artikulacije fiksne baze korijen artikulacije se stavlja na fiksni zglob koji je povezuje ostatak svijeta sa sklopom ili na njegovog pretka (npr. Radni stol). Kod artikulacije lebdeće baze korijen artikulacije se može postaviti na bilo koji zglob s ciljem urednosti strukture stabla [25]. U trenutnom primjeru korijen artikulacije (eng. *Articulation root*) bi bilo tijelo mobilnog robota.

4.4. DODAVANJE INDUSTRIJSKIH ROBOTA

Pomoću ekstenzija za uvoz omogućen je brz i jednostavan unos modela izrađenim CAD softverima. Kao što je prije spomenuto, *URDF* formati modela su najpovoljniji jer su artikulacije zglobova već definirane, te smanjuje vrijeme izrade virtualnog okruženja.



Slika 18 Ekstenzija za uvoz URDF formata Izvor: Autor

Ekstenzija za uvoz u sebi sadržava opcije uvoza kao što je preliminarno postavljanje podložne ravnine i fizičkog sustava (slika 18). Nakon što se odabere željeni *URDF* dokument, objekt se pojavi u *Stage* ekstenziji i vizualno u *Viewport* ekstenziji. Nakon čega se može s njim manipulirati kao i s ostalim objektima. *Isaac Sim* unutar *Nucleus*-a sadrži URDF formate *Universal Robot 10* (*UR10*), *Franka, Carter i Kaya* robota.

4.5. IZGRADNJA VIRTUALNE SCENE U PYTHON PROGRAMSKOM JEZIKU

Postavljanje virtualnog okruženja je moguće obaviti bez korištenja postojećih *Isaac Sim* ekstenzija. Poznavanjem dokumentacije *Isaac Sim* API vrlo jednostavno se može postaviti virtualno okruženje koristeći objektno orijentirano programiranje.

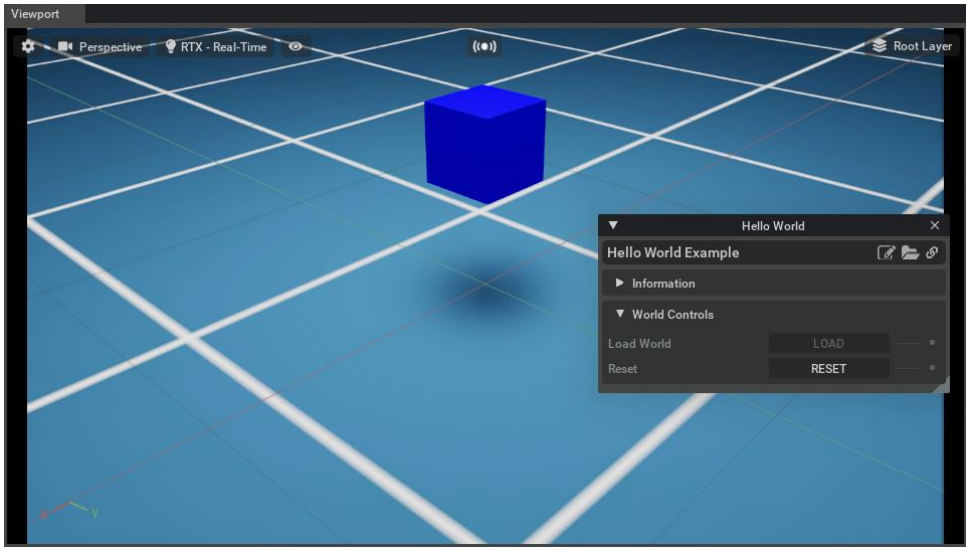
Dizajniranje virtualnog okruženja preko *Python* programskog jezika obavlja se unutar *Visual Basic Code* softvera. Za pravilnu izradu potrebno je imati sučelje i algoritam koji reagira na

interakciju sučelja. U ovom poglavlju će se analizirati algoritmi koji se koriste za kreaciju virtualnog okruženja, a o programiranju sučelja, odnosno ekstenzija će se govoriti u sljedećem poglavlju.

Na dijelu koda je definirana funkcija *setup_scene* koja je zadužena za postavljanje objekata unutar virtualnog okruženja. Unutar funkcije su postavljene naredbe za dohvat podataka o virtualnom svijetu, dodavanja podložne ravnine i dodavanje kocke. Kocka je definirana destinacijom unutar *Stage* ekstenzije, nazivom, pozicijom, orijentacijom i bojom.

```
1.  def setup_scene(self):
2.          world = self.get_world()
3.          world.scene.add_default_ground_plane()
4.          fancy_cube = world.scene.add(
5.              DynamicCuboid(
6.                  prim_path="/World/random_cube", # Destinacije objekta kocke nakon
    dodavanja u Stage
7.                  name="fancy_cube", # Unikatno ime za dohvat podataka o objektu
8.                  position=np.array([0, 0, 0.5]), # Pozicija kocke u prostoru u metrima
9.                  scale=np.array([5, 5, 5]), # Veličina kocke u prostoru u metrima
10.                 color=np.array([0, 0, 1.0]), # RGB boja kocke
11.             ))
12.         return
```

Definirana funkcija se povezuje sa sučeljem, te se pritiskom na odgovarajuću tipku pokreće algoritam. U ovom slučaju, pritiskom na tipku *Load* stvara se podložna ravnina i kocka zadanih atributa (slika 19).



Slika 19 Prikaz rezultata pritiska tipke LOAD

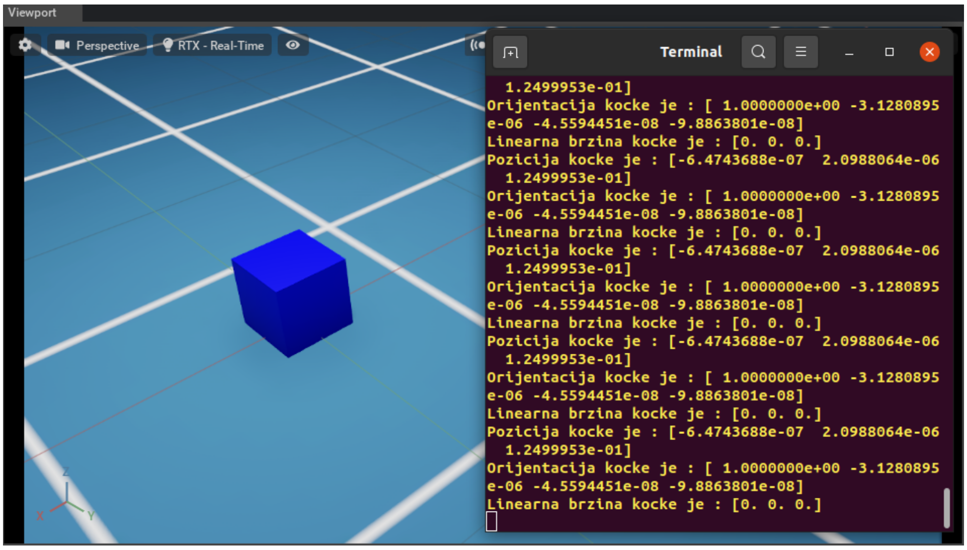
Kako bi se prikazao način na koji *Isaac Sim* dohvaća podatke u virtualnom okruženju u nastavku koda dodaje se definicija za očitavanje podataka o kocki – pozicija, orijentacija, linearna brzina.

```
def setup_scene(self):
        world = self.get_world()
        world.scene.add_default_ground_plane()
        fancy_cube = world.scene.add(
            DynamicCuboid(
                prim_path="/World/random_cube", # Destinacije objekta kocke nakon dodavanja u Stage
                name="fancy_cube", # Unikatno ime za dohvat podataka o objektu
                position=np.array([0, 0, 0.5]), # Pozicija kocke u prostoru u metrima
                scale=np.array([5, 5, 5]), # Veličina kocke u prostoru u metrima
                color=np.array([0, 0, 1.0]), # RGB boja kocke
            ))
        return

    async def setup_post_load(self):
        self._world = self.get_world() #Dohvaćanje podataka o svijetu
        self._cube = self._world.scene.get_object("fancy_cube") # Dohvaćanje podataka o kocki
        self._world.add_physics_callback("sim_step", callback_fn=self.print_cube_info) #Dodavanje fizikalnog povratana informacija
        return

    # Definiranje svake fizikalne povratne informacije da se dogodi prije svakog fizikalnog koraka, sve fizikalne povratne
    # informacije moraju uzet fizikalni korak kao argument
    def print_cube_info(self, step_size):
        position, orientation = self._cube.get_world_pose()
        linear_velocity = self._cube.get_linear_velocity()
        # Sljedeće će se prikazati u terminalu
        print("Pozicija kocke je : " + str(position))
        print("Orijentacija kocke je : " + str(orientation))
        print("Linearna brzina kocke je : " + str(linear_velocity))
```

Kako bi mogli pratiti stanje kocke u stvarnom vremenu potrebno je definirati postupak uzastopnog povrata informacije na svakom simulacijskom koraku i prikazati informacije vizualno s naredbom *print* kao što je prikazano pod *print_cube_info*.



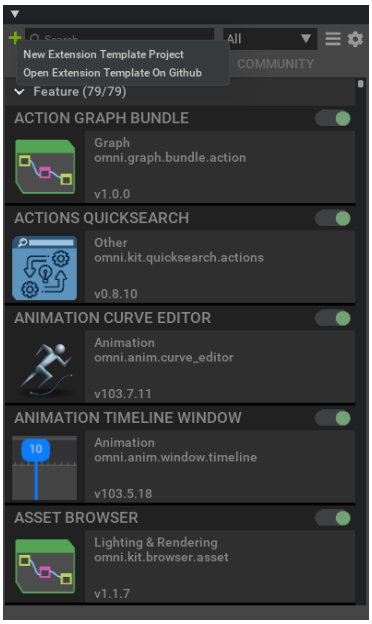
Slika 20 Prikaz zapisa podataka o kocki Izvor: Autor

Pokretanjem simulacije unutar *Isaac Sim* terminala u svakom simulacijskom koraku se zapisuju atributi postavljene kocke (slika 20).

Upute za dodavanje navedenih objekata i otvaranje spomenutih ekstenzija su dane u prilogu ovog rada.

5. IZRADA EKSTENZIJA

Prednost modularne prirode *Omniverse* paketa je personalizacija alata. Pri izradi robotskih aplikacija potrebno je organizirati željene operacije. *Omniverse* omogućava da korisnik jednostavno može započeti u dizajniranju svoje robotske aplikacije preko izrade vlastite ekstenzije. Rad ekstenzije dijele dva aspekta – sučelje ekstenzije i algoritam ekstenzije. Oba aspekta se grade objektno orijentiranim programiranjem unutar *Visual Basic Code* softvera, te se simultano povezuju jedan s drugim. Izvornom kodu svake ekstenzije se može pristupiti preko izbornika za ekstenzije. *Omniverse* i *Isaac Sim* dokumentacija sadržava alate s kojima se počinju graditi oba aspekta ekstenzije. *Isaac Sim* posjeduje izbornik za ekstenzije koji dopušta pregled svih aktivnih i neaktivnih ekstenzija unutar cijelog Omniverse paketa, te omogućava izradu vlastite ekstenzije (slika 21).

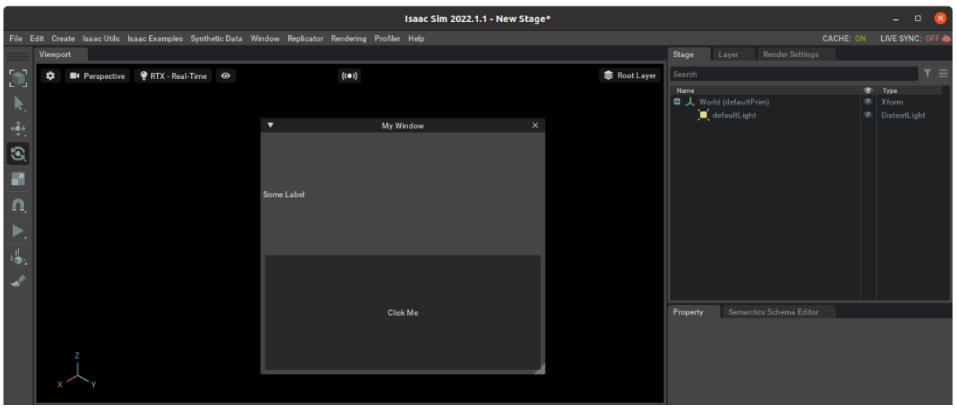


Slika 21 Izbornik za ekstenzije i upute za izradu nove ekstenzije Izvor: Autor

Prilikom izrade vlastite ekstenzije, otvara se *Visual Basic Code* s početnim kodom u kojem je definirano osnovno sučelje kao što je prikazano dolje:

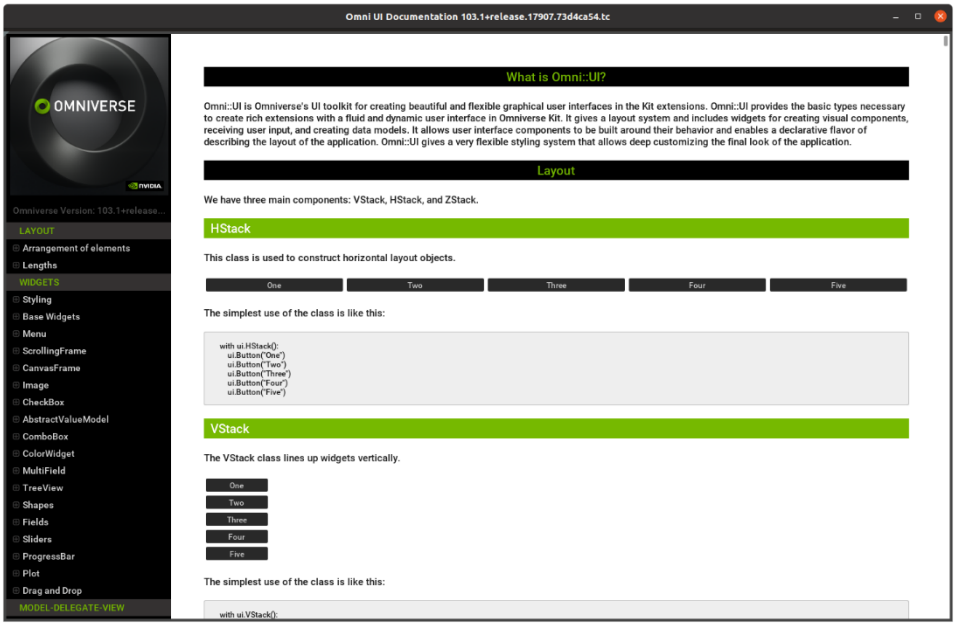
```
1.  import omni.ext
2.  import omni.ui as ui
3.
4.
5.  class MyExtension(omni.ext.IExt):
6.
7.      def on_startup(self, ext_id):
8.          print("[zavrsnirad] MyExtension startup")
9.
10.         self._window = ui.Window("My Window", width=300, height=300)
11.         with self._window.frame:
12.             with ui.VStack():
13.                 ui.Label("Some Label")
14.
15.                 def on_click():
16.                     print("clicked!")
17.
18.                 ui.Button("Click Me", clicked_fn=lambda: on_click())
19.
20.     def on_shutdown(self):
21.         print("[zavrsnirad] MyExtension shutdown")
```

Sučelje se istodobno otvara i u *Isaac Sim*-u u obliku jednostavnog prozora s imenom ekstenzije i gumbom "*Click Me*" (slika 22).



Slika 22 Početno sučelje nove ekstenzije Izvor: Autor

Za dizajniranje sučelja *Omniverse* paket posjeduje dokumentaciju specifično za izradu korisničkog sučelja (eng. *User Interface*), koja se nalazi u izborniku za ekstenzije pod nazivom *Omni UI Documentation.* Unutar dokumentacije su dijelovi koda za dizajniranje elemenata korisničkog sučelja poput : gumbova, potvrdnih okvira, padajućih izbornika, tekstualnih polja, tipkala, klizača i slično. Uz dijelove koda su dani primjeri radi boljeg razumijevanja (slika 23).

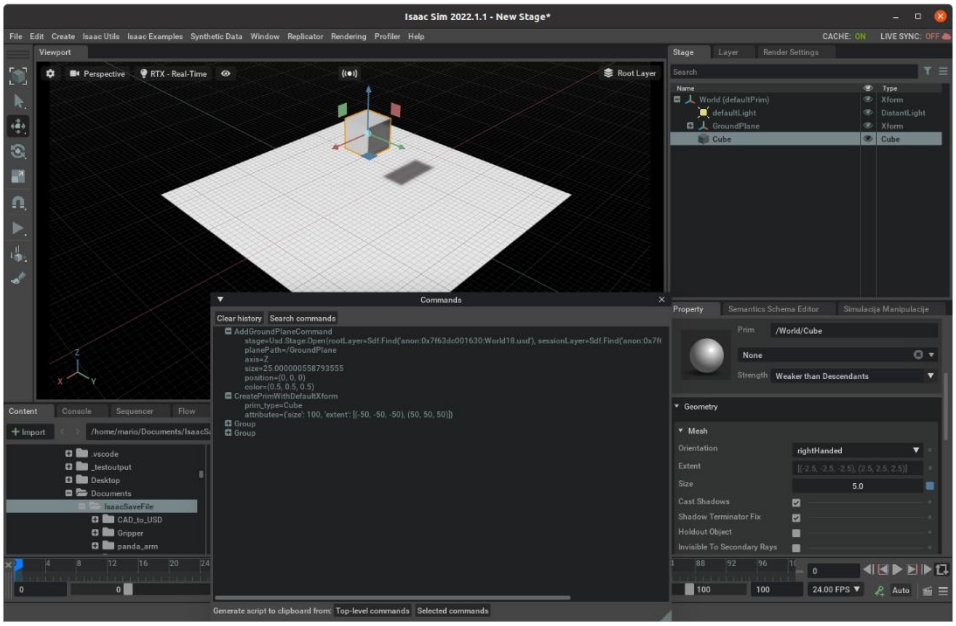


Slika 23 Omniverse dokumentacija za izradu sučelja ekstenzije Izvor: Autor

Također postoje i gotovi predlošci pod imenom *BaseSample* i *BaseSampleExtension* koji se koriste za izgradnju *Omniverse* ekstenzija. Oni se također mogu upotrebljavati, te se prilikom instalacije mogu naći lokalno na tvrdom disku. *BaseSample* i *BaseSampleExtension* su objektno orijentirani kodovi napisani u *Python* programskom jeziku koji definiraju osnovne operacije ekstenzija poput *Load*, *Clear* i *Reset* virtualne simulacije. Isto tako su i definirane naredbe za dohvat podataka o virtualnom okruženju koje čine primaran korak u svim robotskim aplikacijama. Ovi predlošci se mogu pozivati kao i ostatak biblioteka, te se preko njih jednostavno postavljaju osnovne funkcije ekstenzija.

Za realiziranje algoritma ekstenzije korisnik se može pripomoći stvaranjem skripti unutar *Isaac Sim*-a. Ovaj pristup je ograničen, no neke jednostavnije operacije može vrlo brzo pretvoriti u kod. Naime, otvaranjem ekstenzije *Commands* mogu se skriptirati određene radnje. Na primjer, ako se promjeni parametar unutar *Property* kartice dok je *Commands* aktivan, on će nam interpretirati radnju mijenjanja parametra u odgovarajuću *Python* naredbu. Koja se onda može iskoristiti u izgradnji ekstenzije.

U primjeru se stvara podložna ravnina i dodaje se kocka, a *Commands* ekstenzija pretvara radnju stvaranje ravnine i kocke u odgovarajući kod (slika 24).



Slika 24 Prikaz djelovanja Commands ekstenzije Izvor: Autor

Pritiskom na *Selected commands* se kopira interpretirani kod koji je spreman za daljnju upotrebu kao što je prikazano ispod:

```
1.  import omni.kit.commands
2.  from pxr import Gf, Usd
3.
4.  omni.kit.commands.execute('AddGroundPlaneCommand',
5.    stage=Usd.Stage.Open(rootLayer=Sdf.Find('anon:0x18b17e00:World0.usd'),
    sessionLayer=Sdf.Find('anon:0x191ad2c0:World0-session.usda')),
6.    planePath='/GroundPlane',
7.    axis='Z',
8.    size=25.000000558793555,
9.    position=Gf.Vec3f(0.0, 0.0, 0.0),
10.   color=Gf.Vec3f(0.5, 0.5, 0.5))
11.
12. omni.kit.commands.execute('CreatePrimWithDefaultXform',
13.   prim_type='Cube',
14.   attributes={'size': 100, 'extent': [(-50, -50, -50), (50, 50, 50)]})
```

Za kompleksnije algoritme potrebno je koristiti *Isaac Sim* API dokumentaciju na službenim stranicama.

Izrada ekstenzija unutar *Visual Basic Code* donosi benefite ponovnog učitavanja (eng. *Reloading*) ekstenzije s čime se postiže vrlo brzo provjeravanje koda na greške. Spremanjem *Python* datoteke, automatski se ažurira ekstenzija unutar *Isaac Sim*-a , te se tim putem može jednostavno provjeravati funkcionalnost koda. Usporedno se u *Isaac Sim* terminalu zapisuje dnevnik operacija gdje se mogu uočiti greške izvornog koda.

U nastavku je prikazan primjer izrade ekstenzije koja unosi USD datoteku, dodaje kocku i translatira je. Sučelje ekstenzija je definirano funkcijom *_build_window_* u kojoj je opisana

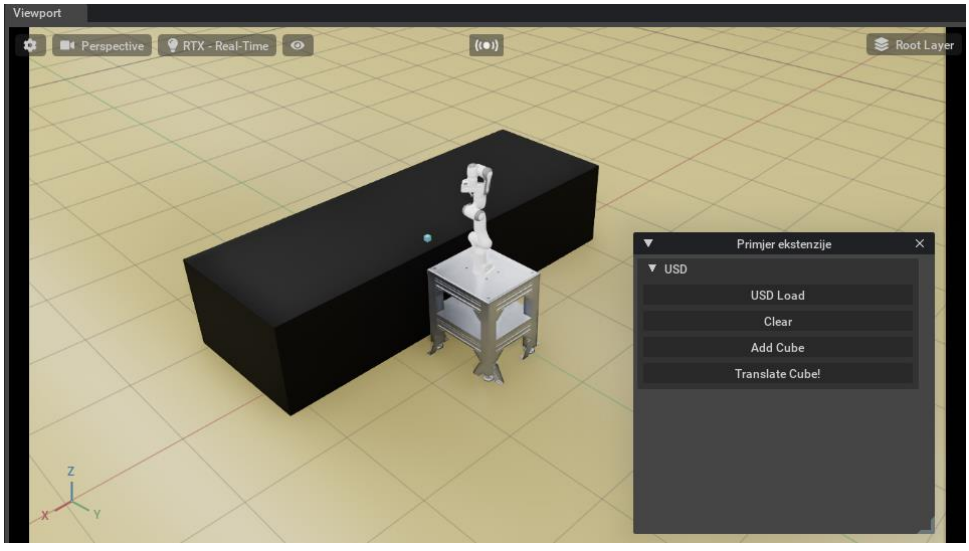
struktura sučelja s jednim padajućim izbornikom i četiri tipke *USD Load, Clear, Add Cube* i *Translate Cube*. Svakoj tipki je dodijeljeno na koju definiciju se mora opozvati prilikom pritiska kao što je prikazano ispod:

```
1.  def _build_window(self):
2.          with ui.ScrollingFrame():
3.              with ui.VStack(spacing = 0, height = 0):
4.                  with ui.CollapsableFrame("USD"):
5.                      with ui.VStack(height=0):
6.                          ui.Button("USD Load", clicked_fn=lambda: clicked_usd_load())
7.                          def clicked_usd_load():
8.                              self._on_usd_load()
9.                              print("USD Loaded")
10.                             return
11.
12.                         ui.Button("Clear", clicked_fn=lambda: clicked_clear())
13.                         def clicked_clear():
14.                             self._on_clear()
15.                             print("Viewport Cleared!")
16.
17.                         ui.Button("Add Cube", clicked_fn=lambda: clicked_add_cube())
18.                         def clicked_add_cube():
19.                             self.add_cube()
20.
21.                             print("Cube Added!")
22.
23.                         ui.Button("Translate Cube!", clicked_fn=lambda:
    clicked_translate_cube())
24.                         def clicked_translate_cube():
25.                             self.translate_cube()
26.
27.                             print("Cube Translated!")
```

Pritiskom na *USDLoad* poziva se definicija *_on_usd_load* koja dodaje odabranu USD datoteku na lokalnom disku u *Stage.* Pritiskom na *AddCube* poziva se definicija *add_cube* koja je ekstrapolirana iz *Commands* ekstenzije, te dodaje kocku unutar scene. Translatiranje je izvedeno na isti način kao i dodavanje, te se iz definicije može očitati postavljanje nove pozicije kocke u odnosu na staru poziciju kao što je prikazano ispod:

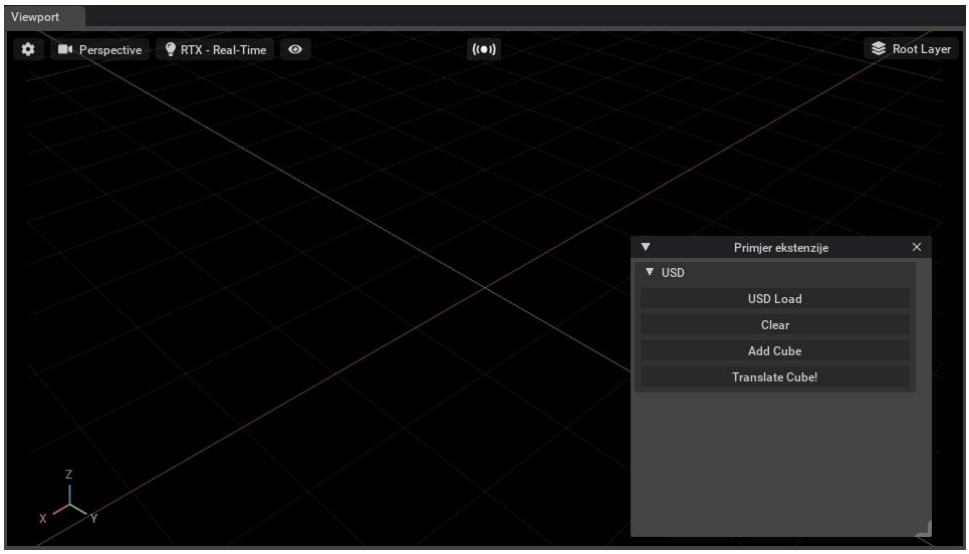
```
1.  def _on_clear(self):
2.          asyncio.ensure_future(self.clear_async())
3.          return
4.
5.      def _on_usd_load(self):
6.          stage = omni.usd.get_context().get_stage()
7.          prim = stage.DefinePrim("/World", "Xform")
8.
    prim.GetReferences().AddReference("/home/mario/Documents/IsaacSaveFile/Pisanje_zavrsnog_m
    anipulator.usd")
9.          return
10.
11.     def add_cube(self):
12.
13.         omni.kit.commands.execute('CreatePrimWithDefaultXform',
14.       prim_type='Cube',
15.         prim_path="/World/Cube",
16.       attributes={'purpose':("render"), 'size': 0.1, 'extent': [(-50, -50, -50), (50, 50,
    50)]})
17.         return
18.
19.     def translate_cube(self):
20.         omni.kit.commands.execute('TransformPrimSRT',
21.       path=Sdf.Path('/World/Cube'),
22.       new_translation=Gf.Vec3d(0.0, 0.0, 1.1),
23.       new_rotation_euler=Gf.Vec3d(0.0, -0.0, 0.0),
24.       new_rotation_order=Gf.Vec3i(0, 1, 2),
25.       new_scale=Gf.Vec3d(1.0, 1.0, 1.0),
26.       old_translation=Gf.Vec3d(0.0, 0.0, 0.0),
27.       old_rotation_euler=Gf.Vec3d(0.0, -0.0, 0.0),
28.       old_rotation_order=Gf.Vec3i(0, 1, 2),
29.       old_scale=Gf.Vec3d(1.0, 1.0, 1.0))
30.         return
```

Definicija *_on_clear* je izvedena uvozom *BaseSample* predloška. Pritiskom na tipku *Clear* definicija se opoziva na klasu *BaseSample* u kojoj je opisana funkcija za čišćenje scene. Ovim putem nije potrebno iznova definirati funkciju, već se samo opozvati na nju. Na slici 25 je prikazan izgled sučelja ekstenzije i krajnji rezultat definiranih funkcija.



Slika 25 Prikaz sučelja i USD datoteke Izvor: Autor

Dok je na slici 26 prikazan rezultat operacije *Clear.*



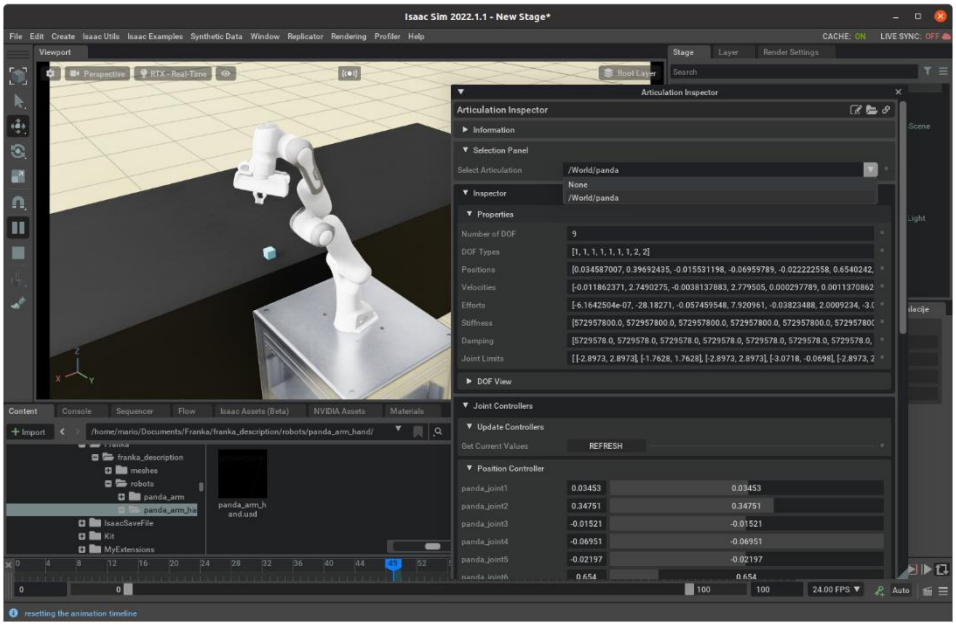
Slika 26 Rezultat operacije Clear Izvor: Autor

Upute za otvaranje navedenih ekstenzija su dane u prilogu ovog rada.

6. PROGRAMSKO RJEŠENJE U PYTHONU ZA SIMULACIJU MANIPULACIJE RADNIH OBJEKATA

Za razvitak robusnog koda za rješavanje robotskih zadataka u *Isaac Sim*-u potrebno je imati adekvatno znanje o objektno orijentiranom programiranju. Naime, *Isaac Sim* paket trenutno nema potrebne ekstenzije za izvođenje zahtjevnijih zadataka s industrijskim robotima kako bi se izbjegao odveć kompleksan i neefikasan rad u *Python* programskom jeziku. Unatoč tome *Isaac Sim* dokumentacija nudi isječke informacija za početnike u pokušaju educiranja svojih korisnika u korištenju njihove vlastite programske dokumentacije koji se sastoje od pojednostavljenih primjera.

Prilikom izrade svog vlastitog okruženja koji se sastoji od industrijskih robota bitno je pravilno postaviti gibanje/kretanje zglobova. Za početak, kako bi osigurali pravilno funkcioniranje artikulacije, *Isaac Sim* u sebi sadrži ekstenziju za kontrolu artikulacije (eng. *Articulation inspector*) čije upute za pokretanje su dane u prilogu ovog rada. Pokretanjem simulacije tipkom *Play* preko spomenute ekstenzije se mogu odabrati postojeće artikulacije unutar postavljenog virtualnog okruženja. Na slici 27 je dan primjer virtualnog okruženja s jednim *Franka* industrijskim robotom. Odabire se artikulacija robotske ruke, te se otvara spektar informacija o svakom postavljenom zglobu. Ovim putem je moguće mijenjati poziciju svakog zgloba te istodobno vizualno pratiti efekte interaktivnosti.



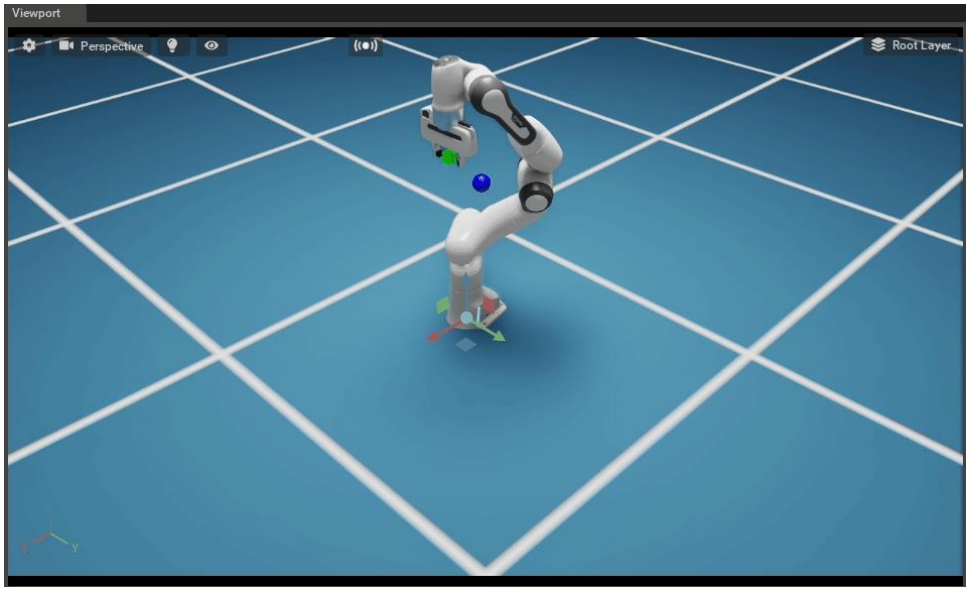
Slika 27 Prikaz Franka industrijskog robota i ekstenzije za kontrolu artikulacije Izvor: Autor

Articulation Inspector omogućava analizu hoda svakog zgloba unutar postavljene artikulacije, postavljanje potrebne krutosti i prigušenja, postavljanje željene brzine i kontroliranje pozicije zgloba. Ovakvo manipuliranje robota preko ekstenzija je postignuto implementacijom kontrolera (eng. *Controllers*).

Implementacija kontrolera je jedna od najbitnijih stavki u stvaranju programskog koda za manipuliranje robota. Korištenjem *Isaac Sim* dokumentacije moguće je isprogramirati vlastite kontrolere, ali se preporuča korištenje već postojećih blokova objektno orijentiranog koda za primjenu kontrolera unutar vlastitih programa. Kontroleri su programski algoritmi pomoću kojih je moguće manipulirati pogonom zglobova robota u cilju izvođenja određenih zadataka. Drugim riječima, kontroleri su živčani sustav robota unutar programskog koda i pomoću njih je moguće definirati kompleksnije robotske zadatke.

Isaac Sim dokumentacija posjeduje svojstvene kontrolere za specifične robote i specifične zadatke od kojih je najrelevantniji kontroler za *Pick and Place* zadatke. *Pick and Place* u žargonu robotike predstavlja proces uzimanja predmeta s jednog mjesta i odlaganja tog istog predmeta na drugo mjesto. Ovaj zadatak je jedan od učestalijih radnji koje roboti izvode u industriji. Također je predstavljen i *RMPFlowController* koji koristi matematičku postavu Riemannian-ovih smjernica kretanja (eng. *Riemannian motion policy*, krat. RMP). Riemannian-ove smjernice kretanja su novi matematički objekt za generaciju kretnji. RMP koristi drugi stupanj dinamičkog sustava u paru sa odgovarajućom Riemannian-ovom metrikom kako bi postigli jednostavno i pouzdanu metodu za kombiniranje višestrukih smjernica kretanja [26].

Robotska ruka koja ima implementirani *RMPFlow* kontroler ima mogućnost automatskog praćenja i izbjegavanje kolizija s objektima u prostoru, što je demonstracija nove tehnologije simuliranja manipulacije prikazano na slici 28.



Slika 28 Prikaz automatskog izbjegavanje objekata Izvor: Autor

U nastavku je analizirana implementacija *Pick and Place* kontrolera u programski kod. U prvom dijelu izvornog koda se uvozi klasa *PickPlaceController* iz Isaac Sim dokumentacija. Tom klasom je definiran kontroler koji je specifično dizajniran za *Pick and Place* radnju. U sljedećem dijelu su postavljene varijable unutar svijeta pomoću kojih se obavlja dohvat podataka o robotu, hvataljkama i objektima manipuliranja (kocka). A u posljednjem dijelu su opisane operacije koje se izvode prilikom pokretanja simulacije kao što je prikazano ispod:

```
1.  from omni.isaac.kit import SimulationApp
2.
3.  simulation_app = SimulationApp({"headless": False})
4.
5.  from omni.isaac.core import World
6.  import numpy as np
7.  from pick_place import PickPlace
8.  from pick_place_controller import PickPlaceController
9.
10. my_world = World(stage_units_in_meters=1.0)
11.
12.
13. target_position = np.array([-0.3, 0.6, 0])
14. target_position[2] = 0.0515 / 2.0
15. my_task = PickPlace(name="denso_pick_place", target_position=target_position)
16.
17. my_world.add_task(my_task)
18. my_world.reset()
19. task_params = my_world.get_task("denso_pick_place").get_params()
20. denso_name = task_params["robot_name"]["value"]
21. my_denso = my_world.scene.get_object(denso_name)
22. my_controller = PickPlaceController(name="controller", robot_articulation=my_denso,
    gripper=my_denso.gripper)
23. task_params = my_world.get_task("denso_pick_place").get_params()
24. articulation_controller = my_denso.get_articulation_controller()
25. i=0
26. while simulation_app.is_running():
27.     my_world.step(render=True)
28.     if my_world.is_playing():
29.         if my_world.current_time_step_index == 0:
30.             my_world.reset()
31.             my_controller.reset()
32.         observations = my_world.get_observations()
33.         #forward the observation values to the controller to get the actions
34.         actions = my_controller.forward(
35.             picking_position=observations[task_params["cube_name"]["value"]]["position"],
36.
    placing_position=observations[task_params["cube_name"]["value"]]["target_position"],
37.
    current_joint_positions=observations[task_params["robot_name"]["value"]]["joint_positions
    "],
38.             # This offset needs tuning as well
39.             end_effector_offset=np.array([0, 0, 0.25]),
40.         )
41.         if my_controller.is_done():
42.             print("done picking and placing")
43.         articulation_controller.apply_action(actions)
44. simulation_app.close()
```

Implementacija ovih alata nije optimizirana za široku uporabu.

7. ZAKLJUČAK

NVIDIA Isaac Sim simulator se razvija u skladu s četvrtom industrijskom revolucijom. *Isaac Sim* je trenutno u fazi razvoja, zbog čega je dokumentacija nepotpuna i ne optimizirana. Za rad u njemu potrebne su napredne vještine u objektno orijentiranom programiranju te je na korisniku da se služi datim primjerima za razvoj individualnih aplikacija.

U službi *Isaac Sim*-a je napredan grafički pokretač koja omogućava precizno i kompleksno simuliranje fizikalnih zakona što odvaja *Isaac Sim* od ostalih simulatora. *NVIDIA* je stvorila *Omniverse* platformu na kojoj se odvija konstantan razvoj novih aplikacija i alata.

Isaac Sim implementira nove tehnologije u robotici te najbolje rezultate daje u suradnji s drugim simulacijskim softverima.

Izrada virtualnog okruženja je vrlo jednostavan i intuitivan proces, no implementacija trenutno postojećih alata za izradu kompleksnijih simulacija je dosta ograničena. Bitno je obratiti pozornost na artikulaciju zglobova pri radu s industrijskim robotima.

Velika prednost *Isaac Sim*-a je njegova modularna priroda, koja omogućava veliku razinu slobode korisniku, odnosno korisnik može izmjenjivati sve alate unutar *Isaac Sim*-a. Osim toga, softver je besplatan i otvorenog izvora zbog čega postoji aktivna komunikacija između kreatora i korisnika, a time i velik broj povratnih informacija.

LITERATURA

[1] John Wiley & Sons Inc., »Handbbook of Simulation,« 1998.. [Mrežno]. Available: shorturl.at/cyUX8.

[2] M. M. Gunal, »Simulation for the Better: The Future in Industry 4.0,« Svibanj 2019. [Mrežno]. Available: https://doi.org/10.1007/978-3-030-04137-3_16. [Pokušaj pristupa 01 2022].

[3] V. Albino, N. Carbonara i I. Giannoccaro, »ScienceDirect,« 2006. [Mrežno]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0925527305000204. [Pokušaj pristupa 16 8 2022].

[4] L. F. McGinnis i O. Rose, »IEEE XPlore,« 3 12 2017. [Mrežno]. Available: https://ieeexplore.ieee.org/abstract/document/8247801/authors#authors. [Pokušaj pristupa 15 Kolovoz 2022.].

[5] Staranowicz, Aaron and Mariottini i L. Gian, »ACM Digital Library Home,« 2011. [Mrežno]. Available: https://dl.acm.org/doi/abs/10.1145/2141622.2141689. [Pokušaj pristupa 16 8 2022].

[6] A. Shaw, »Technia Addnode Group,« [Mrežno]. Available: https://www.technia.com/blog/what-is-delmia/. [Pokušaj pristupa 17 Kolovoz 2022].

[7] C. Connolly, »Emerald Insight,« 2006. [Mrežno]. Available: https://www.emerald.com/insight/content/doi/10.1108/01439910610667863/full/html. [Pokušaj pristupa Kolovoz 2022].

[8] A. Atanassov i A. Garbev, »IEEE Xplore,« 7 Siječanj 2021. [Mrežno]. Available: https://ieeexplore.ieee.org/abstract/document/9311332. [Pokušaj pristupa Kolovoz 2022].

[9] »RoboDK,« RoboDK, 2016. [Mrežno]. Available: https://robodk.com/blog/off-line-programming/. [Pokušaj pristupa Kolovoz 2022].

[10] R. Smith, »TU/e Eindhoven University of Technology,« 29 Svibanj 2004. [Mrežno]. Available:

https://robocup.wtb.tue.nl/svn/techunited/trunk/old/Robocup%20Bremen%202006/src/Turtle1/Libs/ode-0.5/ode/doc/ode.pdf. [Pokušaj pristupa 17 Kolovoz 2022].

[11] T. J. Trappler, »Educause Review,« 30 Lipanj 2009. [Mrežno]. Available: https://er.educause.edu/articles/2009/7/is-there-such-a-thing-as-free-software-the-pros-and-cons-of-opensource-software. [Pokušaj pristupa Kolovoz 2022].

[12] »ROS,« ROS, [Mrežno]. Available: https://www.ros.org/blog/why-ros/. [Pokušaj pristupa Kolovoz 2022].

[13] »NVIDIA,« NVIDIA, [Mrežno]. Available: https://www.nvidia.com/en-us/omniverse/. [Pokušaj pristupa 30 Kolovoz 2022].

[14] »What Is Isaac Sim?,« Omniverse, 30 Kolovoz 2022. [Mrežno]. Available: https://docs.omniverse.nvidia.com/app_isaacsim/app_isaacsim/overview.html. [Pokušaj pristupa 30 Kolovoz 2022].

[15] »API,« Wikipedia, [Mrežno]. Available: https://en.wikipedia.org/wiki/API. [Pokušaj pristupa Kolovoz 2022].

[16] »Nvidia Developer,« NVIDIA, [Mrežno]. Available: https://developer.nvidia.com/isaac-sim. [Pokušaj pristupa 31 Kolovoz 2022].

[17] »PCPARTPICKER,« [Mrežno]. Available: https://pcpartpicker.com. [Pokušaj pristupa 30 Kolovoz 2022].

[18] N. Omniverse, »YouTube,« NVIDIA, 17 Kolovoz 2021. [Mrežno]. Available: https://www.youtube.com/watch?v=nI_jm0BIJGI. [Pokušaj pristupa 30 Kolovoz 2022].

[19] »Omniverse: Create,« NVIDIA, 2 Rujan 2022. [Mrežno]. Available: https://docs.omniverse.nvidia.com/app_create/app_create. [Pokušaj pristupa 1 Rujan 2022].

[20] »NVIDIA PhysX SDK Documentation,« NVIDIA, [Mrežno]. Available: https://documentation.help/NVIDIA-Physx-SDK-Guide/Indeks.html. [Pokušaj pristupa 12. Rujan 2022].

[21] »unstop,« 14 Travanj 2022. [Mrežno]. Available: https://unstop.com/blog/what-is-object-oriented-programming. [Pokušaj pristupa 20 Rujan 2022].

[22] E. Doherty, »educative,« 15 Travanj 2020. [Mrežno]. Available: https://www.educative.io/blog/object-oriented-programming. [Pokušaj pristupa 5 Rujan 2022].

[23] »VisualStudio,« Visual Studio Code, [Mrežno]. Available: https://code.visualstudio.com. [Pokušaj pristupa 13. Rujan 2022].

[24] H. L. Lu, »Youtube,« NVIDIA, 11. Lipanj 2021. [Mrežno]. Available: https://www.youtub.com/watch?v=pxPFr58gHmQ. [Pokušaj pristupa 10. Svibanj 2022].

[25] »Articulations,« NVIDIA, 10. Rujan 2022. [Mrežno]. Available: https://docs.omniverse.nvidia.com/app_create/prod_extensions/ext_physics/articulations.html#articulationroot. [Pokušaj pristupa 10. Rujan 2022.].

[26] N. D. Ratliff, J. Issac, D. Kappler, S. Birchfield i D. Fox, »Cornell University,« 25 Srpanj 2018. [Mrežno]. Available: https://arxiv.org/abs/1801.02854. [Pokušaj pristupa 15. Rujan 2022.].

[27] »Cyberbotics,« Cyberbotics, 2022. [Mrežno]. Available: https://cyberbotics.com/doc/guide/system-requirements. [Pokušaj pristupa Kolovoz 2022].

[28] »Omniverse,« NVIDIA, 1 Rujan 2022. [Mrežno]. Available: https://docs.omniverse.nvidia.com/app_isaacsim/app_isaacsim/requirements.html. [Pokušaj pristupa 1 Rujan 2022].

[29] A. Garbev i A. Atanassov, »IEEE Xplore,« 3 Listopad¸ 2020. [Mrežno]. Available: https://ieeexplore.ieee.org/abstract/document/9311332. [Pokušaj pristupa 19 Rujan 2022].

PRILOG

I. Upute za korištenje osnovnih Isaac Sim alata :
https://github.com/JabloStabuke/IsaacSimUPUTE.git