Digitalni dvojnik robotskog postrojenja za sječenje s korisničkom korekcijom putem virtualne stvarnosti

Arih, Dominik

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 442

DIGITALNI DVOJNIK ROBOTSKOG POSTROJENJA ZA SJEČENJE S KORISNIČKOM KOREKCIJOM PUTEM VIRTUALNE STVARNOSTI

Dominik Arih

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 442

DIGITALNI DVOJNIK ROBOTSKOG POSTROJENJA ZA SJEČENJE S KORISNIČKOM KOREKCIJOM PUTEM VIRTUALNE STVARNOSTI

Dominik Arih

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

DIPLOMSKI ZADATAK br. 442

Pristupnik:	**Dominik Arih (0036526172)**
Studij:	Računarstvo
Profil:	Znanost o mrežama
Mentor:	izv. prof. dr. sc. Mirko Sužnjević

Zadatak: **Digitalni dvojnik robotskog postrojenja za sječenje s korisničkom korekcijom putem virtualne stvarnosti**

Opis zadatka:

Digitalni dvojnici (engl. digital twins) postaju ključan element u robotici, omogućujući precizno modeliranje stvarnih robota u digitalnom prostoru. Ova tehnologija omogućuje inženjerima testiranje i optimizaciju performansi robota prije njihove stvarne primjene, što značajno ubrzava razvoj i smanjuje rizik. Istovremeno, upotreba virtualne stvarnosti u robotici omogućuje inženjerima i operaterima upravljanje robotskim sustavima kroz imerzivan medij, pružajući intuitivniji način za upravljanje, obuku i održavanje. Vaš zadatak je razviti digitalnog dvojnika jednostavnog robotskog postrojenja, u kojem robotska ruka ima zadatak prerezati 3D objekt upravljana jednostavnom umjetnom inteligencijom (UI). Simulaciju je potrebno moći prikazati u virtualnoj stvarnosti. Osim toga, korisniku virtualne stvarnosti trebalo bi biti omogućeno ručno precizno rezanje 3D objekta s mogućnošću pamćenja staze rezanja, orijentacije oštrice te krivulje kojom je oštrica putovala. Ovim pristupom omogućit će se ljudska korekcija ponašanja algoritama UI-a te njihova evolucija kroz podržano učenje.

Rok za predaju rada: 28. lipnja 2024.

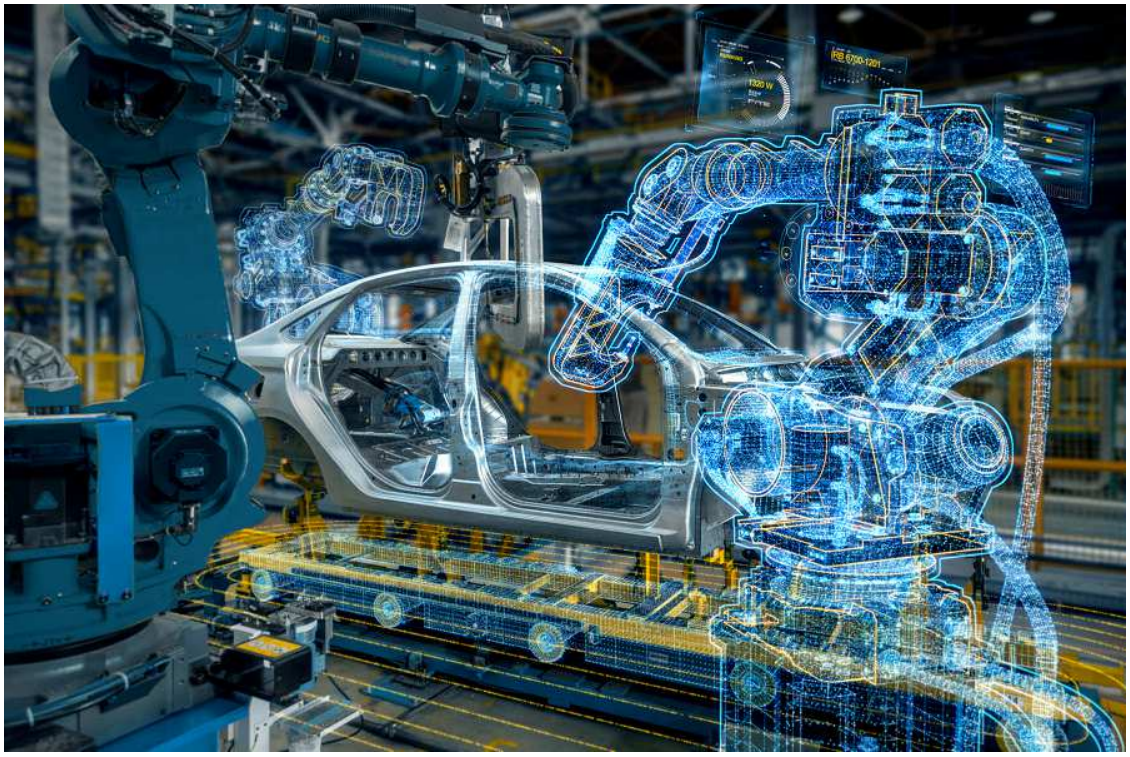
SADRŽAJ

1. Uvod

Digitalni dvojnici (engl. *Digital twins*) postaju sveprisutni element u suvremenom inženjerskom okruženju. Pružaju manipulaciju fizičkim entitetima i ključnu podršku u optimizaciji složenih sustava putem integracije raznolikih tehnoloških komponenti [1]. Također, digitalni dvojnici omogućuju precizno modeliranje stvarnih robota unutar digitalnog prostora te pružaju osnovu za inovativne pristupe testiranju, optimizaciji i razvoju robotskih sustava. Primjenom takvih tehnoloških rješenja ubrzava se proces razvoja robota, ali i smanjuju rizici i troškovi eksperimentalnih faza implementacije u stvarnom svijetu.

Suvremeni inženjerski pristupi sve više prepoznaju važnost simulacija u razvoju kompleksnih sustava, posebno u području robotike. Napredne tehnologije omogućuju stvaranje virtualnog svijeta iz stvarnog te njihovu međusobnu komunikaciju tijekom proizvodnje [2]. S pomoću digitalnih dvojnika inženjerima je omogućeno testiranje i optimizacija performansi robota prije praktične primjene. Postoji mogućnost preciznog modeliranja ponašanja robota u različitim scenarijima, pružajući temelj za kontinuirano poboljšanje dizajna i funkcionalnosti u kasnijim fazama. Smanjenje nesigurnosti, troškova, neučinkovitosti i pogrešaka samo su neke od prednosti koje proizlaze iz korištenja spomenutih simulacija naspram tradicionalnog industrijskog razvoja robota [3].

Cilj ovog istraživanja je predstaviti integrirani pristup razvoju simulacije digitalnog dvojnika jednostavnog robotskog postrojenja, s naglaskom na primjeni u rezanju 3D objekata. Kroz nekoliko poglavlja opisana je ideja implementacije sustava digitalnog dvojnika kroz konkretne algoritme, tehnologije i tehnike korištene u procesu. Osnovni ishod uključuje razvoj jednostavne simulacije koja se može nadograđivati u kompleksni sustav za specifične primjene unutar industrije. Opisana je implementacija **podržanog učenja** (engl. *Reinforcement Learning – **RL***) koje omogućuje interakciju korisnika i prilagodbu ponašanja simulacije prema željenim rezultatima. Također, istražene su i mogućnosti **virtualne stvarnosti** (engl. *Virtual Reality* - **VR**), koja korisnicima pruža intuitivne metode za obuku, održavanje i upravljanje robotskim sustavom.



Slika 1.1: Simbolični prikaz robotskog postrojenja i pripadajućih digitalnih dvojnika. Slika preuzeta iz [4].

Analiza rezultata izvođenja simulacije predstavljena je kroz detaljne zapise generirane tijekom izvršavanja, uz raspravu o uspješnosti postignutih rezultata i identifikaciji osjetljivih faza simulacije s mogućim poboljšanjima. Posebna pažnja pridaje se ideji kompleksnijih sustava, te se raspravlja o prednostima i izazovima njihove implementacije, ističući neka od mogućih rješenja za prevladavanje istih.

Konačno, stavljen je naglasak na važnost ovog istraživanja u brzorastućoj industriji primjene digitalnih dvojnika reguliranih korisničkim interakcijama u virtualnom okruženju. Razumijevanje i implementacija ovakvih sustava imaju potencijal unapređenja načina na koji se robotika koristi u različitim industrijskim sektorima, pridajući prednost učinkovitosti, sigurnosti i praktičnosti. Važnost ovog istraživanja leži u potencijalu za ubrzanje procesa razvoja u primjeni robotskih tehnologija u stvarnom svijetu, kao i unapređenju performansi i funkcionalnosti predstavljenog jednostavnog robotskog postrojenja.

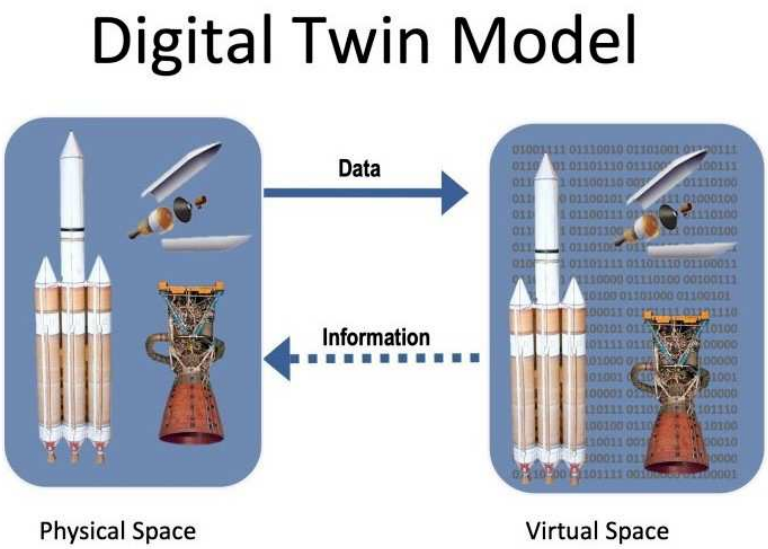
2. Digitalni dvojnici

Koncept digitalnog dvojnika, također poznatog kao digitalni blizanac, ima svoje korijene u NASA-inom programu *Apollo*, gdje je identična svemirska kapsula simulirala ponašanje kapsule u svemiru [5]. Iako je u tom kontekstu riječ samo o fizičkom entitetu, a ne digitalnom, evolucija koncepta digitalnog dvojnika tijekom vremena je izražena, ali i dalje nedovoljno jasno definirana [5]. Pojam digitalnog dvojnika se prvi put koristi tek 2003. godine od strane dr. Michaela Grievesa sa Sveučilišta u Michiganu [5]. S obzirom na suvremene primjene, digitalni dvojnik se može promatrati kao virtualna replika fizičkog entiteta ili sustava koja se konstruira uporabom podataka iz stvarnog svijeta. Takvi digitalni entiteti se oblikuju korištenjem različitih tehnologija poput senzora, softverskih algoritama ili tehnika obrade podataka. Glavna svrha digitalnih dvojnika je modeliranje stvarnog objekta ili sustava na način koji omogućuje praćenje, analizu, simulaciju, predviđanje, dijagnosticiranje i upravljanje fizičkim objektom u virtualnom okruženju [6].

2.1. Prednosti primjene digitalnih dvojnika u robotici

Kao što je već navedeno, primjena digitalnih dvojnika u različitim područjima, uključujući robotiku, donosi niz ključnih prednosti koje značajno unapređuju funkcionalnost, sigurnost i učinkovitost sustava.

Jedna od osnovnih prednosti je kontinuirano nadgledanje stanja robotskog sustava u stvarnom vremenu. Time je omogućena brza detekcija problema ili nepravilnosti te pravovremena reakcija korisnika radi održavanja optimalnih performansi ili ispravljanja eventualnih grešaka. Digitalni dvojnici, kojima nedostaje mogućnost analize scenarija u stvarnom vremenu, nazivaju se "slabim dvojnicima" [8]. Također, upravljanje virtualnim entitetima omogućuje optimizaciju performansi stvarnih entiteta. Korištenjem simulacija unutar digitalnih dvojnika, moguće je testirati različite strategije i algoritme upravljanja robotskim sustavima u virtualnom okruženju prije implementacije u stvarnom svijetu, što omogućuje precizno podešavanje parametara kako bi se



Slika 2.1: Preteča koncepta digitalnog blizanca korištenog u NASA-inom programu *Apollo*. Slika preuzeta iz [7].

postigla maksimalna učinkovitost i preciznost u radu robota. Digitalni dvojnici, kao dio opće digitalne transformacije u svim sektorima industrije, predstavljaju značajan potencijal za povećanje učinkovitosti [9].

U područjima gdje su eksperimenti skupi ili potencijalno opasni, postoji težnja za smanjenjem njihovih troškova i rizika. S obzirom na to da digitalni dvojnici omogućuju simulaciju različitih scenarija i testiranje bez potrebe za fizičkim objektima, smanjuju se troškovi eksperimentalnih testiranja i minimizira rizik od oštećenja ili neželjenih incidenata. Mehanizmi za istraživanje incidenata omogućavaju učenje i bolju pripremu za slične incidente u budućnosti [10]. Digitalni dvojnici su fleksibilni i prilagodljivi. Time je ostvarena brza optimizacija robota, a mogućnost virtualnog modeliranja i simuliranja omogućuje inženjerima da eksperimentiraju s različitim dizajnom ili upravljačkim algoritmima te da ih prilagode bez potrebe za skupim ili rizičnim naknadnim promjenama robotskog sustava.

Analiza podataka iz stvarnog svijeta omogućuje iterativno poboljšanje modela robota te kontinuirano usavršavanje njegovih performansi i funkcionalnosti. Integracija digitalnih dvojnika s **Internetom stvari** (engl. *Internet of Things* - ***IoT***) dodatno proširuje mogućnosti upravljanja i nadzora robotskih sustava, omogućujući daljinsku kontrolu i nadzor robota putem mreže, što je posebno korisno u kontekstu automatizacije i udaljenih operacija. Koncepti digitalnih dvojnika i Interneta stvari se preklapaju u kontekstu opisivanja, otkrivanja i pristupa resursima [11].

2.2. Budućnost i izazovi tehnologije digitalnih dvojnika

Budućnost digitalnih dvojnika leži u kontinuiranom razvoju naprednih simulacijskih tehnika. Smatra se kako će integracija **umjetne inteligencije** (engl. *Artificial Intelligence - **AI***), **strojnog učenja** (engl. *Machine learning*) i **dubokog učenja** (engl. *Deep learning*) omogućiti stvaranje sve realističnijih virtualnih okruženja u kojima će se robotski entiteti moći testirati i optimizirati prije stvarne implementacije. Pošto ovakva tehnologija može donijeti revolucionarne promjene u načinu rada i optimizaciji procesa, postoji težnja za proširenjem njezine primjene na različite industrije i sektore. Također, neosporno je da takav razvoj digitalnih dvojnika zahtijeva suradnju između različitih disciplina i temeljito razumijevanje specifičnih zahtjeva različitih područja. Aspekti poput sigurnosti, kibernetičke sigurnosti i pouzdanosti digitalnih dvojnika i dalje su otvorena pitanja koja nisu potpuno obrađena [12].

S razvojem digitalnih dvojnika, postavlja se pitanje sigurnosti, ali i etičnosti njihove primjene. Osiguranje integriteta i privatnosti podataka u virtualnim okruženjima, kao i sprječavanje potencijalne zloupotrebe tehnologije, samo su neki od izazova za budući razvoj ovakvog pristupa robotici. Korištenje digitalnih blizanaca može pojačati opasnosti od gubitka, curenja ili krađe podataka uslijed sigurnosnih nemara ili propusta [13]. Uz to, potrebno je osmisliti programe obrazovanja koji će novim radnicima omogućiti stjecanje vještina potrebnih za rad s ovakvom tehnologijom, ali isto tako i osigurati kontinuirano usavršavanje za postojeće stručnjake.

Smatra se kako budućnost tehnologije digitalnih dvojnika neće biti ograničena samo na tehnički napredak, već će imati i dubok društveni i ekonomski utjecaj. Prilikom daljnjeg razvoja ove tehnologije potrebno je uzeti u obzir utjecaj na promjene u načinu obavljanja poslova, te razvoj u skladu s etičkim načelima. Osim toga, smatra se da će u budućnosti takva tehnologija omogućiti prijelaz na procese vođene podacima, gdje se parametri prilagođavaju učenjem, a promjene se automatski identificiraju i odobravaju [14].

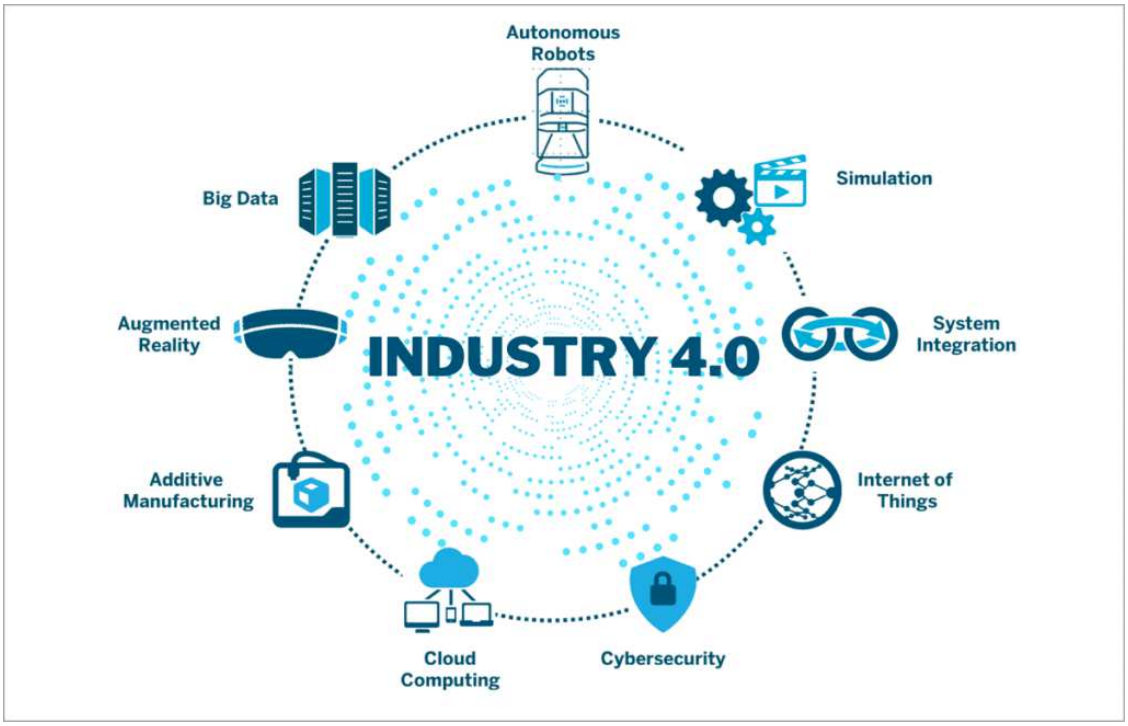
2.2.1. Industrija 4.0

Industrija 4.0 (engl. *Industry 4.0*) predstavlja "Četvrtu industrijsku revoluciju" (**4IR**), odnosno koncept integracije naprednih digitalnih tehnologija u gotovo sve aspekte proizvodnje. Takav proces obuhvaća implementaciju Interneta stvari, umjetne inteligencije, analize velikih količina podataka te automatizacije u proizvodne procese, kako bi se stvorile pametne tvornice i sustavi s visokom razinom autonomije.

Koncepti Industrije 4.0 tvrtkama bi omogućili fleksibilne proizvodne procese i analizu velikih količina podataka u stvarnom vremenu, unaprjeđujući time strateško i operativno donošenje odluka [15].

U kontekstu razvoja opisanog koncepta, upravo digitalni dvojnici mogu poslužiti za ostvarenje inteligentnih, održivih i prilagodljivih postrojenja. Zbog prethodno opisanih prednosti ove tehnologije (Poglavlje 2.1), moguće je očekivati njezin doprinos većoj fleksibilnosti i učinkovitosti industrijskih okruženja. Digitalni dvojnici su temelj za inovacije u proizvodnji, omogućujući tvrtkama bržu reakciju na promjene tržišta i potrebe potrošača. Također, jedna od istaknutih prednosti koju bi oni unutar konteksta Industrije 4.0 mogli donijeti jest sigurnija interakcija i suradnja između čovjeka i robota [16].

Stoga, tehnologiju digitalnih dvojnika možemo smatrati ključnom za ostvarivanje vizije Industrije 4.0 i značajnog utjecaj na budućnost proizvodnje i industrije u cjelini. Slika 2.2 prikazuje neke od tehnologija obuhvaćenih konceptom Industrije 4.0, kao što su autonomni robotski sustavi, simulacije, računalstvo u oblaku, proširena stvarnost i kibernetička sigurnost.



Slika 2.2: Primjeri tehnologija koje obuhvaća koncept Industrije 4.0. Slika preuzeta iz [17].

2.2.2. Digitalni dvojnici temeljeni na računalnom oblaku

Digitalni dvojnici temeljeni na **računalnom oblaku** (engl. *Cloud-based digital twins*) predstavljaju budućnost industrijskih sustava zbog svoje sposobnosti integracije s globalnim mrežama podataka i resursa. Korištenjem računalnog oblaka za pohranu i obradu podataka, digitalni dvojnici postaju dostupni većem rasponu korisnika. Time se također omogućuje brza i fleksibilna razmjena informacija između različitih lokacija i uređaja. Prijelaz na tehnologiju računalnog oblaka predstavlja strateški potez koji oblikuje budućnost inovacija te omogućuje njihov rast i brzu prilagodbu u stvarnom vremenu [18].

Slika 2.3: Preporučene prakse upravljanja podacima digitalnih dvojnika. Slika prevedena iz [18].

Ovakav koncept sadrži potencijal za napredne analize podataka i strojno učenje, što omogućuje digitalnim dvojnicima da postanu sve pametniji i samostalniji s vremenom. Korištenje naprednih algoritama pridonosi sposobnosti predviđanja budućih scenarija, a time i optimizaciji proizvodnih procesa uz samostalno donošenje odluka u stvarnom vremenu. Osim toga, digitalni dvojnici temeljeni na računalnom oblaku ostvaruju

skalabilnost i prilagodljivost potrebnu za obradu dinamičnih zahtjeva modernih industrijskih okruženja. S druge strane, korištenjem **rubnog računalstva** (engl. *Edge computing*) se također ostvaruje potencijal za unaprjeđenjem digitalnih dvojnika, pružajući poboljšanje performansi i veliku operativnu transparentnost [19]. Kako bi organizacije osigurale zadovoljavajuće performanse sustava i točne uvide u podatke, potrebno je uspostaviti dobre prakse upravljanja podacima (Slika 2.3) [18].

Računalni oblak digitalnim dvojnicima omogućuje da, bez obzira na veličinu ili složenost sustava, kontinuirano pružaju visoku razinu usluge. Međutim, iskorištavanje digitalnih dvojnika unutar koncepta računalnog oblaka također donosi svojevrsne izazove. Zahtijeva se spremnost programera i operativnih timova na promjenu načina razmišljanja i pristupa, uključujući prihvaćanje novih modela i prilagodbu prema korisnički orijentiranom pristupu [20].

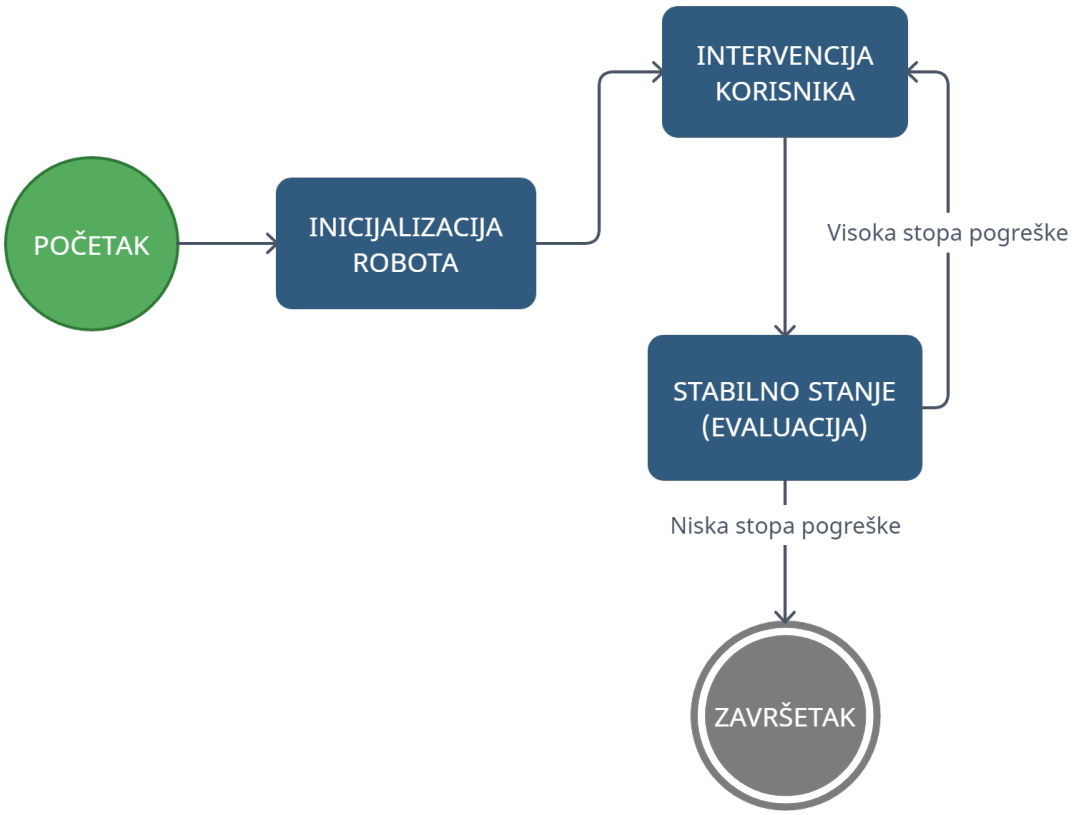
3. Opis sustava

Jedan od ishoda ovog rada obuhvaća izradu prototipa jednostavnog robotskog postrojenja (u daljnjem tekstu "jednostavni sustav"), odnosno robotske ruke koja ima sposobnost rezanja 3D objekata u virtualnom prostoru. Simulacija također omogućuje korisnički unos, odnosno rezanje tih objekata od strane osobe koja upravlja sustavom ili ga nadzire. Tijekom trajanja simulacije kontinuirano se bilježe podaci o uspješnosti, odnosno preciznosti rezova robota ili korisnika. Sustav za evaluaciju rezova izračunava postotke volumena odrezanih dijelova u odnosu na ukupni volumen te, sukladno tome, dodjeljuje svakom rezu opisnu ocjenu kvalitete (loše - *BAD*, srednje - *MEDIUM* ili dobro - *GOOD*). Jednostavan sustav koristi algoritam učenja kako bi s vremenom optimizirao točnost svojih rezova i približio se točnosti korisničkih rezova (koji se smatraju ispravnima). Krajnji cilj izvođenja simulacije jest da sustav, kroz nekoliko iteracija izmjene robotskih i korisničkih rezova, postigne stanje u kojem većina robotskih rezova bude označena kao dobra. Potrebno je napomenuti kako ovakav jednostavan sustav ima ograničenu primjenu i služi samo kao pokazni model za razvoj kompleksnijeg sustava (Poglavlje 3.2) koji koristi napredne tehnologije, a čija izvedba nije trivijalna. Svi detalji vezani uz izvođenje simulacije jednostavnog prototipa, uz potencijalne nedostatke i moguća poboljšanja, raspravljeni su u ostatku ovog poglavlja.

3.1. Funkcionalnosti jednostavnog sustava

Kao što je već navedeno, ovaj sustav je zamišljen kao simulacija kroz nekoliko sekvenci radnji robota ili korisnika, pri čemu sustav prolazi kroz tri stanja: inicijalizaciju robota, intervenciju korisnika te "stabilno" stanje. Inicijalizacija robota uključuje odabir tri nasumična kuta oštrice i izvođenje početnih nasumičnih rezova. Tijekom intervencije korisnika, korisnik izvodi tri reza koja smatra ispravnima, te se ti rezovi kasnije koriste za optimizaciju učinkovitosti robota. Stanje unutar kojeg robot izvodi rezove koji nisu nasumični, već su rezultat prethodnih evaluacija i optimizacijskih algoritama, naziva se stabilnim stanjem. Ovo stanje završava konačnom procjenom us-

pješnosti čitave iteracije. Ako se iteracija smatra neuspješnom, simulacija nastavlja s korisničkim intervencijama i stabilnim stanjima sve dok ponašanje robota ne postane zadovoljavajuće. Stanje evaluacije naziva se stabilnim jer simulacija teoretski može biti beskonačna. Ako robot postigne zadovoljavajuću razinu preciznosti, može nastaviti s radom sve do pojave greške ili naznake potencijalne degradacije kvalitete. U tom slučaju, zbog lakšeg praćenja zapisa u pojedinim stanjima, stanje evaluacije prelazi u završetak, u kojem robot prestaje funkcionirati. Elementi pojedinih stanja detaljnije su opisani u nastavku poglavlja, a Slikom 3.1 prikazan je automat stanja jednostavnog sustava.

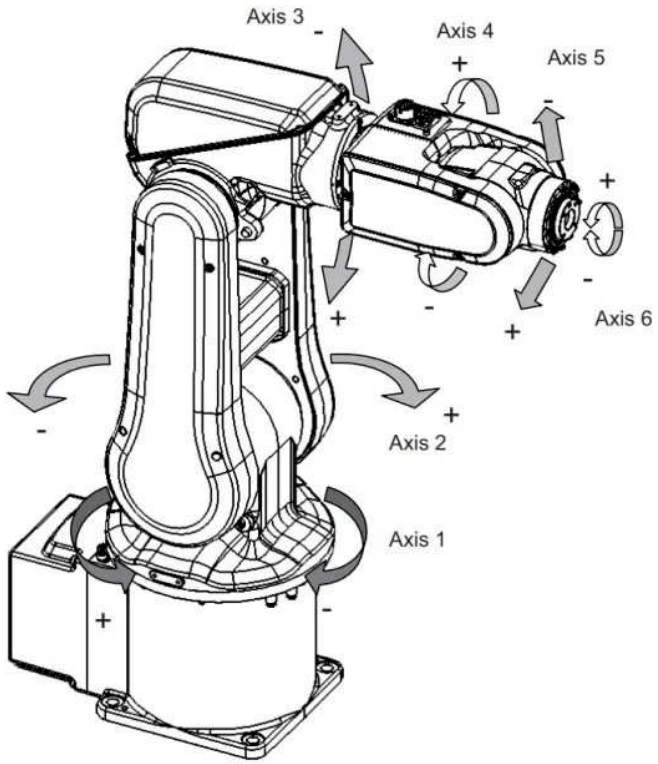


Slika 3.1: Automat stanja jednostavnog sustava za rezanje objekata.

3.1.1. Model robotske ruke

S obzirom na to da je riječ o jednostavnom robotskom postrojenju, za rezanje objekata potrebna je samo jedna robotska ruka. Način kretanja robotske ruke temelji se na složenom sustavu zglobova i veza, omogućavajući visoku razinu fleksibilnosti i preciznosti.

Mogućnosti robota u smislu dosega i kuteva rezanja ovise o njegovoj konstrukciji i dostupnim zglobovima. **Rotacijski zglobovi** omogućuju rotaciju jednog segmenta u odnosu na drugi oko jedne osi, što je najčešći tip zgloba u robotskim rukama. **Prizmatični zglobovi** omogućuju translacijsko kretanje duž jedne osi, tako da se jedan segment linearno pomiče u odnosu na drugi. **Sferni zglobovi** pružaju najviši stupanj slobode jer omogućuju rotaciju oko tri ortogonalne osi. **Cilindrični zglobovi** kombiniraju rotacijsko i translacijsko kretanje duž jedne osi, čime dodaju dodatnu fleksibilnost robotskoj ruci. Slika 3.2 prikazuje robotsku ruku modela **ABB IRB 120** sa šest stupnjeva slobode (engl. *Degrees of Freedom* - ***DoF***) koji omogućuju njezino kretanje unutar tri dimenzije (pokretljivost zglobova označena je strelicama).

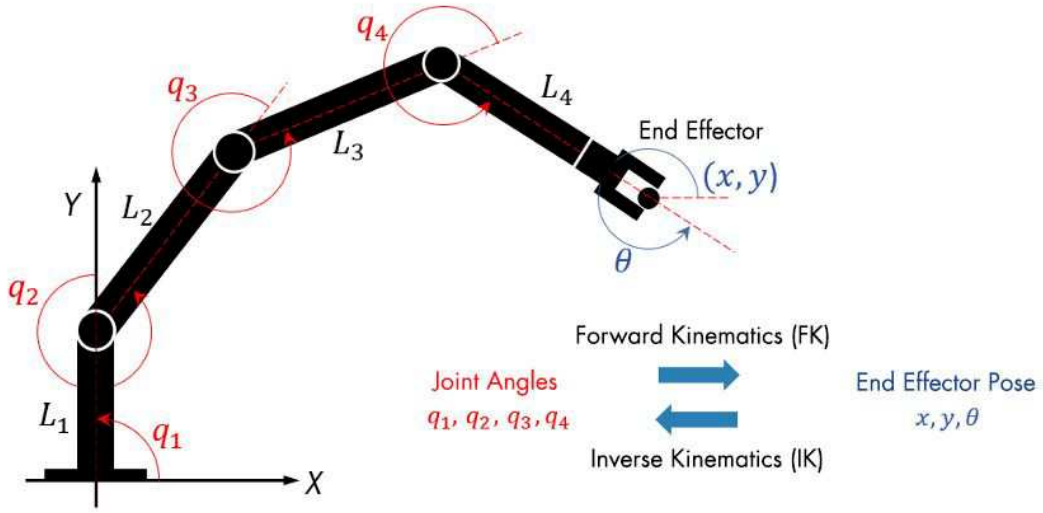


Slika 3.2: Nacrt robotske ruke modela ABB IRB 120 sa šest stupnjeva slobode. Slika preuzeta iz [21].

Koristeći takav ili kompleksniji model unutar virtualnog prostora, moguće je ostvariti zadovoljavajuću razinu prilagodljivosti i preciznosti pokreta za izvođenje rezanja, međutim, to sa sobom nosi i brojne implementacijske izazove. Kretanje oštrice između dvije točke po kojima se izvodi rez svodi se na problem inverzne kinematike.

Problem kretanja oštrice

Inverzna kinematika (engl. *Inverse kinematics*) je proces određivanja potrebnih kutova zglobova robotske ruke kako bi se alat (u ovom slučaju oštrica) postavio u željenu poziciju i orijentaciju. Pri tome se koristi niz jednadžbi koje povezuju željenu poziciju oštrice s kutovima zglobova. Slika 3.3 prikazuje razliku između inverzne kinematike i direktne kinematike, gdje je vidljivo kako se inverzna kinematika koristi za izračunavanje potrebnih kutova zglobova za postizanje određene pozicije i orijentacije krajnjeg alata. Suprotno tome, **direktna kinematika** (engl. *Forward kinematics*) izračunava poziciju i orijentaciju krajnjeg alata na osnovu zadanih kutova zglobova. Problem inverzne kinematike znatno je složeniji od problema direktne kinematike [22].



Slika 3.3: Shematski prikaz razlike koncepta direktne i inverzne kinematike. Slika preuzeta iz [23].

Mogući postupak pomicanja oštrice započinje određivanjem željene pozicije i orijentacije oštrice u virtualnom radnom prostoru. Zatim se robotska ruka može modelirati s pomoću **Denavit-Hartenbergove notacije** (engl. *Denavit–Hartenberg parameters*), koja definira svaku vezu i zglob parametrima kao što su duljina veze, kut između veza i kut zgloba [24]. Nakon toga primjenjuju se trigonometrijske jednadžbe i algebra za izračunavanje kutova zglobova koji oštricu dovode u željenu poziciju.

Na posljetku, budući da inverzna kinematika često može imati više rješenja, potrebno je provjeriti koje rješenje najbolje odgovara fizičkim ograničenjima robotske ruke i specifičnostima zadatka [25].

Kretanje oštrice jednostavnog sustava

Implementacija kretanja oštrice unutar virtualnog prostora po željenoj trajektoriji, koristeći inverznu kinematiku, nije trivijalna. S obzirom na to da ishodi ovog rada uključuju izradu jednostavnog sustava, kretanje robotske ruke unutar prostora je ograničeno i ne uključuje korištenje inverzne kinematike za pomicanje zglobova, kao što je to nužno za kompleksni sustav (Poglavlje 3.2). Robotska ruka u jednostavnoj simulaciji ne prilagođava kuteve svojih zglobova, već samo kut oštrice, po jednoj osi, koja je pričvršćena za vrh ruke.

Takva implementacija ne omogućava rezanje putem krivulje, ali i dalje podržava izvođenje jednog linearnog reza, pri čemu se objekt razdvaja na dijelove različitih volumena. Rotaciju oštrice prema x-osi moguće je mijenjati nasumično ili prema izračunima sustava za optimizaciju preciznosti. Ovakvo kretanje oštrice omogućava bolje razumijevanje korištenog algoritma umjetne inteligencije i jasnije rezultate, jer nije potrebno upravljati s više od jednog parametra (rotacija oštrice).

Konačno, izostanak inverzne kinematike u kretanju robotske ruke jednostavnog sustava pridonosi određenoj razini trivijalnosti, što je prihvatljivo s obzirom na to da je riječ samo o prototipu. Za izradu kompleksnog sustava predlaže se korištenje složenijih koncepata kako bi se ostvarila veća sloboda kretanja, a time i veće mogućnosti prilagodljivosti i raspona rezanja. Kako bi sustav bio iskoristiv i efektivan za konkretnije industrijske primjene, potrebno je ostvariti model robotske ruke s više stupnjeva slobode.

3.1.2. Rezanje objekta

Predviđeno je da se rezanje objekta izvodi jednim rezom. Nakon dostizanja unaprijed izračunate rotacije oštrice, izvodi se animacija spuštanja oštrice nad objektom, čime nastaju dva nova objekta s udjelima volumena izvornog objekta. Proces rezanja može se podijeliti na sljedeće korake:

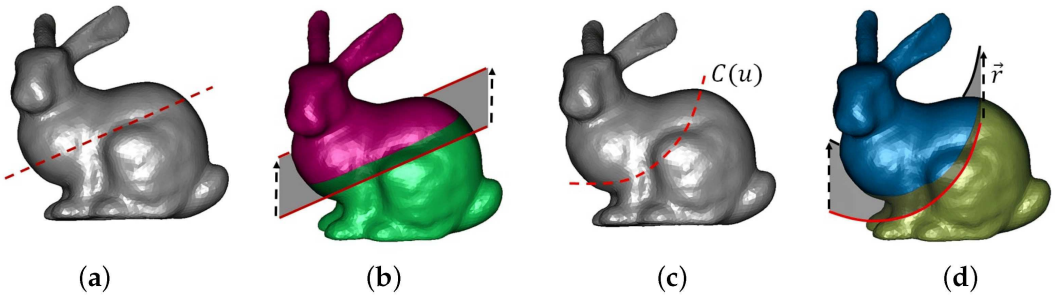
1. **Definiranje ravnine rezanja:**
   - Ova ravnina se određuje s pomoću zrake emitirane iz objekta za rezanje (oštrice) prema objektu koji se reže. Točka presjeka zrake i objekta određuje jednu točku na ravnini, dok smjer zrake definira normalu ravnine. Ravnina se koristi za određivanje strane na kojoj se nalaze pojedinačni trokuti i vrhovi mreže rezanog objekta (engl. *mesh*).

2. **Klasifikacija trokuta:**
   - Svi trokuti mreže se klasificiraju na temelju njihovog položaja u odnosu na ravninu rezanja, pri čemu mogu biti potpuno s jedne strane ravnine, potpuno s druge strane ravnine ili se mogu sjeći s ravninom.

3. **Podjela trokuta:**
   - Trokuti koji se sijeku s ravninom moraju biti podijeljeni dodavanjem novih vrhova na mjestima gdje bridovi trokuta sijeku ravninu rezanja. Svaki izvorni trokut koji se siječe s ravninom dijeli se u dva nova trokuta.

4. **Generiranje novih mreža:**
   - Nakon što su svi trokuti klasificirani i podijeljeni, generira se po jedna nova mreža za svaki dio objekta s obje strane ravnine. Nastaju dva nova objekta s novim skupovima trokuta i vrhova.

Kompleksni sustav zahtijeva drugačiji pristup koji omogućuje rezanje u krivuljama. Takav pristup može uključivati korištenje presjeka između nekoliko uzastopnih rezova kako bi se stvorila ravnina rezanja prije odvajanja trokuta. Međutim, za široku primjenu unutar virtualnog okruženja potrebno je implementirati crtanje ravnine rezanja pokretom ruke unutar virtualnog prostora, što predstavlja složen problem. Slika 3.4 prikazuje razliku između rezultata rezanja objekta ravnom linijom (korišteno u jednostavnom sustavu) te slobodnom krivuljom (predlaže se za kompleksni sustav).



Slika 3.4: Razlika između rezultata rezanja objekta ravnom linijom (a i b) te slobodnom krivuljom (c i d). Slika preuzeta iz [26].

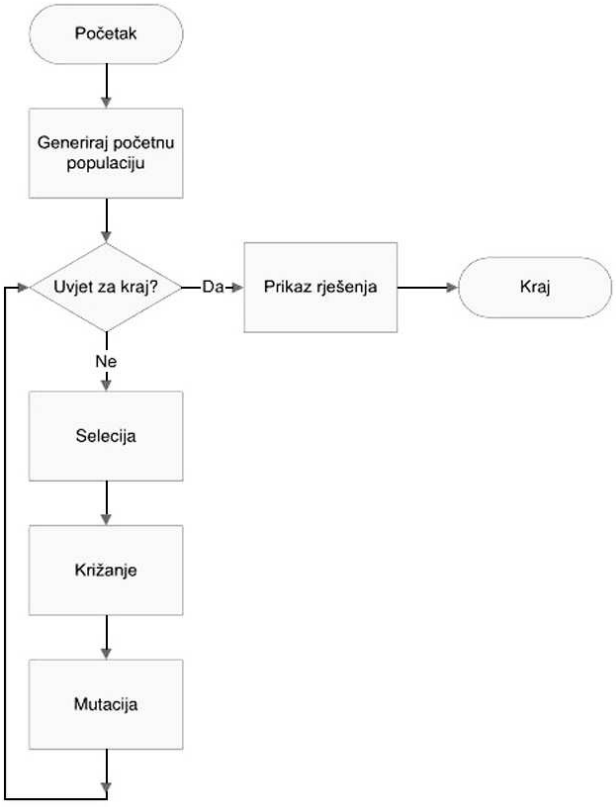
3.1.3. Optimizacija preciznosti rezanja i genetički algoritam

Svaki ostvareni rez opisan je određenom razinom preciznosti koja ovisi o volumenu novonastalih objekata. Nakon uspješnog završetka rezanja, koristeći svojstva

mreže te pojedinačne trokute od kojih se mreža sastoji, izračunava se volumen odrezanih dijelova i uspoređuje s izvornim ukupnim volumenom prije rezanja. Rez kojim su nastala dva dijela jednakog ili približnog volumena smatra se preciznim.

Kao što je spomenuto u Poglavlju 3.1, koristeći rezove robota te rezove nastale korisničkom intervencijom, provodi se postupak optimizacije učinkovitosti rada sustava, odnosno postupak optimizacije preciznosti rezanja. Kako bi ostvarili učenje i konvergenciju prema željenoj razini preciznosti, potrebno je implementirati jednostavni sustav umjetne inteligencije, za što je u ovom slučaju odabran **genetički algoritam** (engl. *Genetic Algorithm* - ***GA***).

Genetički algoritam je optimizacijski algoritam inspiriran procesom genetske evolucije. Koristi populaciju rješenja, koje se iterativno poboljšavaju kroz generacije kako bi se pronašlo optimalno rješenje za složene probleme. Proces se sastoji od operacija **selekcije** (engl. *selection*), **križanja** (engl. *crossover*) i **mutacije** (engl. *mutation*) [27]. Ako se koristi samo mutacija, algoritam je vrlo spor, dok križanje značajno ubrzava algoritam [28].



Slika 3.5: Slijed izvođenja genetičkog algoritma unutar simulacije. Slika preuzeta iz [29].

Općenito, selekcijom se odabiru najbolja rješenja iz trenutne populacije, čime se osigurava da bolje jedinke imaju veću šansu za reprodukciju. Križanjem se kombiniraju dijelovi dvaju roditelja kako bi se stvorili potomci, omogućujući razmjenu genetskog materijala i stvaranje novih rješenja. Mutacija uvodi nasumične promjene u genetski materijal potomaka, čime se povećava raznolikost populacije i izbjegava zaglavljivanje u **lokalnom optimumu**. Konkretno, početna populacija sastoji se od 3 nasumična reza robotske ruke. Procesom selekcije, iz trenutne populacije (koja se u kasnijim koracima sastoji od 6 rezova), odabiru se 3 reza klasificirana najvećom razinom preciznosti i ulaze u proces mutacije s 3 reza nastala korisničkim unosom. S obzirom na to da svaki rez sadrži i informaciju o kutu oštrice pod kojim je nastao, u procesu križanja, nakon sparivanja robotskih i korisničkih rezova na temelju preciznosti, izvodi se **linearna interpolacija** (engl. *Linear interpolation*) kuteva. Linearna interpolacija može se koristiti za procjenu vrijednosti funkcije između dvije poznate točke. Točnije, dana je formulom prema kojoj se procjenjuje vrijednost y za zadanu vrijednost x između dviju točaka (x_1, y_1) i (x_2, y_2):

$$y = y_1 + \frac{(x - x_1)(y_2 - y_1)}{x_2 - x_1}$$

Interpolacijom vrijednosti kuteva oštrice sparenih jedinki nastaju nove vrijednosti kuteva koje su bliže kutevima koje je koristio korisnik za izvođenje "dobrih" rezova. Korištenjem linearne interpolacije želimo ostvariti postupnu konvergenciju prema ispravnijim rješenjima kroz generacije, bez prenaučenosti i zaglavljivanja u lokalnom optimumu. Kako bi dodatno istražili prostor mogućih rješenja i potencijalno ostvarili bržu konvergenciju prema preciznijim ishodima, na interpolirane vrijednosti kuteva nadodaju se vrijednosti izračunate nasumično. Time se dobivaju još 3 kuta slična onima nastalim križanjem, a takav proces naziva se mutacijom i završava s populacijom od 6 potencijalno boljih rezova koje je potrebno evaluirati u sljedećoj iteraciji. Populacija korisničkih rezova se nakon svake mutacije prazni.

Potrebno je napomenuti da o strategiji selekcije, križanja i mutacije ovisi brzina konvergencije prema boljim rješenjima, što je detaljnije razmotreno u Poglavlju 5. Genetički algoritmi su posebno korisni za optimizacijske probleme s velikim i složenim prostorima rješenja, gdje tradicionalne metode mogu biti neučinkovite [28]. U ovom slučaju, postoji iznimno velik raspon mogućih kuteva oštrice i rezultata ostvarenih tim kutevima, pa se ovakav algoritam smatra prikladnim.

3.1.4. Unos unutar virtualnog okruženja

S obzirom na jednostavnost simulacije te ograničenje pokreta korisnika i robotske ruke, integracija sustava unutar virtualnog okruženja je trivijalna. Potrebno je unutar projekta uključiti podršku za VR te, umjesto pritiskom tipke miša, upravljati ekvivalentnom korisničkom akcijom s pomoću pripadajuće VR opreme. Potrebno je napomenuti kako ideja upravljanja kompleksnom verzijom simulacije unutar virtualnog radnog prostora predstavlja značajniju kompleksnost zbog prethodno spomenutog problema stvaranja trajektorije za rezanje objekta (Poglavlje 3.1.2). U tom slučaju, prebacivanje simulacije u virtualni prostor zahtijeva promjene implementacije koje su u potpunosti prilagođene upravljanju VR opremom.

3.2. Ideja kompleksnog sustava

Unutar ovog potpoglavlja ponovno se ističu značajke kompleksnog sustava za rezanje objekata po kojima se on razlikuje od jednostavnog sustava. Razlike pronalazimo u arhitekturi, funkcionalnostima i implementacijskim izazovima. Dok jednostavni sustav služi kao demonstracijski model za osnovne principe i optimizacijske algoritme, kompleksni sustav namijenjen je za napredne industrijske primjene gdje su potrebne veća preciznost i prilagodljivost.

Kompleksni sustav koristi napredniji model robotske ruke s više stupnjeva slobode, što omogućuje složenije pokrete i preciznije rezanje objekata. Dok jednostavni sustav koristi samo jedan kut rotacije oštrice, kompleksni sustav integrira inverznu kinematiku za kontrolu svih zglobova robotske ruke. To omogućuje izvođenje rezova u tri dimenzije i prilagodbu kretanja oštrice na temelju zadane trajektorije. Inverzna kinematika, kao što je objašnjeno, omogućuje izračunavanje potrebnih kutova zglobova za postizanje željene pozicije i orijentacije oštrice. Ovakva arhitektura povećava fleksibilnost sustava i omogućuje rezanje složenih oblika i krivulja, čime se značajno povećava njegova primjenjivost u industriji. Dok jednostavni sustav dijeli objekte linearnim rezom, kompleksni sustav koristi napredne metode za definiranje i praćenje složenih ravnina rezanja. Ovo uključuje generiranje više uzastopnih ravnina i njihovo spajanje kako bi se formirala trajektorija rezanja koja odgovara željenom obliku. Optimizacija preciznosti rezanja u kompleksnom sustavu može koristiti drugačije i složenije algoritme umjetne inteligencije. S pomoću odgovarajućih algoritama, umjesto jednostavne rotacije oštrice, optimiziraju se kutovi svih zglobova robotske ruke kako bi se postigla maksimalna preciznost u složenim rezovima. Trenutno se ne predlažu konkretne

metode optimizacije, već je područje potrebno istražiti ovisno o arhitekturi sustava. Kompleksni sustav potencijalno zahtijeva sofisticiraniju metodu evaluacije kvalitete rezova, koja nadilazi jednostavno korištenje opisnih oznaka i razlike u volumenu.

Integracija kompleksnog sustava u virtualno okruženje zahtijeva napredne metode upravljanja putem VR opreme. Za razliku od jednostavnog sustava, kompleksni sustav zahtijeva detaljnije praćenje pokreta korisnika i preciznu kontrolu robotske ruke. Ovo uključuje implementaciju naprednih algoritama za praćenje pokreta i optimizaciju korisničkog sučelja kako bi se osiguralo intuitivno i efikasno upravljanje.

4. Programsko rješenje

Unutar ovog poglavlja detaljno je opisana programska izvedba jednostavnog sustava za rezanje objekata kroz korištene tehnologije i alate, te komentirane isječke programskog koda kojima su objašnjeni pojedini elementi sustava. Poglavljem su obuhvaćeni specifični aspekti implementacije, uključujući detalje o razvojnom okruženju, korištenim programskim jezicima, bibliotekama i algoritmima te integraciji unutar virtualnog okruženja.

4.1. Korištene tehnologije i alati

Za izradu jednostavnog sustava za rezanje objekata korišteno je razvojno okruženje **Unity** u inačici 2022.3.22f1. Unity je jedno od najpopularnijih i najfleksibilnijih razvojnih okruženja za izradu interaktivnih 3D i 2D sadržaja, kao što su igre, simulacije i drugi virtualni sadržaji. Glavni programski jezik za skriptiranje u Unityju je C#, koji omogućuje manipulaciju objektima i upravljanje događajima unutar aplikacije.

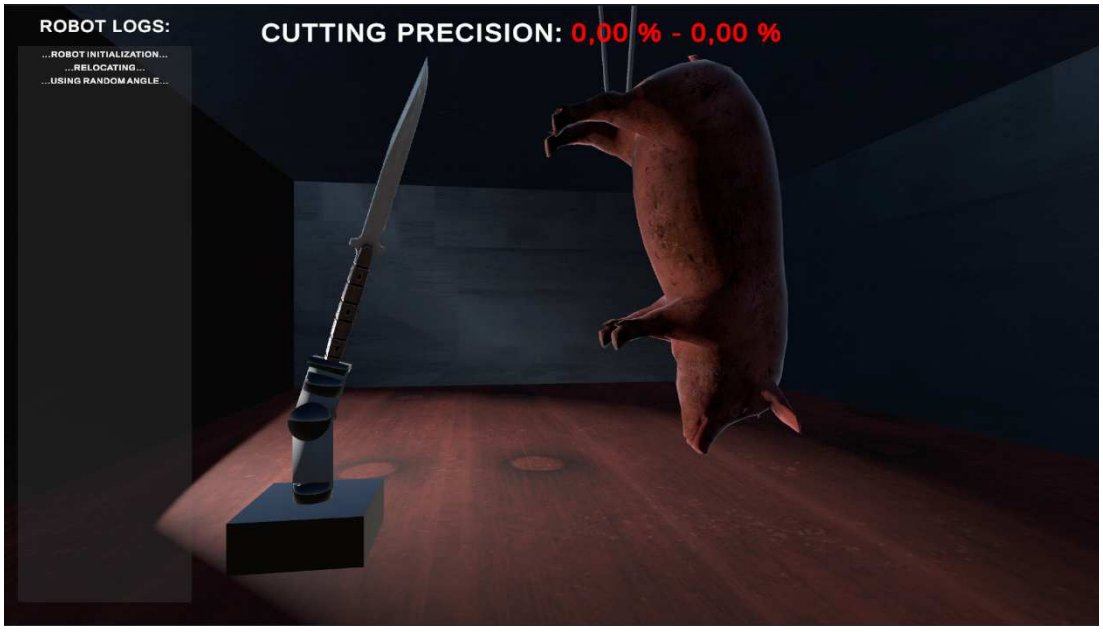
Unity se temelji na principu **komponentno orijentiranog razvoja**, gdje su svi objekti na sceni definirani kao entiteti s pridruženim komponentama koje određuju njihovo ponašanje i izgled. Osnovni elementi takve aplikacije su scene, objekti i njihove komponente. Scene predstavljaju različite faze ili razine izvođenja aplikacije, objekti su entiteti unutar scene, a komponente su moduli koji dodaju specifične funkcionalnosti tim objektima. Kroz korištenje programskih skripti, programeri mogu definirati ponašanje objekata, reagirati na korisničke unose, upravljati animacijama i obavljati mnoge druge zadatke.

Osim toga, Unity nudi podršku za razvoj digitalnih dvojnika, omogućujući stvaranje virtualnih kopija fizičkih objekata ili sustava. Omogućeno je upravljanje podatcima u stvarnom vremenu, vizualizacija podataka i simulacija različitih scenarija. Za izradu jednostavnog sustava ne koristi se tehnologija digitalnih dvojnika, već samo vizualizacija koncepta jednostavne simulacije unutar scene.

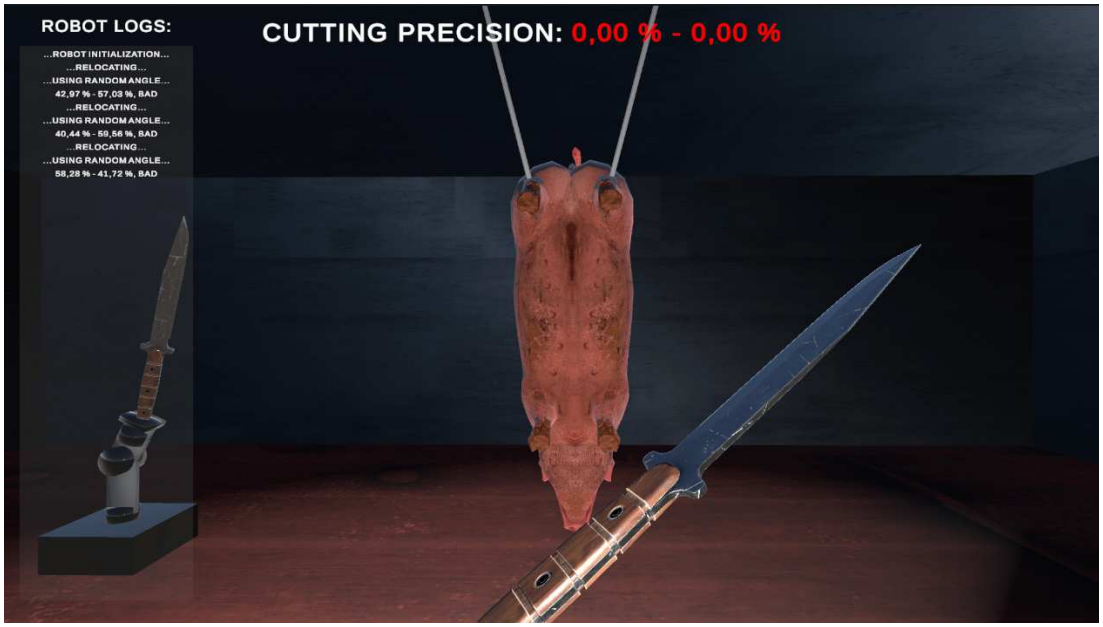
4.2. Postavljanje scene i struktura projekta

Unutar simulacije koristi se samo jedna scena s dva različita pogleda, iz perspektive korisnika i robotske ruke. Svaki pogled definiran je pripadajućim objektima koje je potrebno prikazati ili sakriti, ovisno o tome čija se perspektiva trenutno prikazuje. Izmjena objekata i kamera izvršava se na prijelazima između pojedinih stanja simulacije (Slika 3.1). Slike 4.1 i 4.2 prikazuju izgled scene s pripadajućim objektima u različitim fazama izvršavanja simulacije.

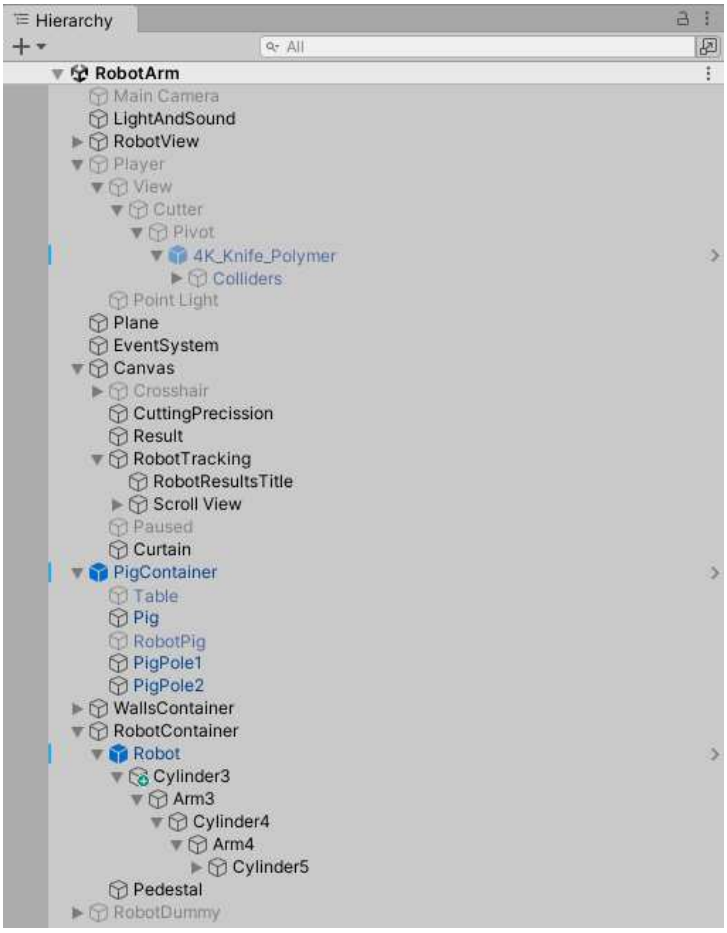
Objekti koji čine scenu uključuju: oštricu kojom upravlja korisnik, improviziranu robotsku ruku s oštricom kao krajnjim alatom, objekt za presijecanje koji je pričvršćen za strop, te elemente korisničkog sučelja za prikazivanje informacija o uspješnosti rezova i napretku simulacije kroz faze. Osim toga, hijerarhija scene sadrži i objekte za osvjetljenje, izvor zvuka, plohe koje sačinjavaju prostoriju te pomoćne objekte poput roditeljskih kontejnera, zglobova robota, animiranih točaka pivotiranja i slično. Slika 4.3 prikazuje proširenu hijerarhiju objekata scene.



Slika 4.1: Raspored objekata na sceni za vrijeme rada robotske ruke.



Slika 4.2: Raspored objekata na sceni za vrijeme korisničke intervencije.



Slika 4.3: Proširena hijerarhija objekata koji čine scenu simulacije.

4.3. Funkcionalnost rezanja

Nakon postavljanja svih objekata na njihove pozicije unutar scene, potrebno je ostvariti osnovnu funkcionalnost rezanja. Rezanje je u osnovi ostvareno korištenjem i nadogradnjom elemenata iz biblioteke ***Triangle Separator***, dostupne na *Unity Asset Storeu*. Korištena biblioteka omogućuje rezanje, razbijanje, sekcioniranje i ekstrakciju mreža trokuta [30].

4.3.1. Strukture podataka

Biblioteka je pisana koristeći programski jezik C# i sadrži nekoliko klasa unutar kojih su podaci strukturirani te sučelja koja definiraju metode za rad s podacima. Unutar ovog i idućeg poglavlja navode se samo ključne klase i metode uz komentare o njihovoj namjeni. Klasa *Triangle* definira jednostavan trokut preko svojih vrhova i, dodatno, indeksa podmreže tog trokuta u nekoj drugoj mreži. Za strukturiranje trokuta nakon rezanja služi skripta *Chunk*, koja sadrži listu pripadajućih trokuta te listu točaka koje čine rubove.

Skripta *ITriangleSeparator* sadrži sučelje koje korisnicima omogućuje prilagodbu razdvajanja mreža na trokute s pomoću parametara kao što su konveksnost, broj niti, dubinsko skeniranje i način razdvajanja. Također pruža metode za provjeru vrhova i izračunavanje novih vrhova za glatke prijelaze između podskupova. Unutar skripte *MeshCalc* pružen je niz pomoćnih metoda za obradu mreža. Metode uključuju normalizaciju vrijednosti, ekstrakciju i mapiranje indeksa, dodavanje i uklanjanje raspona elemenata iz lista i povezanih lista te zamjenu elemenata unutar lista. Osigurava osnovne alate za manipulaciju podacima o mrežama i olakšava složenije operacije razdvajanja i obrade mreža. Skripta *MeshChunkExtractor* omogućuje generiranje i zatvaranje dijelova mreža koristeći podatke izračunate s pomoću klase *TMeshTriangleSeparator*. Upravlja vađenjem i mapiranjem trokuta i rubova iz originalne mreže te omogućuje kreiranje novih mreža iz tih dijelova. Također, pruža metodu za zatvaranje mreže koristeći rubove, što uključuje generiranje novih trokuta za zatvaranje rupa unutar mreže.

Klasa *TMeshTriangleSeparator* pruža krajnju funkcionalnost podjele mreže. Može izračunati fragmente za konveksne i konkavne mreže, koristeći odgovarajuće podatke o mreži. Korištenjem povratnog poziva, svi izračuni mogu se prebaciti na radnu dretvu, a dodatno se koristi višedretveno izvođenje u izračunima konkavnih mreža. Također, omogućuje otkrivanje dvostrukih rubova (vrhova) za preciznije rezultate. Osnovna metoda iz ove klase, koja se poziva prilikom rezanja, naziva se ***divideMesh*** (Programski

isječak 4.4). Ovom se metodom mreža s fragmentima i rubovima dijeli u dvije podskupine. Argumenti metode uključuju ulaznu mrežu, listu fragmenata i izlaznu mrežu koja sadrži podatke indeksirane fragmentima i rubovima. Osim toga, metoda čeka na slobodnu oznaku za sinkronizaciju višedretvenog pristupa.

```
        public bool divideMesh(Mesh m, out List<Chunk>[] chunks, out Mesh mesh)
        {
            if (m == null)
                throw new ArgumentNullException("m", "Mreza ne smije biti null");
            QueueHolder.WaitOne();
            _triangleSeparator = _latestTriangleSeparator;

            _mesh = m;
            mesh = _mesh;
            try
            {
                chunks = null;
                if (rootSeparation())
                {
                    mesh = _mesh;
                    if (!_convex) startMeshDivision(_vertices.Length);
                    loadUp(out chunks);
                    return true;
                }
                mesh = null;
            }
            catch (Exception e)
            {
                UnityEngine.Debug.LogException(e);
                chunks = null;
                mesh = null;
            }
            finally
            {
                cleanUp();
                QueueHolder.Set();
            }
            return false;
        }

```

Programski isječak 4.4: Temeljna metoda *divideMesh* korištena za razdvajanje mreže trokuta.

Za pohranu podataka o populaciji rezova, pojedinačnim rezovima te rezultatima njihova izvođenja, izrađene su klase *CutterPopulation*, *Cutter* i *CutResult* s pripadajućim pomoćnim enumeracijama (Programski isječak 4.5).

```
    public enum CutQuality
    {
        BAD = 0,
        MEDIUM = 1,
        GOOD = 2
    }

    public enum GameState
    {
        ROBOT_INITIALIZATION,
        PLAYER_INTERVENTION,
        STABLE
    }

    public class CutResult
    {
        public float volumeChunk1; // Postotak volumena prvog dijela nakon rezanja
        public float volumeChunk2; // Postotak volumena drugog dijela nakon rezanja
        public CutQuality quality; // Kvaliteta rezultata rezanja
        public bool needsEvaluation = false; // Potreba za evaluacijom reza
    }

    public class Cutter
    {
        public float bladeAngle; // Kut ostrice
        public CutResult cutResult; // Rezultat izvodjenja reza
    }

    public class CutterPopulation
    {
        public List<Cutter> cutters; // Lista rezova u populaciji
    }

```

Programski isječak 4.5: Klase za pohranu podataka o rezovima, populaciji rezova i rezultatima rezanja.

Klasa *CutResult* pohranjuje podatke o volumenu dijelova nakon rezanja, kvaliteti reza i potrebi za evaluacijom, dok klasa *Cutter* sadrži kut oštrice i rezultat reza. Klasa *CutterPopulation* održava listu svih rezova u populaciji, a enumeracije *CutQuality* i *GameState* definiraju opisnu ocjenu reza i stanje igre.

4.3.2. Izvođenje reza

Programska skripta zaslužna za kontrolu tijeka izvođenja reza naziva se ***SimpleChopHandler***. S pomoću nje se, na temelju pozicije oštrice, određuje ravnina rezanja te se pozivaju spomenute metode za smještanje podataka u odgovarajuće strukture i upravljanje tim strukturama. Izvorna skripta je promijenjena tako da uključuje razlike u logici robotskog i korisničkog reza, implementira dodatne metode za detekciju sudara, izračun volumena novonastalih objekata i rotaciju oštrice robota te upravlja kontrolnim varijablama za stanja simulacije i podacima nakon evaluacije preciznosti.

Unutar metode *Update* provjerava se nekoliko uvjeta koji dozvoljavaju reakciju na korisnički unos ili pokretanje izvođenja reza od strane robotske ruke. Uvjeti služe isključivo kako bi se osiguralo da su se objekti između promjene pogleda scene uspješno izmijenili te da je prethodni rez u potpunosti izvršen. Ako je akcija robotske ruke dozvoljena, pozivaju se metode *RotateObjectCoroutine* i *RotateRobotKnife* (Programski isječci 4.6 i 4.7) koje omogućuju prilagodbu kuta oštrice prije izvođenja reza.

```
1   private IEnumerator RotateObjectCoroutine(Transform objectToRotate, float
    rotationSpeed, float targetAngle)
2   {
3       Quaternion startRotation = objectToRotate.rotation;
4       // Smanjivanje kuta zbog inicijalne rotacije
5       float defaultCutterRotation = 30f;
6
7       Quaternion endRotation = Quaternion.Euler(targetAngle - defaultCutterRotation
    , 0, 0);
8       float elapsedTime = 0f;
9
10      while (elapsedTime < 1f)
11      {
12          objectToRotate.rotation = Quaternion.Slerp(startRotation, endRotation,
    elapsedTime);
13          elapsedTime += Time.deltaTime * rotationSpeed;
14          yield return null;
15      }
16
17      // Osiguraj da je zeljena rotacija u potpunosti ispunjena
18      objectToRotate.rotation = endRotation;
19  }
20
```

Programski isječak 4.6: Metoda *RotateObjectCoroutine* kojom se ostvaruje rotacija oštrice do zadanog kuta.

```
    private void RotateRobotKnife()
    {
        if (PrecisionDisplay.recalculateCuttingAngle)
        {
            PrecisionDisplay.recalculateCuttingAngle = false;

            float rotationAngle;
            float rotationSpeed = 2f;

            // Ako postoje kutevi koje je potrebno evaluirati koristi njih
            if (SceneManagement.anglesToValidate.Count > 0)
            {
                rotationAngle = SceneManagement.anglesToValidate[0];
                SceneManagement.anglesToValidate.RemoveAt(0);
            }
            else
            {
                float minAngle = -50f;
                float maxAngle = 90f;

                // U protivnom generiraj nasumicni kut unutar specificiranog raspona
                rotationAngle = Random.Range(minAngle, maxAngle);
            }

            // Zapocni korutinu rotacije
            StartCoroutine(RotateObjectCoroutine(robotPivot.gameObject.transform,
    rotationSpeed, rotationAngle));
        }
    }

```

Programski isječak 4.7: Metoda *RotateRobotKnife* kojom se određuje sljedeći kut rotacije.

U metodi *RotateRobotKnife* provjerava se postoje li kutevi rezanja koje je potrebno validirati, i ako postoje, uzima se prvi kut iz liste. Ako nema kuteva za validaciju, generira se nasumičan kut unutar unaprijed definiranog raspona. Metoda *RotateObjectCoroutine* vrši samu rotaciju objekta do zadanog kuta koristeći **sferičnu linearnu interpolaciju** kako bi se postigla glatka tranzicija u vremenu.

Nakon rotacije oštrice robota ili nakon pritiska tipke miša, poziva se metoda ***CutObject*** koja sadrži cijelu logiku izvođenja reza po reznoj ravnini, ažuriranja varijabli koje kontroliraju stanje simulacije, te stvaranja zapisa o preciznosti i uspješnosti reza. Programski isječci 4.8 i 4.10 prikazuju neke od važnijih dijelova metode za razdvajanje mreže trokuta te stvaranje struktura podataka sa zapisima o uspješnosti.

```
public void CutObject(bool robotCutting, bool mousePressed)
{
    ...
    _toCutter = combinedTransformObject.transform.worldToLocalMatrix * transform.localToWorldMatrix;
    RaycastHit hit;
    Physics.Raycast(blade.transform.position, blade.transform.right, out hit, 100, slicableLayers);
    if (hit.collider != GetComponent<Collider>()) return;
    _slicer.setTriangleSeparator(this);
    // Dijelovi formirani od susjednih trokuta (nisu medjusobno povezani)
    List<Chunk>[] chunks;
    Mesh copy = null;
    if (_slicer.divideMesh(_myMesh, out chunks, out copy)) {
        // Stvaranje dodatne podmreze
        if (!(copy.subMeshCount > 1)) {
            copy.subMeshCount = copy.subMeshCount + 1;
            copy.SetTriangles(new int[] { 0, 0, 0 }, copy.subMeshCount - 1);
        }
        MeshChunkExtractor extractor = MeshChunkExtractor.CreateInstance(copy);
        for (int w = 0; w < 2; w++)
            foreach (Chunk chunk in chunks[w]) {
                if (chunk.chunk.Count <= minChunkSize) continue;
                Mesh cm;
                Dictionary<int, int> reindex = extractor.extractChunk(chunk, out cm);
                float chunkVolume = VolumeOfMesh(cm);
                chunkVolumes[w] = (chunkVolume / originalVolume) * 100f;
                if (capMesh) {
                    foreach (List<int> ed in chunk.edges) {
                        try {
                            List<int> edReindexed = MeshCalc.translateIndices(ed, reindex);
                            MeshChunkExtractor.capMesh(cm, edReindexed, 1);
                        }
                        ...
                    }
                }
                GameObject go = Instantiate(gameObject, transform.position, transform.rotation) as GameObject;
                go.GetComponentInChildren<MeshFilter>().sharedMesh = cm;
                Collider col = go.GetComponent<Collider>();
                ...
                // Dodaj Rigidbody samo ako objekt nije pricvrscen za strop
                ...
            }
        ...

```

Programski isječak 4.8: Dijelovi metode *CutObject* koji sadrže logiku razdvajanja mreže trokuta.

Programski isječak 4.8 prikazuje neke od ključnih dijelova metode *CutObject*. Metoda započinje provjerom sudara oštrice s objektom s pomoću ***Raycast*** tehnologije. Ako sudar postoji, mreža objekta se dijeli na dijelove, pri čemu se stvaraju novi mrežni segmenti. Za svaki segment provjerava se veličina te se računaju volumeni segmenata. Ako je potrebno, rubovi segmenta se zatvaraju kako bi se formirala cjelovita mreža. Na kraju, stvara se novi objekt s izrezanim segmentom mreže, a na njega se može dodati komponenta *Rigidbody* ako objekt nije pričvršćen za strop.

Programski isječak 4.9 prikazuje dvije metode koje služe za izračunavanje volumena trokuta s pomoću zadanih vrhova i ukupnog volumena mreže objekta iteriranjem kroz sve trokute u mreži.

```
        // Izracun volumena trokuta
    float SignedVolumeOfTriangle(Vector3 p1, Vector3 p2, Vector3 p3) {
        float v321 = p3.x * p2.y * p1.z;
        float v231 = p2.x * p3.y * p1.z;
        float v312 = p3.x * p1.y * p2.z;
        float v132 = p1.x * p3.y * p2.z;
        float v213 = p2.x * p1.y * p3.z;
        float v123 = p1.x * p2.y * p3.z;
        return (1.0f / 6.0f) * (-v321 + v231 + v312 - v132 - v213 + v123);
    }
    // Izracun volumena mreze
    float VolumeOfMesh(Mesh mesh) {
        float volume = 0;
        Vector3[] vertices = mesh.vertices;
        int[] triangles = mesh.triangles;
        for (int i = 0; i < mesh.triangles.Length; i += 3) {
            Vector3 p1 = vertices[triangles[i + 0]];
            Vector3 p2 = vertices[triangles[i + 1]];
            Vector3 p3 = vertices[triangles[i + 2]];
            volume += SignedVolumeOfTriangle(p1, p2, p3);
        }
        return Mathf.Abs(volume);
    }

```

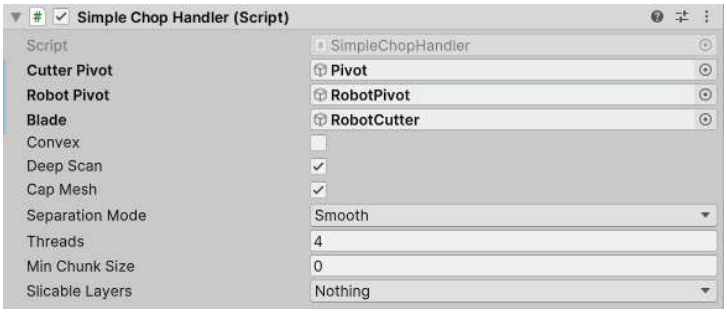
Programski isječak 4.9: Metode iz skripte *SimpleChopHandler* koje služe za izračunavanje volumena mreže i pojedinačnih trokuta.

Programski isječak 4.10 prikazuje dijelove metode *CutObject* koji smještaju podatke o rezu u odgovarajuće strukture i stvaraju zapise o napretku. Nakon što se izračunaju volumeni segmenata nastalih rezanjem, postotci se prikazuju na zaslonu te se inicijaliziraju varijable za resetiranje scene. Zatim se kreira objekt koji pohranjuje podatke o volumenu segmenata i kvaliteti reza. Na kraju, rezultati se formatiraju kao tekst i dodaju u popis rezultata robota koji se prikazuje korisniku.

```
public void CutObject(bool robotCutting, bool mousePressed)
{
        ...
        canvas.gameObject.GetComponent<PrecisionDisplay>().SetChunkVolumes(
chunkVolumes[0], chunkVolumes[1]);
        SceneManagement.resetSwitch = true;
        SceneManagement.currentlyResetting = true;
        KeyValuePair<float, float> pair = new KeyValuePair<float, float>(
chunkVolumes[0], chunkVolumes[1]);
        CutResult cutRes = new CutResult {
            volumeChunk1 = chunkVolumes[0],
            volumeChunk2 = chunkVolumes[1],
            quality = PrecisionDisplay.GetGrade(chunkVolumes[0], chunkVolumes[1])
,
        };
        if (robotCutting == true) {
            // Rukovanje populacijom
            float roundedXRotation =(float)System.Math.Round(UnwrapAngle(
robotPivot.transform.localEulerAngles.x), 3);
            if (SceneManagement.gameState == GameState.STABLE) {
                // Iteriraj kroz svakog rezaca u populaciji robota
                foreach (Cutter cutter in SceneManagement.robotPopulation.cutters
) {
                    if (cutter.bladeAngle == roundedXRotation && cutter.cutResult
 == null) {
                        cutter.cutResult = cutRes;
                        cutter.cutResult.needsEvaluation = true;
                        break;
                    }
                }
            }
            else {
                Cutter currentCutter = new Cutter {
                    bladeAngle = roundedXRotation,
                    cutResult = cutRes,
                };
                SceneManagement.robotPopulation.AddCutter(currentCutter);
            }
            // Rukovanje tekstom
            string resultToAdd = chunkVolumes[0].ToString("0.00") + " %" + " - "
+ chunkVolumes[1].ToString("0.00") + " %, ";
            resultToAdd += PrecisionDisplay.GetGrade(chunkVolumes[0],
chunkVolumes[1]);
            PrecisionDisplay.robotResultsTextChunks.Add(resultToAdd);
            PrecisionDisplay.robotResultsList.Add(pair);
        }
        ...
}

```

Programski isječak 4.10: Dijelovi metode *CutObject* koji sadrže logiku smještanja podataka u odgovarajuće strukture i stvaranja zapisa o napretku.

Konačno, programska skripta *SimpleChopHandler* je unutar razvojnog okruženja pričvršćena na objekt koji je potrebno rezati. Slika 4.11 prikazuje primjer konfiguracije skripte na objektu kojeg reže robotska ruka.



Slika 4.11: Primjer konfiguracije skripte *SimpleChopHandler* na objektu kojeg reže robotska ruka.

4.4. Upravljanje scenom i implementacija genetičkog algoritma

Unutar skripte ***SceneManagement*** implementirana je logika za upravljanje čitavim tijekom simulacije. Sadrži metode za postavljanje i micanje objekata sa scene pri prijelazu iz pojedinih stanja, provjeru prelaska u sljedeće stanje te metode za upravljanje populacijom rezova. Metode za upravljanje populacijom rezova ujedno su i metode genetičkog algoritma te se s pomoću njih, na način opisan u Poglavlju 3.1.3, optimizira uspješnost izvršavanja simulacije. ***ChangeGameState*** je ključna metoda unutar ove skripte koja provjerava trenutno stanje simulacije, varijable koje omogućuju prijelaze te strukture podataka za upravljanje populacijom rezova.

4.4.1. Napredovanje simulacije po stanjima

Programski isječak 4.12 prikazuje provjeru postojanja samo 3 inicijalna reza u trenutnoj populaciji te se prema tome napreduje u odgovarajuća stanja za punjenje populacije. U protivnom se provjerava ima li u populaciji rezova (kuteva) koje je potrebno evaluirati. Ako su svi rezovi izvedeni i evaluirani, provjerava se je li zadovoljena određena razina uspješnosti sustava, inače simulacija napreduje na evaluaciju preostalih rezova.

```
    public static void ChangeGameState()
    {
        if (robotPopulation.cutters.Count == 3 && gameState == GameState.
    ROBOT_INITIALIZATION) {
            if (gameState == GameState.ROBOT_INITIALIZATION) {
            }
            else if (gameState == GameState.PLAYER_INTERVENTION) {
                gameState = GameState.STABLE;
            }
            exchangeObjects = true;
        }
        if (anglesToValidate.Count == 0 && gameState == GameState.STABLE) {
            bool evaluationNeeded = false;
            // Provjera za cutResult koji je null
            foreach (Cutter cutter in robotPopulation.cutters) {
                if (cutter.cutResult.needsEvaluation) {
                    evaluationNeeded = true;
                    break;  // Izadji iz petlje posto postoji null cutResult
                }
            }
            learningBreak = true;
            if (evaluationNeeded) {
                foreach (Cutter cutter in robotPopulation.cutters) {
                    cutter.cutResult.needsEvaluation = false;
                }
                int goodQualityCount = 0;
                foreach (Cutter cutter in SceneManagement.robotPopulation.cutters) {
                    if (cutter.cutResult != null) {
                        if (cutter.cutResult.quality == CutQuality.GOOD) {
                            goodQualityCount++;
                        }
                    }
                }
                if (goodQualityCount < 3) {
                    gameState = GameState.PLAYER_INTERVENTION;
                    exchangeObjects = true;
                    ...
                }
                else {
                    PrecisionDisplay.AddLogLine("CALIBRATION WAS SUCCESSFUL :)");
                }
            }
            else {
                // Metode genetickog algoritma
                ...

```

Programski isječak 4.12: Dijelovi metode *ChangeGameState* zaslužni za napredovanje po stanjima scene.

4.4.2. Metode genetičkog algoritma

Ako se u populaciji trenutno nalazi 6 rezova, od kojih neki ne sadrže potrebne informacije na temelju kojih bi se mogli evaluirati, potrebno je najprije s pomoću metoda genetičkog algoritma izračunati kuteve pod kojima se ti rezovi trebaju izvesti. Metode selekcije, križanja i mutacije izvode se slijedno, a praćene su čišćenjem trenutne populacije rezova igrača i robota (Programski isječak 4.13). Unutar Programskih isječaka 4.14 i 4.15 prikazana je implementacija metoda genetičkog algoritma te su pruženi komentari koji ih pobliže opisuju.

```
1    public static void ChangeGameState()
2    {
3        ...
4        // SELEKCIJA
5        // Izaberi najbolje rezultate iz populacije i sortiraj korisnicke rezove
6        List<Cutter> chosenRobotCutters = SelectBestCutters(robotPopulation);
7        List<Cutter> chosenPlayerCutters = SortPopulationByQuality(playerPopulation);
8
9        // Ocisti populaciju robota
10       robotPopulation.ClearPopulation();
11
12       // KRIZANJE
13       ExecuteCrossing(chosenRobotCutters, chosenPlayerCutters);
14
15       // MUTACIJA
16       List<Cutter> mutated = ExecuteMutation(robotPopulation.cutters);
17       foreach (Cutter cutter in mutated)
18       {
19           robotPopulation.AddCutter(cutter);
20       }
21
22       playerPopulation.ClearPopulation();
23       foreach (Cutter cutter in robotPopulation.cutters)
24       {
25           anglesToValidate.Add(cutter.bladeAngle);
26       }
27       learningBreak = false;
28   }
29
```

Programski isječak 4.13: Dijelovi metode *ChangeGameState* zaslužni za optimizaciju uspješnosti rada sustava.

```
    public static List<Cutter> ExecuteMutation(List<Cutter> robotCutters)
    {
        List<Cutter> mutatedCutters = new List<Cutter>();
        foreach (Cutter cutter in robotCutters) {
            // Stvaranje kopije
            Cutter mutatedCutter = new Cutter {
                bladeAngle = cutter.bladeAngle,
            };
            // Izracunavanje raspona mutacije na temelju trenutnog kuta ostrice
            float mutationRange = Mathf.Abs(mutatedCutter.bladeAngle) * 0.1f;
            // Moguce je proizvoljno mijenjati mnozitelj (0.1f)
            // Izracun nasumicne vrijednosti za mutaciju
            float mutationValue = UnityEngine.Random.Range(-mutationRange,
    mutationRange);
            // Stvaranje novog kuta pomocu vrijednosti za mutaciju
            float mutatedAngle = mutatedCutter.bladeAngle + mutationValue +
    UnityEngine.Random.Range(-mutationRange * 0.5f, mutationRange * 0.5f);
            // Ako zelimo da kut ostrice ostane unutar odredjenog raspona:
            mutatedAngle = Mathf.Clamp(mutatedAngle, cutter.bladeAngle -
    mutationRange, cutter.bladeAngle + mutationRange);
            // Dodavanje rezultata mutacije u populaciju
            mutatedCutter.bladeAngle = (float)System.Math.Round(mutatedAngle, 3);
            mutatedCutters.Add(mutatedCutter);
        }
        return mutatedCutters;
    }

    public static List<Cutter> SortPopulationByQuality(CutterPopulation population)
    {
        List<Cutter> sortedCutters = population.cutters;
        sortedCutters.Sort((c1, c2) => {
            int qualityComparison = ((int)c2.cutResult.quality).CompareTo((int)c1.
    cutResult.quality);
            if (qualityComparison != 0) return qualityComparison;
            // Ako su opisne kvalitete jednake usporedi postotke
            float cuttingDiff1 = Mathf.Abs(50.0f - c1.cutResult.volumeChunk1) + Mathf
    .Abs(50.0f - c1.cutResult.volumeChunk2);
            float cuttingDiff2 = Mathf.Abs(50.0f - c2.cutResult.volumeChunk1) + Mathf
    .Abs(50.0f - c2.cutResult.volumeChunk2);

            return cuttingDiff1.CompareTo(cuttingDiff2);
        });
        return sortedCutters;
    }

```

Programski isječak 4.14: Metode *SortPopulationByQuality* i *ExecuteMutation* koje služe za sortiranje populacije prema kvaliteti rezova i izvođenje mutacije.

```
    public static List<Cutter> SelectBestCutters(CutterPopulation population)
    {
        // Sortiraj prema opisnoj ocjeni pa prema postocima
        List<Cutter> sortedCutters = SortPopulationByQuality(population);

        // Izaberi tri najbolja rezultata
        int count = Mathf.Min(3, sortedCutters.Count);
        return sortedCutters.GetRange(0, count);
    }

    public static void ExecuteCrossing(List<Cutter> chosenRobotCutters, List<Cutter>
    chosenPlayerCutters)
    {
        for (int i = 0; i < chosenRobotCutters.Count; i++) {
            // Stvaranje parova za krizanje
            // Moguce je najbolji s najboljim ili najbolji s najlosijim
            int correspondingPlayerCutterIndex = (chosenPlayerCutters.Count - 1) - i;
            Cutter playerCutter = chosenPlayerCutters[correspondingPlayerCutterIndex
    ];

            // Moguce je koristiti i nasumicno sparivanje neovisno o kvaliteti

            // Izvodjenje linerane interpolacije i stvaranje novog kuta
            float interpolatedAngle = Mathf.Lerp(chosenRobotCutters[i].bladeAngle,
    playerCutter.bladeAngle, 0.5f);

            // Stvaranje novog objekta s interpoliranim kutem i dodavanje u
    populaciju
            Cutter newCutter = new Cutter { bladeAngle = (float)System.Math.Round(
    interpolatedAngle, 3), cutResult = null };
            robotPopulation.AddCutter(newCutter);
        }
    }

```

Programski isječak 4.15: Metode *SelectBestCutters* i *ExecuteCrossing* koje služe za selekciju i izvođenje križanja.

4.4.3. Stvaranje i prikaz zapisa o izvođenju simulacije

Unutar nekih od prethodno prikazanih programskih isječaka nalaze se pojedine linije programskog koda zaslužne za dodavanje novih zapisa o izvođenju simulacije u liste ili datoteku. Primjer stvaranja novih zapisa nalazi se unutar Programskog isječka 4.10 u odjeljku rukovanja tekstom, gdje se koristi skripta ***PrecisionDisplay***.

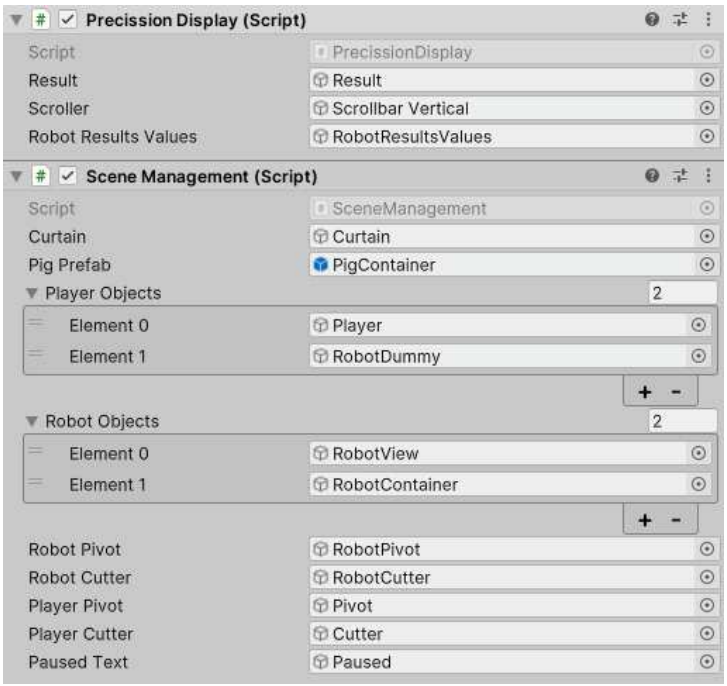
Metode skripte *PrecisionDisplay* omogućuju dodavanje novih zapisa u dnevnik (posebna datoteka) koji ostaju sačuvani i nakon izvođenja simulacije, te dodavanje zapisa o napredovanju koji su pohranjeni i vidljivi samo za vrijeme rada sustava.

Programski isječak 4.16 prikazuje neke od dijelova skripte koji su zaslužni za stvaranje zapisa i pridjeljivanje boje tekstu, ovisno o uspješnosti.

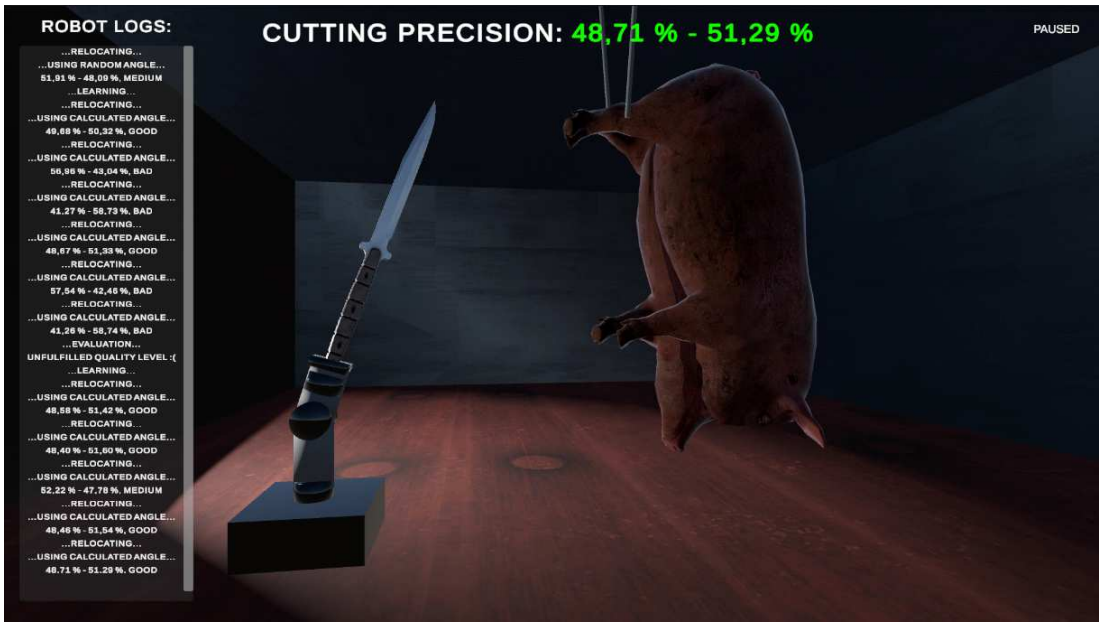
```
    void Update()
    {
        Result.gameObject.GetComponent<TextMeshProUGUI>().text = chunkVolume_1.
    ToString("0.00") + " %" + " - " + chunkVolume_2.ToString("0.00") + " %";

        StringBuilder stringBuilder = new StringBuilder();
        foreach (string result in robotResultsTextChunks) {
            stringBuilder.AppendLine(result);
        }
        robotResultsText = stringBuilder.ToString();
        if (!SceneManagement.simulationPaused) {
            ScrollToTheBottom();
        }
        RobotResultsValues.GetComponent<TextMeshProUGUI>().text = robotResultsText;
        AssignColor(GetGrade(chunkVolume_1, chunkVolume_2));
    }
    private void AssignColor(CutQuality grade)
    {
        switch (grade) {
            case CutQuality.GOOD:
                Result.gameObject.GetComponent<TextMeshProUGUI>().color = Color.green
    ;
                break;
            ...
        }
    }
    public static void AddLogLine(string log)
    {
        using (StreamWriter writer = new StreamWriter(filePath, true)) {
            writer.WriteLine(log);
        }
    }

```

Programski isječak 4.16: Metode iz skripte *PrecisionDisplay* koje služe za stvaranje zapisa o izvođenju simulacije.

Skripta *PrecisionDisplay* se, kao i navedena skripta *SceneManagement*, nalazi na glavnom objektu vrste *Canvas*, koji sadrži sve tekstualne objekte korisničkog sučelja. Konfiguracija tih skripti prikazana je na Slici 4.17, a primjer prikaza zapisa tijekom izvođenja simulacije prikazan je na Slici 4.18.



Slika 4.17: Konfiguracija skripti *PrecisionDisplay* i *SceneManagement* na objektu vrste *Canvas*.



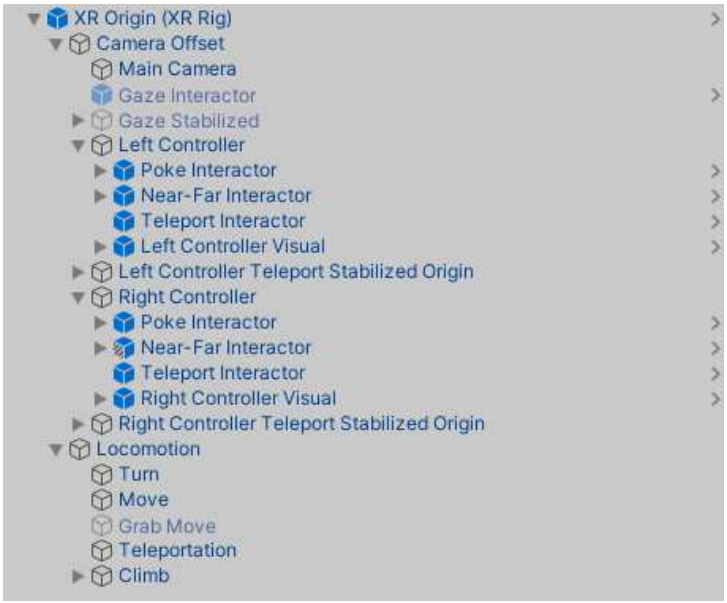
Slika 4.18: Prikaz zapisa o napredovanju tijekom izvođenja simulacije.

4.5. Integracija unutar virtualnog okruženja

Kao što se navodi unutar Poglavlja 3.1.4, jednostavni sustav nema omogućeno kretanje unutar prostora niti kompleksno upravljanje oštricom. Kako bi se ostvarila podrška za virtualna okruženja, potrebno je u postavkama projekta instalirati paket ***XR Plugin Management*** uz opciju *Open XR*. Dodatno, potrebno je instalirati skup alata *XR Interaction Toolkit*. Također, iz scene je potrebno ukloniti objekte koji predstavljaju glavnu kameru za igrača i robota te ih zamijeniti objektima tipa ***XR Rig*** iz spomenutog skupa alata, koji omogućuju osnovne funkcionalnosti upravljanja korisnikom unutar virtualnog okruženja.

Objekt *XR Rig* pruža sveobuhvatan skup komponenti za interakciju u virtualnom okruženju. Omogućava praćenje pozicije i rotacije glave korisnika te osigurava pravilno postavljanje kamere za realno prikazivanje perspektive. Komponenta *Input Action Manager* mapira različite unose, poput pokreta ruku i pritiska tipki, na specifične akcije, a *Locomotion System* podržava metode kretanja poput teleportacije ili kontinuiranog hodanja, omogućujući intuitivno kretanje unutar virtualnog prostora.

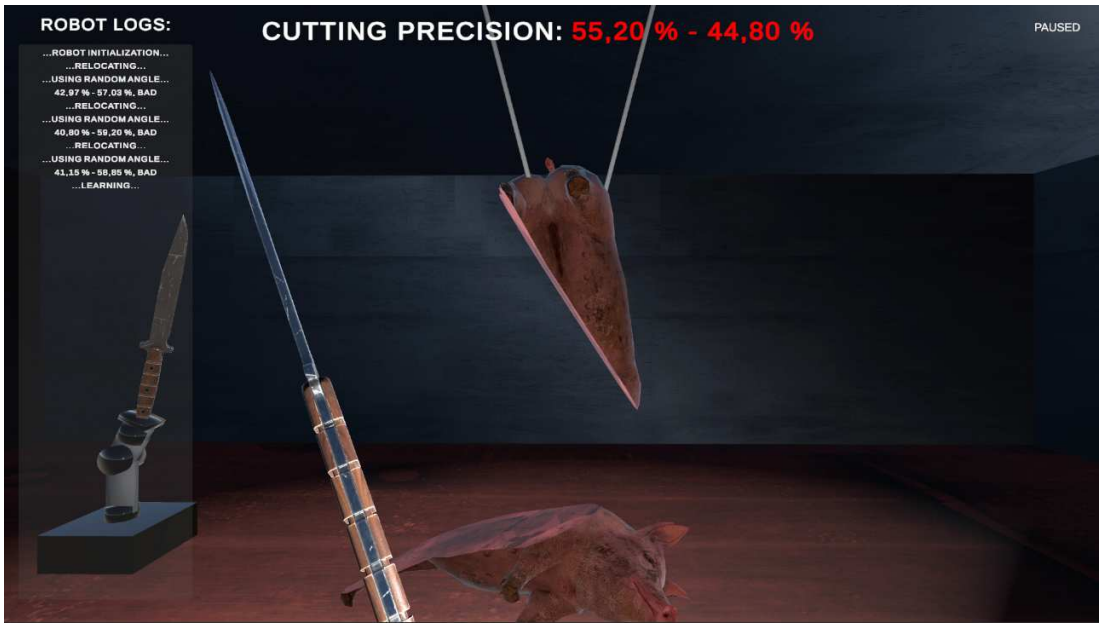
Kako bi se očuvala funkcionalnost simulacije, potrebno je ograničiti funkcionalnosti navedenih objekata na one neophodne za specifične zahtjeve projekta. Primjerice, potrebno je onemogućiti nepotrebne metode kretanja ili interakcije koje nisu relevantne za zadane ciljeve simulacije. Ideja kompleksnog sustava zahtijeva drugačiju integraciju podrške za VR, shodno proširenim funkcionalnostima.



Slika 4.19: Struktura objekta *XR Rig* unutar hijerarhije.

5. Rezultati i rasprava

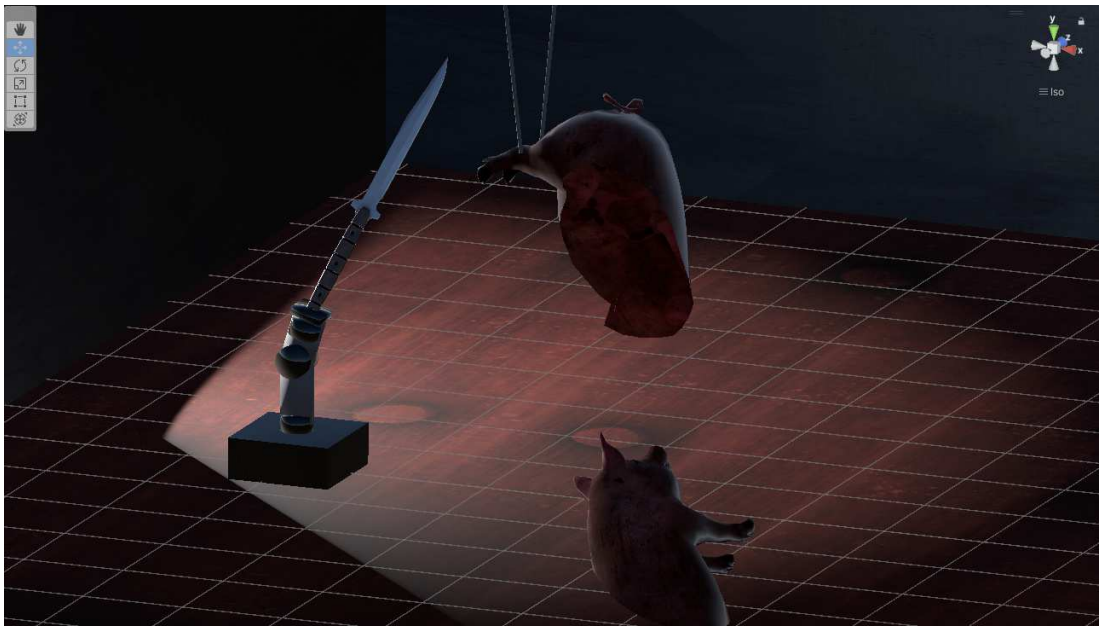
Korištenjem navedenih koncepata i tehnologija, konačno je ostvarena funkcionalna simulacija jednostavnog robotskog sustava za rezanje objekata. Slikama 5.1 i 5.2 prikazani su rezultati izvođenja preciznog i nepreciznog reza od strane korisnika. Na slikama su, osim prerezanog objekta, vidljive povratne informacije o preciznosti izvođenja reza.



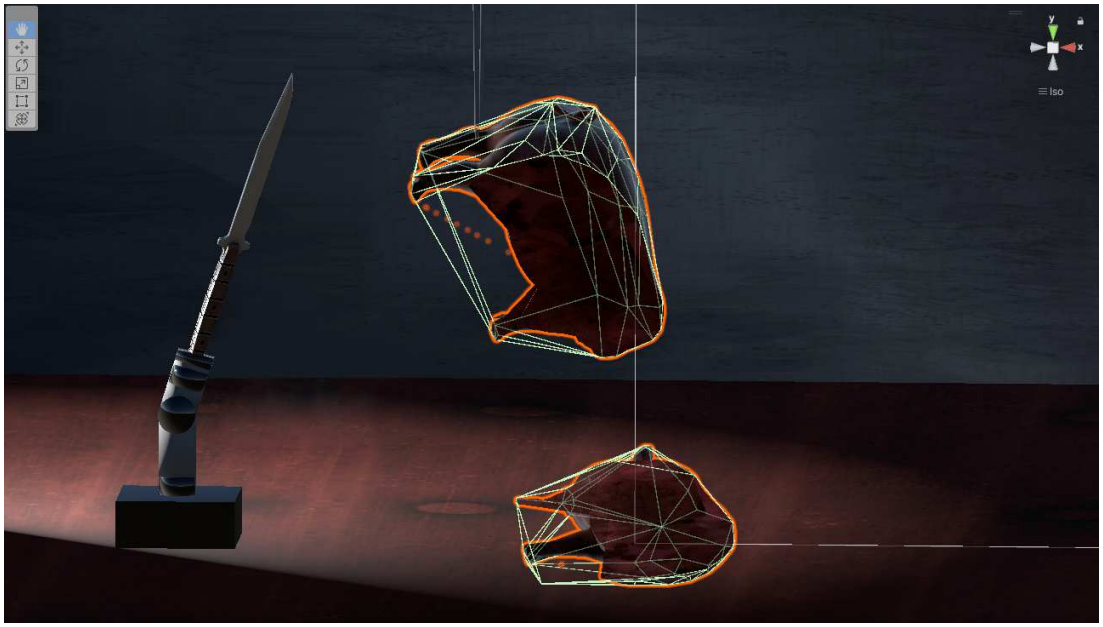
Slika 5.1: Rezultati izvođenja nepreciznog reza od strane korisnika.

Osim toga, Slika 5.3 prikazuje stanje unutar scene nakon izvođenja reza od strane robotske ruke, a na Slici 5.4 prikazana je komponenta *Mesh Collider* nad objektima koji su rezultat rezanja. Iz toga je moguće primijetiti kako su iz originalne mreže trokuta nastale dvije nove i jedinstvene mreže za stvorene objekte.

Slika 5.2: Rezultati izvođenja preciznog reza od strane korisnika.

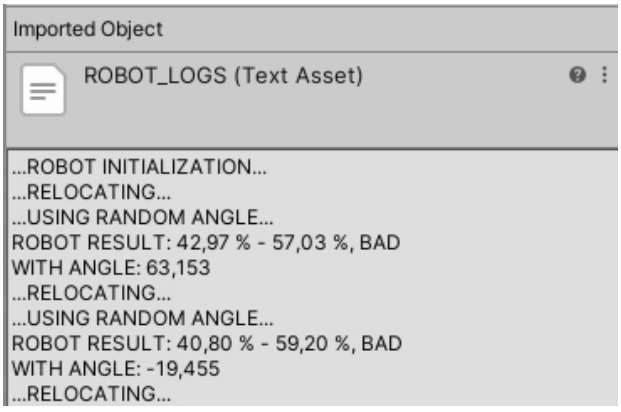


Slika 5.3: Stanje unutar scene nakon izvođenja reza od strane robotske ruke.



Slika 5.4: Nove mreže trokuta na objektima nastalim kao rezultat rezanja.

Ipak, značaj ovog sustava počiva na sposobnosti optimizacije preciznosti kroz faze izvođenja simulacije. Na zapisima o napretku simulacije sa Slike 4.18 moguće je primijetiti kako robotski sustav kontinuirano isprobava nasumične i izračunate kuteve kojima ostvaruje rezultate od lošijih prema boljima, ovisno o iteraciji u kojoj se trenutno nalazi. U nastavku ovog poglavlja razmatra se konvergencija robotskih rezultata na temelju različitih konfiguracija sustava za optimizaciju. Za usporedbe konfiguracija korišten je dnevnik zapisa u obliku datoteke pod nazivom *ROBOT_LOGS*, koja sadrži konkretne informacije o svim kutevima prije i nakon pojedine faze genetičkog algoritma.



Slika 5.5: Prikaz tekstualnog objekta *ROBOT_LOGS* koji sadrži potpune informacije o cijelom tijeku izvođenja simulacije.

5.1. Konvergencija rezultata

Konvergencija rezultata unutar genetičkog algoritma predstavlja ključni aspekt optimizacije performansi simuliranog robotskog sustava za rezanje objekata. Kroz različite faze iteracija, algoritam teži postizanju što preciznijih rezova, oslanjajući se na nekoliko bitnih parametara i metoda koje utječu na rezultate. U ovom poglavlju detaljno se razmatraju različite vrste konvergencije, utjecaj početnih uvjeta te uloga mutacije u procesu konvergencije.

1. **Konvergencija kod selekcije i križanja**
   Prilikom procesa selekcije i križanja, važno je uzeti u obzir način na koji su odabrani rezultati spareni, jer to može uvelike utjecati na brzinu učenja sustava. Postoji nekoliko osnovnih pristupa sparivanju:
   - **Nasumično sparivanje**: U ovom pristupu, rezovi se sparuju bez obzira na njihovu kvalitetu. Ovo može dovesti do širokog raspona rezultata jer se spajaju i dobri i loši rezovi, što može usporiti proces konvergencije. Nasumično sparivanje omogućava veću genetsku raznolikost, ali može zahtijevati više iteracija da bi se postigli optimalni rezultati.
   - **Sparivanje boljih s lošijima**: Ovo može ubrzati konvergenciju jer se dobre karakteristike brže prenose na sljedeće generacije. Međutim, postoji rizik da se sustav prerano zaglavi u lokalnom optimumu.
   - **Sparivanje boljih s boljima**: Sparivanje najboljih rezova među sobom može brzo dovesti do vrlo dobrih rješenja, ali također povećava rizik od lokalnih optimuma i smanjenja genetske raznolikosti.

2. **Uloga uniformnosti rezultata korisnika za konvergenciju**
   Na konvergenciju rezultata također mogu utjecati akcije korisnika. S obzirom na to da korisnik ima ulogu učenja sustava, smatra se da rezovi koje on izvodi moraju biti izvedeni sa što većom i stabilnijom razinom preciznosti. Ako korisnik izvede tri identična reza, sustav ima jasnu smjernicu za optimizaciju, što može ubrzati konvergenciju. Suprotno tome, ako korisnik napravi tri različita reza, sustav se suočava s većim izazovom jer mora optimizirati različite ishode. Ovo povećava potrebu za dodatnim iteracijama kako bi se pronašla optimalna rješenja za svaku vrstu reza.

3. **Utjecaj početnih kuteva na konvergenciju**
   Već je poznato da sustav na početku rada inicijalizira početnu populaciju rezova s nasumičnim kutevima. Način na koji se određuju nasumični kutevi također utječe na konvergenciju rezultata. Ako su početne vrijednosti izrazito loše, sustavu treba više iteracija da bi interpolirao do dobrih rješenja. U ovom slučaju može biti korisno uvesti dodatne parametre za ograničavanje kuta rezanja, kako bi se smanjila količina nepotrebnih iteracija i ubrzao proces konvergencije. Ako su početne vrijednosti postavljene relativno blizu optimalnih rješenja, proces konvergencije je brži, s manje potrebnih iteracija.

4. **Uloga mutacije u konvergenciji**
   Proces mutacije je ključni faktor koji utječe na konvergenciju jer dodaje element nasumičnosti i omogućava istraživanje većeg prostora mogućih rješenja. Veća razina nasumičnosti povećava prostor istraživanja, što može dovesti do ranog pronalaženja dobrih rješenja koja se kasnije koriste za križanje. Međutim, prevelika nasumičnost može usporiti proces konvergencije jer se sustav kontinuirano udaljava od potencijalno dobrih rješenja. Povećanje prostora istraživanja može omogućiti pronalazak inovativnih rješenja koja inače ne bi bila otkrivena. Ipak, postoji rizik da se sustav zaglavi u lokalnom optimumu ako prerano pronađe dobra rješenja i previše se oslanja na njih u kasnijim iteracijama.

5. **Utjecaj veličine inicijalne populacije na konvergenciju**
   Povećani broj inicijalnih rezova, bilo od strane robota ili korisnika, može različito utjecati na konvergenciju. Veći broj inicijalnih rezova omogućuje algoritmu bolji uvid u prostor rješenja, ali može zahtijevati više vremena za procesiranje i analizu. Istovremeno, može povećati šanse za pronalaženje optimalnih rješenja u ranoj fazi.

5.2. Prednosti i nedostatci brze konvergencije

Brza konvergencija ima svoje prednosti i nedostatke, koje je važno razmotriti kako bi se postigla ravnoteža između brzine i kvalitete rješenja. Jedna od glavnih prednosti brze konvergencije je ušteda vremena i računalnih resursa. Brza konvergencija omogućava postizanje optimalnih rješenja u kraćem periodu, što je posebno korisno u sustavima gdje je vrijeme ključni faktor. U industrijskim okruženjima, brza konvergencija može značiti povećanje produktivnosti i smanjenje troškova operacija. Također, omogućava brzu prilagodbu promjenama u ulaznim parametrima ili uvjetima

rada, što je ključno za održavanje visoke razine učinkovitosti u dinamičnim okruženjima.

Međutim, brza konvergencija nosi i određene rizike. Jedan od glavnih nedostataka je mogućnost zaglavljivanja u lokalnom optimumu. Kada algoritam prerano konvergira, postoji rizik da se rješenja stabiliziraju na suboptimalnim vrijednostima, a da se pritom ne istraži cijeli prostor mogućih rješenja. Ovo može rezultirati rješenjima koja su manje precizna nego što bi mogla biti, što je posebno problematično u kompleksnim problemima s više lokalnih optimuma. Još jedan nedostatak brze konvergencije je smanjenje genetske raznolikosti unutar populacije. Kako algoritam brzo konvergira, genetska raznolikost može se smanjiti, što ograničava sposobnost sustava da istraži nova i potencijalno bolja rješenja. Ovo može dovesti do stagnacije i smanjene prilagodljivosti sustava na promjene u okolini ili ulaznim parametrima. Važno je napomenuti da balansiranje između brzine konvergencije i kvalitete zahtijeva pažljivo podešavanje parametara algoritma, kao što su stopa mutacije, veličina populacije, kriteriji za selekciju i nasumičnost početne populacije rezova.

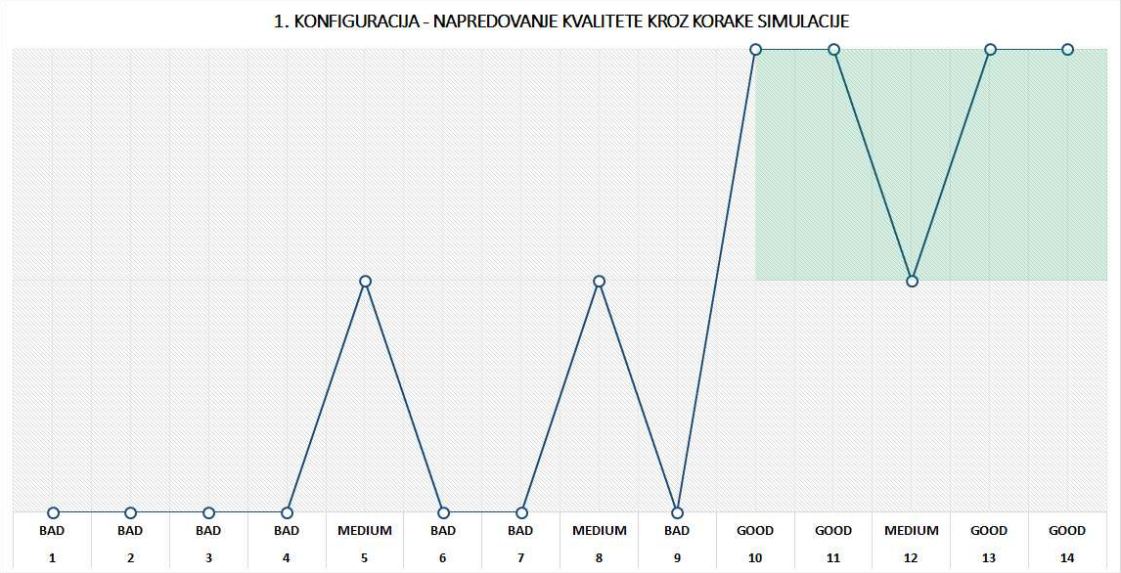
Zaključno, brza konvergencija genetičkog algoritma može značajno povećati učinkovitost i produktivnost sustava za rezanje objekata, ali nosi i rizike koji mogu ugroziti kvalitetu konačnih rješenja. Ključ je u pronalaženju ravnoteže koja omogućava brzo postizanje kvalitetnih rješenja uz održavanje dovoljno genetske raznolikosti za dugoročnu prilagodljivost i optimizaciju.

5.3. Analiza grafova napredovanja kvalitete različitih konfiguracija genetičkog algoritma

Obradom zapisa o izvođenju simulacije pri različitim konfiguracijama genetičkog algoritma, koje utječu na konvergenciju (Poglavlje 5.1), nastali su grafovi napredovanja kvalitete rezova kroz korake simulacije. Analizom grafova cilj je prikazati utjecaj različitih konfiguracija na brzinu i stabilnost konvergencije sustava prema zadovoljavajućoj razini preciznosti. U nastavku ovog poglavlja detaljno se analiziraju sve tri konfiguracije te se uspoređuju rezultati kako bi se identificirali ključni faktori koji doprinose učinkovitosti konvergencije. Na grafovima je obojenim pravokutnikom naznačeno stanje unutar kojeg se razina preciznosti smatra zadovoljavajućom.

1. **Konfiguracija**

   Ova konfiguracija koristi postavke koje uključuju sparivanje najboljih rezultata s najlošijim kod križanja, različite rezove korisnika, inicijalni vrlo široki raspon kuteva te relativno mali faktor mutacije. U prvih nekoliko koraka simulacije (od 1. do 4. koraka), kvaliteta je loša. Međutim, već na 5. koraku dolazi do oporavka kvalitete, gdje kvaliteta prelazi na *MEDIUM*. Ova promjena ukazuje na početnu fazu optimizacije, gdje algoritam pokušava pronaći bolja rješenja unutar širokog raspona mogućnosti naspram početnih nasumičnih kuteva. Daljnji tijek simulacije pokazuje oscilacije između različitih razina kvalitete. Iako algoritam povremeno pronalazi bolja rješenja, ona nisu uvijek stabilna. To je rezultat istraživanja prostora rješenja s pomoću različitih rezova korisnika i mutacije, koji uvode varijabilnost u proces optimizacije. U konačnici, algoritam pronalazi dobra rješenja te je moguće primijetiti kako se željena razina preciznosti postiže, iako je proces konvergencije spor i varira u ranim fazama.

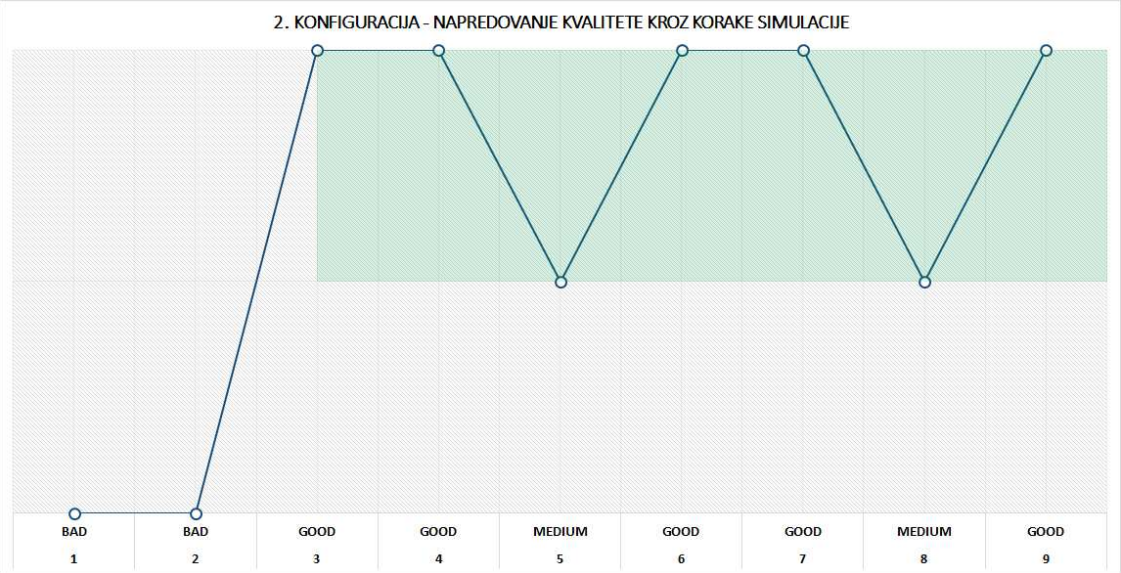


Slika 5.6: Napredovanje kvalitete kroz korake simulacije koristeći prvu konfiguraciju genetičkog algoritma.

2. **Konfiguracija**

   Ova konfiguracija koristi postavke koje uključuju sparivanje najboljih rezultata s najboljima, identične rezove korisnika, smanjeni raspon inicijalnih kuteva oštrice te relativno mali faktor mutacije. Kvaliteta vrlo brzo napreduje od najlošije prema najboljoj unutar prvih nekoliko koraka simulacije. Nakon toga, kvaliteta ostaje stabilna na visokoj razini, uz oscilacije uzrokovane mutacijom. To

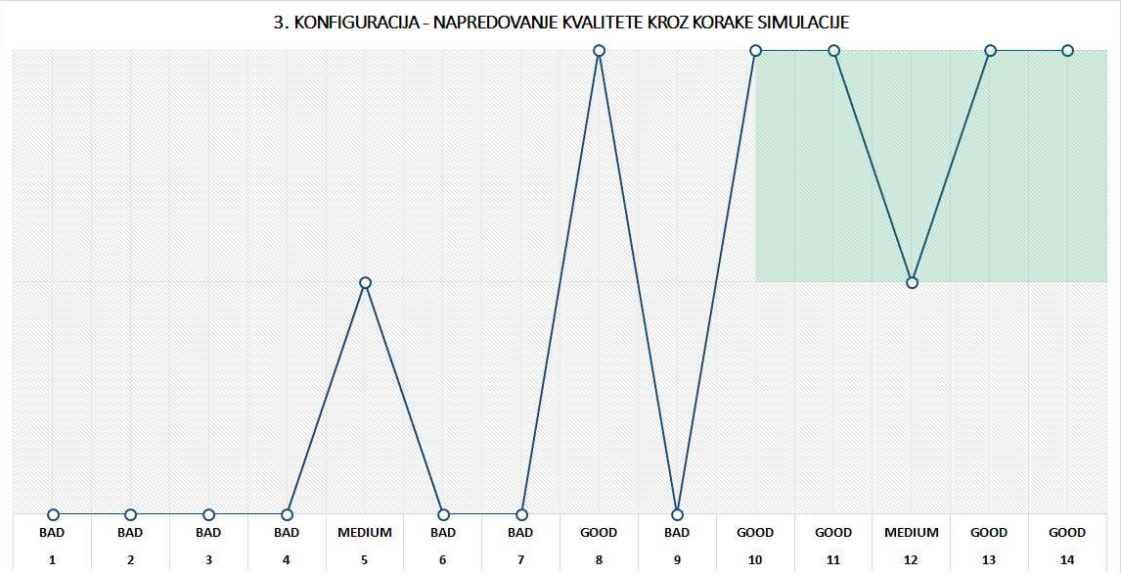
sugerira da sparivanje najboljih rezova s najboljima i smanjeni raspon kuteva omogućuju algoritmu da vrlo brzo pronađe prihvatljiva rješenja. Ova konfiguracija pokazuje najbržu konvergenciju ostvarenu u najmanje koraka, međutim, zbog toga se može smatrati "naivnom".



Slika 5.7: Napredovanje kvalitete kroz korake simulacije koristeći drugu konfiguraciju genetičkog algoritma.

3. **Konfiguracija**

   Ova konfiguracija koristi postavke koje uključuju sparivanje najboljih rezultata s najlošijima, različite rezove korisnika, inicijalni vrlo široki raspon kuteva te povećani faktor mutacije. Ponovno, u prvih nekoliko koraka simulacije, kvaliteta je loša te na 5. koraku dolazi do poboljšanja kvalitete zbog početne faze optimizacije (kao i u prvoj konfiguraciji). No, ono što se ističe kod ove konfiguracije je značajna varijabilnost kvalitete u sredini simulacije. Primjerice, kvaliteta na 9. koraku iz *GOOD* ponovno naglo pada na *BAD*, a zatim se opet vraća na *GOOD*. Ove oscilacije ukazuju na to da povećani faktor mutacije uz široki raspon kuteva uvodi veliku nestabilnost u proces optimizacije, otežavajući algoritmu održavanje visoke razine kvalitete u ranim fazama. Zadovoljavajuća razina preciznosti se ponovno ostvaruje tek u 10. koraku.



Slika 5.8: Napredovanje kvalitete kroz korake simulacije koristeći treću konfiguraciju genetičkog algoritma.

Zaključno, analiza napredovanja kvalitete kroz korake simulacije za tri različite konfiguracije genetičkog algoritma pokazuje značajne razlike u brzini i stabilnosti konvergencije. Prva konfiguracija pokazuje sporiji proces konvergencije s oscilacijama između različitih razina kvalitete. Postepeno se dolazi od loših prema dobrim rješenjima uz manja odstupanja u konačnoj fazi uzrokovanih mutacijom. Druga konfiguracija postiže najbržu konvergenciju, brzo prelazeći na visoku kvalitetu koja ostaje stabilna uz oscilacije uzrokovane mutacijom. Kvaliteta nakon dostizanja zadovoljavajuće razine više ne pada na najnižu razinu. Kod treće konfiguracije, iako dolazi do početnog poboljšanja kvalitete, sredina simulacije pokazuje značajnu varijabilnost s velikim promjenama kvalitete. Povećani faktor mutacije i široki raspon kuteva uzrokuju nestabilnost, otežavajući algoritmu održavanje visoke razine kvalitete u ranim fazama. Za postizanje veće učinkovitosti sustava potrebno je odrediti konfiguraciju koja istovremeno ostvaruje što bržu konvergenciju uz što manji gubitak varijabilnosti, kako sustav ne bi zaglavio u lokalnom optimumu.

6. Zaključak

Unutar ovog rada pružen je pregled teorije i tehnoloških aspekata tehnologije digitalnih dvojnika, ističući njihovu važnost u suvremenom inženjerskom okruženju. Riječ je o pristupu koji omogućava precizno modeliranje stvarnih robota unutar digitalnog prostora, pružajući osnovu za inovativne pristupe testiranju, optimizaciji i razvoju robotskih sustava. Brojni izazovi, poput sigurnosti, kibernetičke sigurnosti i pouzdanosti, neizostavan su element budućnosti ovakve tehnologije.

Također, detaljno je predstavljena funkcionalnost jednostavnog robotskog postrojenja, koje je korišteno za rezanje 3D objekata. Opisani su svi elementi sustava, uključujući model robotske ruke, metode rezanja objekata i optimizaciju preciznosti rezanja s pomoću genetičkog algoritma. Posebno je naglašena važnost integracije sustava unutar virtualne stvarnosti, koja korisnicima pruža intuitivne metode za obuku, održavanje i upravljanje robotskim sustavom. Nakon što su opisane tehnike implementacije ovog sustava, provedena je analiza konvergencije rezultata, prednosti i nedostataka brze konvergencije, te analiza grafova napredovanja kvalitete različitih konfiguracija genetičkog algoritma. Rezultati su pokazali da različite konfiguracije imaju značajan utjecaj na brzinu i stabilnost konvergencije. Brza konvergencija može značajno povećati učinkovitost sustava, no nosi rizike poput smanjenja genetske raznolikosti i mogućnosti zaglavljivanja u lokalnim optimumima. Analiza grafova napredovanja kvalitete različitih konfiguracija pokazala je da pažljivo podešavanje parametara algoritma može osigurati ravnotežu između brzine konvergencije i kvalitete rješenja.

Buduća istraživanja trebala bi se usmjeriti na razvoj kompleksne inačice sustava, koja uključuje poboljšanje arhitekture i korištenih algoritama u svrhu ostvarenja različitih konkretnih industrijskih primjena. Korištenje naprednih tehnologija uključuje brojne implementacijske izazove, ali rezultat bi trebao predstavljati sustav digitalnog dvojnika viših razina učinkovitosti, sigurnosti i praktičnosti u primjeni.

LITERATURA

[1] C. Pylianidis, S. Osinga, and I. N. Athanasiadis, "Introducing digital twins to agriculture," *Computers and Electronics in Agriculture*, vol. 184, p. 105942, 2021.

[2] S. Zacher, "Digital twins for education and study of engineering sciences," *International Journal on Engineering, Science and Technology*, vol. 2, no. 2, pp. 61–69, 2020.

[3] M. Singh, E. Fuenmayor, E. P. Hinchy, Y. Qiao, N. Murray, and D. Devine, "Digital twin: Origin to future," *Applied System Innovation*, vol. 4, no. 2, p. 36, 2021.

[4] B. Ratawal, "Virtual twins for industrial robotics," 2023. Pristupljeno: 2024-05-05.

[5] H. Van der Valk, H. Haße, F. Möller, M. Arbter, J.-L. Henning, and B. Otto, "A taxonomy of digital twins.," in *AMCIS*, 2020.

[6] M. Segovia and J. Garcia-Alfaro, "Design, modeling and implementation of digital twins," *Sensors*, vol. 22, no. 14, p. 5396, 2022.

[7] M. Grieves, "Physical twins, digital twins, and the apollo myth," 2022. Pristupljeno: 2024-05-06.

[8] G. S. Saini, A. Fallah, P. Ashok, and E. van Oort, "Digital twins for real-time scenario analysis during well construction operations," *Energies*, vol. 15, no. 18, p. 6584, 2022.

[9] N. Simchenko, S. Tsohla, I. Molchanov, and N. Molchanova, "Digital twins in industry: Benefits and traps," *Revista Inclusiones*, pp. 351–366, 2021.

[10] S. Suhail, R. Jurdak, and R. Hussain, "Security attacks and solutions for digital twins," *arXiv preprint arXiv:2202.12501*, 2022.

[11] M. Jacoby and T. Usländer, "Digital twin and internet of things—current standards landscape," *Applied Sciences*, vol. 10, no. 18, p. 6519, 2020.

[12] E. Zio and L. Miqueles, "Digital twins in safety analysis, risk assessment and emergency management," *Reliability Engineering & System Safety*, p. 110040, 2024.

[13] E. O. Popa, M. van Hilten, E. Oosterkamp, and M.-J. Bogaardt, "The use of digital twins in healthcare: socio-ethical benefits and socio-ethical risks," *Life sciences, society and policy*, vol. 17, pp. 1–25, 2021.

[14] P. Palensky, M. Cvetkovic, D. Gusain, and A. Joseph, "Digital twins and their use in future power systems," *Digital Twin*, vol. 1, p. 4, 2022.

[15] L. S. Dalenogare, G. B. Benitez, N. F. Ayala, and A. G. Frank, "The expected contribution of industry 4.0 technologies for industrial performance," *International Journal of production economics*, vol. 204, pp. 383–394, 2018.

[16] Z. Huang, Y. Shen, J. Li, M. Fey, and C. Brecher, "A survey on ai-driven digital twins in industry 4.0: Smart manufacturing and advanced robotics," *Sensors*, vol. 21, no. 19, p. 6340, 2021.

[17] S. Meloeny, "What is industry 4.0?." `https://www.calsoft.com/what-is-industry-4-0/`. 2022. Pristupljeno: 2024-05-12.

[18] M. Neumann, "Why cloud-based digital twins are the future," 2024. Pristupljeno: 2024-05-12.

[19] Y.-K. Wong, "Digital twins built on a cloud platform bring much-needed benefits for businesses," 2023. Pristupljeno: 2024-05-12.

[20] B. Ibryam, "The role of digital twins in unlocking the cloud's potential," 2023. Pristupljeno: 2024-05-12.

[21] `https://www.shgongboshi.com/sale-14003447-abb-irb-120-6-axis-industrial-robotic-arm-for-flexible-and-compact-production-industrial-robot-arm-p.html`. Pristupljeno: 2024-05-17.

[22] S. Kucuk and Z. Bingul, *Robot kinematics: Forward and inverse kinematics*. INTECH Open Access Publisher London, UK, 2006.

[23] https://www.mathworks.com/discovery/inverse-kinematics.html. Pristupljeno: 2024-05-18.

[24] M. M. U. Atique, M. R. I. Sarker, and M. A. R. Ahad, "Development of an 8dof quadruped robot and implementation of inverse kinematics using denavit-hartenberg convention," *Heliyon*, vol. 4, no. 12, 2018.

[25] S. Tabandeh, C. Clark, and W. Melek, "A genetic algorithm approach to solve for multiple solutions of inverse kinematics using adaptive niching and clustering," in *2006 IEEE International Conference on Evolutionary Computation*, pp. 1815–1822, IEEE, 2006.

[26] S.-Y. Lee, S.-H. Kweon, and S.-H. Yoon, "An effective method for slicing triangle meshes using a freeform curve," *Mathematics*, vol. 12, no. 10, p. 1432, 2024.

[27] L. M. Schmitt, "Theory of genetic algorithms," *Theoretical Computer Science*, vol. 259, no. 1-2, pp. 1–61, 2001.

[28] T. V. Mathew, "Genetic algorithm," *Report submitted at IIT Bombay*, vol. 53, 2012.

[29] https://www.slideserve.com/morna/primjena-genetskih-algoritama-za-optimiranje-pogonskih-stanja-razdjelnih-mre-a. Pristupljeno: 2024-05-19.

[30] https://assetstore.unity.com/packages/tools/modeling/triangle-separator-90709. Pristupljeno: 2024-05-26.

DIGITALNI DVOJNIK ROBOTSKOG POSTROJENJA ZA SJEČENJE OBJEKATA S POMOĆU KORISNIČKE KOREKCIJE PUTEM VIRTUALNE STVARNOSTI

Sažetak

Digitalni dvojnici su ključni za precizno modeliranje robota unutar digitalnog prostora. Oni omogućavaju testiranje i optimizaciju performansi prije praktične uporabe, čime se značajno ubrzava razvoj i smanjuje rizik. U ovom radu opisan je razvoj simulacije jednostavnog robotskog sustava namijenjenog sječenju 3D objekata. Primijenjeni su algoritmi za optimizaciju preciznosti, uz mogućnost korisničkog unosa u virtualnom okruženju. Provedena je analiza rezultata i evaluacija učinkovitosti sustava, s posebnim naglaskom na izazove i potencijale za konkretnije primjene u industriji. Ovo istraživanje naglašava važnost integriranih pristupa u napretku razvoja i primjeni robotskih sustava.

Ključne riječi: Unity, digitalni dvojnici, genetički algoritam, virtualna stvarnost

DIGITAL TWIN OF A ROBOTIC SYSTEM FOR CUTTING OBJECTS WITH USER CORRECTION THROUGH VIRTUAL REALITY

Abstract

Digital twins are crucial for precise modeling of robots within a digital space. They enable testing and performance optimization before practical use, significantly accelerating development and reducing risk. This paper describes the development of a simulation for a simple robotic system designed for cutting 3D objects. Algorithms for precision optimization were applied, along with the capability for user input in a virtual environment. An analysis of the results and an evaluation of system effectiveness were conducted, with a special focus on challenges and potentials for more concrete industrial applications. This research highlights the importance of integrated approaches in the advancement of the development and application of robotic systems.

Keywords: Unity, digital twins, genetic algorithm, virtual reality